Intel® Manycore Platform Software Stack (Intel® MPSS)

User's Guide

Revision: 3.4

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

Disclaimer and Legal Information

Revision History

Release Number	Description	Revision Date
3.3	Revised section 14.3.9, 2.2, 2.3. Created sections 11.4, 11.4.1, and 11.4.2, edited and updated data for release 3.4. , Edited section 14.4.5.4, Added Conventions and Symbols table to Section 1.3. Edited sections 7.5, 7.6, 7.7, 7.10, 7.12, and 9.1. Corrected figure 2 in Section 13. Edited Section 7.16., Reviewed and corrected notes where necessary. Added Chapter 18 General Services Tutorial. Edited sections 14.4.5.3, 15.3.12.2, and 15.3.13, Replaced Sections 2.1 – 2.5	April 2014
3.3	Minor edits Chapter 2,9 , Title changes to Chapter 2 section headings,	May 2014
3.3	Minor edits throughout, Adjusted User Prompts to reflect card vs. host.	June 2014
3.3	7.5 added method 1 and method 2 to differentiate between the two options. Added to 7.6 example [host]# micctrl --rpmdir=<path to K1OM directory> Modified section 7.10 instructions to fix holes and broken directory structures. Fixed several indententations, and removed a few page breaks that were no longer needed through out section 7. Fixed host based authentication instructions. Still need to be verified. changed several commands that spanned two lines / into single line commands. Document needs fonts, font size, and indentation cleanup. Most changes have been verified by two or more people. It's not perfect, but we are out of itme. Minor edits throughout the document.	July 2014
3.4	Moved footnote reference to beginning of the "service" command. Multiple edits for 3.4.	August 2014

Contents

List of Figures

List of Tables

1 Overview

This User's Guide is for the Intel® Manycore Platform Software Stack (Intel® MPSS) build revision 3.4. Intel MPSS 3.4 encompasses the Linux* driver and supporting tools from gold_update_3 forward. This document contains advanced configuration information and procedures.

NOTE: If using sudo, the full path of the command is required (or run as root):

```
[host]$ sudo /usr/sbin/micctrl --<option>
```

Add */usr/sbin* and */sbin* to the path to avoid entering the full path of the command:

```
export PATH=$PATH:/usr/sbin:/sbin
```

Optionally, add */usr/sbin* and */sbin* to the user's **.bashrc** file to eliminate the need to set the path in each session.

Warning: To begin Intel MPSS installation, you must start with **Section 2** of the *Intel MPSS Readme* document -- **DON'T START HERE**. That Section also includes flash update and SSH access instructions.

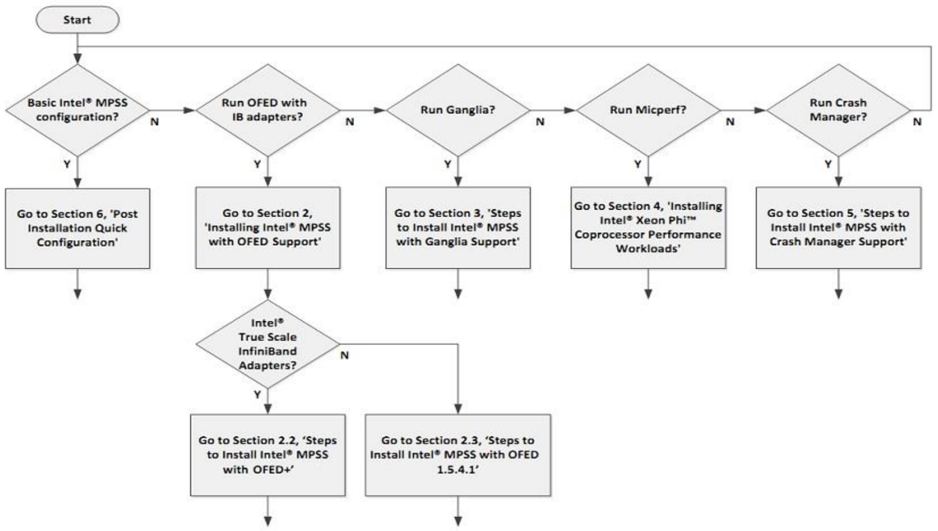
Start
Basic Intel® MPSS configuration?
N
Y
Run OFED with IB adapters?
N
Y
Run Ganglia?
N
Y
Run Micperf?
N
Y
Run Crash Manager?
N
Y
Go to Section 6, 'Post Installation Quick Configuration'
Go to Section 2, 'Installing Intel® MPSS with OFED Support'
Go to Section 3, 'Steps to Install Intel® MPSS with Ganglia Support'
Go to Section 4, 'Installing Intel® Xeon Phi™ Coprocessor Performance Workloads'
Go to Section 5, 'Steps to Install Intel® MPSS with Crash Manager Support'
Intel® True Scale InfiniBand Adapters?
N
Y
Go to Section 2.2, 'Steps to Install Intel® MPSS with OFED+'
Go to Section 2.3, 'Steps to Install Intel® MPSS with OFED 1.5.4.1'

Figure 1 Common Intel MPSS Installation/Configuration Workflows

1.1 Technology Previews

This release includes a technology preview of CCL-Direct for kernel mode clients. This includes an experimental version of kernel mode InfiniBand verbs and RDMA_CM and an experimental version of IPoIB. This experimental version of CCL-Direct kernel mode support has also been tested with a Lustre client. Refer to the document (/usr/share/doc/ofed-driver-*/lustre-phi.txt) for information on how to build and install a Lustre client on the Intel® Xeon Phi™ coprocessor. This preview only supports the Mellanox* mlx4 driver and associated hardware and currently only supports the OFED-1.5.4.1 version of OFED software. Support for the new Mellanox* mlx5 driver and the OFED-3.5-MIC release will be added in a future version of Intel MPSS.

1.2 Supported Intel® Tools

The following table lists compatible version numbers of Intel® tools that are supported with Intel MPSS release 3.4.

Table 1 Intel® Tools Supported with Intel MPSS Release 3.4

Name of Tool	Supported Version
Intel® Composer XE	SP1 2013
Intel® C++ Compiler	14.0.0
Intel® Integrated Performance Primitives for Linux*	8.0.1
Intel® Math Kernel Library for Linux*	11.1.0
Intel® Threading Building Blocks for Linux*	4.2.0
Intel® VTune™ Amplifier XE	2013_update6
Intel® SEP	Sep3_10 (3.x fix)

The intel-composerxe-compat-k1om RPM temporarily provides backward compatibility to the icc compiler before version 14.0.0 via the soft links to /opt/mpss/[version number]/sysroot. It is not a separate set of binaries for the x86_64-k1om-linux architecture used in Intel MPSS 2.1.6720.

1.3 Conventions and Symbols

Table 2 Conventions and Symbols lists conventions used in this docuement.

Table 2 Conventions and Symbols

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, command, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Used to highlight the elements of a graphical user interface such as buttons and menu names.
This type style	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder. Also used to indicate new terms, URLs, email addresses, filenames, and file extensions.
[*items*]	Indicates that the items enclosed in brackets are optional.
{ *item* | *item* }	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.
\ (backslash)	Indicates continuation of a command onto the next line in the document.
micN	Denotes a name such as mic0, mic1, etc. where N=0,1, 2, … For example, the file name micN.conf denotes file names mic0.conf, mic1.conf, etc.
[host] $	Denotes a command entered on the host with user privileges.
[host] #	Denotes a command entered on the host with administrative (root) privileges.
[micN] $	Denotes a command entered on a coprocessor N with user privileges.
[micN] #	Denotes a command entered on a coprocessor N with administrative privileges.
*	Other names and brands may be claimed as the property of others.
<command>[1]	The number [1] is a hyperlink to a footnote pertaining to this command.

[1] When running Intel® MPSS on RHEL 7.0, please replace:
service mpss unload
with
systemctl disable
modprobe -r mic
For all other service commands, replace:
service <daemon> <action>
with
systemctl <action> <daemon>

2 Installing Intel MPSS with OFED Support (optional)

2.1 Requirements

1) OFED-1.5.4.1 typical requirements
   - gcc-c++
   - bison
   - flex
   - tk
   - tcl-devel
   - zlib-devel
   - libstdc++-devel
   - libgfortran43
2) Red Hat* Enterprise Linux* (RHEL) specific:
   - gcc
   - rpmbuild
   - kernel-devel
3) SUSE* Linux* Enterprise Server (SLES) specific:
   - gcc
   - rpmbuild
   - kernel-default-devel

NOTE: libgfortran43 is not included in SLES 11.2/11.3 x86_64 base install DVDs or the SDK, however libgfortran46 is included. See Section 2.3 Steps to Install Intel MPSS using OFED 1.5.4.1 for instructions for modifying the OFED install script to allow the installation to work with libgfortran46.

Several different OFED distributions are supported. Select one which matches hardware and software requirements and install using instructions from the accompanying section.

Each OFED distribution supports a subset of the Intel MPSS supported OS distros; most support SLES 11 SP2/3 and RHEL 6.3/4/5. RHEL 7 is only supported by 3.5-2-MIC as of this writing. Check the release notes for the exact supported distros.

Table 3 OFED Distribution versus Supported Features

OFED Distribution (install section)	Mlx4 (kernel mode)	Mlx5 (kernel mode)	Intel offload	Scif (native mode)	ccl-proxy
Ofed+ (2.2)	No (no)	No (no)	Yes	Yes	No
Ofed 1.5.4.1 (2.3)	Yes (yes)	No (no)	Yes	Yes	Yes
Ofed 3.5-2 MIC (2.4)	Yes (yes)	Yes (no)	Yes	Yes	Yes
Mellanox* Ofed 2.1/2.2 (2.6)	Yes (yes)	Yes (no)	Yes	Yes	Yes

Several different OFED distributions are supported. Select one which matches hardware and software requirements and install using instructions from the accompanying section.

2.2 Steps to Install Intel MPSS using OFED+

OFED+ is the Intel® True Scale Fabric Host Channel Adapter Drivers and Software stack

WARNING: Installing OFED+ support will replace the OFED components in your standard distribution. This section describes the steps to install Intel® True Scale Fabric Host Channel Adapter Drivers and Software stack (OFED+), an enhanced implementation of OFED that supports Intel® True Scale Fabric Host Channel Adapters (HCA), and enables communication between an Intel® Xeon Phi™ coprocessor and an Intel® True Scale Fabric HCA. This installation may overlay some of the RDMA/InfiniBand* components in your Red Hat* or SUSE* distribution. As a result, the Linux* kernel will not load kernel mode software that was built against the Red Hat* or SUSE* RDMA/InfiniBand* software in your distribution. This may require that you rebuild such software against the installed OFED, or obtain a version of the software that was so built. For example, an implementation of the Lustre* file system that was built against a Red Hat* or SUSE* distribution will not be loaded by the Linux* kernel, and must be rebuilt against the installed OFED.

NOTE: User mode applications will not need to be rebuilt due to this installation.

The following installation should be performed on any compute node in which an Intel® True Scale Fabric HCA is installed.

After a successful installation, an 'ibv_devices' command issued on the host will show both 'qib0' and 'scif_0', while an ibv_devices command issued on the Intel® Xeon Phi™ coprocessor will show only 'scif_0'.

NOTE: When running MPI in Symmetric mode with more than 16 processes per node, PSM_RANKS_PER_CONTEXT=<value> needs to be specified (the value can be 2,3 or 4 the default value is 1) so that the available 16 contexts can be shared by the ranks.

Intel® True Scale Fabric Host Channel Adapter Drivers and Software (OFED+) including the PSM library is available as a free download on http://downloadcenter.intel.com

Download and Installation Instructions for All Supported Versions of Linux 6.X.

NOTE: If using RHEL 7, please skip to Section 2.4

1) Go to http://downloadcenter.intel.com/
2) Under "Search Downloads" on the left half of the screen, type "True Scale" and hit "Enter".
3) Narrow the results by selecting the appropriate operating system.
4) Select the version of Intel® True Scale Fabric Host Channel Adapter Host Drivers and Software that supports your operating system. Details of the operating system can be found by clicking on a version and then clicking the "Release Notes (pdf)" link at the bottom of the section.

   NOTE: Intel® True Scale Fabric Host Channel Adapter Host Drivers and Software versions 7.2.x and previous support OFED 1.5.4.1 while version 7.3 and higher support OFED 3.5-2 or later. The specific supported OFED version is listed in the *Detailed Description* section of the selected version.
5) Download the appropriate "IB-Basic" file as well as the related publications file.
6) The Software Installation Guide (IFS_FabricSW_InstallationGuide*.pdf) is contained in the Publications_HCA_SW*.zip download. *Chapter 4 Install OFED+ Host Software* in the *Software Installation Guide* provides detailed installation instructions.
7) After rebooting the system as recommended by the previous Install step, stop openibd service and ensure that openibd does not start automatically after every reboot:

```
[host]# [1]service openibd stop
[host]# chkconfig --level=123456 openibd off
```

8) If using the 7.2 OFED+ version, ensure kernel-ib-devel, kernel-ib, and dapl packages are not installed.

```
[host]# rpm -e kernel-ib-devel kernel-ib
[host]# rpm -e {dapl,dapl-{devel,devel-static,utils}}
```

   If using the 7.3 OFED+ version, ensure compat-rdma-devel, compat-rdma and dapl packages are not installed.

```
[host]# rpm -e compat-rdma-devel compat-rdma
[host]# rpm -e {dapl,dapl-{devel,devel-static,utils}}
```

9) If using yum to install Intel MPSS, it is also necessary to remove infinipath-libs and infinipath-devel prior to installing Intel MPSS:

```
[host]# rpm -e --nodeps --allmatches infinipath-libs \
infinipath-devel
```

```
Install Intel MPSS OFED.
[host]$ cd mpss-3.4
[host]$ cp ofed/modules/*`uname -r`*.rpm ofed
```

- Red Hat* Enterprise Linux*

  ```
  [host]# yum install ofed/*.rpm
  ```

- SUSE* Linux* Enterprise Server

  ```
  [host]# zypper install ofed/*.rpm
  ```

10) Install required PSM (Performance Scaled Messaging) libraries and drivers as follows:

- Red Hat* Enterprise Linux*

  ```
  [host]# yum install psm/*.rpm
  ```

- SUSE* Linux* Enterprise Server

  ```
  [host]# zypper install psm/*.rpm
  ```

11) To run MPI applications using Intel MPI over tmi fabric, the tmi.conf file should be copied to the Intel® Xeon Phi™ coprocessor using following procedure:

    a) Create a directory "etc" in "/var/mpss/common/" directory.

    b) Copy the tmi.conf file from <impi_install_dir>/etc64/ directory to /var/mpss/common/etc directory.

2.3 Steps to Install Intel MPSS using OFED 1.5.4.1

WARNING: This installation may overlay some of the RDMA/InfiniBand components in your distribution. As a result, the Linux* kernel may not load the kernel mode software that was built against the Red Hat* or SUSE* RDMA/InfiniBand software in your distribution. This may require that you rebuild such software against the installed OFED, or obtain a version of the software that was so built. For example, an implementation of the Lustre* file system that was built against a Red Hat* or SUSE* distribution may not be loaded by the Linux* kernel, and may need to be rebuilt against the installed OFED. Note that user mode applications will not need to be rebuilt due to this installation.

Background:
This section describes the steps to install an enhanced implementation of OFED 1.5.4.1 that supports CCL (Coprocessor Communication Link) and CCL-Proxy. CCL enables native Intel® Xeon Phi™ coprocessor applications to communicate directly with certain Mellanox* InfiniBand adapters.

Option 1:
The Offload computing model is characterized by MPI communication (if used at all) taking place only between the host processors in a cluster. In this model, Intel® Xeon Phi™ coprocessors are accessed exclusively through the offload capabilities of products like the Intel® C, C++, and Fortran Compilers, and the Intel® Math Kernel Library (Intel MKL). This mode of operation does not require CCL, and therefore the OFED version in a Red Hat* or SUSE* distribution can be used.

Option 2:
If MPI ranks are to be executed on Intel® Xeon Phi™ coprocessors, and if it is required that these ranks communicate directly with an InfiniBand adapter, then the following

installation should be performed. The ibscif virtual adapter will provide the best host-to-coprocessor and coprocessor-to-coprocessor transfer performance on systems without an InfiniBand adapter.

Steps:

1) Download OFED 1.5.4.1.

```
[host]$ wget \
https://www.openfabrics.org/downloads/OFED/ofed-1.5.4/OFED-1.5.4.1.tgz
```

2) Untar OFED 1.5.4.1 and access the untar folder.

```
[host]$ tar xf OFED-1.5.4.1.tgz
[host]$ cd OFED-1.5.4.1
```

NOTE: Step 3 applies only when using SLES 11 SP2 or SP3. If your host operating system is not SLES 11 SP2 or SP3, skip ahead to Step 4.

3) In SLES 11 SP2 or SP3, when choosing to install the MPI/gcc packages from the OFED package, it is necessary to make the following changes to the OFED 'install.pl' script:

   a) Find the line containing "$libstdc = 'libstdc++46';" and add the following line immediately below it:

   ```
   $libgfortran = 'libgfortran46';
   ```

   b) Find the line containing "} elsif ($dist_rpm =~ /sles-release-11.2/) {" and add the following three lines immediately above it:

   ```
   } elsif ($dist_rpm =~ /sles-release-11.3/) {
      $DISTRO = "SLES11.3";
       $rpm_distro = "sles11sp3";
   ```

   The diff example shown on the next page illustrates the changes to be applied to the OFED 'install.pl' script.

NOTE: In this PDF document, consecutive spaces from the diff are not present in the text. The example is shown for reference only, and will not produce a working patch when copied and pasted from this document. To create a working patch, manually type the file, including space characters where appropriate.

```
diff -Naur install.pl-old install.pl
--- install.pl-old   2012-02-07 15:22:26.000000000 +0000
+++ install.pl  2013-10-30 15:33:00. 738745000 +0000
@@ -217,6 +217,9 @@
 } elsif ($dist_rpm =~ /openSUSE/) {
     $DISTRO = "openSUSE";
     $rpm_distro = "opensuse11sp0";
+} elsif ($dist_rpm =~ /sles-release-11.3/) {
+    $DISTRO = "SLES11.3";
+    $rpm_distro = "sles11sp3";
```

```
 } elsif ($dist_rpm =~ /sles-release-11.2/) {
     $DISTRO = "SLES11.2";
     $rpm_distro = "sles11sp2";
@@ -374,6 +377,7 @@
     $curl_devel = 'libcurl-devel';
     if ($rpm_distro eq "sles11sp2") {
        $libstdc = 'libstdc++46';
+        $libgfortran = 'libgfortran46';
     }
 } elsif ($DISTRO =~ m/RHEL|OEL|FC/) {
     $libstdc = 'libstdc++';
```

4) Install the OFED stack as instructed in OFED README.txt, with a few exceptions regarding installed packages.

```
[host]$ less README.txt
[host]# perl install.pl
```

During installation, select:

- Option 2 (Install OFED Software)
- Option 4 (Customize)
- ...exclude kernel-ib* and dapl* packages...
- "Install 32-bit packages? [y/N]", answer N
- “Enable ROMIO support [Y/n]”, answer Y
- “Enable shared library support [Y/n]”, answer Y
- “Enable Checkpoint-Restart support [Y/n]”, answer N

NOTE: It is recommended not to install the 32-bit packages when installing OFED on RHEL 6.5. Installing 32-bit support on RHEL 6.5 requires an older version of glibc-devel.i686.

5) Install Intel MPSS OFED.

```
[host]$ cd mpss-3.4
[host]$ cp ofed/modules/*`uname -r`*.rpm ofed
[host]# rpm -Uvh ofed/*.rpm
```

2.4 Steps to Install Intel MPSS using OFED-3.5-2-mic

NOTE: As of release time, this OFED is currently in beta. See the Intel *MPSS Release Notes* for changes and distro support.

1) Install the Intel MPSS stack as instructed in the Intel MPSS Readme.
2) Download the latest distribution tarball from:

```
https://www.openfabrics.org/downloads/ofed-mic/ofed-3.5-2-mic/
```

3) Untar OFED-3.5* and access the untar folder.

```
[host]$ tar xf OFED-3.5*.tgz
[host]$ cd OFED-3.5*
```

4) Install the OFED stack as instructed in OFED README.txt.

```
[host]$ less README.txt
[host]# perl install.pl
```

2.5 Steps to Install Intel MPSS using Mellanox* OFED 2.1/2.2

1) Download Mellanox* OFED 2.1.x/2.2.x from:

   http://www.Mellanox.com/page/products_dyn?product_family=26

2) Untar, read the documentation, install as normal.

```
   Install Intel MPSS OFED ibpd rpm:
   [host]# rpm -U ofed/ofed-ibpd*.rpm
```

3) From the src/ folder of the Intel MPSS installation, compile dapl, libibscif, and ofed-driver source RPMs:

```
   [host]# rpmbuild --rebuild --define "MOFED 1" \
src/dapl*.src.rpm src/libibscif*.src.rpm src/ofed- \
driver*.src.rpm
```

4) Install the resultant RPMs:

```
   # on RHEL systems:
   [host]# rpm -U ~/rpmbuild/RPMS/x86_64/*rpm
   # on SLES systems:
   [host]# rpm -U /usr/src/packages/RPMS/x86_64/*rpm
```

2.6

2.7 Starting Intel MPSS with OFED Support

1) After installing Intel MPSS, ensure the Intel MPSS service is started by using the Linux* service command:

```
[host]# 1service mpss status
```

Do not proceed any further if Intel MPSS is not started.

2) If using Intel® True Scale Fabric HCAs, or using Mellanox** InfiniBand adapters and/or the ibscif virtual adapter, start the IB and HCA services by doing the following:

```
[host]# 1service openibd start
```

NOTE: If needed, start the opensmd service to configure the fabric:

```
[host]# 1service opensmd start
```

NOTE: If using Intel® True Scale Fabric HCAs and Intel® True Scale Fabric switches, it is recommended to use the Intel® Fabric Manager, rather than the opensm. Visit http://www.intel.com/infiniband for information on Intel's fabric management and software tools, downloads and support contacts.

3) If using CCL-Direct and IPoIB with Mellanox** InfiniBand adapter you can enable IPoIB module to be loaded as part of the ofed-mic service (see Section 19.2) and configure the IP Address and Netmask by editing the /etc/mpss/ipoib.conf which contains instructions for how to make these changes. See example ipoib.conf script in Section 19.2. Note that IPoIB is only a technology preview and currently only works with OFED-1.5.4.1 on the Mellanox* mlx4 driver and hardware.

4) If using Intel® True Scale Fabric HCAs, or using Mellanox** InfiniBand adapters and/or the ibscif virtual adapter, then start the Intel® Xeon Phi™ coprocessor specific OFED service on the host using:

```
[host]# 1service ofed-mic start
```

5) The use of ccl-proxy service is applicable only if using Mellanox** InfiniBand adapters. To start the ccl-proxy service (see configuration in: /etc/mpxyd.conf):

```
[host]# 1service mpxyd start
```

2.8 Stopping Intel MPSS with OFED Support

To stop all OFED support on all variants, stop the following services in order:

```
[host]# 1service mpxyd stop
[host]# 1service opensmd stop
[host]# 1service ofed-mic stop
[host]# 1service openibd stop
```

3 Installing Intel MPSS with GANGLIA* Support (optional)

3.1 Requirements

1) Red Hat* Enterprise
   - apr
   - apr-devel
   - expat
   - expat-devel
   - gcc-c++
   - libconfuse
   - libconfuse-devel
   - libtool
   - rpmbuild
   - rrdtool
   - rrdtool-devel

2) SUSE* Linux* Enterprise Server (SLES)
   - gcc-c++
   - libapr1
   - libapr1-devel
   - libconfuse0
   - libconfuse-devel
   - libexpat0
   - libexpat-devel
   - libtool
   - rpmbuild
   - rrdtool
   - rrdtool-devel

3.2 Steps to Install Intel MPSS with GANGLIA* Support

The default path for the GANGLIA* web page is /usr/share/ganglia. If the ganglia-web RPM was installed, the files conf.php, get_context.php and host_view.php will be overwritten. Only GANGLIA* 3.1.7 is currently supported.

For additional information on the installation of GANGLIA*, consult the documentation at http://ganglia.sourceforge.net

NOTE: Before executing steps 1 - 12, create the directories */var/lib/ganglia/rrds* and */var/www/html* if they do not already exist:

```
[host]# mkdir -p /var/lib/ganglia/rrds
[host]# mkdir -p /var/www/html
```

Steps:

1) Download GANGLIA* 3.1.7 from http://ganglia.info/?p=269.
2) Untar GANGLIA* 3.1.7 package and access the untar folder.
```
[host]$ tar xf ganglia-3.1.7.tar.gz
[host]$ cd ganglia-3.1.7
```
3) Execute the configure tool.
```
[host]$ ./configure --with-gmetad \
--with-libpcre=no --sysconfdir=/etc/ganglia
```
4) Build GANGLIA* content and install binaries.
```
[host]# make
[host]# make install
```
5) Generate default configuration for gmond.
```
[host]# gmond --default_config >/etc/ganglia/gmond.conf
```
6) Edit (as root or superuser) /etc/ganglia/gmond.conf to configure a udp_recv_channel, add the following entry on line 54:
```
udp_recv_channel {  port = 8649  }
```
7) Edit (as root or superuser) /etc/ganglia/gmetad.conf to configure the cluster name in the "data_source" line.
```
data_source "mic_cluster" localhost
```
8) Change the owner of the RRD folder.
```
[host]# chown -R nobody /var/lib/ganglia/rrds
```
9) Copy GANGLIA* web content to local web path.
```
[host]$ cp -r web <web_path>/ganglia
```
10) Start gmond and gmetad daemons.
```
[host]# gmond
[host]# gmetad
```

11) Install web front end for Intel MPSS GANGLIA*.

- Red Hat* Enterprise Linux*

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

- SUSE* Linux* Enterprise Server

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

12) Copy the web content under /usr/share/mpss/ganglia to the GANGLIA* web path.

```
[host]# cp -r /usr/share/mpss/ganglia/* <web_path>/ganglia/
```

3.2.1 Installing Intel MPSS GANGLIA* RPMs in the Card

To install GANGLIA* RPM files in the card, do the following:

1) Refer to Section 11.3.2 "Copy RPMs to Card Using a Repo and Zypper (via HTTP)" for example of card side installation procedure.
2) Proceed to Section 3.3 to start the coprocessor specific GANGLIA* stack.

NOTE: If there are multiple cards, repeat step 2 for the remaining card(s).

3.3 Starting Intel MPSS with GANGLIA* Support

1) Configure file /etc/ganglia/gmond.conf for the host and the cards as needed.

   The CPU metrics are disabled by default; enabling them will cause a performance penalty. To enable CPU metrics, uncomment the block in line 146.
2) The Intel® Xeon Phi™ coprocessor specific GANGLIA* stack is started by executing:

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

3.4 Stopping Intel MPSS with GANGLIA* Support

Stop the Gmond for all installed coprocessors in the system. Change N to the corresponding number for each coprocessor.

```
[host]# ssh micN killall gmond
```

4 Installing Intel® Xeon Phi™ Coprocessor Performance Workloads (optional)

4.1 Requirements

1) Intel® Composer XE Requirements

   There are two options to installing the Intel® Composer XE requirements. The first option is to install the full Intel® Composer XE package and source the compilervars.sh or compilervars.csh script 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 has an install.sh script for installation. After installation, there are compilervars.sh and compilervars.csh scripts which serve a similar purpose to those scripts in the full Intel® Composer XE distribution and must be sourced at run time.

   Besides the shared object libraries, the Intel MKL Linpack benchmark is also a requirement. This is also 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, but 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 (in bash or Bourne shell):

   ```
   export MKLROOT=<directory_path>/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. Note that the version of linpack linked above may be newer than 11.1.2, and MKLROOT variable should reflect this.

2) MATPLOTLIB Requirements

   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 only requires MATPLOTLIB for this use case. MATPLOTLIB must be installed in order to create plots. Download it from:

   matplotlib.sourceforge.net

4.2 Distributed Files

This package is distributed as two RPM files:

micperf-3.*.rpm
micperf-data-3.*.rpm

The first of these packages contains everything except the reference performance measurements, which are distributed in the second package.

4.3 RPM Installation

To install the RPM files, simply access the *perf* directory, located in the path where the Intel MPSS package was extracted.

- Red Hat* Enterprise Linux*

```
[host]# yum install *.rpm
```

- SUSE* Linux* Enterprise Server

```
[host]# zypper install *.rpm
```

This installs files to the following directories:

• Source code:	/usr/src/micperf
• Documentation and licenses:	/usr/share/doc/micperf-[version number]
• Benchmark binaries:	/usr/libexec/micperf
• Reference data:	/usr/share/micperf/micp
• Links to executables:	/usr/bin

4.4 Python Installation

Once the RPM packages have been installed, an additional step may 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 install into the Python site packages:

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

This method provides access to the micp package for all non-root users who use the same Python version as the root user (sudoer). If Python is in the default location and uses a standard configuration, setup.py installs the micp package to the 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, and this module does not support uninstall. If installing on a Linux system where Python is configured in a standard way, it should be possible to uninstall with the following commands:

```
[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]*
```

4.5 Alternative to Python Installation

Another way to access the micp package after installing the RPMs is to alter the shell run time environment of a user. To set up your bash or Bourne shell environment:

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

To set up your csh run time environment:

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

5 Installing Intel MPSS with Reliability Monitor Support (optional)

5.1 Overview

Reliability Monitor is designed to monitor overall health of compute nodes on cluster level. It is running on head node, or management node. Reliability Monitor works closely with RAS agent running on each service node. Any uncorrectable error or crash symptoms will be reported to Reliability Monitor.

5.2 Requirements

1) Install Reliability Monitor RPM on head node, or management node.
2) Install and start Intel MPSS (see the Intel MPSS Readme, Section 2).
3) Micrasd installed on each compute node and micras service started.

5.3 Steps to Install Intel MPSS with Reliability Monitor Support

Only install Reliability Monitor on head node, or management node.

The default path for Reliability Monitor node configuration file is */etc/mpss*.

Steps:

Install Intel MPSS Reliability Monitor:

- Red Hat* Enterprise Linux*

```
[host]$ cd mpss-[version number]/relmon
[host]# yum install mpss-sysmgmt-relmon-3.*.rpm
```

- SUSE* Linux* Enterprise Server

```
[host]$ cd mpss-[version number]/relmon
[host]# zypper install mpss-sysmgmt-relmon-3.*.rpm
```

5.4 Starting Intel MPSS with Reliability Monitor Support

1) On each compute node, make sure Intel mpss service and micras service are up and running. If Intel mpss service and micras service are not running, use:

```
[host]# 1service mpss start
[host]# 1service micras start
```

2) On head node, start Reliability Monitor service by using:

```
[host]# 1service relmon start
```

5.5 Stopping Intel MPSS with Reliability Monitor Support

On head node, stop Reliability Monitor service by using:

```
[host]# 1service relmon stop
```

5.6 Reliability Monitor Configuration File and Log

The node configuration file "mic_node.cfg" for Reliability Monitor is located under /etc/mpss. The file is in comma-separated values (CSV) format so it is supported by almost all spreadsheets and database management systems.

The errors will be logged into Linux* syslog /var/log/messages. You can check the error log by using:

```
[host]# cat /var/log/messages | grep relmon
```

Reliability Monitor is installed in /usr/bin. After relmon service is running, you can issue commands to monitor node status and error information by using:

```
[host]$ relmond --cmd shownode
[host]$ relmond --cmd showerr
```

For more information about Reliability Monitor, refer to:

```
[host]$ relmond --help
```

6 Post Installation Quick Configuration

After the installation of the RPMs (consult the Intel MPSS Readme for installation instructions), the system administrator must complete the Intel® Xeon Phi™ coprocessor configuration before starting the Intel MPSS service.

6.1 Step 1: Ensure Root Access

User access to the Intel® Xeon Phi™ coprocessor node is provided through the secure shell utilities. Ensure the **root** user has ssh keys by looking in the */root/.ssh* directory for either the **id_rsa.pub** or **id_dsa.pub** key files. If no SSH keys exist, use the **ssh-keygen** command to generate a set:

```
[host]# ssh-keygen
```

6.2 Step 2: Generate the Default Configuration

Each Intel® Xeon Phi™ coprocessor has a unique configuration file in the */etc/mpss* directory. Initialize the default configuration for the Intel® Xeon Phi™ coprocessors installed on the system:

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

The **micctrl --initdefaults** command creates and populates default configuration values into Intel MPSS specific configuration files. These configuration files, default.conf and micN.conf, are located at */etc/mpss/*, where N is an integer number (0, 1, 2, 3, etc.) that identifies each coprocessor installed in the system. The default.conf file is common to all installed Intel® Xeon Phi™ coprocessors.

6.3 Step 3: Change Configuration

Examine the default.conf and micN.conf files in the /etc/mpss directory. If the default configuration meets the requirements of the system, continue to Step 4. Otherwise, change the values using the **micctrl** utility (refer to Section 14 "Configuration").

6.4 Step 4: Start the Intel MPSS Service

The default configuration specifies that each Intel® Xeon Phi™ coprocessor is booted when the Intel MPSS service is started. To start the Intel MPSS service, execute the Linux* service command:

```
[host]# [1]service mpss start
```

The call to service will exit when it determines the Intel® Xeon Phi™ coprocessors have either booted successfully or failed to boot, and the status of the cards will be displayed.

7 Intel MPSS Configuration

7.1 Intel MPSS Configuration Overview

Intel® Xeon Phi™ coprocessor configuration files are located at /etc/mpss/default.conf and /etc/mpss/micN.conf, where N is an integer number (0, 1, 2, 3, etc.) that identifies each coprocessor installed in the system. The Intel MPSS device driver installs default.conf. This file serves as the centralized configuration file for all installed coprocessors. Additionally, the driver installs the file micN.conf, which allows the system administrator to configure each Intel® Xeon Phi™ coprocessor individually.

Sections 12 through 18 explain in detail how to modify Intel MPSS configuration files. Refer to those sections to complete the configuration steps.

7.2 Clock Source for the Intel® Xeon Phi™ Coprocessor

By default, the clocksource has been set to TSC. The power management software for the coprocessor will keep the TSC clocksource calibrated even when deep sleep states are enabled. Calibration was implemented to avoid clock drift and keep users from needing to use the micetc clocksource. Using micetc will result in 100x slower return from gettimeofday compared to using TSC.

7.3 Peer to Peer (P2P) Support

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

P2P is enabled by default for Intel MPSS.

- To run with P2P disabled, the module parameter control file /etc/modprobe.d/mic.conf must be edited.

  Change "p2p=1" to "p2p=0" in the options line for the "mic" module.
- A driver reload is required. Follow this procedure:

```
[host]# [1]service mpss unload
[host]# [1]service mpss start
```

7.4 NFS Mounting a Host Export

For NFS to work, the host firewall or iptables may need to be configured to allow the following ports:

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

The user can mount an NFS file system exported from the host. As superuser, first add the exports to the /etc/exports file.

For example: To provide access from the "/micNfs" directory to the first Intel® Xeon Phi™ Coprocessor (micN) in the system, using a default configuration, where micN host assignment ip: 172.31.1.254 and coprocessor assignment ip: 172.31.1.1, execute the following sequence:

1) On the NFS host:
   a) Create the “/micNfs” directory on the host system.
      ```
      [host]# mkdir /micNfs
      ```
   b) Append */etc/exports* with the line:
      ```
      /micNfs 172.31.1.1(rw,no_root_squash)
      ```
   c) Add in */etc/hosts.allow* file:
      ```
            ALL:172.31.1.1
      ```
   d) Let NFS know the files have changed by running:
      ```
      [host]# exportfs –a
      ```
   e) Add the following to /etc/exports file:
      ```
      /srv/michome 172.31.0.0/16(rw)
      ```

2) On the Intel® Xeon Phi™ Coprocessor host system:
   Using the micctrl utility, modify the Intel® Xeon Phi™ Coprocessor’s /etc/fstab file to find the exported NFS file system.
   ```
   [host]# micctrl --addnfs=172.31.1.254:/micNfs \
   --dir=/micNfs
   ```
3) Back on the NFS host, by default, the user is only created on the coprocessor. Set up the user to allow Intel® Xeon Phi™ coprocessor to write back to the host via NFS with the following steps.
   a) Stop Intel MPSS service.
   ```
   [host]# [1]service mpss stop
   ```
   b) Set up micuser. (Optional if micuser already exsists)
   ```
      [host]# useradd -U -m -u 400 micuser
   [host]# groupmod -g 400 micuser
   ```

c) Create /srv/michome required home directories in /srv/michome (specifically /srv/michome/micuser).

```
[host]# mkdir -p /srv/michome/micuser
```

d) Set owner of michome directories as appropriate.

```
[host]# chown micuser /srv/michome/micuser/
```

e) Confirm the following was added to the /etc/exports file:

```
/srv/michome 172.31.0.0/16(rw)
```

f) Restart NFS service.

```
    [host]# exportfs -a
[host]# 1service nfs restart
```

g) Configure the coprocessor.

```
[host]# micctrl --addnfs=/srv/michome --dir=/home
```

h) Start the Intel MPSS service.

```
[host]# 1service mpss start
```

Considerations:

- If you perform a 'service mpss restart', you might need to execute the following command as root on the coprocessor to get the NFS links operational again:

  ```
  [micN]# mount -a
  ```

- For SUSE*, to auto-start the NFS server on reboots, change the Intel MPSS script (/etc/init.d/mpss) line as follows below:

  From:

  ```
  # Required-Start:
  ```

  To:

  ```
  # Required-Start: nfsserver
  ```

- For RHEL*, to auto-start the NFS server on reboots, ensure the NFS server priority is higher than that of Intel MPSS.

7.5 How to Login to the Intel® Xeon Phi™ Coprocessor by LDAP Account

Prerequisite

Configure the network as an external bridge so the LDAP server can be reached from the coprocessor. See Section 19.1 for an example of how to configure an external bridge.

There are two ways to activate LDAP on an Intel® Xeon Phi™ coprocessor:

Method 1 - Enable LDAP with micctrl:

1. If not already done, extract and specify the k1om rpm directory:

```
[host]# micctrl --rpmdir=/path/to/k1om
```

2. Specify the LDAP server:

```
[host]# micctrl --ldap=<LDAP server IP address> \
--base=”example.com”
```

Example: `[host]#` *micctrl --ldap=192.168.122.129 --base=”example.com*”.

Method 2 - Manually configure LDAP (using the procedure below).

NOTE: The steps described in this section are volatile: all of the steps must be repeated every time the card is rebooted.

NOTE: This configuration does not allow changing the user’s password from the coprocessor.

Procedure

1) Install libldap, nss-ldap, pam-ldap RPM files on the coprocessor.

```
[micN]# rpm -ivh libldap-2.4-2-2.4.23-r1.k1om.rpm
[micN]# rpm -ivh nss-ldap-265-r0.k1om.rpm
[micN]# rpm -ivh pam-ldap-186-r0.k1om.rpm
```

2) Configure nss-ldap on the Coprocessor.

```
[micN]# cp /etc/nsswitch.ldap /etc/nsswitch.conf
[micN]# sed -ie"/^hosts:/s/dns ldap/files/" /etc/nsswitch.conf
[micN]# SelfIp=`/sbin/ifconfig micN | grep \
"inet addr" | cut -d":" -f2 | cut -d" " -f1`
[micN]# echo ${SelfIp} `hostname` `hostname -s` >>/etc/hosts
```

3) Configure ldap on the Coprocessor.

```
[micN]# cp /etc/openldap/ldap.conf /etc
[micN]# echo \
"URI ldap://<LDAP server IP address>/" >>/etc/ldap.conf
```

Example: echo "URI ldap://*192.168.122.129/*" >>/etc/ldap.conf

```
[micN]# echo "BASE dc=example,dc=com" >>/etc/ldap.conf \
“auth sufficient pam_ldap.so”
```

4) Configure PAM for SSH and others on the Coprocessor.

```
[micN]# sed -ie"s/^$/auth sufficient pam_ldap.so/" \
/etc/pam.d/common-auth

[micN]# sed -ie"/session/s/required/optional/"\
/etc/pam.d/sshd
```

7.6 How to Login to the Intel® Xeon Phi™ Coprocessor by NIS/YP Account

There are two ways to activate NIS on an Intel® Xeon Phi™ coprocessor:

- Follow the procedure below
- Use micctrl, such as **micctrl --nis=<*NIS/YP server IP address*> --domain="example.com"**

  Example:

```
[MicN]# micctrl --rpmdir=<path to K1OM directory>
[MicN]# micctrl --nis=192.168.122.129 --domain="example.com"
```

In both cases, the prerequisite below is required. For more information about micctrl, refer to its help page:

```
[host]# micctrl --help
```

Prerequisite

Configure the network as an external bridge so the NIS/YP server can be reached from the coprocessor. See Section 19.1.2 for an example of how to configure an external bridge.

The above configuration does not allow changing the user's password from the coprocessor.

NOTE: The steps described in this section are volatile: all of the steps must be repeated every time the card is rebooted.

Procedure

1) Install libtirpc1,rpcbind, ypbind-mt, yp-tools, and glibc-extra-nss RPM files on the coprocessor.

```
[micN]# rpm -ivh libtirpc1-0.*.k1om.rpm
[micN]# rpm -ivh rpcbind-0.*.k1om.rpm
[micN]# rpm -ivh yp-tools-*.k1om.rpm
[micN]# rpm -ivh ypbind-mt-*.k1om.rpm
[micN]# rpm -ivh glibc-extra-nss-2.*.k1om.rpm
```

2) Start rpcbind daemon.

```
[micN]# /etc/init.d/rpcbind start
```

3) Start ypbind daemon.

```
[micN]# echo \
"domain <domain name> server <server IP address>" >>/etc/yp.conf
[micN]# domainname <domain name>
[micN]# /etc/init.d/ypbind start
```

Example:

```
[micN]#  echo "domain caz.ra.intel.com server 192.168.122.136" \

>>/etc/yp.conf
[micN]#  domainname caz.ra.intel.com
[micN]#  /etc/init.d/ypbind start
```

4) Configure nss-ldap on the Coprocessor.

```
[micN]$ cat <<EOF >>/etc/nsswitch.conf
passwd: nis files
shadow: nis files
group: nis files
EOF
```

5) Configure sshd on the Coprocessor.

```
[micN]# echo "UsePAM yes" >>/etc/ssh/sshd_config
```

6) Configure PAM for SSH and others on the Coprocessor.

```
[micN]# sed -ie"s/^$/auth sufficient pam_ldap.so/" \
/etc/pam.d/common-auth

[micN]# sed -ie"/session/s/required/optional/" \
/etc/pam.d/sshd
```

7) Restart sshd (the above changes will take effect).

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

7.7 How to Enable NFS Auto Mount with NIS/YP on the Intel® Xeon Phi™ Coprocessor

Prerequisite

Configure the network as an external bridge so the NIS/YP server can be reached from the coprocessor. See Section 19.1.2 for an example of how to configure an external bridge.

The above configuration does not allow changing user's password from the coprocessor.

NOTE: The steps described in Section 7.7 are volatile: all of the steps must be repeated every time the card is rebooted.

Procedure

1) Install libtirpc1,rpcbind, ypbind-mt, yp-tools, and glibc-extra-nss RPM files from the K1OM tar package to the coprocessor:

```
[micN]# rpm -ivh libtirpc1-0.*.k1om.rpm rpcbind-0.*.k1om.rpm yp-
tools-*.k1om.rpm ypbind-mt-*.k1om.rpm glibc-extra-nss-
2.*.k1om.rpm nfs-utils-client-*.k1om.rpm autofs-5.*.k1om.rpm
```

2) Start rpcbind daemon.

```
[micN]# /etc/init.d/rpcbind start
```

3) Start ypbind daemon.

```
[micN]# echo "domain <domain name> \
server <server IP address>" >>/etc/yp.conf
[micN]# domainname <domain name>
[micN]# /etc/init.d/ypbind start
```

4) Configure nss-ldap on the Coprocessor.

```
[micN]$ cat <<EOF >>/etc/nsswitch.conf
passwd: nis files
shadow: nis files
group: nis files
EOF
```

5) Configure sshd on the Coprocessor.

```
[micN]# echo "UsePAM yes" >>/etc/ssh/sshd_config
```

6) Configure PAM for SSH and others on the Coprocessor.

```
[micN]# sed -ie"s/^$/auth sufficient pam_ldap.so/" \
/etc/pam.d/common-auth

[micN]# sed -ie"/session/s/required/optional/" \ /etc/pam.d/sshd
```

7) Configure autofs and re-start autofs/automount daemon with the configuration.

```
[micN]# echo "/home /etc/auto.misc " >>/etc/auto.master
[micN]# /etc/init.d/autofs stop
[micN]# sleep 2 # need.
[micN]# /etc/init.d/autofs start
```

8) Restart sshd (the above changes will take effect).

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

7.8 How to Enable Host Based Authentication on SSH

On Intel® Xeon Phi™ coprocessor – ssh server c

1) Configure sshd to enable host based authentication.

```
[micN]# cat <<EOF >>/etc/ssh/sshd_config

HostbasedAuthentication yes
IgnoreRhosts no
EOF
```

2) Register SSH client to a user.

```
[host]$ cat <<EOF >><home directory>/.shosts

<micN>
<server IP address>
EOF

[host]# chmod 600 <home directory>/.shosts

[host]# chown <owner:group> <home directory>/.shosts
```

Example:

```
 cat <<EOF >>~caz/.shosts

HostBasedAuthClient
192.168.122.50
EOF

 chmod 600 ~caz/.shosts

 chown caz:caz ~caz/.shosts
```

3) Create an entry for SSH client in user's known_hosts.

```
[host]$ ssh <user>@<micN>
```

Example:

```
[host}# su caz

 ssh HostBasedAuthClient

Are you sure you want to continue connecting (yes/no)? yes
<user>@<server IP address>'s password:
Exit
```

NOTE: The prompt `Are you sure you want to continue connecting (yes/no)?` will only be seen the first time you connect to the system.

4) On Intel® Xeon Phi™ coprocessor – ssh server restart SSH daemon.

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

5) Remove SSH key to make sure user based authentication is not used.

```
[host]$ cd <home directory>/.ssh
[host]$ rm -f authorized_keys id_rsa*
```

7.9 How to Mount a Share using NFS v4

On Intel® Xeon Phi™ coprocessor – NFS client

1) Install required RPM files

```
[micN]# rpm -ivh util-linux-mount-2.*.k1om.rpm \
           util-linux-umount-2.*.k1om.rpm
[micN]# nfs-utils-client-*.k1om.rpm
[micN]# nfs-utils-1*.k1om.rpm \
           libnfsidmap0-0.*.k1om.rpm libevent-2.*.k1om.rpm
```

NOTE: When installing nfs-utils, the prompt, "starting idmapd: no /etc/idmapd.conf" should be expected, because you have not yet created /etc/idmapd.conf.

2) Create or modify nss configuration file.

```
[micN]$ cat <<EOF >>/etc/nsswitch.conf
passwd:     files
shadow:     files
group:      files
EOF
```

3) Create an idmapd configuration file.

```
[micN]$ cat <<EOF >>/etc/idmapd.conf
[General]
Domain = localdomain

[Mapping]
Nobody-User = nobody
Nobody-Group = nogroup

[Translation]
Method = nsswitch, static
EOF
```

4) Optionally, you may add nobody and nogroup.

```
[micN]$ groupadd -g4294967294 nogroup
```

```
[micN]$ useradd -gnogroup -u4294967294 nobody
```

5) Start idmap daemon.

```
[micN]# /etc/init.d/idmapd start
```

6) Try to mount.

```
[micN]# mount -tnfs <server IP address>:/home/<dir>
/mnt -v -onfsvers=4,soft
[micN]# ls -l /mnt
```

NOTE: There is no symbolic link created in /etc/rcN.d for /etc/init.d/nfsserver while /etc/init.d/nfsserver is created.

7.10 How to Customize MIC OS

Unpack initramfs – Intel® Xeon Phi™ coprocessor file system

```
[host]# mkdir /provision
[host]# cd /provision
[host]# export initramfs_dir=/usr/share/mpss/boot
[host]# export \
initramfs_cpio_gz=initramfs-knightscorner.cpio.gz
[host]# gunzip <${initramfs_dir}/${initramfs_cpio_gz} | cpio –I
If using Sudo:
sudo gunzip <${initramfs_dir}/${initramfs_cpio_gz} | sudo cpio -I
```

You now have the following file system.

```
drwxr-xr-x  2 root root 4096 Nov 25 09:17 bin
drwxr-xr-x  2 root root 4096 Nov 25 09:17 boot
drwxr-xr-x  2 root root 4096 Nov 25 09:17 dev
drwxr-xr-x 26 root root 4096 Nov 25 09:17 etc
drwxr-sr-x  3 root root 4096 Nov 25 09:17 home
-rwxr-xr-x  1 root root 3512 Nov 25 09:17 init
drwxr-xr-x  3 root root 4096 Nov 25 09:17 lib
drwxr-xr-x  4 root root 4096 Nov 25 09:17 lib64
drwxr-xr-x 10 root root 4096 Nov 25 09:17 media
```

```
drwxr-xr-x  2 root root 4096 Nov 25 09:17 mnt
drwxr-xr-x  2 root root 4096 Nov 25 09:17 proc
drwxr-xr-x  2 root root 4096 Nov 25 09:17 sbin
drwxr-xr-x  2 root root 4096 Nov 25 09:17 sys
drwxrwxrwt  2 root root 4096 Nov 25 09:17 tmp
drwxr-xr-x 11 root root 4096 Nov 25 09:17 usr
drwxr-xr-x  7 root root 4096 Nov 25 09:17 var
```

Customize initramfs

As an example of how to customize, the following commands enable LDAP.

1) Unpack required packages.

```
[host]# cp /<k1om_path>/*ldap* /provision/
[host]# rpm2cpio libldap-2.*.k1om.rpm | cpio -i
[host]# rpm2cpio nss-ldap-2*.k1om.rpm | cpio -i
[host]# rpm2cpio pam-ldap-1*.k1om.rpm | cpio -i
[host]# rm -f *ldap*.rpm
```

2) Configure for LDAP.

```
[host]# sed -i -e"s/^$/auth sufficient pam_ldap.so/"
etc/pam.d/common-auth
[host]# sed -i - e"/session/s/required/optional/"
etc/pam.d/sshd
[host]# echo "UsePAM yes" >>etc/ssh/sshd_config
[host]$ cat <<EOF >>etc/ldap.conf
URI ldap://<server IP address>/
BASE dc=example,dc=com
bind_policy=soft
EOF
```

Example:

```
[host]$  cat <<EOF >etc/ldap.conf
URI ldap://192.168.122.47/
BASE dc=example,dc=com
bind_policy=soft
EOF
```

3) Create an init script to prevent boot freeze.

```
[host]# cat <<EOF >etc/init.d/ldap_client
case "\$1" in
start)
```

```
sed -i -e"/^passwd:/s/files\$/files ldap/" \
etc/nsswitch.conf
   sed -i -e"/^group:/s/files\$/files ldap/" \
   etc/nsswitch.conf
    ;;
   stop)
   sed -i -e"/^passwd:/s/files ldap/files/" \
         etc/nsswitch.conf
   sed -i -e"/^group:/s/files ldap/files/" \
               etc/nsswitch.conf
    ;;
*)
         echo usage \$0 \[start\|stop\]
         exit 1
         ;;
esac
exit 0
EOF

[host]# chmod a+x etc/init.d/ldap_client
[host]# cd etc/rc5.d/
[host]# ln -s ../init.d/ldap_client S99ldap_client
[host]# cd ../..
```

Pack initramfs

```
[host]# find . |cpio -o -c |gzip >../${initramfs_cpio_gz}
[host]# rm -f ${initramfs_dir}/${initramfs_cpio_gz}
[host]# mv ../${initramfs_cpio_gz} ${initramfs_dir}
```

Modify nsswitch.conf to prevent boot freeze

```
[host]# cp etc/nsswitch.conf \
/var/mpss/micN/etc/nsswitch.conf
[host]# sed -i -e"/^passwd:/s/files ldap/files/" \
/var/mpss/micN/etc/nsswitch.conf
[host]# sed -i -e"/^group:/s/files ldap/files/" \
/var/mpss/micN/etc/nsswitch.conf
```

```
[host]# sed -i -e"/^hosts:/s/dns ldap/files/" \
/var/mpss/micN/etc/nsswitch.conf
```

7.11 Virtual Console Configuration and Access

When using SUSE, minicom prompts for a username and password when logging into the coprocessor. Use *micctrl --passwd<=user>* to set the password for a user before using the virtual console on minicom.

If there are multiple coprocessors, the virtual console devices are */dev/ttyMICN* for the first coprocessor*, /dev/ttyMICN* for the second coprocessor, and so on.

To configure minicom for virtual console access, perform the following instructions:

For each coprocessor:

1. `[host]# minicom -s`
2. Go to "`Serial Port Setup`".
   a. Choose option: **A - Serial Device**
   b. Edit Serial Device to /dev/ttyMICN
   c. Hit <**Enter**> twice.

```
Go to “Modem and dialing”.
a. Choose option: A - Init string
b. Erase the entire line.
c. Hit <Enter> twice.
```

3. Go to "**`Save setup as`**.."
   a. An input prompt "*`Give name to save this configuration?`*" will appear.
   b. Enter the preferred name, for example: `micN <`**`Enter`**`>`
4. Select "**`Exit`** *`from Minicom`*".

Repeat the above steps for the remaining coprocessor(s). Each coprocessor should have its own config name.

To open the virtual console for micN coprocessor where the config name is micN:

```
[host]# minicom micN
```

Minicom will prompt for a login and password.

```
To exit minicom: <CTRL+a> <x> <Enter>
```

7.12 Enable Virtio Block Device for the Intel® Xeon Phi™ Coprocessor

The virtio block device (virtblk) is accessed via the virtio data transfer mechanism. Virtio is designed for virtualization environment like KVM. Virtblk is available on the Intel® Xeon Phi™ coprocessor.

To use virtio block device as an ext2 file system, do the following:

1) Host side:

   a) Start Intel MPSS service.

      ```
      [host]# ¹service mpss start
      ```

   b) Identify which file or block device is a block device of virtblk.

      ```
      [host]# echo dev/<ForVirtioBlockTest> >\
      /sys/class/mic/<micN>/virtblk_file
      ```

   The /dev/<VirtioTestTarget> file can be one of the following:

      + A regular file like /srv/aaa, or
      + LVM (Logical Volume Manager) volume, or
      + A physical device like /dev/sda*

   Additionally, /sys/class/mic/micN/virtblk_file is the sysfs file for setting a block device of virtblk.

2) Coprocessor side:

   a) Login to the coprocessor as superuser.

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

   b) Load the virtblk driver.

      ```
      [micN]# modprobe mic_virtblk
      ```

   c) Create ext2 file system on virtblk and mount it on /mnt/vda.

      ```
      [micN]# mkdir -p /mnt/vda
      [micN]# mkfs.ext2 /dev/vda
      [micN]# mount -t ext2 \
         /dev/vda /mnt/vda
      ```

   d) Unmount virtblk.

      ```
      [micN]# umount /mnt/vda
      ```

   e) Unload virtblk driver.

      ```
      [micN]# modprobe -r \
      mic_virtblk
      ```

   f) Exit from the coprocessor session.

      ```
      [micN]# exit
      ```

7.12.1 To use virtio as a swap device file system

1) Host side:

```
[host]# bash
[host]# echo /dev/<VirtioSwap>\
>/sys/class/mic/<micN>/virtblk_file
[host]# exit
```

2) Coprocessor side:

   a) Login to the coprocessors.

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

   b) Load the virtblk driver.

   ```
   [micN]# modprobe mic_virtblk
   ```

   c) Assign a swap device and confirm.

   ```
   [micN]# mkswap /dev/vda
   [micN]# swapon /dev/vda
   [micN]# more /proc/swaps
   ```

   d) Stop using swap device.

   ```
   [micN]# swapoff -a
   ```

   e) Exit from the coprocessor session.

   ```
   [micN]# exit
   ```

To use multiple virtio block devices, create multiple partitions in a virtio block device file. Those partitions are referenced as */dev/vda1*, */dev/vda2*.

If the system administrator does not assign a virtio block device file, unloading the Intel MPSS host driver will trigger the message "*request comes in while coprocessor side driver is not loaded yet. Ignore*" in *dmesg* and */var/log/messages*.

To load the coprocessor side driver, mic_virtblk, without assigning virtio block device file, the error message, "*Have set virtblk file?*" will be displayed in dmesg and */var/log/messages*.

Do not use multiple sysfs device entries from */sys/class/mic/micN/virtblk_file*.

7.13 Kernel Crash Dump Support for the Intel® Xeon Phi™ Coprocessor

1) The host driver configuration option to enable/disable coprocessor kernel crash dumps is located in /etc/modprobe.d/mic.conf.

   # Set crash_dump= to '1' to enable or '0' to disable kernel crash dump captures.

2) The mpssd daemon configuration options to tune crash dump storage location and storage limit are at */etc/mpss/default.conf*

   # Storage location and size for MIC kernel crash dumps

   ```
   CrashDump /var/crash/mic/ 16
   ```

3) If a custom user space utility other than the mpssd daemon is being used, then the steps below should be followed to obtain the kernel crash dump. The algorithm used by mpssd to obtain the kernel crash dump is as follows:

   a) Poll the sysfs entry for coprocessor state changes at /sys/devices/virtual/mic/ctrl/subsystem/micN/state, for example.

   b) Upon detection of the "lost" state, read from /proc/mic_vmcore/ and write the contents to a crash dump file.

   c) Gzip the content of the file.

   d) Now reset the card and reboot it if required.

4) A gzipped kernel crash dump core file should now be available if a coprocessor OS crash occurs at the storage location configured in step 2.

5) Install crash utility on host to analyze the crash dump (RHEL example shown):

   ```
   [host]# yum install crash
   ```

6) An example on how a crash can be analyzed is shown below:

   ```
   cd /var/crash/mic/micN/

   gunzip vmcore-xxxx.gz

   cp /opt/mpss/[version number]/sysroots/k1om-mpss-linux/ \
   boot/vmlinux-2.6.38.8+mpss[version number] .

   …x86_64-k1om-linux-elfedit --output-mach x86-64 vmlinux

   crash vmlinux vmcore-2012-9-24-15\:50\:29
   ```

NOTE: Useful commands include `bt`, `foreach bt`, `ps`, `log`, etc.
Refer to http://people.redhat.com/anderson/crash_whitepaper/#HELP

7.14 Offload User Options

COI Security:

_Authorized

The coi_daemon supports a mode that allows system administrators to configure the coi_daemon to spawn processes as the same user on the host. This is set up via the /etc/coi.conf or /etc/sysconfig/coi files, with /etc/sysconfig/coi taking precedence in case of conflict. This is supported through the --coiuser option when launching the coi_daemon directly, or through the coiparams='--coiuser=<option>' when using a configuration file.

The authentication occurs using a .mpsscookie file located in the user's home directory created and managed by the host mpss daemon.

_Dynamic

The daemon will also create a new user for each COI Process spawned on the card. Each COI Process is owned by this new user and can only access its own created temp files and directories, effectively isolating all COI Processes from each other for better security. To make the coi_daemon launch with this setting by default, create or modify the /etc/coi.conf or /etc/sysconfig/coi.conf files on the card, with /etc/sysconfig/coi.conf taking precedence in case of conflict. Then, change or add the line with *coiparams='--coiuser=micuser'* to *coiparams='--coiuser=_Dynamic'*. With this change, whenever you start the **coi_daemon**, the daemon will start with the new functionality. To permanently change the configuration, refer to the documentation on micctrl and file overlays.

You can view the existing */etc/init.d/coi* for samples and other settings.ples.

- Default setting:
  coiparams='--coiuser=micuser'
- To set authenticated users, use:
  coiparams='--coiuser=_Authorized'
- To set dynamic users, use:
  ='--coiuser=_Dynamic'

Example:

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

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

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

```
[micN]# coi_daemon --help
```

7.15 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 memory on the device.

To run 200 concurrent processes, users will need to modify the following parameters. Changes to the configuration will not persist when modifying the files directly on the card;

a reboot will reset these settings. To permanently change the configuration, refer to the documentation on micctrl and file overlays.

1) On the coprocessor, log into the card as superuser.

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

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 2000

[micN]# /usr/bin/coi_daemon \

--coiuser=micuser --max-connections=200 &

[micN]# exit
```

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

```
coi_daemon --help
```

7.16 Coprocessor Post Codes

Like any other Intel® IA-32, Intel® 64 or IA-64 platform, the Intel® Xeon Phi™ coprocessor produces POST codes at power on and boot to identify the stage that the card is at during the boot process. These POST codes can be viewed using the Linux* command "dmesg" after a system power on. The POST codes can also be viewed by "tailing" /var/log/messages :

```
[host]# tail -f /var/log/messages | grep \
"Post Code"
```

The POST codes are defined as follow:

"01"	LIDT
"02"	SBOX initialization
"03"	Set GDDR top
"04"	Begin memory test
"05"	Program E820 table
"06"	Initialize DBOX
"09"	Enable caching
"0b"	Pass initialization parameters to APs
"0c"	Cache C code
"0d"	Program MP table

"0E"	Copy AP boot code to GDDR
"0F"	Wake up APs
"10"	Wait for APs to boot
"11"	Signal host to download Coprocessor OS
"12"	Wait for Coprocessor OS download - this also known as the "ready" state. The coprocessor will be in this state after powering on, running "micctrl -r" or "[1]service mpss stop". It means that the coprocessor is ready to receive the coprocessor OS either by a "[1]service mpss start", "[1]service mpss restart" or "micctrl -b" depending on how the coprocessor got into this state. It is not an error condition for the coprocessor to be in this state. See the sections above to learn how to start Intel MPSS when the card is showing a state of POST code 12
"13"	Signal received from host to boot Coprocessor OS
"15"	Report platform information
"17"	Page table setup
"30"	Begin memory training
"31"	Begin GDDR training to query memory modules
"32"	Find GDDR training parameters in flash
"33"	Begin GDDR MMIO training
"34"	Begin GDDR RCOMP training
"35"	Begin GDDR DCC disable training
"36"	Begin GDDR HCK training
"37"	Begin GDDR ucode training
"38"	Begin GDDR vendor specific training
"39"	Begin GDDR address training
"3A"	Begin GDDR memory module identification
"3b"	Begin GDDR WCK training
"3C"	Begin GDDR read training with CDR enabled
"3d"	Begin GDDR read training with CDR disabled
"3E"	Begin GDDR write training
"3F"	Finalize GDDR training
"40"	Begin Coprocessor OS authentication
"50"-"5F"	Coprocessor OS loading and setup
"6P"	int 13 General Protection
"75"	int 10 Invalid TSS
"87"	int 16 x87 FPU Floating Point Error
"AC"	int 17 Alignment Check
"bP"	int 3 Breakpoint

"br"	int 5 BOUND Range Exceeded
"CC"	int 18 Machine Check
"co"	int 9 Coprocessor Segment Overrun
"db"	int 1 Debug
"dE"	int 0 Divide Error
"dF"	int 8 Double Fault
"EE"	Memory test failed
"F0"	GDDR parameters not found in flash
"F1"	GBOX PLL lock failure
"F2"	GDDR failed memory training
"F3"	GDDR memory module query failed
"F4"	Memory preservation failure
"F5"	int 12 Stack Fault
"FF"	Bootstrap finished execution
"FP"	int 19 SIMD Floating Point
"Ld"	Locking down hardware access
"nA"	uOS image failed authentication
"nd"	int 7 Device Not Available
"no"	int 2 Non-maskable Interrupt
"nP"	int 11 Segment Not Present
"oF"	int 4 Overflow
"PF"	int 14 Page fault
"r5"	int 15 reserved
"ud"	int 6 Invalid opcode

8 Tools

Intel® Xeon Phi™ coprocessor tools are located in the /usr/bin and /usr/sbin directories.

NOTE: Executing some tools without superuser privileges may result in reduced functionality of the tool.

8.1 Micinfo

Micinfo provides information about system configuration. It includes information about current Intel® Xeon Phi™ coprocessor hardware and the driver.

For detailed information about micinfo, refer to the micinfo man page.

```
    [host]# man micinfo
```

8.2 Micflash

Micflash is a flash utility for the Intel® Xeon Phi™ coprocessors. It is capable of updating the card's flash, saving the existing flash from the card to a file on the host, and displaying the current flash version that is loaded on a card.

For detailed information about micflash, refer to the micflash man page.

```
    [host]$ man micflash
```

8.3 Micsmc

Micsmc is the binary executable for the Intel® Xeon Phi™ coprocessor Platform Status Panel. The micsmc tool can function in two modes: GUI mode and command-line (CLI) mode. GUI mode provides real-time monitoring of all detected Intel® Xeon Phi™ coprocessors installed in the system. The CLI mode produces a snap-shot view of the status, which allows CLI mode to be used in cluster scripting applications. The micsmc tool monitors core utilization, temperature, memory usage, power usage statistics, and error logs, among other features.

The micsmc tool is based on the work of the Qwt project (http://qwt.sf.net)

The Status Panel User Guide is available in all supported languages, in PDF and HTML formats, at:

```
    /usr/share/doc/micmgmt/<lang_folder>
```

Or refer to the micsmc man pages to use GUI or CLI modes:

```
    [host]$ man micsmc
```

8.4 Miccheck

The miccheck utility is used to verify the configuration and current status of the Intel® Xeon Phi™ coprocessor software stack. It performs sanity checks on a host system with Intel® Xeon Phi™ coprocessor(s) installed, by running a suite of diagnostic tests. The default behavior is to run all enabled tests on the host system first, and then on each Intel® Xeon Phi™ coprocessor in turn.

There are two forms of the miccheck utility in Linux*:

1) A Python program, callable directly as “miccheck.py”.

2) A binary program, executable as “miccheck”.

Both versions behave exactly the same and will produce the same output.

For detailed information about miccheck, refer to the miccheck man page, or the help option of the program:

```
[host]$ man miccheck
[host]$ miccheck --help
```

8.5 Micnativeloadex

The micnativeloadex utility will copy an Intel® Xeon Phi™ coprocessor native binary to a specified Intel® Xeon Phi™ Coprocessor and execute it. The utility automatically checks library dependencies for the application. If they are found in the default search path (set using the SINK_LD_LIBRARY_PATH environment variable), the libraries will be copied to the card prior to execution. This simplifies running Intel® Xeon Phi™ coprocessor native applications.

In addition, the utility can also redirect output from an application running remotely on the Intel® Xeon Phi™ coprocessor back to the local console. This feature is enabled by default but can be disabled with a command line option. For further details on command line options see the help section below.

Note that if the application has any library dependencies, then the SINK_LD_LIBRARY_PATH environment variable must be set to include those directories. This environment variable works just like LD_LIBRARY_PATH for normal Linux* applications. To help determine the required libraries, execute micnativeloadex with the -l command line option. This will display the list of dependencies and which ones have been found. Any dependencies not found will likely need to be included in the SINK_LD_LIBRARY_PATH.

NOTE: The SINK_LD_LIBRARY_PATH must include the directory path for libcoi_host.so library

For example:

```
[host]# SINK_LD_LIBRARY_PATH=/usr/lib64/

[host]# /usr/bin/micnativeloadex <native_app> -l
```

For more information about micnativeloadex, refer to:

```
[host]# /usr/bin/micnativeloadex –h
```

CAUTION: When linking in libraries installed in /lib64, do not add "/lib64" to the LD_LIBRARY_PATH environment variable. This path is already implicit in the dynamic linker/loader's search path, and modifying the path variable will result in breaking the order in which library paths are searched for offload compilation.

8.6 Micctrl

The micctrl command is a helper for the system administrator. See Section 15 of this User's Guide for more information.

For more information about micctrl, refer to:

```
[host]# micctrl --help
```

8.7 Micrasd

Micrasd is the application running on the Host to handle and log the hardware errors reported by Intel® MIC devices. It is a daemon that is started by the micras service.

Using the "-daemon" option will run micrasd in daemon mode. In this case, micrasd will run in background and handle/log errors silently. In daemon mode, micrasd log messages are logged in the /var/log/micras.log file.

Using the "-maint" option will enable Maintenance mode for error test and repair.

If micrasd is executed with no arguments, it runs at the console prompt, connects to devices, and waits for errors. Use Ctrl-C to exit micrasd and return to the console prompt.

Micrasd can also be run as a Linux* system service. The Linux* "service" command gives access to start and stop the micras service. For micras service use:

```
[host]# 1service micras start
[host]# 1service micras stop
```

Be aware that micras service has a dependency on the mpss service. The micras service must be started after the mpss service, and be stopped prior to stopping the mpss service. To automatically start the micras service, use the command:

```
[host]# chkconfig micras on
```

```
For RHEL 7:
[host]# systemctl enable micras
```

Use "off" as an argument to disable automatically starting the coprocessor when the host boots.

The errors will be logged into Linux* syslog under /var/log/messages. Use "micras" tag to locate them.

For more information about micrasd, refer to:

```
[host]# micrasd -help
```

8.8 Mpssflash

The mpssflash utility is the POSIX version of the micflash tool. For more information, refer to the man page.

```
[host]$ man mpssflash
```

8.9 Mpssinfo

The mpssinfo utility is the POSIX version of the micinfo tool. For more information, refer to the man page.

```
[host]$ man mpssinfo
```

8.10 Intel® Xeon Phi™ Coprocessor Shell Environment

The Linux* environment on the coprocessor utilizes BusyBox to provide a number of Linux* utilities. The usage of these tools may differ slightly when compared to the usage of similar tools provided with the host Linux* distribution.

For example, the usage of netcat in the Intel® Xeon Phi™ coprocessor environment is:

```
nc [-iN] [-wN] [-l] [-p PORT] [-f FILE|IPADDR PORT] [-e PROG]
```

However, the conventional netcat utility usage is:

```
nc [-46DdhklnrStUuvzC] [-i interval] [-p source_port]
[-s source_ip_address] [-T ToS] [-w timeout] [-X proxy_protocol]
[-x proxy_address[:port]] [hostname] [port[s]]
```

To listen to a tcp port 45678, the user will need to use the following command:

```
nc -lp 45678
```

Instead of

```
nc -l 45678
```

For more information regarding BusyBox, see the link http://www.busybox.net/

9 Recompiling Modules and RPMs from the Intel MPSS Release

The source RPMs required for rebuilding the MPSS and ofed driver modules (see readme.txt, Sec. 2.1 ,“Requirements”, under “Resolution 2”, as well as Sec. 9.1 in this User’s Guide) are included in the Intel MPSS package tar archive.

- Extract the Intel MPSS package. The source RPM files will be included in the mpss-[version number]/src/ folder.

  ```
  [host]$ tar xvf mpss-[version number]\
  -[distribution].tar
  ```

The source code required to build GPL binaries is included in the source tar archive.

1) Go to the Intel® Developer Zone website (Intel® DZ): http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

   Download the mpss-src-[version number].tar file from the “SOURCE” link associated with your Intel MPSS release.
2) Extract the source archive. The archive will be included in the mpss-[version number]/src/ folder.

   ```
   [host]$ tar xvf mpss-src-[version number].tar
   ```

9.1 Recompiling the Intel MPSS RPM specifically for OFED

NOTE: The uninstall script will not uninstall rebuilt drivers. You must remove them manually before running the uninstall.sh script.

On RedHat*:*
```
[host]# yum erase <name of rebuilt package\ rpm>
```

On Suse Linux :
```
[host]# zypper remove <name of rebuilt\
package rpm>
```

The ofed kernel modules need to be recompiled if the kernel is upgraded or changed. First, follow the instructions in readme.txt, Section 2.1, “Requirements”, under “Resolution 2”, and install the resultant mpss-modules and mpss-modules-dev RPMs. Then, rebuild the ofed-driver modules against the new mpss-modules and kernel:

- Red Hat* Enterprise Linux* systems:

1) Install the kernel building prerequisites:

   ```
   [host]# yum install kernel-headers kernel-devel
   ```

2) Rebuild the RPMs from the source RPMs:

   ```
   [host]$ cd <folder where extracted tar file\
   expanded>/src/
   ```

```
[host]$ rpmbuild --rebuild ofed-driver-*.src.rpm
```

3) Install the OFED binary RPMs located under $HOME/rpmbuild:

   ```
   [host]# rpm -ivh $HOME/rpmbuild/RPMS/x86_64/*rpm
   ```

- SUSE* Linux* Enterprise Server (SLES) 11 systems:

1) Install the kernel building prerequisites:

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

2) Rebuild the RPMs from the source RPMs:

   ```
   [host]$ cd <folder where extracted tar \
   file expanded>/src/
   ```

   ```
   [host]$ rpmbuild --rebuild ofed-driver-*.src.rpm
   ```

3) Install the OFED binary RPMs located under /usr/src:

   ```
   [host]# rpm -ivh /usr/src/packages/RPMS/x86_64/*rpm
   ```

Once these steps are completed, restart Intel MPSS by rebooting or performing the following:

```
[host]# 1service ofed-mic stop
[host]# 1service openibd stop
[host]# 1service mpss unload
[host]# 1service mpss start
[host]# 1service openibd start
[host]# 1service ofed-mic start
```

9.2 Recompiling the Intel MPSS GANGLIA* Modules

NOTE: If you recompile you must manually uninstall the new rpm using "*yum erase <my re-compiled rpm>*" before running the uninstall script.

Support enabled for Red Hat* Enterprise Linux* 6.3, 6.4, 6.5, and SUSE* Linux* Enterprise Server (SLES) 11 SP2 and SP3.

1) Install prerequisites (see Section 3.1, "Requirements").
2) Install Intel MPSS (see the Intel MPSS Readme, Section 2.2, "Steps to Install Intel MPSS using OFED+).

3) Extract mpss-ganglia-mpss.tar.bz2.

```
[host]# tar xvf mpss-ganglia-mpss.tar.bz2
```

4) Define the environment variable CROSS_COMPILE.

```
[host]# export CROSS_COMPILE=/opt/mpss/[version\ number]
/sysroots/x86_64-mpsssdk-linux/usr/bin/k1om\   -mpss-linux/k1om-
mpss-linux
```

5) Regenerate the GANGLIA* modules.

```
[host]# make
```

NOTE: Ensure that ganglia-devel-3.1.7, apr-devel-1.3.9-3, and mpss-sdk-k1om-[version number] are installed. It may be required to setup include paths.

9.3 Recompiling the Intel MPSS MIC Management Modules

Support enabled for Red Hat* Enterprise Linux* 6.3, 6.4, 6.5, and SUSE* Linux* Enterprise Server (SLES) 11 SP2 and SP3.

Make sure that RPMs mpss-modules-headers-[version number], mpss-modules-headers-dev, libmicmgmt0, libmicmgmt-dev, libscif0-[version number], libscif-dev and all prerequisites are installed.

Make sure that the mpss-metadata-[version number].tar.bz2 source tarball is untarred in a directory of your choice. You can find the mpss-metadata-[version number].tar.bz2 file in mpss-src-[version number].tar.

1) Install Intel MPSS (see the Intel MPSS Readme, Section 2.2, "Steps to Install Intel MPSS").

2) Install the “a2x” utility.

3) Extract mpss-micmgmt-[version number].tar.bz2

```
[host]# tar xvf mpss-src-[version number].tar
[host]# cd mpss-[version number]/src
[host]# tar xvf mpss-micmgmt-[version number].tar.bz2
[host]# tar xvf mpss-metadata-[version number].tar.bz2
```

4) Regenerate the Intel MPSS MIC Management modules:

```
[host]# cd mpss-micmgmt-[version number]
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.mk miclib/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.c miclib/
```

```
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.mk apps/mpssinfo/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.c apps/mpssinfo/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.mk apps/mpssflash/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.c apps/mpssflash/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.mk apps/micsmc/
[host]# cp ../mpss-metadata-[version number]\
/mpss-metadata.c apps/micsmc/
[host]# make lib
```

Set the DESTDIR environment variable to the desired install path.

```
[host]# make install_lib
[host]# make
[host]# make install
```

A build directory will be created at DESTDIR, and everything will be installed there.

NOTE: If **DESTDIR=/** is specified, the files are installed to their default locations under the /usr directory.

9.4 How to Extract and Use the MYO Open Source Distribution

MYO source is delivered in the file mpss-myo-[version number].tar.bz2. In the tar file, the files are a tree relative to the mpss-myo-[version number] directory. Extract the archive to the desired directory with the following steps.

```
[host]$ cd [directory of choice]
[host]$ tar -xf mpss-myo-[version number].tar.bz2
[host]$ cd mpss-myo-[version number]
```

After MYO is extracted, the myo directory tree will be similar to the following:

```
 mpss-myo-[version number]
   |-- include            Contains header files used in the build of applications and
tutorials.
   |-- src                          Contains source files of MYO runtime library.
   |-- src/include        Contains header files used in the build of the library.
   |-- tools                        Contains the environment setting configuration files.
   |-- docs               Contains myo reference documents.
   `-- docs/tutorials     Contains tutorials that demonstrate how to program with MYO.
```

Drilling down further into the src directories, a map of where features are implemented is as follows:

```
|-- allocator          Contains the memory management module (Memory
|                      Allocator).
|-- communication      Contains the implementation of the communication.
|-- consistent                 Contains the implementation of VSM protocol.
|-- include                    Contains all internal header files.
|-- machinedep         Contains the platform dependent portion of MYO.
|-- misc                       Contains all other files.
|-- Makefile                   Makefile for MYO libraries.
|-- pinnedmem          Contains the implementation of get pinned memory.
|-- rfunc              Contains the implementation of APIs related with remote
|                      function call.
|-- svar                       Contains the implementation of APIs related with shared
|                                      variables.
`-- sync               Contains global (cross-bus) sync operations (mutex, barrier and
                       semaphore).
```

In the mpss-myo-[version number] directory, the README text file explains the purpose, content, and use of the MYO Open Source Distribution. It includes information about compiler selection, building and installing the Myo libraries, Myo system requirements, and the Myo tutorials.

For information about MYO API function calls, refer to the MYO API man pages.

They can be viewed by entering:

```
[host]$ man <function-name>
```

The list of MYO API man pages follows:

myoAcquire.3
myoAcquireOwnership.3
myoArenaAcquire.3
myoArenaAcquireOwnership.3
myoArenaAlignedFree.3
myoArenaAlignedMalloc.3
myoArenaCreate.3
myoArenaDestroy.3
myoArenaFree.3
myoArenaGetHandle.3
myoArenaMalloc.3
myoArenaRelease.3
myoArenaReleaseOwnership.3
myoBarrierCreate.3
myoBarrierDestroy.3
myoiRemoteCall.3
myoiRemoteFuncLookupByAddr.3
myoiRemoteFuncLookupByName.3
myoiRemoteFuncRegister.3
myoiRemoteThunkCall.3
myoiSetMemConsistent.3
myoiSetMemNonConsistent.3
myoiTargetFptrTableRegister.3
myoiTargetSharedMallocTableRegister.3
myoiVarRegister.3
myoMutexCreate.3
myoMutexDestroy.3
myoMutexLock.3
myoMutexTryLock.3
myoMutexUnlock.3

myoBarrierWait.3
myoGetMemUsage.3
myoHTimeOn.3
myoiCheckResult.3
myoiGetResult.3
myoiHostFptrTableRegister.3
myoiHostSharedMallocTableRegister.3
myoiHostVarTablePropagate.3
myoiLibFini.3
myoiLibInit.3
myoiMicVarTableRegister.3
myoRelease.3
myoReleaseOwnership.3
myoSemCreate.3
myoSemDestroy.3
myoSemPost.3
myoSemTryWait.3
myoSemWait.3
mySharedAlignedFree.3
myoSharedAlignedMalloc.3
myoSharedFree.3
myoSharedMalloc.3

9.5 How to Extract and Use the COI Open Source Distribution

COI source is delivered in the file mpss-coi-[version number].src.tar.bz2. In the tar file, the files are packaged with paths relative to the original source directory structure.

To extract and build the COI source, do the following:

Ensure the Intel MPSS driver and Intel MPSS SDK RPM's are installed from the Intel MPSS-[version number].tar package.

```
[host]$ tar xvf mpss-src-[version number].tar
[host]$ cd mpss-[version number]/src/
[host]$ tar xvf mpss-coi-[version number].tar.bz2
```

After COI is extracted, the COI directory tree will be similar to the following:

```
|-- docs                Contains the COI_getting_started.pdf for instructions
|                       on working with COI.
|-- src                 Contains source files of COI runtime library. Also
|                       contains the release_notes.txt document, which
|                       describes new features and known issues.
|-- src/api             Contains the upper-level source files for API calls into
|                       COI.
|-- src/docs_config     Contains the scripts for creating the PDF documents for
|                       COI.
```

```
  |-- src/include                          Contains header files used in the build of
the COI
  |                                                  library.
  |-- src/legal                            Contains the source for specific license
headers for
  |                                                  parts of COI.
  |-- src/mechanism                  Contains the low-level API source files that make
up the
  |                                                  core of the COI
functionality.
  |-- src/policy                           Contains the code for the Task Scheduling
design in COI.
  |-- src/tools                            Contains the source for the tools that
debug and utilize
  |                                                  COI.
  |-- src/tutorial                         Contains tutorials that demonstrate how
to program
  |                                                  with COI.
  |-- Makefile                             Makefile used to build all COI source
code.
  |-- coi_daemon_files.mk          Sub Makefile config for building the COI daemon
files.
  |-- coi_device_files.mk          Sub Makefile config for building the COI device
files.
  |-- coi_host_files.mk            Sub Makefile config for building the COI host files.
  `-- coi_host_version_files.mk  Sub Makefile config for building the COI host version files.

  |-- build                                        Overall Build output directory.
  |-- build/device-linux-debug   Debug built libraries and binaries for coprocessor-side
  |                                                  COI.
  |-- build/device-linux-release Release built libraries and binaries for coprocessor-side
  |                                  COI.
  |-- build/host-linux-debug           Debug built libraries and binaries for host-side
COI.
  `-- build/host-linux-release     Release built libraries and binaries for host-side COI.
```

NOTE: Execute the next line only if mpss-metadata-[version number].tar.bz2 is not extracted yet.

```
[host]$ tar xvf mpss-metadata-[version number].tar.bz2
[host]$ cd mpss-coi-[version number]
[host]$ make -I ../mpss-metadata-[version number]/
```

To directly install the COI libraries and binaries, first make sure that the Intel MPSS driver is running, then do the following:

Installing Host Binaries and Libraries:

```
[host]$ cp build/host-linux-[debug|release]/libcoi_host.so\
/usr/lib64/
[host]$ cd /usr/lib64/
[host]# ln -s libcoi_host.so libcoi_host.so.0
```

Installing Card-side Binaries and Libraries:

```
[host]# ssh mic[card number]
```

NOTE: Here we need to kill the coi_daemon so that the new one can be installed

```
[micN]# killall -9 coi_daemon
[micN]# exit
[host]# scp build/device-linux-[debug|release]/coi_daemon
mic[card\ number]:/usr/bin/coi_daemon
[host]# scp build/device-linux-[debug|release]/libcoi_device.so
\ mic[card number]:/usr/lib64/libcoi_device.so
[host]# ssh mic[card number]
[micN]# cd /usr/lib64/
[micN]# ln -s libcoi_device.so libcoi_device.so.0
[micN]# coi_daemon --coiuser=micuser&
[micN]# exit
```

Once installed, and now that the new coi_daemon is running, the new COI binaries and libraries will be in use in the current running driver.

Building the COI stack also builds the tools as well. If you wish to install the newly built tools coitrace and micnativeloadex, do the following:

```
[host]# cp build/host-linux-[debug|release]/coitrace /usr/bin/
[host]# cp build/host-linux-[debug|release] \
/libcoitracelib.so /usr/lib64/
[host]# cp build/host-linux-[debug|release] \
/micnativeloadex /usr/bin/
[host]# cd /usr/lib64/
[host]# ln -s libcoitracelib.so libcoitracelib.so.0
```

COI Tutorial Build and Execution Instructions:

To build and run the COI tutorials, follow the instructions below:

1) Ensure the prerequisites are installed:
   - mpss-[version number]*.rpm must be installed.
2) Build instructions:

   Change to the src/tutorial/<tutorial> directory and run 'make'.

   ```
   [host]$ cd <tutorial>
   ```

```
[host]# make
```

3) Execution instructions:

a) Ensure the driver is running. Set default configuration values and start the Intel MPSS service.

```
[host]# micctrl --initdefaults
[host]# [1]service mpss start
```

b) Execute the tutorial.

```
[host]$ cd <tutorial>/debug
[host]# ./<tutorial>_source_host
```

10 Supported Environments for the Intel MPSS 3.4 Release

For more information about the supported Intel® Xeon Phi™ coprocessor instructions, refer to the Intel® Xeon Phi™ Coprocessor Instruction Set Reference Manual located at http://software.intel.com/mic-developer.

10.1 Supported Environments

Compiling for the Intel MPSS 3.4 release is only supported in:

- Red Hat* Enterprise Linux* 64-bit 6.2 kernel 2.6.32-220
- Red Hat* Enterprise Linux* 64-bit 6.3 kernel 2.6.32-279
- Red Hat* Enterprise Linux* 64-bit 6.4 kernel 2.6.32-358
- Red Hat* Enterprise Linux* 64-bit 6.5 kernel 2.6.32-431
- Red Hat* Enterprise Linux* 64-bit 7.0 kernel 3.10.0-123
- SUSE* Linux* Enterprise Server SLES 11 SP2 kernel 3.0.13-0.27-default
- SUSE* Linux* Enterprise Server SLES 11 SP3 kernel 3.0.76-0.11-default

Intel strongly recommends using the included GCC compiler ONLY when building the kernel, LSB, and the Symmetric Communications Interface (SCIF) driver. For all other builds, using the Intel® C Compiler or the Intel® FORTRAN Compiler is strongly recommended. Using inappropriate compilers may lead to unpredictable results and is not an Intel tested or supported configuration.

10.2 Compiling Supported Environments

The Linux* host driver source is supplied. This makes it possible to run on unsupported Linux* releases, which may produce driver errors and incompatibilities with the supplied host libraries. The supplied host libraries have been produced with particular versions of the GCC compiler and may not link correctly to code produced by older or newer GCC compilers.

10.3 User Mode Code for Symmetric Communications Interface (SCIF)

SCIF applications must be compiled for both the Intel® Xeon Phi™ coprocessor Linux* OS and the host Linux* OS, depending on the direction of connection establishment. For testing purposes, every test can be compiled for both sides and run as required.

An example to establish a connection with the scif_connect() function has been provided. The following Makefile will compile versions for both the host operating system and the coprocessor Linux* operating system.

```
CC=icc
XCC=icc -mmic
XCCPATH=/usr/linux-k1om-4.7/bin


all:    scif_contest_mic scif_contest_host
scif_contest_mic: scif_contest.c
        (export PATH=$$PATH:$(XCCPATH); \
          $(XCC) -o scif_contest_mic scif_contest.c -lscif -lpthread)
scif_contest_host: scif_contest.c
    $(CC) -DHOST -o scif_contest_host scif_contest.c -lscif -lpthread
clean:
        rm -f scif_contest_mic scif_contest_host
```

10.4 Registration Caching in SCIF

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

Registration caching is a SCIF feature intended to improve the performance of scif_vreadfrom()/scif_vwriteto(). When registration caching is enabled, SCIF caches virtual to physical address translations of the virtual addresses passed to scif_vreadfrom()/scif_vwriteto(), thus eliminating the overhead of pinning pages when the same virtual range is specified in future calls. Registration caching is enabled by default.

- To disable registration caching, the module parameter control file /etc/modprobe.d/mic.conf must be edited.

  Set "reg_cache" equal to zero in the options line for the "mic" module.

- A driver reload is required. Follow the procedure:

```
[host]# 1service mpss unload
[host]# 1service mpss start
```

There is a per-node tunable limit on the maximum number of pinned pages per SCIF endpoint. This limit can only be modified by the root user.

- To set the maximum number of pinned pages when the driver is loaded:

  ```
  [host]# echo limit > /proc/scif/reg_cache_limit
  ```

  Where limit is the number of 4K pages in decimal.

- To disable caching at runtime, set the limit to 0 on each node.

10.5 GNU Debugger (GDB) for the Intel® Xeon Phi™ Coprocessor

- Running natively on the Intel® Xeon Phi™ coprocessors

  Users need to install gdb-7.5+mpss*.k1om.rpm to each respective Intel® Xeon Phi™ coprocessor. The mpss-[version-number]-k1om.tar file contains this RPM.

  Refer to Section 11.3, "Installing Card Side RPMs" for card side installation procedures.

- Running remote GDB on the Intel® Xeon Phi™ coprocessors

  The remote Intel® Xeon Phi™ coprocessor enabled GDB client is located on the host at:

  /opt/mpss/[version number]/sysroots/x86_64-mpsssdk-linux/usr/bin/k1om-mpss-linux/k1om-mpss-linux-gdb

  Where [version number] is the Intel MPSS version number: 3.x.

  The GDB Server is pre-installed in the card by default and is located under:

  /usr/bin/gdbserver

  For complete GDB remote debugging instructions, refer to the chapter "Debugging Remote Programs" in the GDB manual.

- GDB remote support for data race detection

  GDB supports data race detection based on Intel® PDBX data race detector for Intel® Many Integrated Core (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.

- Debugging heterogeneous/offload applications

  Heterogeneous application debugging is supported in Eclipse*. This requires the installation of an Eclipse* plugin. Install mpss-eclipse-cdt-mpm-*.x86_64.rpm.

  Installation steps for the Eclipse* plugin:

  1) From the Eclipse* menu use "Help" -> "Install new Software".
  2) Click on "Add..."
  3) Click on "Local..."
  4) Use the "/usr/share/eclipse/mic_plugin" path and click "OK".
  5) Click "OK" again in the popup window.
  6) Unselect the following two checkboxes: "Group items by category" and "Contact all update sites during install..."
  7) Select the plugin using the corresponding checkbox, then click "Next".
  8) Click "Next" .
  9) Accept the license agreement and click "Finish".
  10) In the "Security Warning" popup, click "OK".
  11) Restart the Eclipse* IDE.

- 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 to the card side offload processes. To do this, execute the following step:

```
[host]$ export AMPLXE_COI_DEBUG_SUPPORT=TRUE
```

10.6 Ulimit Checks for Max Locked Memory in SCIF

This feature enforces ulimit checks of the memory that scif_register() locks. Pages locked using scif_register() are counted towards the ulimit.

NOTE: In kernel versions later 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.

This feature is disabled by default.

- To enable the feature, the module parameter control file /etc/modprobe.d/mic.conf must be edited:

  Set "ulimit" equal to one in the options line for the "mic" module.

- A driver reload is required. Follow the procedure:

```
[host]# 1service mpss unload
[host]# 1service mpss start
```

11 Important Considerations

11.1 Disabling and Enabling Power Management

Package C3 (PC3), and Package C6 (PC6) are power management states that are entered when the coprocessor is idle for a period of time. Power management states are enabled by default in Intel MPSS. Under normal circumstances, the host sends traffic to awaken the coprocessor and resume execution. The following instructions indicate how to disable and re-enable power management.

PREREQUISITES:

The following settings can only be modified by the root user:

- To disable power management package states and allow successful execution of an application that uses the sleep() function on the coprocessor, change the line in each /etc/mpss/micN.conf file to this:

  PowerManagement "cpufreq_on;corec6_off;pc3_off;pc6_off"

- To re-enable power management, change the line in each /etc/mpss/micN.conf file to this:

  PowerManagement "cpufreq_on;corec6_on;pc3_on;pc6_on"

  Where N is an integer number (0, 1, 2, 3, etc.) that identifies each coprocessor installed in the system.

The values may also be changed with the **micctrl --pm=<value>** command. To disable power management, set **<value>** to **off**. To enable, set **<value>** to **default**.

NOTE: Restarting of the coprocessor(s) is required after modifications to the micN.conf files. Use the following command to restart:

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

11.2 Disabling and Enabling the Memory Control Group (cgroup)

The memory Control Group is disabled by default in this release, but it can be enabled in the */etc/mpss/micN.conf* files. Enabling the memory cgroup decreases the amount of memory available to applications on the coprocessor by about 120MB.

PREREQUISITES:

The following settings can only be modified by the root user:

- To enable the memory cgroup, set the 'Cgroup memory' variable to ‘enabled’ in the /etc/mpss/mic*N*.conf files and then restart the Intel MPSS service:
  1) In /etc/mpss/mic*N*.conf:

     Cgroup memory=enabled
  2) Restart the Intel MPSS service:

     ```
     [host]# 1service mpss restart
     ```
- To disable the memory cgroup, set the 'Cgroup memory' variable to ‘disabled’ in the /etc/mpss/mic*N*.conf files and then restart the Intel MPSS service:
  1) In */etc/mpss/micN.conf*:

     Cgroup memory=disabled
  2) Restart the Intel MPSS service:

     ```
     [host]# 1service mpss restart
     ```

11.3 Installing Card Side RPMs

To install card side RPMs, use one of the procedures outlined below in Sections 11.3.1 through 11.3.3.

Currently, the mpss-[version number]-k1om.tar file contains the card-side RPMs. Go to the Intel® Developer Zone website (Intel® DZ): http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss. Download the mpss-[version number]-k1om.tar file from the “Software for Coprocessor OS” link associated with your Intel MPSS release.

In order to use zypper to install RPM files located on the card side, coreutils must first be installed.

To do this:

```
[host]$ xvf mpss-[version number]-k1om.tar
[host]$ cd mpss-[version number]/k1om
[host]# scp coreutils*.rpm micN:.
[host]# scp libgmp*.rpm micN:.
[host]# ssh micN
[micN]# rpm -ihv coreutils*.rpm libgmp*.rpm
```

11.3.1 Copy RPMs to the Card Using SCP

Steps:

1) Extract the mpss-[version number]-k1om.tar file. This will create the mpss-[version number]/k1om directory.

   ```
   [host]$ tar xvf mpss-[version number]-k1om.tar
   ```

2) Change to the extracted folder.

```
[host]$ cd mpss-[version number]/k1om
```

3) SCP the RPMs to the card.

```
[host]# scp <rpm_packages> micN:/tmp
```

4) SSH to the card.

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

5) Install the RPMs via rpm or zypper utility (zypper example shown).

```
[micN]# zypper install /tmp/<rpm_package_name>
```

6) Repeat steps 3-5 for the remaining card(s).

   Where N is an integer number (0, 1, 2, 3, etc.) that identifies each coprocessor installed in the system.

11.3.2 Copy RPMs to Card Using a Repo and Zypper (via HTTP)

Set up an http server (reachable from the card) offering Intel MPSS RPM repodata. The steps in this section require that Python and the createrepo tool be previously installed.

Steps:

1) Create a folder to place the Intel MPSS tarball.

```
[host]$ mkdir mpss-repo
```

2) Go to the Intel® Developer Zone website (Intel® DZ): http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

   Download the mpss-[version number]-k1om.tar file from the “Software for Coprocessor OS” link associated with your Intel MPSS release.

3) Extract the tarball file.

```
[host]$ tar xvf mpss-[version number]-k1om.tar
```

4) Change to the extracted folder.

```
[host]$ cd mpss-[version number]
```

5) Use the createrepo tool to create a new repo.

```
[host]$ createrepo .
```

6) Start the http server as follows:

```
[host]# python -m SimpleHTTPServer
```

7) Create a service to provide RPM installation at every card boot. To achieve this, create a folder under /var/mpss/common/etc/init.d

```
[host]# mkdir -p /var/mpss/common/etc/init.d
```

8) Under /var/mpss/common/etc/init.d create the service file named install-service as follows (example shown applies to GANGLIA* installation):

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          install-service
# Required-Start:
# Required-Stop:     $local_fs
# Default-Start:     S
# Default-Stop:      0 6
# Short-Description: Load node dependencies before notified as
online
# Description:       Load node dependencies before notified as
online
zypper ar http://host:8000 k1om-mpss-[version number]
zypper install ganglia
zypper install mpss-ganglia
```

9) Restart the mpss service.

```
[host]# 1service mpss restart
```

11.3.3 Use the Micctrl Utility

Steps:

1) In 3.x-based releases, place the RPM(s) for the card in a known location on the host.
2) Execute the micctrl command:

```
[host]# micctrl --overlay=rpm \
--source=<location of file> --state=on
```

Where location is the absolute path to a single RPM or a directory where all RPMs contained in it are installed on the card at boot. The state parameter specifies whether or not to include the RPM(s) on the file system.

NOTE: When using the micctrl utility to install card side RPMs, it is not necessary to start the ofed-mic service (Section **2.5**, step 4 is not required).

11.4 BIOS Setting and Process Affinity

BIOS setting and process affinity have a significant performance impact on SCIF. Without the correct BIOS and user process affinity settings, you can expect to see considerable performance penalties.

11.4.1 BIOS Setting

Make sure Intel® Turbo Boost Technology is *enabled* in the BIOS.

11.4.2 Process Affinity

In the Intel® Xeon Phi™ product, each coprocessor belongs to a particular NUMA (non-uniform memory access) node. A device that is connected to a particular node can access the memory in this node faster than it can access the memory in the other nodes. To get the maximum SCIF bandwidth, ensure your process runs on the same node as the coprocessor you communicate with by setting the process affinity.

12 Intel MPSS Boot Configuration Details

The Intel® Xeon Phi™ coprocessors are PCIe based add-in cards that run a version of Linux* tailored for these coprocessors. The Linux* OS for Intel® Xeon Phi™ coprocessors, as well as a range of drivers and utilities, are included in the Intel® Manycore Platform Software Stack (Intel MPSS). The responsibilities of these drivers and utilities include:

- Placing the Linux* boot image and root file system into coprocessor memory.
- Controlling coprocessor booting, shutdown and reset.
- Providing a virtual console.
- Providing an IP (over PCIe) networking connection to each coprocessor.
- Directing power management of each coprocessor.
- Supporting high speed data transfer to and from the coprocessor.

The PCIe bus is the only communication channel available to the Intel ® Xeon Phi™ coprocessors. Therefore configuration and provisioning of the OS to be executed on each Intel® Xeon Phi™ coprocessor is performed by the host system in which the coprocessor is installed.

The Linux* kernel and file system image for the Intel ® Xeon Phi™ coprocessors are installed into the host file system as part of Intel MPSS installation. The coprocessor file system image can be configured through the use of the **micctrl** utility described below and/or directly by the host root.

The **mic.ko** driver is the component of Intel MPSS that provides PCIe bus access and implements the coprocessor boot process. To boot a coprocessor, mic.ko injects the Linux* kernel image and a kernel command line into coprocessor memory and signals it to begin execution. A virtual console driver and a virtual network driver are also built into mic.ko. Finally, mic.ko directs power management of the installed coprocessors and provides a high speed data transfers over PCIe through its Intel® Symmetric Communications Interface (SCIF) driver.

The **mpssd** daemon directs the initialization and booting of the Intel® Xeon Phi™ coprocessors based on a set of configuration files. mpssd is started and stopped with the Linux* service **mpss**, and instructs the cards to boot or shutdown. It supplies the final file system image to the cards when requested. mpssd will also log debug information from each coprocessor.

micctrl is a utility through which the user can control (boot, shutdown, reset) each of the installed Intel® Xeon Phi™ coprocessors. micctrl also offers numerous options to simplify the process of configuring each coprocessor. Section 15 of this document, “The micctrl Utility”, describes the **micctrl** utility in detail.

13 Intel MPSS Boot Process

Booting the Linux* kernel on the Intel® Xeon Phi™ coprocessor requires a number of steps. Figure 2 Boot process for Intel MPSS shows the sequence of steps that are performed during the Intel MPSS boot process.

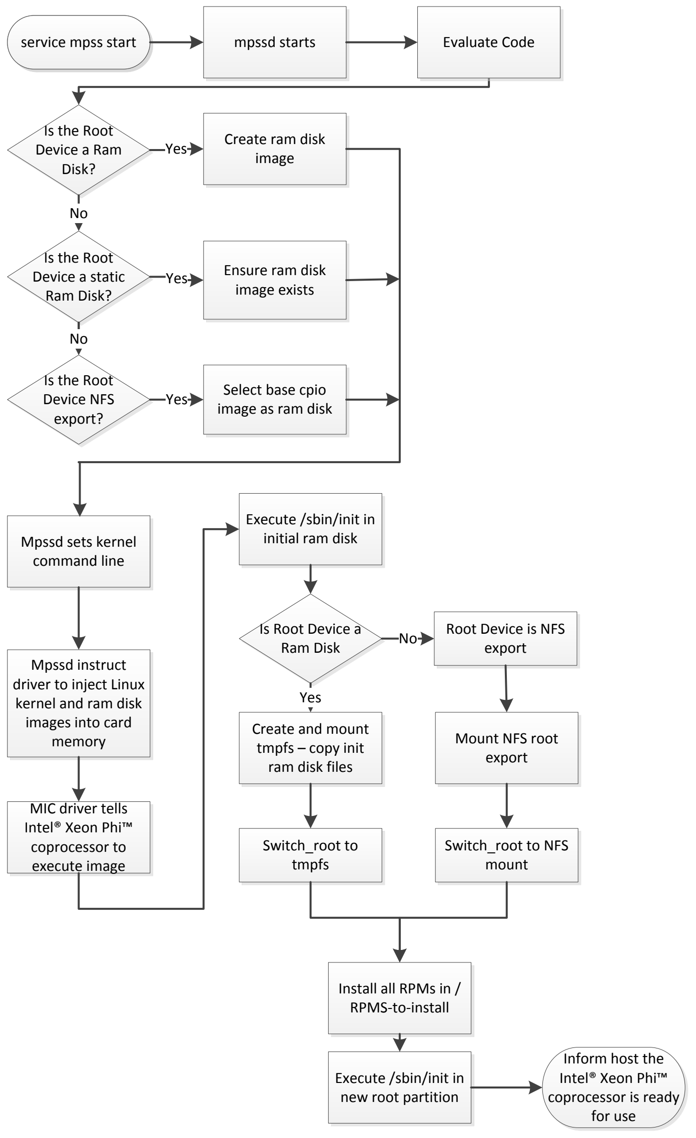
service mpss start
mpssd starts
Evaluate Code
Is the Root Device a Ram Disk?
Yes
Create ram disk image
No
Is the Root Device a static Ram Disk?
Yes
Ensure ram disk image exists
No
Is the Root Device NFS export?
Yes
Select base cpio image as ram disk
Mpssd sets kernel command line
Mpssd instruct driver to inject Linux kernel and ram disk images into card memory
MIC driver tells Intel® Xeon Phi™ coprocessor to execute image
Execute /sbin/init in initial ram disk
Is Root Device a Ram Disk
No
Root Device is NFS export
Yes
Create and mount tmpfs – copy init ram disk files
Mount NFS root export
Switch_root to tmpfs
Switch_root to NFS mount
Install all RPMs in / RPMS-to-install
Execute /sbin/init in new root partition
Inform host the Intel® Xeon Phi™ coprocessor is ready for use

Figure 2 Boot process for Intel MPSS

13.1 Booting the Intel® Xeon Phi™ Coprocessor

This section describes the key steps that are performed during the Intel MPSS boot process on the Intel® Xeon Phi™ coprocessor.

13.1.1 Kernel Command Line

On most Intel® based systems, loading and executing the Linux* kernel image is controlled by the **grub** boot loader. In the **grub** configuration file, each possible kernel definition contains a number of parameters to be passed to Linux* through its kernel command line. In the Intel MPSS boot process, this is done by the **mpssd** daemon parsing its configuration files. The kernel command line is created based on values in the configuration files and placed in */sys/class/mic/mic<id>/cmdline* for the driver to retrieve it.

13.1.2 Instruct the Driver to Boot the Intel® Xeon Phi™ Coprocessor

The **mpssd** daemon requests the Intel® Xeon Phi™ coprocessor to start executing the Linux* image by writing a boot string to the */sys/class/mic/mic<id>/state* file. This file is a link into the MIC driver through a Linux* sysfs entry. The format of the request must be:

```
boot:linux:<Linux* image file name>:<ram disk file name>
```

The options **reset** and **shutdown** may also be written to **state** entry and will be discussed later.

The second part of the boot argument indicates to boot a Linux* image. It may also be set to **elf** to indicate booting a standard ELF format file. Documenting non-Linux* boot is beyond the scope of this document.

When the driver receives the boot request, it first checks to see whether the card is in the **ready** state. If the card is not ready to boot it will return an error through the write call to the sysfs entry and not attempt to boot the card. Otherwise it sets the state of the Intel® Xeon Phi™ coprocessor to **booting**.

The driver then saves the image and initial ram disk file names for later retrieval through the */sys/class/mic/mic<id>/image* and */sys/class/mic/mic<id>/initramfs* sysfs entries. It also sets the mode to indicate it is booting a Linux* image

The driver will copy the kernel command line setting requested by the **mpssd** daemon, along with a number of addresses in host memory required by various drivers in the Linux* image to its location in card memory. It then copies the requested Linux* image and ram disk files into the Intel® Xeon Phi™ coprocessor's memory.

The last step is to write to the Intel® Xeon Phi™ coprocessor register instructing it to start executing the injected image.

13.1.3 Linux* Kernel Executes

Executing the Linux* kernel code functions as it does on any Intel® based machine. It initializes hardware, starts kernel services, and sets all the CPUs to the “online” state. When the kernel is ready, it initializes its attached initial ram disk image and starts executing the **init** program in the image.

As on any Intel® based Linux* system, the initial ram disk contains the loadable modules required for the real root file system. Many of the arguments passed in the kernel command line are addresses required for the modules to access host memory. The **init** program parses the kernel command line for needed information and creates the */etc/modprobe.conf* file needed by the card’s init process.

The last step is for the **init** program to check the **root=** parameter in the kernel command line for the type of device containing the root file system, and take the appropriate actions.

13.1.4 Root is the Initial Ram Disk

This option is no longer available.

13.1.5 Root is a Ram Disk Image

If the root is set to be a ram file system, the **init** program creates a **tmpfs** (Linux* ram disk file system type) in Intel® Xeon Phi™ coprocessor memory. It then copies all the files from the initial ram disk image into the new tmpfs mount.

If any RPM files exist in the */RPMS-to-install* directory, they will be installed. After installation this directory is removed to free disk space.

The ram disk image is activated as the root device by calling the Linux* **switch_root** utility. This special utility instructs the Linux* kernel to remount the root device on the tmpfs mount directory, release all file system memory references to the old initial ram disk and start executing the new */sbin/init* function.

/sbin/init performs the normal Linux* user level initialization. All the information required must have already been in the compressed cpio file.

13.1.6 Root is an NFS Export

If an NFS mount is indicated to supply the root device, the **init** program will initialize the **micN** virtual network interface to the IP address supplied on the kernel command line and mount the NFS export from the host.

As in the ram disk image, the NFS mount is activated as the root device by calling the Linux* **switch_root** utility. This special utility instructs the Linux* kernel to remount the root device on the NFS mount directory, release all file system memory references to the old initial ram disk and start executing the new */sbin/init* function.

/sbin/init performs the normal Linux* user level initialization. All the information required must have already been in the NFS export.

13.1.7 Notify the Host that the Intel® Xeon Phi™ Coprocessor System is Ready

The last step of any of the three initializations is to notify the host that the coprocessor is ready for access. It does this by writing to its */sys/class/micnotify/notify/host_notified* entry. This causes an interrupt into the host driver which updates the card's state to **online**.

14 Configuration

This section focuses on configuring Intel® Xeon Phi™ coprocessors, including configuration files, kernel command line parameters, and authentication.

14.1 Configurable Components

On a typical Linux* system, the installation and configuration process is performed as a series of questions posed by the system and answered by the installer/operator. Since the Intel® Xeon Phi™ coprocessors do not have a file system of their own, this process is replaced by the installation of RPMs containing the required software on the host and then configured by a combination of editing of configuration files and using the **micctrl** utility. In most cases, direct editing of the configuration files has been deprecated in favor of using the micctrl command directly.

The configuration parameters have three categories:

1) Parameters that control loading the Intel® Xeon Phi™ coprocessor Linux* kernel onto the card and initiating the boot process.
2) Parameters to define the root file system to be used on the card.
3) Parameters to configure the host end of the virtual Ethernet connection.

The current configuration parameters can be displayed with the **micctrl --config** command. For example, the default configuration on most systems looks like the following:

```
micN:
=============================================================
   Linux Kernel:   /usr/share/mpss/boot/bzImage-knightscorner
   BootOnStart:    Enabled
   Shutdowntimeout: 300 seconds

   ExtraCommandLine: highres=off
   PowerManagement: cpufreq_on;corec6_off;pc3_on;pc6_on

   UserAuthentication: Local

   Root Device:   Dynamic Ram Filesystem /var/mpss/micN.image.gz from:
            Base:      CPIO /usr/share/mpss/boot/mpss-image-minimal-knightscorner.cpio.gz
            CommonDir: /var/mpss/common
            MicDir:    /var/mpss/micN

   Network:        Static Pair
      Hostname:    eagles-micN.music.local
      MIC IP:      172.31.1.1
```

```
    Host IP:      172.31.1.254
    Net Bits:     24
    NetMask:   255.255.255.0
    MtuSize:      64512
    MIC MAC:  4c:79:ba:1e:01:4a
    Host MAC: 4c:79:ba:1e:01:4b

  Console:              hvc0
  VerboseLogging:       Disabled
  CrashDump:            /var/crash/mic 16GB
```

14.2 Configuration Files

This section briefly discusses configuration file formats and use of the **Include** parameter to **micctrl**.

14.2.1 Configuration File Location

Configuration is controlled by per card configuration files located, by default, in the */etc/mpss* directory. Each card has an associated micN.conf configuration file, where N is the integer ID of the particular coprocessor it defines (i.e., micN.conf, micN.conf, etc.).

System administrators may wish to maintain more than one set of configuration information for the coprocessors in their systems. Configurations may be switched by specifying different configuration files. The location of any set of configuration files may be specified in the special configuration file **/etc/sysconfig/mpss.conf**. The top level directory to be used for the configuration files (instead of */etc/mpss*) is indicated by adding a line with the format **MPSS_CONFIGDIR=<new location>**. The updated location is honored by both the **mpssd** daemon and the **micctrl** utility.

The configuration directory may also be specified with **micctrl --configdir**. For example, the system administrator may have four different configurations for the coprocessors based on which user will be using the card. To accomplish this, the administrator creates the directories */micuser/john*, */micuser/paul*, */micuser/george* and */micuser/ringo*, corresponding to each of the four users.

```
[host]# mkdir -p /micuser/john /micuser/paul \ /micuser/george
/micuser/ringo
```

In each of these directories, the subdirectories *config* and *var* are added (examples not shown). To configure the system, the administrator first creates the four configurations:

```
[host]# micctrl --configdir=/micuser/john/config \
--initdefaults
[host]# micctrl --configdir=/micuser/paul/config \
--initdefaults
[host]# micctrl --configdir=/micuser/george/config \
--initdefaults
[host]# micctrl --configdir=/micuser/ringo/config \
--initdefaults
```

Next, the administrator differentiates the file system information for each of the users with the micctrl **--micdir** and **--commondir** options, and copies files to the file system parts as required for each user. Since each user has a unique configuration, the system administrator must switch configurations as required. Using a text editor, the system administrator creates or edits the **/etc/sysconfig/mpss.conf** file. For example, to specify Paul as the user, the mpss.conf file would contain:

```
MPSS_CONFIGDIR=/micusers/paul/config
```

Then, the system administrator must perform a “[1]service mpss restart” to restart mpssd with the new configuration location.

14.2.2 Configuration File Format

Each of the per card configuration 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 same line.

14.2.3 Configuration Version

Parameter Syntax:

```
Version <major number> <minor number>
```

The **Version** parameter sets the coprocessor configuration file version. As new releases are produced, the version is used by the **micctrl --initdefaults** command to identify where to update configuration files. This parameter should not be manually edited.

14.2.4 Including Other Configuration Files

Parameter Syntax:

```
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 be found in *etc/mpss*. The configuration parser processes each parameter sequentially. When the **Include** parameter is encountered, the included configuration file(s) are immediately processed. If a parameter is set multiple times, the last instance of the parameter setting will be applied.

By default, the */etc/mpss/default.conf* file is included at the start of each coprocessor specific file (e.g. micN.conf, micN.conf, etc.). This allows the coprocessor specific files to override any parameter set in **default.conf**.

The second entry in the **mic<ID>.conf** files is typically (and by default) the line:

```
Include conf.d/*.conf
```

This is a special rule, specifying that all the files in the */etc/mpss/conf.d* directory will be included. Including both files at the top allows the card specific configuration file to override any common values that were specified.

14.3 Configuring Boot Parameters

The host system boots the Intel® Xeon Phi™ coprocessor by injecting the Linux* kernel image and kernel command line into coprocessor memory and then instructing the coprocessor to start. To perform this operation, the host system reads the configuration files, and builds the kernel command line from relevant parameters. By default, the boot parameters are placed in the per-card micN.conf files, allowing each card to be configured independently of the other cards. If a boot parameter is placed in the defaults.conf file, then it will apply to all cards unless overridden

14.3.1 What to Boot

Parameter Syntax:

```
OSimage <linux_kernel_image> <system_address_map_file>
```

The OSimage parameter specifies the Intel® Xeon Phi™ coprocessor Linux* OS boot image and its associated system address map file. The default values are */usr/share/mpss/boot/bzImage-knightscorner* and */usr/share/mpss/boot/System.map-knightscorner*.

The system owner can specify a different kernel image by using the **micctrl --osimage** command or by editing this parameter.

The change takes effect upon executing either [1]**service mpss start** or **micctrl -b**.

When to Boot

Parameter Syntax:

```
BootOnStart <Enabled | Disabled>
```

The **BootOnStart** parameter controls whether the Intel® Xeon Phi™ coprocessor is booted when the Intel MPSS service starts. If set to Enabled, the **mpssd** daemon will attempt to boot the Intel® Xeon Phi™ coprocessor when [1]**service mpss start** is called.

BootOnStart may be changed using the **micctrl --autoboot** command.

14.3.2 VerboseLogging Kernel Command Line Parameter

Parameter Syntax:

```
VerboseLogging <Enabled | Disabled>
```

The **VerboseLogging** parameter specifies whether the **quiet** kernel command line parameter is passed to the Intel® Xeon Phi™ coprocessor on boot. The **quiet** kernel parameter suppresses most kernel messages during kernel boot. **VerboseLogging** is disabled by default. Enabling **VerboseLogging** will increase boot times.

Changes to **VerboseLogging** take effect upon executing [1]**service mpss start** or **micctrl -b**.

NOTE: This parameter may be deprecated in future releases.

14.3.3 Console Kernel Command Line Parameter

Parameter Syntax:

```
Console "<console device>"
```

Intel MPSS software provides a PCIe bus virtual console driver. Its device node (hvc0) is the default value assigned to the **Console** parameter. Other possible values are intended for internal use.

Changes to **Console** take effect upon executing [1]**service mpss start**.

14.3.4 ExtraCommandLine Kernel Command Line Parameter

Parameter Syntax:

```
ExtraCommandLine "<string>"
```

The **ExtraCommandLine** parameter specifies kernel command line parameters to pass to the Intel® Xeon Phi™ coprocessor 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.

CAUTION: Exercise caution when editing this parameter.

Changes to **ExtraCommandLine** take effect upon executing [1]**service mpss start** or **micctrl -b**.

14.3.5 PowerManagement Kernel Command Line Parameter

Parameter Syntax:

```
PowerManagement
"cpufreq_<on|off>;corec6_<on|off>:pc3_<on|off>:pc6_<on|off>"
```

The **PowerManagement** parameter is a string of four attributes passed directly to the kernel command line for the card's power management driver. The **mpssd** daemon and **micctrl** utility do not validate any of the parameters in this string or its format. Consult power management documentation for correct values.

Changes to power management parameters may be specified using the **micctrl --pm** command.

Changes to **PowerManagement** take effect upon executing [1]**service mpss start** or **micctrl -b**.

14.3.6 ShutdownTimeout Parameter

Parameter Syntax:

```
    ShutdownTimeout [value]
```

Setting **value** to a positive integer specifies the maximum number of seconds to wait for the card to shut down. If shut down time exceeds the value it is reset.

Setting **value** to any negative value indicates to wait indefinitely for the card to shut down.

Setting **value** to zero indicates to simply reset the card without waiting for it to shut down.

The default value for shutdown is set to 300 seconds by the **micctrl --initdefaults** command. Changing the value requires a restart of the mpss service to activate it.

14.3.7 CrashDump Parameter

Parameter Syntax:

```
    CrashDump <dirname> <limit>
```

The **CrashDump** parameter specifies the directory in which to place coprocessor crash dump files and the maximum size of the files to create. The **micctrl --initdefaults** command sets defaults to the directory */var/crash/mpss* and a maximum size of 16 gigabytes.

14.3.8 Cgroup Parameter

Parameter Syntax:

```
    Cgroup [memory=<disabled|enabled>]
```

The **Cgroup** parameter implementation is currently limited to specifying the status of enabling the memory cgroup category. Cgroup memory support in the Linux* kernel adds significant overhead. In most cases, customers will want leave it disabled. The default value created by the **micctrl --initdefaults** command is to set the functionality to disabled. Cgroup's memory support may be enabled by editing the configuration file and changing the parameter or using the **micctrl --cgroup** command. In either case the functionality will be enabled on the next boot of the associated coprocessor.

14.3.9 RootDevice Kernel Command Line Parameter

Parameter Syntax:

```
RootDevice <type> [<location> [</usr location>]]
```

The **RootDevice** parameter defines the type of root device to mount. The **type** argument is a string specifying the device type. The **location** argument is the location information of the file system for the Intel® Xeon Phi™ coprocessor.

Current supported types are **RamFS**, **StaticRamFS**, **NFS**, and **SplitNFS**.

The **RamFS** type builds a compressed cpio ram disk image when a request to boot is received. The file created is specified by the **location** parameter. The image is used as the contents to be loaded into the root ***tmpfs*** file system.

The **StaticRamFS** type causes the compressed cpio image at **location** to be used as the contents of the root file system for the booting coprocessor. The **StaticRamFS** boot will fail if the image file is not already present at **location**.

The static ramfs image may have been previously created by booting with **RootDevice** set to **RamFS**. Optionally, when **RootDevice** is **StaticRamFS**, the **micctrl --updateramfs** command causes a compressed cpio image to be built and placed at the **location** argument to **StaticRamFS**. System administrators may also supply their own initial ram disk image.

The **NFS** type instructs the booting coprocessor to mount the NFS share specified by the **location** argument as the root file system. The location must be a fully qualified NFS mount location with the format "server:location". For example it may look like 10.10.10.12:/export/micN or host:/var/mpss/micN.export.

The **SplitNFS** type is the same as **NFS** except it also provides a separate NFS share at **/usr location** to mount as the **/usr** directory on the card.

The effects of changes to **RootDev** take effect upon executing [1]**service mpss start** or **micctrl -b**.

For more information, refer to Section 16.1, "The File System Creation Process".

14.4 Root File System

Every Linux* system needs a root file system with a minimal set of files. Other nonessential files may be on the root or they may be on secondary mounts. Most modern Linux* OS releases assume the root file system will be large enough to install the complete release into. The Intel® Xeon Phi™ coprocessor embedded file system currently follows the same rule.

Files on the root fall into three categories: the binaries installed with the system, the files in the */etc* directory, which are used for configuring parameters uniquely to an individual system, and the set of files for the users of the system.

Intel MPSS provides a set of configuration parameters that are used in building the root file system image. When a root file system image is (re)built it is controlled by the **RootDevice** parameter. Refer to Section 14.3, "Configuring Boot Parameters" and Section 16.1, "The File System Creation Process" for more information.

14.4.1 File Location Parameters

The **mpssd** daemon and **micctrl** command require the location of the files to be placed in the final root disk image to be used on the card. The files are located using the four configuration parameters **Base**, **CommonDir**, **Overlay**, and **MicDir**. Of the four, the Overlay parameter is the only one allowed to be specified multiple times.

Parameter Syntax:

```
Base <type> <target>
```

The **Base** parameter specifies most of the standard binaries for the Intel® Xeon Phi™ coprocessor. The **type** argument must be set to **CPIO** or **DIR**. The default is **CPIO** and the **location** argument is set to the */usr/share/mpss/boot/initramfs-knightscorner.cpio.gz* file installed from the Intel MPSS RPMs.

Setting the **Base** parameter to **DIR** requires the location argument to be set to the directory where the CPIO file has been expanded. Use the **micctrl --base** command to help create the new directory. Once the CPIO file has been expanded, the system administrator may update the files as desired.

Changes to the **Base** parameter take effect the next time the Intel® Xeon Phi™ coprocessor is booted.

Parameter Syntax:

```
CommonDir <source> <target>
```

The **CommonDir** parameter defines a set of files that the system administrator wishes to have on all the Intel® Xeon Phi™ coprocessor file systems. There are no files installed in this directory by default, and added files will be maintained between updates to the Intel MPSS installation.

```
The source argument is the directory containing the files to be
placed in the card's file system. The target argument is a list
of the files with information on type, ownership, and
permissions.  The use of the target parameter has been
deprecated. New configuration files generated with the micctrl -
-initdefaults command will not include it. If the micctrl --
resetdefaults command is executed the target argument will be
removed.
```

Parameter Syntax:

```
Overlay <type> <source> <target> <on|off>
```

The **Overlay** parameter is the only one of the set that can be used multiple times. Each time it is listed, it may specify an additional set of files to add to the file system. This parameter is used to add additional software to be automatically included.

There are four types of overlays and are specified by the string **FileList**, **Simple**, **File**, and **RPM**.

Setting the **type** argument to **FileList** places files from the directory **source** onto the ram file system image based on specifications in the **target** file. As in the **CommonDir** parameter, the **source** argument defines the directory containing the files to be placed on the card's file system, and the **target** file contains the descriptors.

Setting the **type** argument to **Simple** specifies to copy all files found under the directory **source** into the directory **target** in the card's file system with owner and permission matching the files on the host.

Setting the **type** argument to **File** specifies to copy the single file **source** to **target** in the card's file system.

Setting the **type** argument to **RPM** specifies placing the single Knight's Corner RPM file **source** or all the Knight's Corner RPM files in the directory **source** in the */RPMs-to-install* directory on the Card's file system. During boot of the card, the **init** program will attempt to install any RPM so placed into the ram file system.

Parameter Syntax:

```
MicDir <location> <descriptor file>
```

These parameters collectively specify all the files from which a root file system cpio image is to be built. Each parameter has two required arguments. The **location** parameter is a host subdirectory. The **descriptor file** parameter identifies a file that describes where files in the directory subtree at **location** are to be placed in the Intel® Xeon Phi™ coprocessor's file system, and the permissions of those files.

The use of the **descriptor file** parameter has been deprecated. New configuration files generated with the **micctrl --initdefaults** command will not include it. If the **micctrl --resetdefaults** command is executed the **descriptor file** argument will be removed.

The **MicDir** parameter defines the per card information required to make the Intel® Xeon Phi™ coprocessor unique. There are no files installed in this directory and most of its content is created by the configuration process (**micctrl** in particular). Specifically, user access and network configuration each has its own set of configuration parameters.

14.4.2 Intel MPSS RPM Location

Parameter Syntax:

```
K1omRpms <location>
```

The file system format for the Intel MPSS Linux* release is based on and includes the control files to install RPM formatted installation files. Some **micctrl** options require the location of k1om type Intel MPSS RPM files.

The **KlomRpms** parameter defines a location on the host file system where the Intel MPSS RPMS (klom type) are to be found. The System Administrator must set this option and ensure the RPM files actually exist at that location. The easiest way to set this parameter is with the **micctrl --rpmdir** command.

The actual use of this parameter is very limited and it may be replaced with more useful functionality in the future.

14.4.3 User Access

Parameter Syntax:

```
UserAuthentication None
UserAuthentication Local <low uid> <high uid>
```

The **UserAuthentication** parameter has been removed. Refer to the sections on micctrl specificiation of users for the cards for configuration user access.

14.4.4 Service Startup

NOTE: This section is still functional but there are no longer default services using it. It may be fully depricated and removed in the future.

During boot, the embedded Linux* OS on the Intel® Xeon Phi™ coprocessor executes the script files in the */etc/rc5.d* directory. These entries are links to the actual script files in the */etc/init.d* directory. The links are named with the standard Linux* custom starting with an 'S' for start or 'K' for stop followed by the position parameter and then the file name from the init.d directory. The position parameter is a number from 01 to 99 establishing the order in which the scripts are executed.

```
Service <name> <start> <stop> <state>
```

The Intel MPSS stack installs several pieces of software with various service scripts. The system administration may not want all of them to start at boot. To support this functionality, the configuration files specify the creation of the files in */etc/rc5.d* based on the **Service** configuration parameter. Each file in */etc/init.d* will require a **Service** entry in an Intel® Xeon Phi™ coprocessor configuration file.

The **name** argument is the name of the actual script found in the */etc/init.d* directory.

The **start** argument defines the order the service start relevant to other scripts. It will be a value from 1 to 99. As an example the network interface must be initialized before the secure shell daemon can be started. The **network** script is assigned a start value of 21 and sshd is assigned 80.

The **stop** argument is the opposite of the start parameter and is generally set to 100 minus the **start** value. This will assure on shutdown the secure shell daemon at 5 will shut down before the network is unconfigured at 79.

The **state** argument determines whether the links specifies an 'S' for start or 'K' for stop. It follows the **chkconfig** utility convention of **on** for start and **off** for stop.

14.4.5 Network Access

Network access to the Intel® Xeon Phi™ coprocessors is a complicated process depending on the network topology to support. The supported coprocessor topologies have been categorized into static pair, internal bridge and external bridge topologies. Using a combination of the **Bridge** and **Network** configuration parameters allows a diverse and robust network setup.

Each Linux* system in a network uses a host name to identify itself. The **Hostname** parameter is used to configure it.

Each network interface is identified by its **MAC** address. The virtual network devices between the host and card needs its own address. These addresses are configured using the **MacAddrs** parameter.

On the host operating system, files are added to the network configuration based on the host OS type (Red Hat* or SUSE*). On the card file systems the files added are:

/etc/network/interfaces

/etc/hostname

/etc/ssh/ssh_host_key

/etc/ssh/ssh_host_key.pub

/etc/ssh/ssh_host_rsa_key

/etc/ssh/ssh_host_rsa_key.pub

/etc/ssh/ssh_host_dsa_key

/etc/ssh/ssh_host_dsa_key.pub

/etc/ssh/ssh_host_ecdsa_key *# if present*

/etc/ssh/ssh_host_ecdsa_key.pub *# if present*

/etc/resolv.conf

/etc/nsswitch.conf

```
/etc/hosts
```

All network configuration parameters take effect upon executing [1]**service mpss start**.

14.4.5.1 Host Name Assignment

Parameter Syntax:

```
    Hostname <name>
The Hostname parameter defines the value assigned to the host
name on the Intel® Xeon Phi™ coprocessor.  The initial value
from the micctrl --initdefaults command is set to the hostname
with a dash and the card name appended to it.  The host name
string may be edited in the card specific configuration.
```

14.4.5.2 MAC Address Assignment

Configuring the virtual network interface between Intel® Xeon Phi™ coprocessors and the host is a non-trivial process and differs based on the required topology. However, as a prerequisite, both ends of the virtual network need to have **MAC** addresses assigned.

```
MacAddrs Serial

MacAddrs Random

    MacAddrs <host MAC>:<card MAC>
```

The current driver associated with the Intel MPSS release creates MAC addresses based on the serial number of the Intel® Xeon Phi™ coprocessor. If the card is an older version with an invalid serial number, it will randomly assign the MAC address.

The system administrator may override the default created by the driver with the **MacAddrs** configuration parameter. The micctrl --mac command can be used to set this parameter.

Both the driver and micctrl will create MAC addresses where the lower bit being set indicates the host side of the virtual network connection.

The current driver also by default assigns address starting with an IEEE assigned 4C:97:BA to easily identify the Intel® Xeon Phi™ coprocessor interfaces.

14.4.5.3 Static Pair (Default) Topology

Parameter Syntax:

```
Network class=StaticPair \
    micip=<cardIP> hostip=<hostIP> \
    mtu=<mtu size> netbits=<netbits> \
modhost=<yes|no> modcard=<yes|no>
```

In the static pair topology, every Intel® Xeon Phi™ coprocessor is assigned to a separate subnet known only to the host.

The IP address for the Intel® Xeon Phi™ coprocessor and host ends of the virtual network connection must be a fully qualified IP address and the first three quads of the addresses must match. The **micctrl --initdefaults** or **micctrl --resetconfig** commands assign a default value to the top two quads of “172.31”. The third quad indicates the card number. The Intel® Xeon Phi™ coprocessor end of the virtual connection is assigned “1” for the last quad and the host end is assigned “254”. For example, the host end of the **micN** card will have the IP address “172.31.1.254” and the card end will be assigned the IP address “172.31.1.1”.

The **netbits** argument specifies the prefix parameter to be used to formulate the NETMASK to assign to the interface. For a static pair configuration it should never be necessary to change this parameter.

The **mtu** parameter will specify the packet size to use over the virtual network connection. The default value of 64k has been shown to provide the highest network performance.

It is up to the system administrator to correctly route the virtual Ethernet nodes to the external network or each other.

The **modhost** parameter determines whether the system administrator wants to modify the host */etc/hosts* file. The **modcard** parameter determines whether the system administrator wants the configuration process to create the */etc/hosts* on the card. Using either will override the use of the old deprecated **hosts** parameter. The **hosts** parameter may still be used and setting it will be treated as setting **modhost** and **modcard** with the specified parameter.

NOTE: Setting **--modhost=no** will remove any pre-existing */etc/hosts* entries that contain the comment "#Generated-by-micctrl".

The static pair network configuration can be changed by editing a card’s configuration file **Network** parameter. The recommended method of changing the **Network** parameter is to use the **micctrl --network** command (see the **micctrl** section of this document). The **--network** method explicitly knows the previous configuration and can remove it before creating the new one. In either case, all of the network control files will be created when the operation is done.

14.4.5.4 Internal Bridge Topology

Parameter Syntax:

```
Bridge [name] Internal [IP] [netbits] <mtu>
Network class=StaticBridge bridge=<name> \
micip=<cardIP> modhost=<yes/no> modcard=<yes/no>
```

Linux* provides a mechanism for bridging network devices to a common network. The terminology “internal bridge”, in the context of Intel® Xeon Phi™ coprocessor configuration, refers to the process of bridging together multiple Intel® Xeon Phi™ coprocessor virtual network interfaces, on the same host.

Internal bridge configuration is specified by adding a **Bridge** parameter to the default.conf file to specify the bridge information, and then changing the **Network** parameter to point to it.

The first argument to the **Bridge** parameter specifies the name of the bridge and will be used to create the host bridge configuration file. The same name will be used in the **Network** parameter to identify the bridge it will be added to.

The IP address must be set to a fully qualified address.

The **NetBits** parameter is set to 24 by default and will rarely need to be changed.

The Optional **MTU** parameter sets the packet size used over the virtual Ethernet. The driver uses a default size of 64k if this value does not change. Testing has shown this to be the optimal size for the virtual network.

Each Intel® Xeon Phi™ coprocessor's configuration must contain a **Network** entry pointing to the added bridge.

The **bridge** argument must match the name of a bridge specified with a **Bridge** parameter.

The **modhost** parameter determines if the system administrator wants to modify the host */etc/hosts* file or not. The **modcard** determines if the system administrator wants the configuration process to create the */etc/hosts* on the card. Using either will override the use of the old deprecated **hosts** parameter. The **hosts** parameter may still be used and setting it will be treated as setting **modhost** and **modcard** with the specified parameter.

NOTE: Setting **--modhost=no** will remove any pre-existing */etc/hosts* entries that contain the comment "#Generated-by-micctrl".

The resulting configuration files will use the bridge configuration for the **MTU** and **NetBits** to ensure they match.

The internal bridge network configuration can be changed by editing a card's configuration file **Network** parameter. The recommended method of changing the **Network** parameter is to use the **micctrl --network** command (see the **micctrl** section of this document). The
--network method explicitly knows the previous configuration and can remove it before creating the new one. In either case, all the network control files will be created when the operation is done.

14.4.5.5 External Bridge Topology

Parameter Syntax:

```
Bridge [name] External [IP] [netbits] [MTU]
Network class=StaticBridge bridge=<name> \
<micip=<cardIP> modhost=<yes/no> modcard=<yes/no>>
   Network class=Bridge bridge=<name>
```

The Linux* bridging mechanism can also bridge the Intel® Xeon Phi™ coprocessor virtual connections to a physical Ethernet device. In this topology, the virtual network interfaces become configurable to the wider subnet. The Intel® Xeon Phi™ coprocessor configuration must become aware of the network bridge before virtual network interface can be attached to it.

The **Bridge** configuration parameter with the **type** argument set to **External** creates this mapping. If the corresponding bridge networking configuration file (ex: ifcfg-br0) does not exist then configuring this parameter will cause it to be generated. It will not generate the physical interface file to attach it to the physical network. The system administrator will need to perform this step. For example, on Red Hat*, a file to link the **eth0** interface to the bridge would be */etc/sysconfig/network-scripts/ifcfg-eth0*:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

On SUSE* releases, the physical port name will also need to be added to the **BRIDGE_PORTS** entry in the bridge configuration file.

The rest of the arguments to the **Bridge** configuration parameter are the same as the internal bridge configuration with exception of **MTU** size. The default value is set to 1500 bytes to match default physical network settings. If attaching to a pre-existing external bridge configuration the system admin must ensure these settings match the setting in the system configuration file. For example, on Red Hat* if the */etc/sysconfig/network-scripts/ifcfg-br0* file contains the line “MTU=9000” then the MTU field must be set to 9000 to match.

If attaching to a bridge that is a DHCP configuration, then set the **IP** value to the string “dhcp”.

Like the internal method, the **Network** parameter connects a virtual network interface to the defined bridge.

If the **Network** parameter specifies a type of **StaticBridge**, the **micip** argument is required and, along with the **modhost**, and **modcard** parameters works the same as the **StaticBridge** configuration.

NOTE: Setting **--modhost=no** will remove any pre-existing */etc/hosts* entries that contain the comment "#Generated-by-micctrl".

If the **Network** parameter specifies a type of **Bridge** then the cards network files are created using DHCP to retrieve the correct IP address for the card. The **modhost** and **modcard** parameters are not required because it is assumed the IP address for each card will be retrievable from a name server on the net.

14.4.5.6 Host SSH Keys

The secure shell utilities recognize a Linux* system on the network by its “host key files”. These files are found in the */etc/ssh* directory. The host key values, like the MAC addresses, are considered to be highly persistent, and the **micctrl** command will retain their values if they exist.

In some clusters, detecting and protecting against “man in the middle” and other such attacks might not be required. In this case, the system administrator may use the **micctrl --hostkeys** command to set the host SSH keys to be the same, cluster wide.

14.4.5.7 Name Resolution Configuration

Name resolution on the card is set by creating the */etc/nsswitch* file and copying the */etc/resolv.conf* file from the host to the Intel® Xeon Phi™ coprocessor file system.

15 The micctrl Utility

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

- Card state control – boot, shutdown and reset control while the **mpssd** daemon is running.
- Configuration files initialization and propagation of values.
- Helper functions for modifying configuration parameters.
- Helper functions for modifying the root file system directory or associated download image.

The **micctrl** utility requires a first argument specifying the action to perform, followed by option-specific arguments. The arguments may be followed by a list of Intel® Xeon Phi™ coprocessor names, which is shown in the syntax statements as [mic card list]. If no cards are specified, the **mic.ko** driver must be loaded and the existing card list is probed. Otherwise, the card will be a list of the card names. For example, the list may be “micN micN”, if these are the cards to control.

NOTE: The mic.ko driver module must be loaded before using the micctrl utility.

15.1 Card State Control

Starting the **mpssd** daemon may initiate booting of the Intel® Xeon Phi™ coprocessors. On a system with multiple cards it would be intrusive to shut down and restart the daemon to reboot one of the cards. It would force all of them to be rebooted. Therefore, the **micctrl** utility provides mechanisms for individual card control.

Micctrl controls Intel® Xeon Phi™ coprocessor state and queries card state via the **sysfs** entry */sys/class/mic/<micname>/state*. The *micname* value is literally the name of the coprocessor and will be in the format *micN*, *micN*, etc.

Reading from the state will show the current run state of the corresponding Intel® Xeon Phi™ coprocessor. Writing to the *state* may cause the corresponding card to change states, and is restricted to the root user.

15.1.1 Waiting for Intel® Xeon Phi™ Coprocessor State Change

Command Syntax:

```
micctrl -w [mic card list]

micctrl --wait [mic card list]
```

The **wait** (or **w**) option waits for the status of the Intel® Xeon Phi™ coprocessor to be either “online” or “ready”. It also allows for a brief pause to the “ready” state during **mpssd** startup. It is intended for users to verify the **mpssd** daemon startup, shutdown, or reset procedure is complete. It has a built-in timeout value of 300 seconds.

15.1.2 Booting Intel® Xeon Phi™ Coprocessors

Command Syntax:

```
micctrl -b [-w] [mic card list]

micctrl --boot [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor(s) must be in the “ready” state. This command writes the string “boot:linux:<image>” (where image is the OSimage configuration parameter) to the */sys/class/mic/<micname>/state* sysfs file. The driver will inject the indicated Linux* image into the cards memory and start it booting.

15.1.3 Shutting Down Intel® Xeon Phi™ Coprocessors

Command Syntax:

```
micctrl -S [-w] [mic card list]

micctrl --shutdown [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor must be in the “online” state. This command writes the string “shutdown” to the */sys/class/mic/<micname>/state* sysfs file. The driver instructs the card to perform an orderly shutdown and wait for completion. It will then reset the card to place it again in the boot ready state.

15.1.4 Rebooting Intel® Xeon Phi™ Coprocessors

Command Syntax:

```
micctrl -R [-w] [mic card list]

micctrl --reboot [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor must be in the “online” state. This command sequentially performs the shutdown and boot functions described in Sections 15.1.2 and 15.1.3.

15.1.5 Resetting Intel® Xeon Phi™ Coprocessors

Command Syntax:

```
micctrl -r [-w] [mic card list]

micctrl --reset [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor can be in any state. This command writes the string "reset" to the */sys/class/mic/<micname>/state* sysfs file. The driver will perform a soft reset on the card by setting the correct card PCI mapped register.

NOTE: Performing a reset may result in the loss of file data that has not been flushed to a remote file. It is therefore recommended to perform a shutdown where possible instead of a reset.

15.1.6 Intel® Xeon Phi™ Coprocessor Status

Command Syntax:

```
micctrl -s [mic card list]
micctrl --status [mic card list]
```

The **status** option displays the status of the Intel® Xeon Phi™ coprocessors in the system. It the status is "online" or "booting" it also displays the name of the associated boot image.

15.2 Configuration Initialization and Propagation

This section discusses the micctrl command options for initializing configuration files, and propagating, resetting, and cleaning configuration parameters.

15.2.1 Initializing the Configuration Files

Command Syntax:

```
micctrl --initdefaults \
[--users=<none | overlay | merge | nochange] \
[--pass=<none | shadow>] [--nocreate] \
[--modhost=<yes | no>] [--modcard=<yes | no>] \
[mic card list]
```

The Intel MPSS installation does not provide configuration files described earlier in Configuration. Instead, these files are created by the **micctrl --initdefaults** command. **micctrl --initdefaults** can be run anytime but will not change files if they already exist and have valid information.

The **--initdefaults** option first checks to see if the */etc/mpss/default.conf* file is present. If not, it creates the default version of it. Then, for each supplied card, it checks for the existence of the card-specific configuration file */etc/mpss/<micname>.conf*. If it is not present, it creates a default version with an **Include** parameter including the **default.conf** file.

The **--initdefaults** option then proceeds to parse the per card configuration files. For each parameter that is not set, it will add a default value to the per card configuration file. At the same time it will check for deprecated parameters and transform them to the updated parameters. For example: the deprecated **FileSystem** parameter is updated to **RootDevice RamFS**.

The **--initdefaults** option also added the following options in the 3.2 release:

With the elimination of the **UserAuthentication** parameter initdefaults needs some guidance on users to add to the */etc/passwd* file. If no --users argument is specified then it defaults to the **nochange.**

If **--users** is set to **nochange** and if no /etc/passwd exists then it defaults to the functionality of setting **overlay**. Otherwise no changes to the */etc/passwd* file will occur.

If **--users** is set to **none** then only the minimal set of users including root, sshd, micuser, nobody and nfsnobody.

If **--users** is set to **overlay** then the information in */etc/passwd* and */etc/shadow* will be replace with the minimal users plus all users in the host's */etc/passwd* file.

If **--users** is set to **merge** then the host's */etc/passwd* file will be checked for any users not in the cards file and added in.

The **--pass** argument allows the system administrator to decide whether the pass word for the user from the host will be copied to the card or not. If it is set to none then the pass word field in the */etc/shadow* file will be filed with a '*'. If it is set to **shadow** then the hosts pass word information for the user on the host will be used. It should be noted this does not apply to SUSE* hosts using the default Blowfish encryption since the card will not understand it.

The **--nocreate** argument specified to not create the associated home directory for the user. This is intended to be used by systems where the users home directory will be remote mounted.

Setting the **--modhost** parameter indicates to micctrl whether to add an entry for the Intel® Xeon Phi™ coprocessor IP address to the host's */etc/hosts* file. If **modhost=yes**, then micctrl will modify the */etc/hosts* file; if **modhost=no**, then the system administrator should add the entry to the */etc/hosts* file.

The **--modcard** parameter determines whether the system administrator wants the configuration process to create the */etc/hosts* file on the card. if **modcard=no**, then the system administrator should create the */etc/hosts* file on the card.

Consult the documentation for setting network functons for the results of using the **--modhost** and **--modcard** options.

15.2.2 Propagating Changed Configuration Parameters

Command Syntax:

```
   micctrl --resetconfig \
[--users=<none | overlay | merge | nochange] \
[--pass=<none | shadow>] [--nocreate] \
[--modhost=<yes | no>] [--modcard=<yes | no>] \
[mic card list]
```

Changes to the configuration files are propagated with the **micctrl --resetconfig** command. The **--resetconfig** option first removes the files in **MicDir** created by the configuration process, with the exception of the highly persistent ssh host key files. It then regenerates those files according to the parameters in the */etc/mpss/<micname>.conf and /etc/mpss/default.conf* files. This process will not add default parameters, but only causes the changed parameters to be propagated.

The **--resetconfig** option added several new options with the 3.2 release. Consult the previous documentation for the **--initdefaults** option.

NOTE: This command is deprecated, and may be removed in future releases.

15.2.3 Resetting Configuration Parameters

Command Syntax:

```
micctrl --resetdefaults \
[--users=<none | overlay | merge | nochange] \
[--pass=<none | shadow>] [--nocreate] \
[--modhost=<yes | no>] [--modcard=<yes | no>] \
[mic card list]
```

In the event of a failed or problematic configuration process, the best remedy may be to start again. The **micctrl --resetdefaults** command deletes the configuration files and executes the same process as the **--initdefaults** option.

Since **--resetdefaults** only affects the files known to the configuration, it does not delete any files the system administrator has added to a card's file system.

The **--resetdefaults** option has a number of new options with the 3.2 release. Consult the previous documentation for the **--initdefaults** option.

15.2.4 Cleaning Configuration Parameters

Command Syntax:

```
micctrl --cleanconfig [mic card list]
```

Since Intel MPSS configuration commands will replace configuration parameters that have been deprecated, the **micctrl --resetdefaults** and **micctrl --resetconfig** commands may not restore the deprecated commands of some previous version of Intel MPSS. Recalling the earlier example in Section 15.2.1, "Initializing the Configuration Files", the **micctrl --resetdefaults** and **micctrl --resetconfig** commands will not automatically revert "RootDevice RamFS" back to the "FileSystem" parameter. If this is the desired goal, then removing the whole card configuration may be required.

The **--cleanconfig** option not only removes a card's configuration files, but also removes all files in the **MicDir** parameter directory along with the other values specified by **RootDevice**.

15.3 Helper Functions for Configuration Parameters

15.3.1 Change the Host Networking Configuration Daemon

Warning: Configuring the virtual network interfaces for the Intel® Xeon Phi™ coprocessors has displayed a number of irregularities in both the Red Hat* and SUSE* implementations of "NetworkManager", due to assumptions they make about non-hardwired network interfaces. It is now required to use the older and more server-oriented "network" daemon instead. To switch to network daemon, perform the following on the host:

For RedHat6.x and SuSE:

```
[host]# service NetworkManager stop
[host]# chkconfig NetworkManager off
[host]# chkconfig network on
```

For RedHat Linux 7.x:

```
[host]# systemctl stop NetworkManager
[host]# systemctl disable NetworkManager
[host]# systemctl enable network
```

15.3.1.1 MAC Address Assignment

Command Syntax:

```
micctrl --mac=serial
micctrl --mac=random
micctrl --mac=<MAC address>
```

The **--mac** option allows the system administrator to set the value of the **MacAddrs** parameters. If the Intel® Xeon Phi™ coprocessor has a valid serial number then using the --mac=serial option will tell the configuration to use the MAC address created from the card's serial number.

Using the **--mac=random** option will configure the interface to use whatever address the driver assigns.

The system administrator may set any valid MAC address in the format XX:XX:XX:XX:XX:XX. This address will be assigned to the Intel® Xeon Phi™ coprocessor end of virtual network interface and this number with the last segment incremented by one to the host end of the virtual network.

15.3.1.2 Resetting to Default

Command Syntax:

```
micctrl --network=default [mic card list]
```

The **--network** option set to **default** restores the network setting for the list of coprocessors to the value originally created by the **micctrl --initdefaults** command.

15.3.1.3 Static Pair

Command Syntax:

```
micctrl --network=static [--ip=<mic_IP>]\
[--mtu=<mic_mtu>]\

[--modhost=<yes|no>] [--modcard=<yes|no>] [mic card list]
```

NOTE: Throughout the rest of this section unless otherwise specified - **IP** in the Command Syntax applies to the mic card and **mtu** applies to the network interface for the card.

The **--network** option set to **static** without specifying a bridge name provides an easy method for changing the **Network** configuration parameter to a static pair type. If no IP address is included, then the network interfaces for the listed cards is set to the default values provided by the original **micctrl --initdefaults** command.

If the **IP** address specified is the first two quads, then **micctrl** will finish the address by setting the third quad of each address to the card's ID + 1, and the fourth quad to 1 on the card and 254 on the host. For example, on a two card system specifying an **IP** value of "10.10" will create micN values of "10.10.1.1" and "10.10.1.254" and micN values of "10.10.2.1" and 10.10.2.254".

It is possible to explicitly set the values assigned to the IP addresses. The **IP** argument must be set with the format **cardIP,hostIP:cardIP,hostIP….** Each **cardIP - hostIP** pair specifies a card's values. For example, if there are two cards in the system and the entry is "10.10.10.1,10.10.10.2:10.10.11.1,10.10.11.2" then micN will be assigned the card IP address of 10.10.10.1 and host of 10.10.10.2. MicN will be assigned 10.10.11.1 and 10.10.11.2.

The default size of MTU is set to max IPV4 size of 64k. This value can be changed by specifying the **MTU** value. This should only be used for performance testing since tests have shown the default value to have the best performance.

The **modhost** parameter determines if the system administrator wants to modify the host */etc/hosts* file or not. The **modcard** determines if the system administrator wants the configuration process to create the */etc/hosts* on the card. Using either will override the use of the old deprecated **hosts** parameter. The **hosts** parameter may still be used and setting it will be treated as setting **modhost** and **modcard** with the specified parameter.

15.3.1.4 Internal Bridging

Command Syntax:

```
micctrl --addbridge=<brname> --type=internal --ip=<bridge_ip>\
[--netbits=<bits>] [--mtu=<MTU>]
```

```
micctrl --network=static [--bridge=<bridge_name>] \
[--ip=<mic_IP>] [--mtu=<mic_mtu>][--modhost=<yes|no>]\
[--modcard=<yes|no>] [mic card list]
```

The internal bridging network topology connects the Intel® Xeon Phi™ coprocessors in a system together with one IP address for the host. This is accomplished by first creating the bridge interface on the host and then connecting the virtual network interfaces to it.

Micctrl creates bridge interfaces with the **--addbridge** option. The **name** of the bridge must be specified and generally on Linux* is **br0** or **br1** . Setting the **type** to **internal** will cause **micctrl** to always create the correct network configuration files for the host.

Like static pair, you may specify the netbits value and the mtu value but it has no real effect. Each of the virtual Ethernet interfaces added to the bridge will inherit these values from the bridge specification.

Virtual network interfaces are added to the bridge with the **micctrl --network** command. The **bridge** argument must be present and the **IP** argument must specify IP addresses with the first 3 quads matching those of the bridge IP address.

The IP address must be a fully qualified dot notated address. If more than one card is specified, each card will get the same IP address with the cards number added to the fourth quad. For example, if the address 10.10.10.12 is specified, the micN will receive 10.10.10.12 and micN 10.10.10.13.

The **modhost** parameter determines if the system administrator wants to modify the host */etc/hosts* file or not. The **modcard** determines if the system administrator wants the configuration process to create the */etc/hosts* on the card. Using either will override the use of the old deprecated **hosts** parameter. The **hosts** parameter may still be used and setting it will be treated as setting **modhost** and **modcard** with the specified parameter.

15.3.1.5 External Bridging

Command Syntax:

```
micctrl --addbridge=<brname> --type=external --ip=<bridge_IP> \
[--netbits=<bits>] [--mtu=<MTU>]
```

```
micctrl --network=static [--ip=<mic_IP>] [--bridge=<brname>] \
[--mtu=<mic_mtu>] [--modhost=<yes|no>] [--modcard=<yes|no>] [mic card list]
```

```
micctrl --network=dhcp --bridge=<brname> [--modhost=<yes|no>]\
[--modcard=<yes|no>] [mic card list]
```

The **external** bridge type definition differs from **internal** bridge only in that **micctrl** does not create the network configuration files for the host. It assumes that the bridge is already connected to a physical Ethernet device. It is a task of the system administrator to modify the configuration file for the physical Ethernet port to attach to the bridge.

External bridges may also be declared as **dhcp** instead of **static**. During boot, the Intel® Xeon Phi™ coprocessor Linux* OS will attempt to retrieve an IP address from a DHCP server. See *micctrl --help* for more details.

15.3.1.6 Modifying Existing Network Definitions

```
Command Syntax:

micctrl --modbridge=<brname> --ip=<IP> \
[--netbits=<bits>] [--mtu=<MTU>]

   micctrl --delbridge=<brname>

micctrl --network=static [--ip=<mic_IP>] \
[--mtu=<mic_mtu>] [--modhost=<yes|no>] \
[--modcard=<yes|no>] [mic card list]

   //bridge_ip for all
```

If changes to any of the bridges are required, the **modbridge** option to **micctrl** will allow the change of any combination of **ip, netbits** or **MTU.** In addition any changes to **MTU** or **netbits** will be propagated to any of the virtual network configuration files.

If a bridge is no longer needed, the **delbridge** option will remove it from the Intel® Xeon Phi™ coprocessor configuration. If the bridge is not external, it will also remove the corresponding host network configuration file.

NOTE: Intel® Xeon™ coprocessors must be detached before a bridge can be deleted. You can use **micctrl --network=default** to remove coprocessors.

The **micctrl --network** command may also be used to change the node bridge parameters for a set of interfaces. Specifying **--network** without a particular option will cause **micctrl** to attempt to change the particular parameter.

15.3.1.7 Extra Networking Notes

On SUSE*, upon completion of all network change commands, run [1]**service networking restart**. **Micctrl** does not yet understand how to flush the name server cache.

15.3.2 Change the UserAuthentication Configuration Parameter

Command Syntax:

```
micctrl --configuser=none [-ids] [mic card list]

micctrl --configuser=local [--low=<low uid>] \
[--high=<high uid] [-ids] [mic card list]
```

This command was removed. Refer to the section on the **micctrl --userupdate** command for its functional replacement.

15.3.3 Initializing The Intel® Xeon Phi™ Coprocessor Password File

Command Syntax:

```
micctrl --userupdate=<mode> [--pass=<none | shadow>] \
[--nocreate] [mic card list]
```

It is necessary to control user access to the Intel® Xeon Phi™ coprocessor(s). The **micctrl --userupdate** command allows the system administrator to specify when to reset or add entries to the passwd file.

Using **--userupdate** requires the mode of use to be set and may be **none, overlay, merge** or **nochange**. The **nochange** option will have no effect in most cases.

If **mode** is set to **nochange** and if no /etc/passwd exists then it defaults to the functionality of setting **overlay**. Otherwise no changes to the */etc/passwd* file will occur.

If **mode** is set to **none** then only the minimal set of users including root, sshd, micuser, nobody and nfsnobody.

If **mode** is set to **overlay** then the information in */etc/passwd* and */etc/shadow* will be replace with the minimal users plus all users in the host's */etc/passwd* file.

If **mode** is set to **merge** then the host's */etc/passwd* file will be checked for any users not in the cards file and added in.

The **--pass** argument allows the system administrator to decide whether the pass word for the user from the host will be copied to the card or not. If it is set to none then the pass word field in the */etc/shadow* file will be filed with a '*'. If it is set to **shadow** then the hosts pass word information for the user on the host will be used. It should be noted this does not apply to SUSE* hosts using the default Blowfish encryption since the card will not understand it.

The **--nocreate** argument specified to not create the associated home directory for the user. This is intended to be used by systems where the users home directory will be remote mounted.

15.3.4 Adding Users to the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
    micctrl --useradd=<name> --uid=<uid> --gid=<gid> \
[--home=<dir>] [--comment=<string>] [--app=<exec>] \
[--sshkeys=<keydir>] [--nocreate] \
[--non-unique] [mic card list]
```

The **--useradd** option adds the specified user name to the */etc/passwd* and */etc/shadow* files on the Intel® Xeon Phi™ coprocessor file system. The system administrator must specify the correct user and group IDs for the user that is being added.

NOTE: The behavior of **micctrl --useradd** differs from Linux* **useradd**. It does not check for conflicting uid or gid for other users on the ldap server. System administrators should not expect identical behavior between **micctrl --useradd** and Linux* **useradd**.

The **--home** argument specifies the user's home directory in the card file system, and will cause the directory to be created. The default home directory for user <name> is */home/<name>.*

The **--comment** argument specifies a comment string to be added to the comment field in */etc/passwd*. The default comment string is the user names.

The **--app** argument specifies the default application executed by the user. The default app is */bin/sh*.

The **--sshkeys** argument specifies the host directory in which the user's secure shell key files are to be found. The default is */home/<name>/.ssh*.

The **--nocreate** option specifies to not create the home directory for the user and is intended to be used when the users home directories are remote mounted.

The **--non-unique** option will allow the user to be added to the card's */etc/passwd* and */etc/shadow* files with the specified **uid** even if a user with that uid already exists.

In addition, a default **.profile** file will be added for the user.

The latest implementation will also add the user to the currently running coprocessor if it is in the **online** state.

15.3.5 Removing Users from the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --userdel=<name> [--remove] [mic card list]
```

The **--userdel** option removes the specified user from the card's */etc/passwd* and */etc/shadow* files. It also removes the directory stored in the **home** field of the */etc/passwd* file.

The current implementation also removes the user from the */etc/passd* and */etc/shadow* files on the currently running coprocessor if it is in the **online** state. By default, it does not remove the user's home directory and is intended to prevent the removal of a user's remote mounted home directory. System administrators can force the removal of the user's home directory by including the **--remove** option.

15.3.6 Changing the Password for Users on the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --passwd=<name> [--pass=<newpw>] [mic card list]
```

The **--passwd** option allows the system administrator to change the password for a user on the coprocessors, or for users to change their own passwords. The system administrator may also pass a new password on the command line with the **--pass** command line option. If the **--pass** option is not used or a non superuser calls this option then they will be prompted for the current passwd in the file before a change will be allowed.

In the current implementation the new passwd will also set on the running coprocessor if its state is set to **online**.

15.3.7 Updating a Users SSH Keys on the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --sshkeys=<name> [--dir=<dir>] [mic card list]
```

The **--sshkeys** option allows a user's SSH keys to be updated on the coprocessor file system. The directory is the location of all the keys required. This command copies the key files with correct owner and permissions to the cards file system. It also creates entries in the **authorized_keys** file for any file(s) with a "**.pub**" extension if it does not already exist.

15.3.8 Adding Groups to the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --groupadd=<name> --gid=<gid> [mic card list]
```

The **--groupadd** option adds the specified group name and ID to the card's */etc/group* file. In the current implementation the group will also be added to the running coprocessor's */etc/group* file if it is in the **online** state.

15.3.9 Removing Groups from the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --groupdel=<name> [mic card list]
```

The **--groupdel** option removes the specified group name entry from the card's */etc/group* file. In the current implementation the group will also be removed from the running coprocessor's */etc/group* file if it is in the **online** state.

15.3.10 Configuring LDAP on the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --ldap=<server> --base=<domain> [mic card list]
```

The **--ldap** option configures the coprocessor to use **LDAP** for user authentication. The **server** argument species the LDAP authentication server and the **base** argument specifies the domain to use.

If the **server** argument is set to **remove** then the use of **LDAP** on the card will be disabled.

Before this command can be used, **micctrl** must know where to find the required LDAP_PAM and LDAP_NSS RPM files. These optional files must be available on the host system and the directory where they exist must be configured with the **micctrl --rpmdir** command.

15.3.11 Configuring NIS on the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --nis=<server> --domain=<domain> [mic card list]
```

The **--nis** option configures the coprocessor to use **NIS** for user authentication. The **server** argument species the NIS/YP server and the **domain** argument specifies the domain to use.

If the **server** argument is set to **remove** then the use of **NIS** on the card will be disabled.

Before this command can be used, **micctrl** must know where to find the required NIS_YPTOOLS, NIS_YPBIND, NIS_NSS and NIS_RPC RPM files. These optional files must be available on the host system and the directory where they exist must be configured with the **micctrl --rpmdir** command.

15.3.12 Setting the Root Device

The **--rootdev** option changes the configured **RootDevice** parameter. It defines whether the Intel® Xeon Phi™ coprocessor file root system will be mounted from the initial ram disk, a downloaded ram file system, or an NFS export.

15.3.12.1 Ram Root File System

Command Syntax:

```
micctrl --rootdev=RamFS --target=<location> \
[mic card list]
micctrl --rootdev=StaticRamFS --target=<location> \
[mic card list]
```

Specifying a rootdev type of either **RamsFS** or **StaticRamFS** instructs the booting card to mount its root file system by creating an image file specified by the **target** argument. Target is the name of the compressed cpio image to be used for the Intel® Xeon Phi™ coprocessor file system. If it is not specified the default value of */var/mpss/micN.image* will be used. The **init** program will create a ram disk, load the files from **target** into it and do a switch_root to the new file system.

The difference between **RamFs** and **StaticRamFS** is **RamFS** will build **target** each and every time download is requested from the **Base**, **CommonDir**, **MicDir**, and any **Overlay** parameters. The static definition will use **target** as it exists and error if it does not exist.

15.3.12.2 NFS Root File System

Command Syntax:

```
micctrl --rootdev=NFS --target=<location> -d -c \
[mic card list]
micctrl --rootdev=SplitNFS --target=<location> \
--usr=<usr location> -d -c [mic card list]
```

Specifying the **NFS** or **SplitNFS** options instructs the booting card to mount its root file system from the NFS export specified by **target**. The **target** parameter will usually be specified in the traditional nfs mount systems of *<server>:<export>*. **micctrl** will use the IP address of the host as the server name. If target is not specified, then the default value of */var/mpss/micN.export* will be used. The NFS export should include **rw** and **no_root_squash** in its definition. If the **-d** option is included, the old root device **target** value will be deleted. If **-c** is included, **micctrl** will create it.

SplitNFS differs from **NFS** in that it also creates the correct files for a shared NFS export for the */usr* directory specified by **usr location**. If it is not specified, then it will default to */var/mpss/usr.export*. The system administrator must also add this to the exports file.

15.3.13 Adding an NFS Mount

Command Syntax:

```
micctrl --addnfs=<NFS export> --dir=<mount dir> \
[--options=<option>[,option,…]] [mic card list]
```

Add the **NFS export** to the MIC card's */etc/fstab*.

NFS export specifies the server and NFS export in the traditional <server>:<export> format.

The **--options** or **-o** argument allows the adding of the standard NFS mount options. Check NFS documentation for more information. The string supplied is directly placed into the options field in the card's */etc/fstab* file.

Additional configuration for SUSE* based host systems: If NFS file system mounts are added and the **chckconfig** utility is used to indicate starting the Intel MPSS stack at host boot time, SUSE* does not ensure the NFS server starts before the coprocessors boot. To ensure the NFS server is available, edit the */etc/init.d/mpss* file and change the "# Required-Start:" line to read "# Required-Start: nfsserver".

15.3.14 Removing an NFS Mount

Command Syntax:

```
micctrl --remnfs=<mount dir> [mic card list]
```

The **--remnfs** option searches the */etc/fstab* for the Intel® Xeon Phi™ coprocessor for the specified mount point and removes the mount point from the file.

15.3.15 Specifying the Host Secure Shell Keys

Command Syntax:

```
micctrl --hostkeys=<keydir> [mic card list]
```

The **--hostkeys** option removes the host keys copied by the **--initdefaults** command and replaces it with the files from the specified directory. Since these files are considered to be highly persistent they should stay resident unless the **--resetdefaults** or **--cleanconfig** option is performed.

15.3.16 Setting Startup Script Parameters

Command Syntax:

```
micctrl --service
micctrl --service=<name> --state=<on|off> \
[--start=<num>I [--stop=<num]] [mic card list]
```

The Intel® Xeon Phi™ coprocessor Linux* OS, like any Linux* OS, executes a series of scripts on boot, which are located in /etc/init.d. To determine which of the installed scripts are executed on any boot, links to these scripts are created in runlevel directory. The card's OS runs at level 5, defining the runlevel directory to be */etc/rc5.d*.

On most Linux* systems, the service scripts to be executed are enabled or disabled using the **chkconfig** command. On the Intel MPSS stack this is performed by the **micctrl --service** command.

The **state** option must be set to **on** or **off** and determines whether the script will execute on boot. Services already included in the configuration may have their state changed without specifying new **start** or **stop** values.

The **start** and **stop** parameters must be between 1 and 100, and determine the order in which the services are executed. If **stop** is not specified, then it will be set to **100 – start**.

Add on software containing a service script will include the **Service** parameter associated with it. Modifying the default value included in its own configuration file will cause an overriding entry to be set in the **micN.conf** file.

Micctrl --service may be called with no arguments and will display a list of current service settings. Currently, no services are configured by default.

15.3.17 Overlaying File System with More Files

Command Syntax:

```
micctrl --overlay
micctrl --overlay=<type> --state=<on|off|delete> \
--source=<dir>I --target=<target> [mic card list]
```

The **Overlay** parameter specifies a block of files to be included in the Intel® Xeon Phi™ coprocessor file system. The **type** parameter specifies different methods of including files known with the values **filelist**, **simple**, **file**, and **rpm**.

NOTE: Do not add overlays to the /tmp directory on the card, as it gets cleared each time the card boots.

If the type is set to **filelist,** then files from the **source** directory will be placed on the card's file system, based on entries in a file specified by the **target** parameter. The **filelist** format allows for copying random, or disassociated files to the card. It also provides a mechanism for creating special file types and setting absolute privileges. For more information on its format, check Section 16.1.

If the type is set to **simple** then the files from the **source** directory are copied directly to the card's file system under the **target** directory.

Setting the **type** argument to **File** specifies to copy the single file **source** to **target** in the card's file system.

If the type is set to **rpm** then the **target** parameter specifies either a single RPM or a directory of RPMs to be installed on the coprocessor file system when an individual coprocessor boots.

The **state** argument determines if the files are to be copied or not. Additional software will be added by creating a configuration file in the */etc/mpss/conf.d* directory with an **Overlay** parameter included. Because **micctrl** cannot modify these files, changing the default value of **state** will be done by duplicate entries in the **micN.conf** files. If **state** is on, the files will be copied to the coprocessor's file system; if **state** is off, the files will not be copied. Calling **micctrl --overlay** with state set to **delete** will cause the entry to be removed from the **micN.conf** file.

Micctrl --overlay may be called with no arguments and will display the current overlay status.

15.3.18 Base Files Location

Command Syntax:

```
micctrl --base
micctrl --base=<default|cpio|dir> \
[--new=<location>] [mic card list]
```

The **base** argument modifies the **Base** parameter in the configuration files. By default, micctrl --initdefaults points this parameter to a CPIO initial ram disk image installed by the Intel® Xeon Phi™ coprocessor software. The **Base** parameter is normally in the **default.conf** file. Changing it, by specifying the **--base** argument, will add an entry to the **micN.conf** files and override the default value.

If **base** is set to **default** then the value is reset to the default location of the initial ram disk CPIO image file.

If **base** is set to CPIO the **Base** parameter will be set to the new location. If the file does not exist then the last value for **Base** will be copied to the new location.

If **base** is set to **dir** the new location must be a directory to contain the files previously contained in the initial ram disk CPIO image. If the new directory does not exist it will be created and the previous setting for the will be either copied there or expanded if it was a CPIO file.

If **--base** is not specified then a list of the cards and their **Base**, **CommonDir**, and **MicDir** parameters will be displayed.

15.3.19 Common Files Location

Command Syntax:

```
micctrl --commondir
micctrl --commondir=<commondir> [mic card list]
```

The **commondir** argument modifies the **CommonDir** parameter in the configuration files. This parameter points to a directory of files to be downloaded to all Intel® Xeon Phi™ coprocessors in a host system. The **micctrl --initdefaults** option creates an empty common directory if it does not exist. The **CommonDir** parameter is normally in the **default.conf** file. Changing it, by specifying the **commondir** argument, will add an entry to the **micN.conf** files and override the default value.

If the **commondir** directory does not exist then the location of the old **CommonDir** parameter will be copied to the new location. After the files have been copied, the configurations for all known cards in the system are checked for references to the old **CommonDir** values and if no references exist the files will be deleted.

If **commondir** is not specified then a list of the cards and their **Base, CommonDir** and **MicDir** parameters will be displayed.

15.3.20 Coprocessor Unique Files Location

Command Syntax:

```
micctrl --micdir
micctrl --micdir=<micdir> [mic card list]
```

The **micdir** argument modifies the **MicDir** parameter in the configuration files. This parameter points to a directory of files to be downloaded to all Intel® Xeon Phi™ coprocessors in a host system, containing files unique to each card instance. The **micctrl --initdefaults** option creates many default files here. Other **micctrl** commands modify many of these files.

If the directory specified by --micdir= does not already exist, the files in the previous location specified by the MicDir parameter in the micX.conf will be copied to the new location, and the old location will be deleted.

If **micdir** is not specified then a list of the cards and their **Base, CommonDir** and **MicDir** parameters will be displayed.

15.3.21 Location of Additional RPMs for the Intel® Xeon Phi™ Coprocessor File System

Command Syntax:

```
micctrl --rpmdir=<location> [mic card list]
```

The **rpmdir** argument specifies the location for other **micctrl** commands to find required coprocessor RPM files. This functionality has been implemented specifically to support the **--ldap** and **--nis** options. It may change in future releases.

15.3.22 Coprocessor Linux* Image Location

Command Syntax:

```
micctrl --osimage
micctrl --osimage=<osimage> [mic card list]
```

The **osimage** option sets the location of the **OSimage** parameter. The **OSimage** defines the Linux* operating system image to boot. The default value of this location is */usr/share/mpss/boot/bzImage-knightscorner* and is installed by the Intel MPSS software.

15.3.23 Boot On Intel MPSS Service Start

Command Syntax:

```
micctrl --autoboot
micctrl --autoboot=<yes|no> [mic card list]
```

The **autoboot** argument sets the **BootOnStart** configuration parameter. This parameter controls whether or not a coprocessor boots when the Intel MPSS service is started. A **yes** value indicates to boot the card. If no value is specified then a list of the cards and the current values of **BootOnStart** will be displayed.

15.3.24 Power Management Configuration

Command Syntax:

```
micctrl --pm

micctrl --pm=<default|off|set> \

[--corec6=<on|off>] [--pc3=<on|off>] \

[--pc6=<on|off>] [--cpufreq=<on|off>] \

[mic card list]
```

Setting the **pm** argument to **default** resets the values of power management for the list of cards to the default values. In the current release, all power management parameters other than **cpufreq** are set to off. This will change in a future release.

Setting the **pm** argument to **off** will also set all parameters to off other than **cpufreq**.

Setting the **pm** argument to **set** enables the use of the optional parameters **corec6**, **pc3**, **pc6**, and **cpufreq**. Each optional parameter can be individually enabled or disabled by setting the **on** or **off** values.

15.3.25 Cgroups Configuration

Command Syntax:

```
micctrl --cgroup --memory=<enable|disable> \
[mic card list]
```

The **cgroup** argument currently only supports enabling or disabling the memory cgroup parameter. The default value is set to **disable** by the **micctrl --initdefaults** command.

15.3.26 Syslog Configuration

Command Syntax:

```
micctrl --syslog=<buffer|file|remote> \
[--host=<targethost[:port]>] [--logfile=<location>] \
[--loglevel=<loglevel] [mic card list]
```

The **syslog** argument modifies the */etc/syslog-startup.conf* file for the list of coprocessors.

If **syslog** is set to **buffer,** it sets the syslog daemon to not take messages from the coprocessor's dmesg buffer. This effectively disables logging.

If **syslog** is set to **file**, it set the syslog daemon to log to the optional **location** in the coprocessor's filesystem or to */var/log/messages* by default.

If **syslog** is set to **remote**, the syslog daemon is instructed to log to a the remote specifiled by the optional **host** argument. When **remote** is used the **targethost** is required but the **port** is optional and will be set to default 514 if not specified.

Changes to the logfile location take effect immediately on the coprocessor if it is running and its state is currently **online**.

15.4 Other File System Helper Functions

15.4.1 Updating the Compressed CPIO Image

Command Syntax:

```
micctrl --updateramfs [mic card list]
```

The **StaticRamFS** root file system image is only changed when the system administrator requests it. In many cluster systems this image will be built externally and put in place.

The **--updateramfs** option updates the image from the same parameters used by the **RamFS** specification. The new image will be used the next time the card boots.

15.4.2 Updating the NFS Root Export

Command Syntax:

```
micctrl --updatenfs [mic card list]
micctrl --updateusr [mic card list]
```

When new software is added through updating the Intel MPSS stack or add-on products, those changes must be propagated, or installed into the NFS root file system exports. The **micctrl --updatenfs** command performs this process.

If the root device type has been set to **SplitNFS** it may also change files located in the common /usr directory. The **--updateusr** command line option will complete this process.

16 *Adding Software*

Typical installations are not static, and usually require the system administrator to add additional files or directories to the Intel® Xeon Phi™ root file system that is downloaded to the card.

16.1 The File System Creation Process

As previously described in Section 14.3.9, the **RootDevice** parameter defines the type of root device to mount. When the **type** argument to **RootDevice** is **RamFS** or **StaticRamFS**, a file system image is pushed to the card as its ram file system root file system. In this section we describe the process of building such a root file system.

The process is driven by the configuration parameters **Base, CommonDir, MicDir** and **Overlay**. Files in these directories must have the correct owners and permissions for targeted coprocessor file system. The **directories** are processed in the following order: **Base, CommonDir, MicDir** and **Overlay**.

16.2 Creating the Download Image File

The download image file for the **RamFS** root device type is created by processing the configuration directives **Base**, **CommonDir**, **MicDir**, and any **Overlays**, in that order. As the configuration directives are processed, a tree of filenames and their information is built.

When the tree is completely processed, the **mpssd** daemon or **micctrl --updateramfs** will create a cpio entry for the file and append it to the filename specified by the **RootDevice** directive. When processing is complete it then compresses the file.

16.3 Adding Files to the Root File System

Adding a file to the root file system can be done in two ways. The system administrator can add an entry to some existing **Base**, **CommonDir**, **MicDir**, or **Overlay** by simply copying to its location. Alternatively, the system administrator can add a new **Overlay** configuration parameter with **location** and **descriptor file** arguments that describe the files to be added.

16.3.1 Adding an Overlay

To add an **Overlay** set, the file must be placed in the correct location specified under **Overlay** and added to the **filelist** file specified. The power of the **Overlay** is that it may be called from a new configuration file in */etc/mpss/conf.d*. Refer to the next section for an example.

16.3.2 Example: Adding a New Global File Set

As an example, adding the /usr/bin/myutil to the Intel® Xeon Phi™ coprocessor file system as an overlay would be done by adding the */etc/mpss/conf.d/mytuil.conf* file and with an **Overlay** parameter. This would contain:

```
Overlay Filelist /var/mpss/myutil /var/mpss/myutil/myutil.filelist on
```

The **location** argument to **Overlay** is */var/mpss/myutil*, the directory under which the new binary **myutil** is placed on the host. The **descriptor file** argument, */var/mpss/myutil/myutil.filelist*, identifies a **filelist** which describes the files to include **myutil** on the filesystem. This file will contain:

```
dir /usr 755 0 0
dir /usr/bin 755 0 0
file /usr/bin/myutil myutil 755 0 0
```

17 Linux* SYSFS Entries

The driver supplies configuration and control information to host software through the Linux* Sysfs file system. The driver presents two sets of information:

- Driver global information is presented in the */sys/class/mic/ctrl* directory.
- Information unique to an Intel® Xeon Phi™ coprocessor instance is presented in the */sys/class/mic/micN* directories, where N is an integer number (0, 1, 2, 3, etc.) that identifies the card instance.

17.1 The Global Mic.ko Driver SYSFS Entries

17.1.1 Revision Information

Sysfs Entries:

/sys/class/mic/ctrl/version

This entry is read-only. The **version** sysfs entry displays a string containing the ID of the build producing the current installed software.

17.1.2 Other Global SYSFS Entries

Sysfs Entries:

/sys/class/mic/ctrl/peer2peer

/sys/class/mic/ctrl/vnet

The **peer2peer** entry contains the state of the Intel® Symmetric Communications Interface (SCIF) peer-to-peer transfer code. It will be either set to **enable** or **disable.**

The vnet entry will contain the number of active links to the virtual Ethernet.

17.2 The Intel® Xeon Phi™ Mic.ko Driver SYSFS Entries

Hardware Information

Sysfs Entries:

/sys/class/mic/micN/family

/sys/class/mic/micN/sku

/sys/class/mic/micN/stepping

/sys/class/mic/micN/active_cores

/sys/class/mic/micN/memsize

These sysfs entries are all read-only.

The **family** sysfs node reports the family the Intel® Xeon Phi™ coprocessor belongs to. At this time the family should always report the string “x100”.

The **sku** sysfs node returns a string defining the device type. As an example it may report the string “C0-3120/3120A”.

The **stepping** node returns the processor stepping. Typically it will be **B0, B1, C0,** etc.

The **active_cores** node reports the actual number of working cores on the card.

The **memsize** node returns the size of memory on the Intel® Xeon Phi™ coprocessor.

17.2.1 State SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/state

/sys/class/mic/micN/mode

/sys/class/mic/micN/image

/sys/class/mic/micN/cmdline

/sys/class/mic/micN/kernel_cmdline

The **state** and **cmdline** sysfs nodes are read and write. The others are read-only.

The **state** sysfs node will show one of the following values:

• **ready**	card is ready for a boot command
• **booting**	card is currently booting
• **no response**	card is not responding
• **boot failed**	card failed to boot
• **online**	card is currently booted
• **shutdown**	card is currently shutting down

• **lost**	booted card is not responding
• **resetting**	card is processing soft reset
• **reset failed**	card cannot be reset – non recoverable

Additionally, if the state is **booting, online** or **shutdown**, the state is modified by the information from the **mode** and **image** sysfs nodes. The **mode** will be either **linux** or **elf.** The image file will contain the name of the file used to boot into the associated mode.

Writing to the **state** sysfs node requests the driver to initiate a change in state. The allowable requests are to boot, reset or shutdown the Intel® Xeon Phi™ coprocessor.

To boot a card, the string to write has the format "boot:linux:<image name>". The **mpssd** daemon uses its **OSimage** parameter to fill in the image name. For example the default Linux* image for the Intel®Xeon Phi™ coprocessor will create the string "boot:linux: /usr/share/mpss/boot/bzImage-knightscorner". After a successful boot the **state** will be **online**, **mode** will be **linux,** and **image** will be **/usr/share/mpss/boot/bzImage-knightscorner**.

The **cmdline** parameter is set by user software, normally the **mpssd** daemon or **micctrl** utility, to pass kernel command line parameters to the Intel® Xeon Phi™ coprocessor Linux* boot process. Current parameters include root file system, console device information, power management options and verbose parameters. When the **state** sysfs node requests the card to boot, the driver adds other kernel command line information to the string and records the complete string that was passed to the booting embedded Linux* OS in the **kernel_cmdline** sysfs node.

17.2.2 Statistics

Sysfs Entries:

/sys/class/mic/micN/boot_count

/sys/class/mic/micN/crash_count

These entries are read-only. **The boot_count** sysfs node returns the number of times that the Intel® Xeon Phi™ coprocessor has booted to the online state. The **crash_count** sysfs node records the number of times that the card has crashed.

17.2.3 Debug SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/platform

/sys/class/mic/micN/post_code

/sys/class/mic/micN/scif_status

/sys/class/mic/micN/log_buf_addr

/sys/class/mic/micN/log_buf_len

/sys/class/mic/micN/virtblk_file

The **platform, post_code,** and **scif_status** entries are read-only; the log_buf_addr, log_buf_len, and virtblk_file entries are read and write.

The **platform** sysfs node should always return a zero value.

The **post_code** sysfs node returns the contents of the hardware register containing the state of the boot loader code. Reading it always returns two ASCII characters. Possible values of note are the strings "12", "FF" and any starting with the character '3'. A string of "12" indicates the Intel® Xeon Phi™ coprocessor is in the ready state and waiting for a command to start executing. A string of "FF" indicates the coprocessor is executing code. A string starting with the character '3' indicates the coprocessor is in the process of training memory. Any other value should be transitory. Any other value remaining for any length of time indicates an error and should be reported to Intel.

The **log_buf_addr** and **log_buf_len** parameters inform the driver of the memory address in the Intel® Xeon Phi™ coprocessor memory to read its Linux* kernel log buffer. The correct values to set are found by looking for the strings "log_buf_addr" and log_buf_len" in the Linux* system map file associated with the file in the **OSimage** parameter, and are typically set by the **mpssd** daemon.

The **virtblk_file** sysfs node indicates the file assigned to the virtio block interface.

17.2.4 Flash SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/flashversion

/sys/class/mic/micN/flash_update

/sys/class/mic/micN/fail_safe_offset

These nodes are all read-only. The **flashversion** sysfs node returns the current version of the flash image installed on the card by the **micflash** utility. The other two are used by the **micflash** command. Root privileges are required to read **flash_update** and **fail_safe_offset** entries.

17.2.5 Power Management SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/pc3_enabled

/sys/class/mic/micN/pc6_enabled

The pc3_enabled node reports the current setting of the pc3 power management setting. If pc3 power management is causing errors, writing a "0" to this setting will disable pc3 power management.

The pc6_enabled node reports the current setting of the pc6 power management setting. If pc6 power management is causing errors, writing a “0” to this setting will disable pc6 power management.

17.2.6 Other SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/extended_family

/sys/class/mic/micN/extended_model

/sys/class/mic/micN/fuse_config_rev

/sys/class/mic/micN/meminfo

/sys/class/mic/micN/memoryfrequency

/sys/class/mic/micN/memoryvoltage

/sys/class/mic/micN/model

/sys/class/mic/micN/stepping

/sys/class/mic/micN/stepping_data

These sysfs nodes are all read-only and return the contents of a particular hardware register. They are used by the micinfo command.

18 General Services Tutorial

This is a brief description of how services start on supported Linux host Operating Systems and the Intel® Xeon Phi™ coprocessor. This is intended for customers who are adding custom services that may interact with the services supplied with Intel MPSS. In all cases the described priority only applies to initial boot and run level changes. The priority or dependencies are not checked when services are manually started and stopped.

18.1 Service Startup by Priorities (Redhat 6.x)

Redhat traditionally uses this method of startup and shutdown. A line is added to the top of the service script that defines the run levels and priority of when a service starts at boot time.

Here is an example snippet from the top of a service file:

```
#!/bin/bash
# chkconfig:  2345 10 90
# ...
```

This tells the startup daemon in Linux to shut down the service early (priority 10) and start the service late (priority 90) when entering Linux run levels 2, 3, 4, and 5. If a service A depends on another service B the shutdown and startup priorities should reflect the relative priorities sooner and later respectively.

```
#!/bin/bash
# Service B
# chkconfig:     2345 10 90
```

and…

```
#!/bin/bash
# Service A
# chkconfig:     2345 9 91
```

NOTE: The priority increases in time for both shut down and start of a service. Now service A will start after service B and service B will shutdown after service A.

When you have multiple dependencies make sure the new service’s shutdown time in the minimum of the dependencies minus 1 and the start priority is max of dependencies + 1.

There is a tool for managing services runlevels and priorities called chkconfig. For more details please see:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-services-chkconfig.html

18.2 Service Startup by Dependencies (SuSE 11 SP2 & SP3)

In addition to the chkconfig comment line from the Redhat distribution priority method, SuSE 11 adds a new concept to the startup order, dependencies. The chkconfig method is present for backward compatibility.

Here is a snippet we can refer to:

```
#!/bin/bash
# chkconfig: 35 75 54
# description: Novell Identity Manager User Application
#
### BEGIN INIT INFO
# Provides: userapp
# Required-Start: $ndsd $network $time
# Required-Stop:
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Short-Description: Novell IDM UserApp
# Description: Novell Identity Manager User Application
### END INIT INFO
```

Some short definitions:

Provides	- The name used to identify this service in the init daemon
Required-Start	- Space delimited Provides names of services to start before this service
Required-Stop	- Space delimited Provides names of services to stop before this service
Default-Start	- Space delimited list of run levels to start when transitioning run levels
Default-Stop	- Space delimited list of run levels to stop when transitioning run levels
Short-Description	- Short display name of service
Description	- Full display name of service

To make sure the service start order is correct, pick the list of service dependencies and list them on the *Required-Start* line. Make sure to fill in the start and stop run levels as appropriate. Optionally list the services to stop after the service represented by this script.

NOTE: All names used for service reference must be the *Provides* name and not the file name of the script!

For more details on this method see:

http://www.novell.com/support/kb/doc.php?id=7002295

18.3 Xeon Phi™ Coprocessor Method for Service Start Priority

The Intel® Xeon Phi™ coprocessor's init daemon using the SuSE 11 dependency system described above in the previous section.

19 Configuration Examples

19.1 Network Configuration

Linux* provides functionality for creating a new network interface for physical interfaces to slave to. Network packets received on any of the slaves are passed unchanged to the bridge. The bridge is assigned the IP address associated with the system it exists on.

Network packets arriving on any of the physical interfaces are passed to the bridge. If the destination for the packet is the IP address assigned to the bridge, it is passed to the TCP/IP stack on the system. If it is any other destination, the bridge performs the role of a network switch and passes it to the correct physical interface for retransmit.

The Intel MPSS software uses the bridging functionality to place the virtual network for each card on the same subnet allowing standard networking software to function seamlessly.

19.1.1 Internal Bridge Example

Internal bridging is a term created to describe a networking topology with Intel® Xeon Phi™ coprocessors connected through a bridged configuration. The advantage of the internal bridge over the default static pair network configuration is the ability for the cards to communicate with each other as well as the host.

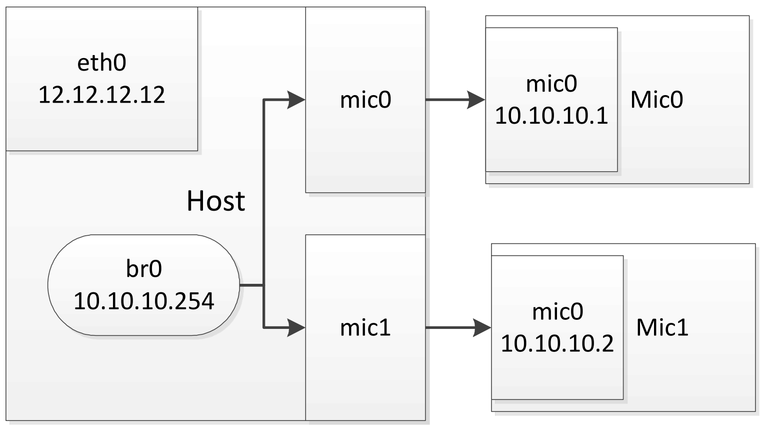

Figure 3 Internal bridge network

If the cards must communicate with the broader physical network, the bridge must be routed to physical Ethernet.

Figure 3 Internal bridge network illustrates the internal bridged network topology. In this example the host and the cards can all communicate through the 10.10.10 subnet. The host can communicate outside through the 12.12.12 subnet but the cards cannot. The series of commands to create this topology would be:

```
[host]# micctrl --addbridge=br0 --type=internal \
--ip=10.10.10.254
[host]# micctrl --network=static --bridge=br0 --ip=10.10.10.1
```

The **micctrl --addbridge** command performs a series of steps starting with removing the old network configuration. Then, if the host does not have a configuration for the designated bridge in */etc/sysconfig/network-scripts/ifcfg-br0,* it creates it containing:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=10.10.10.254
NETMASK=255.255.255.0
```

The **micctrl** utility then executes an “ifup br0” command to enable the new bridge interface. After the bridge is enabled the */etc/mpss/default.conf* file has the additional line:

```
Bridge br0 Internal 10.10.10.254 24
```

The value of 24 at the end of the line is the default **NetBits** or **PREFIX** value of 24 defining a netmask of FFFFFF00. If the **--mtu** option had been used on the **micctrl** command line then it would have followed the 24.

The **micctrl --network** command slaves the host ends of the virtual network, connects to the designated bridge **br0**, and replaces the network configuration files for the Intel® Xeon Phi™ coprocessors with a configuration for the new IP addresses. The first step of this process is to shutdown the current virtual network connections with the **ifdown** command, then create new configuration files. The */etc/sysconfig/network-scripts/ifcfg-micN* becomes:

```
DEVICE=micN
ONBOOT=yes
NM_CONTROLLED="no"
BRIDGE=br0
```

The */etc/sysconfig/network-scripts/ifcfg-micN* file contains the same information with the exception of **DEVICE**. If the associated bridge configuration had set an MTU value other than the default, this file would also contain an MTU value assignment.

When this is complete **micctrl** executes **ifup** commands on both **micN's**. At the end of this process, the **brctl show** command can be used to check the status of the bridge. Its output should be:

```
bridge name     bridge id

STP enabled     interfaces
br0             8000.66a8476a8f15
no                      micN

                        micN
```

The **ifconfig** command relevant output should be:

```
br0         Link encap:Ethernet
            inet addr:10.10.10.254  Bcast:10.10.10.255  Mask:255.255.255.0
micN    Link encap:Ethernet
micN    Link encap:Ethernet
```

These commands show the micN and micN virtual network interfaces are slaved to bridge br0. Bridge br0 has been assigned the host IP address and the slaves do not have their own IP addresses.

The old Network configuration parameters in each card's configuration file are then replaced with the new line. For example the */etc/mpss/micN.conf* file now has the Network configuration line:

```
Network StaticBridge br0 10.10.10.1 yes
```

The */etc/mpss/micN.conf* file will have the same line with the exception of the IP address being assigned the value 10.10.10.2.

Micctrl then creates the network configuration files for the Intel® Xeon Phi™ coprocessor file system. It will first create the network interface configuration file */var/mpss/micN/etc/network/interfaces* with the contents:

```
auto micN
iface micN inet static
address 10.10.10.1
gateway 10.10.10.254
netmask 255.255.255.0
```

The */var/mpss/micN/etc/network/interfaces* file is similar, with correct address, gateway, and netmask values corresponding to coprocessor micN.

Next it will create the */var/mpss/micN/etc/hosts* with the content similar to:

```
127.0.0.1       zappa-micN.music.local micN localhost.localdomain localhost
::1             zappa-micN.music.local micN localhost.localdomain localhost
10.10.10.254    host zappa.music.local
```

```
10.10.10.1            micN zappa-micN.music.local micN
```

NOTE: The entries contained in the configuration file Hostname parameter are related to the IP addresses of other reachable network interfaces, along with the shortened form host, micN, micN, etc.

The last of the process is to add entries to the */etc/hosts* file on the host. This last process could be avoided if the **micctrl --network** command specifies the option **--modhost=no**.

The next boot of the Intel® Xeon Phi™ coprocessors, by either [1]**service mpss start** or **micctrl -b** will use the new network configuration and the cards will be able to ssh to each other.

19.1.2 Basic External Bridge Example

External bridging is a term used in the Intel MPSS software to describe a network topology where the virtual network interfaces are bridged to a physical network interface. This will be the desired configuration in clusters. The topology diagram in Figure 4 External bridge network shows individual hosts and their Intel® Xeon Phi™ coprocessors become a part of the larger subnet.

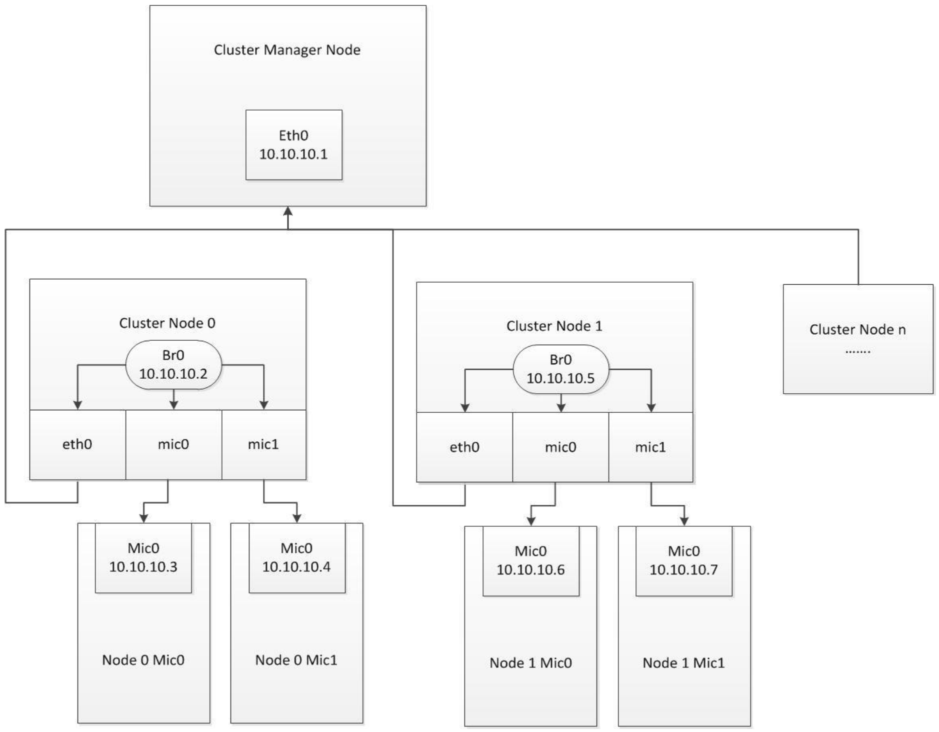

Figure 4 External bridge network

This section discusses the steps required to create an external bridge topology as illustrated above.

On the host node0 the commands to configure the virtual network interfaces are:

```
[host]# micctrl --addbridge=br0 \
--type=external --ip=10.10.10.2
[host]# micctrl --network=static --bridge=br0 \
--ip=10.10.10.3
```

And on the host node1 the commands are:

```
[host]# micctrl --addbridge=br0 \
--type=external --ip=10.10.10.5
[host]# micctrl --network=static --bridge=br0 \
--ip=10.10.10.6
```

Normally, the system administrator will have configured the bridge br0 and tied the eth0 interface to it before these configuration calls. If they did not, then **micctrl** will create the */etc/sysconfig/network-scripts/ifcfg-br0*. It will not, however, create the configuration file for the eth0 physical network interface; the system administrator will need to do this step. An example of */etc/sysconfig/network-scripts/ifcfg-eth0* file is:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

When this is completed, perform a [1]**service network restart**.

In the process of configuring an external bridge, **micctrl** assumes that the network packet MTU size must be set to the Ethernet standard 1500 bytes. This value significantly slows down data transfer across the virtual Ethernet interfaces. If the physical network hardware allows, it would be better to increase the MTU size to the biggest possible value. Typically most hardware will support at least 9000 byte mtu sizes.

This could have been set by the original **micctrl --addbridge** command with the addition of the **--mtu=9000** argument. If this was not done and the system administrator wishes to increase the value, this may be done with the command:

```
[host]# micctrl --modbridge --mtu=9000
```

19.2 IPoIB Networking Configuration

The OFED IPoIB driver is an implementation of the IP over InfiniBand protocol as specified by RFC 4391 and 4392, issued by the IETF IPoIB working group. It is a native implementation in the sense of setting the interface type to ARPHRD_INFINIBAND and the hardware address length to 20 versus implementations that are masqueraded to the kernel as Ethernet interfaces.

The code base is a direct port from OFED 1.5.4.1, without change. The module runs on top of Intel® Xeon Phi™ CCL-Direct Kernel IB Verbs. As a result, most of the functional and performance characteristics are bound by CCL-Direct restrictions. The driver is released to enable InfiniBand-based Lustre* solutions that require IPoIB interface regardless of LNET configurations.

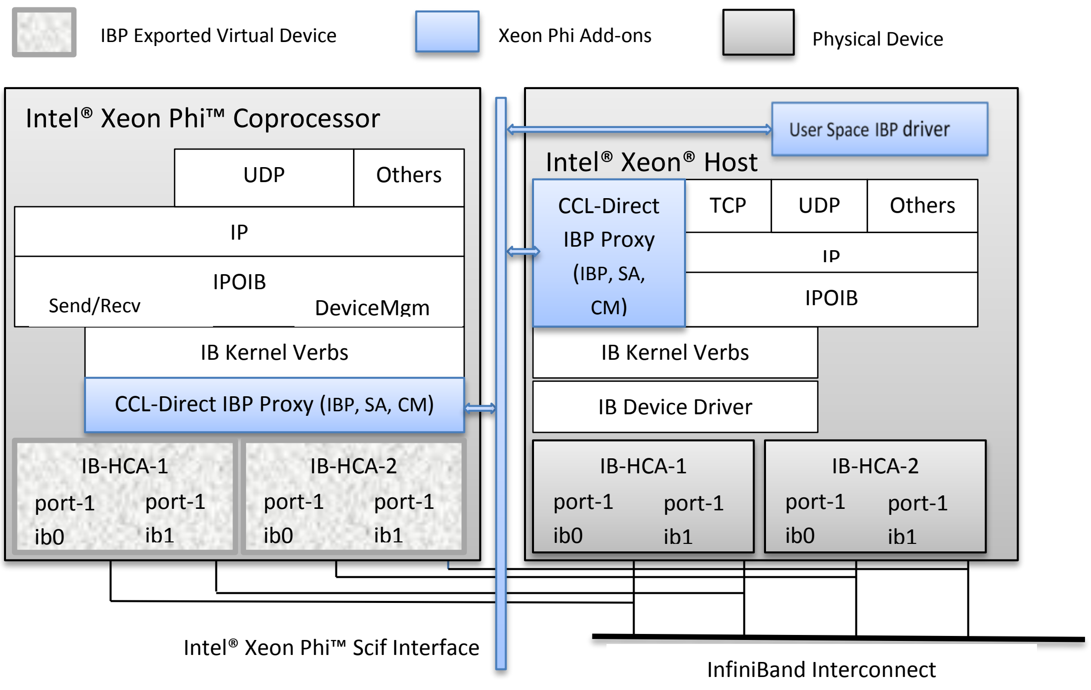

Figure 5 One-to-One IB Device (HCA, Port) Mapping between Host and Coprocessor

19.2.1 Managing the IPoIB Interface

The Intel® Xeon Phi™ coprocessor IPoIB currently manages the virtual IB devices via CCL-Direct IBP proxy drivers. Its existing configuration parameters are inherited from OFED setting without change.

To enable the IPoIB interface on the Intel® Xeon Phi™ coprocessor from the host, edit /etc/mpss/ipoib.conf to bring up the “ib0” interface on a coprocessor with the default hostname (micN):

```
ipoib_enabled=yes
micN_ib0=192.168.100.100
```

19.2.2 IP Addressing

Unlike the Intel® Xeon Phi™ coprocessor Ethernet virtual driver, IPoIB does not require bridging or routing to be configured. In the default case, there is an automatically created one-to-one mapping of (HCA, Port) pair between the host and coprocessor. Figure 1 shows an example configuration with two 2-port HCAs on the host. All 8 ports (host and coprocessor combined) can be individually configured by “net-if” commands. On the Intel® Xeon Phi™ node, the setting is configured by *ifconfig* command, by adding a configuration file in /etc/sysconfig/network, or by editing /etc/mpss/ipoib.conf. The host side follows the host OS conventions.

19.2.3 Datagram vs. Connected Modes

The driver supports two modes of operation: datagram and connected. The mode is set and read through the interface’s /sys/class/net/<intf name>/mode file. Datagram is the default mode.

In datagram mode, the CCL-Direct IB UD transport is used, and the IPoIB MTU is equal to the IB L2 MTU minus the IPoIB encapsulation header (4 bytes). For example, in a typical IB fabric with a 2K MTU, the IPoIB MTU will be 2048 - 4 = 2044 bytes.

In connected mode, the IB RC transport is used. Connected mode takes advantage of the connected nature of the IB transport that allows an MTU up to the maximal IP packet size of 64K. This reduces the number of IP packets needed for handling large UDP datagrams and TCP segments, and increases the performance for large messages.

20 Intel MPSS Cluster Setup Guide

This section is designed to outline the basic steps necessary to configure Intel® Xeon Phi™ coprocessors, based on Intel® Many Integrated Core (Intel® MIC) Architecture, in a wide variety of cluster environments. Some examples in this document may not apply to specific cluster environments, but were created in a general way so that they may be adapted. The following items are covered:

- Setting up an environment that enables communication between Intel® Xeon Phi™ coprocessors in the cluster to enable the user to run the Intel® Message Passing Interface (Intel® MPI) application via a Static Bridge or DHCP configuration.
- Enabling users to connect to and from the Intel® Xeon Phi™ coprocessor by using SSH without a password.
- Enabling administrators to mount the home file system.
- Bulk Coprocessor Flash update procedure in a cluster

The flowchart in Figure 6 Setup process flowchart shows the set up process that is covered in this document:

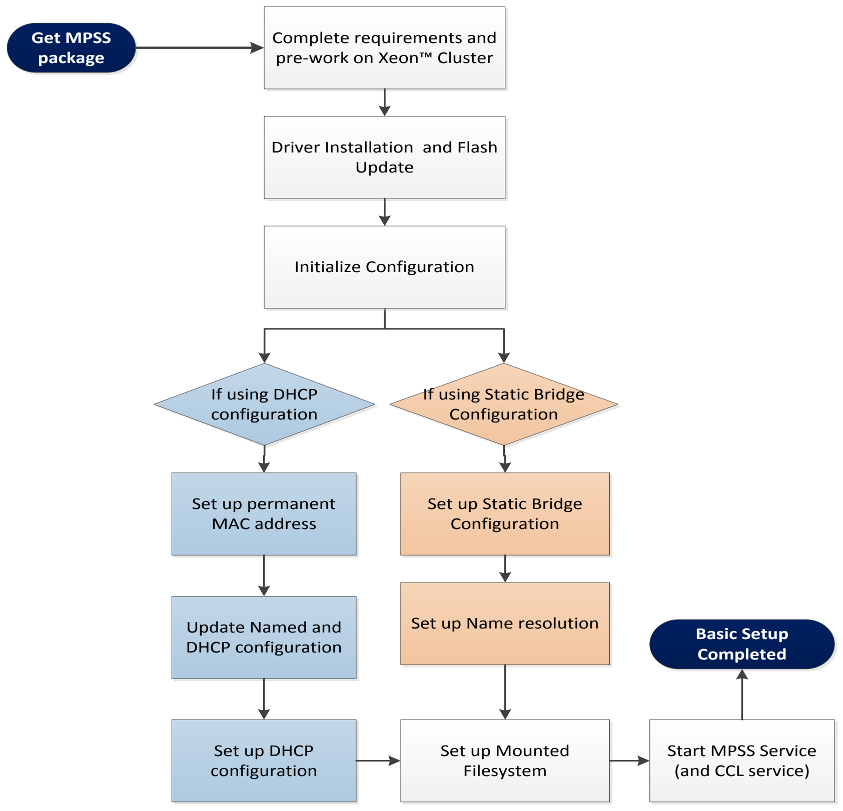
Get MPSS package
Complete requirements and pre-work on Xeon™ Cluster
Driver Installation and Flash Update
Initialize Configuration
If using DHCP configuration
If using Static Bridge Configuration
Set up permanent MAC address
Set up Static Bridge Configuration
Update Named and DHCP configuration
Set up Name resolution
Basic Setup Completed
Set up DHCP configuration
Set up Mounted Filesystem
Start MPSS Service (and CCL service)

Figure 6 Setup process flowchart

More features and functionalities are available for additional configuration but will not be covered here. See Sections 12 through 18 of this User's Guide for detailed information on using the micctrl tool and setting up various coprocessor network options.

20.1 Pre-work

Ensure that the following requirements are met before installing the Intel MPSS RPMs and updating the coprocessor Flash image.

1) Verify administrative access privileges (i.e. root access).
2) Verify that the kernel version of the Operating System installed in the cluster is supported for Intel® Manycore Platform Software Stack (Intel MPSS). If not, then it will be necessary to rebuild the kmod*.rpms (see readme.txt, Section 2.1, "Requirements", under "Resolution 2").
3) Execute uname -a on the console to verify the version of the Linux* Kernel that is running.
4) Match the Intel MPSS RPM package build to the Linux* Kernel or operating system running on the cluster. For example, see the supported Linux* Kernel versions below:
   - Red Hat*/Centos* 6.3 = 2.6.32-279
   - Red Hat*/Centos* 6.4 = 2.6.32-358
   - Red Hat*/Centos* 6.5 = 2.6.32-431
   - SUSE* 11 SP2 = 3.0.13-0.27
   - SUSE* 11 SP3 = 3.0.76-0.11
5) Ensure that the Intel® Xeon Phi™ coprocessors are installed on the Intel® Xeon® host systems of the cluster, and are visible through the standard lspci command:

   ```
   [host]# lspci | grep -i co-processor
   ```

7) You should see something like this on the console (actual bus number and device ID may vary based on coprocessor SKU and PCIe slot used):

   08:00.0 Co-processor: Intel Corporation Device 225c (rev 20)

7) Ethernet network connection between the cluster head node and the compute nodes are functional.
8) Intel® ComposerXE and Intel® MPI packages (download from Intel® Premier) are installed and sourced as needed.

20.2 RSA/DSA Key Creation

RSA/DSA keys for root and user (when available) should be created prior to Intel MPSS installation. This will allow the mpssd daemon to automatically obtain the keys and copy them to the coprocessor when the service is started.

20.3 IP Assignment

In order to get the Intel® Xeon Phi™ coprocessor to communicate with cards within the compute node as well as cards outside the compute node, all assigned IP addresses for the coprocessor have to be set to the same subnet as the rest of the cluster. There are various options for implementing this. See section 19.1 Network Configuration to learn how to set up the bridging options using the micctrl utility.

20.4 Flash Update in Cluster Environment

To update the coprocessor flash image in a cluster environment, it is recommended to use a utility like PDSH (Parallel Distributed Shell) to initiate the flash update over a large number of cards in parallel. To update the flash manually one node at a time, see Section 2.4, "Intel® Xeon Phi™ Coprocessor Flash & SMC Update", in the Intel MPSS Readme.

In a cluster environment, it is recommended to use a tool like the open source Parallel Distributed Shell (PDSH), found in http://sourceforge.net/projects/pdsh, to update all the compute nodes at the same time, although any parallel shell tool can be used. The generic command usage for pdsh is:

```
[host]# pdsh -w <node range>  <command>
```

To update the coprocessor flash on many cards in a clustered setup follow these basic steps below:

1) Ensure that an ssh session from the head node to the compute node(s) is successful without a password since psdh will also use ssh to communicate to all coprocessors in the cluster.
2) Ensure that the flash RPM is installed, which contains the flash update files needed.
3) Ensure that the coprocessors are in the ready state in preparation of the card flash update procedure and Intel MPSS service is not running.

```
[host]# pdsh -w node[1-4] 'micctrl -r'
```

4) Ensure that the required Intel MPSS configuration has been initialized in /etc/mpss by executing the following command:

```
[host]# pdsh -w node[1-4] 'micctrl --initdefaults'
```

Update the coprocessor SMC and flash image by executing the following commands:

```
[host]# pdsh -w node[1-4] 'micflash -update \
/usr/share/mpss/flash/EXT_HP2_SMC_Bootloader_1_8_4326.css_ab \
-device all -noreboot'
```

NOTE: The above command is only required for B0 or B1 silicon stepping, skip for C-stepping.

```
[host]# pdsh -w node[1-4] 'micflash –update \
/usr/share/mpss/flash/EXT_HP2_XX_XXXX-XX.rom.smc \
-device all -silent -log root/micflash_update_log.txt \
-noreboot'
```

A reboot of each compute node that has a coprocessor installed is required after the flash update is successful.

The micflash tool will report the status of the coprocessor flash update on each card in the cluster. Search for the final status in the log files created across the nodes, such as:

```
[host]# pdsh -w node[1-4] grep status \
/root/micflash_update_log.txt
```

Verify that the flash and SMC update indicates success (i.e. no errors). If you see errors on some nodes, confirm that the coprocessors on these nodes were in the 'ready' state and not the 'online' state. You can also issue a 'micctrl -s' command to check the state of the coprocessors on each node to ensure they were in the 'ready' state as required for the flash update tool.

20.5 Initialize Configuration

Configuration should already be initialized during the flash update process in the previous chapter. However, if the node is reimaged or a new coprocessor is added to the server, the configuration must be reinitialized:

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

This will generate the required *default.conf* and *micN.conf* files for all detected Intel® Xeon Phi™ coprocessors.

20.6 Set Up Static Bridge Configuration

This section will cover how to set up the Static Bridge configuration. Skip this section if you are setting up the DHCP configuration. Also see Section 14 for more info on coprocessor network configuration options using the micctrl tool.

20.6.1 Modify the Intel MPSS Configuration

There are two ways to set up the coprocessor configuration file. The first (and recommended) option is to use micctrl to set up the environment:

1) Add the coprocessors to the preexisting bridge:

```
[host]# micctrl --addbridge=br0 --type=External \
--ip=$HostIP --netbits=24 micN micN
where $HostIP is the IP address of the br0.
```

2) Set up the IP for the coprocessor:

```
[host]# micctrl --network=static --bridge=br0 \
--ip=$MICNIP:$MICNIP --netbits=24 micN micN
where $MICNIP is the IP for micN, and $MICNIP is the IP for
micN.
```

3) Additional step for SUSE* Linux* Enterprise Server: Make sure to restart the network so the compute node's network stack can reflect the changes.

```
[host]# 1service network restart
```

Alternatively, system administrators can directly modify the configuration files located in the /etc/mpss/ directory:

1) Add the following line in the default.conf file, so that it has the bridge setup:

   Bridge $Bridgename External $Bridge_or_Host_IP $Netbits $MTUSize

   Where:

   a) $Bridgename can be the host's preexisting bridge name (usually set as br0).
   b) $Bridge_or_Host_IP should have the same value as what is assigned to the network bridge of choice.
   c) $Netbits value should be equivalent to the bridge's netmask.
   d) $MTUSize value should be identical to the bridge's MTU size.

   Example line for default.conf for node with bridge br0 with IP 192.168.0.1:

```
Bridge br0 External 192.168.0.1 24 9000
```

2) Modify the Network configuration line in the micN.conf to make sure the Static Pair configuration is changed to Static Bridge:

   Network StaticBridge $Bridgename $MICnIPAddr $Y_N_Update_etc_host

   Where:

   a) $Bridgename is the bridge that is defined in the default.conf.
   b) $MICnIPAddr is the IP address for MICN (micN, micN,…).
   c) $Y_N_Update_etc_host is to decide whether to update the */etc/hosts*.

Example for micN with IP of 192.168.0.2 and choose to update the */etc/hosts*:
Original line:

```
Network class=StaticPair micip=172.31.1.1
hostip=172.31.1.254 mtu=64512 netbits=24 modhost=yes
modcard=yes
```

Replace with:

```
   Network class=StaticBridge bridge=br0 micip=192.168.0.2
  modhost=yes modcard=yes
```

3) Additional step for SUSE* Linux* Enterprise Server: Make sure to restart the network so the compute node's network stack can reflect the changes.

```
[host]$ 1service network restart
```

20.6.2 Set Up Name Resolution for Static Bridge Configuration

20.6.2.1 Modify the host's /etc/hosts

All compute nodes have to be able to identify each other as well as the Intel® Xeon Phi™ coprocessors. A simple way to ensure this is to populate the entire compute node's */etc/hosts* files with all available cards' IPs as shown below:

```
127.0.0.1                 localhost

192.168.0.254             CChead.cluster CChead
master.cluster master
#Xeon Servers
192.168.0.1     node1
192.168.0.4     node2
192.168.0.7     node3
192.168.0.10    node4
#MICs
192.168.0.2     node1-micN
192.168.0.3     node1-micN
192.168.0.5     node2-micN
192.168.0.6     node2-micN
192.168.0.8     node3-micN
192.168.0.9     node3-micN
192.168.0.11    node4-micN
192.168.0.12    node4-micN
```

20.6.2.2 Modify the coprocessors' /etc/hosts

Modify the following */etc/hosts* located on the host:

/var/mpss/micN/etc/hosts (for micN)
/var/mpss/micN/etc/hosts (for micN)

Example for the coprocessors' */etc/hosts* (for micN):

```
127.0.0.1       node1-micN micN localhost.localdomain
localhost
::1             node1-micN micN localhost.localdomain
localhost
192.168.0.1     host node1
192.168.0.3     micN node1-micN
192.168.0.4     node2
192.168.0.5     node2-micN
192.168.0.6     node2-micN
192.168.0.7     node3
192.168.0.8     node3-micN
192.168.0.9     node3-micN
192.168.0.10    node4
192.168.0.11    node4-micN
192.168.0.12    node4-micN
```

20.7 Set up DHCP Configuration

This section will cover how to set up a DHCP networked configuration. Skip this section if setting up the Static Bridge configuration. A permanent MAC address is required to make sure the DHCP IP for the coprocessor will never change. System administrators have the option to use the default DHCP leases, however it will not be recommended since name resolution cannot be set definitively.

20.7.1 Modify the Intel MPSS Configuration

There are two ways to set up the DHCP networked configuration. The first (and recommended) option is to use micctrl to set up the environment:

1) Add the coprocessors to the preexisting bridge:

```
[host]# micctrl --addbridge=br0 --type=External \
--ip=dhcp micN micN
```

2) Set up DHCP for the coprocessor:

```
[host]# micctrl --network=dhcp --bridge=br0 \
--ip=dhcp micN micN
```

Alternatively, system administrators can directly modify the configuration files located in the /etc/mpss/ directory:

3) Add a line in default.conf with the bridge configuration:

    Bridge $Bridgename External dhcp 24 9000

   Example:

```
Bridge br0 External dhcp 24 9000
```

4) Modify the Network configuration line in the micN.conf to change the Static Pair configuration so that it will be redirected to the bridge instead:

   Network Bridge $Bridgename

   Where:

   $Bridgename is the bridge that is defined in the default.conf

   Example:

```
Network class=Bridge bridge=br0
```

20.7.2 Set Permanent MAC Address for the Coprocessor

Since DHCP mainly relies on the MAC address, the coprocessors' MAC addresses (`MicMacAddress`) listed in the *micN.conf* should be unique and permanent. One way to ensure this is to retain a permanent list of the coprocessors' MAC addresses, and set the configuration such that it will never change.

Steps to do this:

1) Create a list of MAC addresses for the available coprocessor. This can be found in the micN.conf that is located in the /etc/mpss/ folder.

   Example of the list in a text file:

```
node1-micN      ca:7a:05:e1:52:65       192.168.0.2
node1-micN      76:f2:07:ff:00:3c       192.168.0.3
node2-micN      fe:fc:cd:15:95:8a       192.168.0.5
node2-micN      72:0c:4c:b7:94:86       192.168.0.6
```

2) If this is not the initial DHCP setup for the coprocessors, use the text file above and replace the assigned MicMacAddress to the one on the list. This will remove the need to constantly update the DHCP reservation and the name resolution setup.

20.7.4 Modify the DHCP Configuration File

Add the list of DHCP configuration for the coprocessors to the dhcp configuration file. DHCP configuration should already be available for the compute node. Configuration format should be similar to the basic DHCP setup for compute nodes. For example:

```
host mnode01-micN.mynetwork {
  hardware ethernet  ca:7a:05:e1:52:65;
  fixed-address 192.168.0.2;
}

host mnode01-micN. mynetwork {
  hardware ethernet 76:f2:07:ff:00:3c;
  fixed-address 192.168.0.3;
}

host mnode02-micN. mynetwork {
  hardware ethernet fe:fc:cd:15:95:8a;
  fixed-address 192.168.0.5;
}

host mnode02-micN. mynetwork {
  hardware ethernet 72:0c:4c:b7:94:86;
  fixed-address 192.168.0.6;
}
```

20.7.5 Modify the Name Resolution Configuration File

For every coprocessor, make sure the name resolution is set. This will remove the need to guess or dig through the message log to figure out what IP address a particular coprocessor was assigned. Setting up the name resolution on the coprocessor is the same process for the regular compute nodes.

For example:

```
node01-micN  IN A 192.168.0.2
node01-micN  IN A 192.168.0.3
node02-micN  IN A 192.168.0.5
node02-micN  IN A 192.168.0.6
```

20.8 Set Up Mounted File System

1) Make sure the filesystem that needs to be mounted is accessible from the coprocessor and properly exported.
2) Edit the /var/mpss/micN/etc/fstab (and /var/mpss/micN/etc/fstab when micN is available) to ensure the mounted /home filesystem is listed. For example:

```
192.168.0.254:/home /home nfs    rsize=8192,wsize=8192,nolock,intr,hard   0 0
```

20.9 Intel® Xeon Phi™ Coprocessor User Access

Running Intel® MPI requires user access. Intel MPSS service will automatically set up the passwordless authentication if the rsa key for the user has been created before [1]service mpss start is invoked.

Alternatively, users can be added later by specifying the useradd option that is provided by the micctrl tool.

20.10 Starting Intel MPSS Service

At this point, all necessary configurations should be set up and Intel MPSS service is ready to be started.

Start the Intel MPSS service:

```
[host]# 1service mpss start
```

20.11 Starting OFED-MIC Service

Do this only if OFED* 1.5.4.1 and ofed-mic RPMs are installed.

1) Restart the openibd service.

```
[host]# 1service openibd restart
```

2) Start OFED-MIC service:

```
[host]# 1service ofed-mic start
```

20.12 Ensure Services Are Running After Reboot

NOTE: The following should only be applied if the compute node is not being reimaged on every reboot, and all configurations are set.

To automatically run Intel MPSS service:

```
[host]# chkconfig MPSS on
```

To automatically run OFED-MIC service:

```
[host]# chkconfig ofed-mic on
```

21 Sample Cluster Configuration Scripts

21.1 ifcfg-br0 Configuration File

```
DEVICE=br0

TYPE=Bridge

ONBOOT=yes

DELAY=0

NM_CONTROLLED="no"

MTU=9000

BOOTPROTO=dhcp

NOZEROCONF=yes
```

21.2 ifcfg-eth0 Configuration File

```
DEVICE=eth0

ONBOOT=yes

BRIDGE=br0

MTU=9000
```

21.3 Hostlist Configuration File

```
192.168.0.1        node1

192.168.0.2        node1-micN

192.168.0.3        node1-micN

192.168.0.5        node2

192.168.0.6        node2-micN
```

```
192.168.0.7       node2-micN

192.168.0.9       node3

192.168.0.10      node3-micN

192.168.0.11      node3-micN

192.168.0.13      node4

192.168.0.14      node4-micN

192.168.0.15      node4-micN
```

21.4 OFED* 1.5.4.1 Custom Configuration Answer

```
kernel-ib=n

core=y

mthca=y

mlx4=y

mlx4_en=y

ipoib=y

sdp=y

srp=y

srpt=y

rds=y

kernel-ib-devel=n

libibverbs=y

libibverbs-devel=y

libibverbs-devel-static=y

libibverbs-utils=y

libibverbs-debuginfo=y

libmthca=y

libmthca-devel-static=y
```

```
libmthca-debuginfo=y

libmlx4=y

libmlx4-devel=y

libmlx4-debuginfo=y

libcxgb3=n

libcxgb3-devel=n

libcxgb3-debuginfo=n

libcxgb4=n

libcxgb4-devel=n

libcxgb4-debuginfo=n

libnes=n

libnes-devel-static=n

libnes-debuginfo=n

libipathverbs=y

libipathverbs-devel=y

libipathverbs-debuginfo=y

libibcm=y

libibcm-devel=y

libibcm-debuginfo=y

libibumad=y

libibumad-devel=y

libibumad-static=y

libibumad-debuginfo=y

libibmad=y

libibmad-devel=y

libibmad-static=y

libibmad-debuginfo=y
```

```
ibsim=y

ibsim-debuginfo=y

ibacm=y

librdmacm=y

librdmacm-utils=y

librdmacm-devel=y

librdmacm-debuginfo=y

libsdp=y

libsdp-devel=y

libsdp-debuginfo=y

opensm=n

opensm-libs=n

opensm-devel=n

opensm-debuginfo=n

opensm-static=n

compat-dapl=y

compat-dapl-devel=y

dapl=n

dapl-devel=n

dapl-devel-static=n

dapl-utils=n

dapl-debuginfo=n

perftest=y

mstflint=y

sdpnetstat=y

srptools=y

rds-tools=y
```

```
rds-devel=y

ibutils=y

infiniband-diags=y

qperf=y

qperf-debuginfo=y

ofed-docs=y

ofed-scripts=y

infinipath-psm=y

infinipath-psm-devel=y

mpi-selector=y

mvapich_gcc=y

mvapich2_gcc=y

openmpi_gcc=y

mpitests_mvapich_gcc=y

mpitests_mvapich2_gcc=y

mpitests_openmpi_gcc=y

build32=0

prefix=/usr

mvapich2_conf_impl=ofa

mvapich2_conf_romio=1

mvapich2_conf_shared_libs=1

mvapich2_conf_ckpt=0

mvapich2_conf_vcluster=small
```

21.5 Pre-Config Script for Static Bridge Configuration

```
/*

* Copyright (C) 2013 Intel Corporation

*

* Intel makes no warranty of any kind regarding this code. This  *
code is licensed on an "AS IS" basis and Intel will not provide  *
any support, assistance, installation, training,  or other     *
services. Intel may not provide any updates, enhancements or   *
extensions to this code. Intel specifically disclaims any      *
warranty of merchantability, non-infringement, fitness for any *
particular purpose, or any other warranty. Intel disclaims all *
liability, including liability for infringement of any         *
proprietary rights, relating to use of the code. No license,   *
express or implied, by estoppel or otherwise, to any           *
intellectual property rights is granted herein.

*/



rm -rf /etc/sysconfig/network-scripts/ifcfg-eth0

cp -f /donotremove/ifcfg-eth0 /etc/sysconfig/network-scripts/

cp -f /donotremove/ifcfg-br0 /etc/sysconfig/network-scripts/

micctrl --initdefaults

service network restart

sleep 5

Value=`ifconfig br0 | grep "inet addr" | gawk -F: '{print $2}' |
gawk '{print $1}' | awk '{print substr($0,11,4)}'`

HostIP=`ifconfig br0 | grep "inet addr" | gawk -F: '{print $2}' |
gawk '{print $1}'`

IPQuad=`ifconfig br0 | grep "inet addr" | gawk -F: '{print $2}' |
gawk '{print $1}' | awk '{print substr($0,0,9)}'`

newvalue1=`expr $Value + 1`

newvalue2=`expr $Value + 2`

micctrl --addbridge=br0 --type=External --ip=$HostIP  micN micN
micctrl --network=static --bridge=br0 --
ip=$IPQuad.$newvalue1:$IPQuad.$newvalue2 micN micN

1service mpss start
```

21.6 Pre-Config Script for DHCP configuration

```
/*

* Copyright (C) 2013 Intel Corporation

*

* Intel makes no warranty of any kind regarding this code. This  *
code is licensed on an "AS IS" basis and Intel will not provide *
any support, assistance, installation, training,  or other     *
services. Intel may not provide any updates, enhancements or   *
extensions to this code. Intel specifically disclaims any      *
warranty of merchantability, non-infringement, fitness for any *
particular purpose, or any other warranty. Intel disclaims all *
liability, including liability for infringement of any         *
proprietary rights, relating to use of the code. No license,   *
express or implied, by estoppel or otherwise, to any           *
intellectual property rights is granted herein.

# Intel MPSS Cluster Init Script

micctrl --cleanconfig

micctrl --initdefaults --users=overlay --pass=shadow --modhost=no -
-modcard=no

# Create Bridge Configuration on MIC filesystem and generate br0 on
compute node

micctrl --addbridge=br0 --type=External --ip=dhcp --mtu=1500

# Make a ifcfg file needed to add eth0 to the new bridge created by
micctrl above

echo "DEVICE=eth0

ONBOOT=yes

BRIDGE=br0

MTU=1500

" > /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
# Restart networking with the new Bridge

service network restart

sleep 5

# Update each card so it's part of the bridge

micctrl --network=dhcp --bridge=br0

# NFS mount the head node /home onto each mic card

micctrl --addnfs=192.168.0.1:/home --dir=/home

# Copy host /etc/hosts to mic card file system

cp -f /etc/hosts /var/mpss/mic0/etc

cp -f /etc/hosts /var/mpss/mic1/etc

micctrl –updateramfs


# Start Intel MPSS services

[1]service mpss start

micctrl -w


# Start infiniband and OFED services

[1]service openibd restart

[1]service ofed-mic start
```

22 Related Documentation

This section contains a listing of MYO, COI, and SCIF documentation, as well as links to various Intel® Xeon Phi™ Coprocessor collateral documents.

22.1 MYO Documentation

MYO is a library and API to support virtual shared memory between processes on a host process and Intel® Xeon Phi™ coprocessor cards. MYO is supplementary to other Intel® Xeon Phi™ hardware and software, and is intended for researchers and advanced users.

22.1.1 MYO Man Page Location

/usr/share/man/man3

22.1.2 MYO Tutorials & Other Document Location on Linux*

/usr/share/doc/myo

22.2 COI Documentation

Intel® Coprocessor Offload Infrastructure (Intel® COI) provides a set of APIs to simplify development of tools and other applications using offload and reverse accelerator models.

22.2.1 COI Documentation for Linux*

/usr/share/doc/intel-coi-[version number]/	-release_notes.txt
/usr/share/doc/intel-coi-[version number]/	-MIC_COI_API_Reference_Manual_0_65.pdf
/usr/share/doc/intel-coi-[version number]/	-coi_getting_started.pdf
/usr/include/intel-coi/	-header files contain full API descriptions
/usr/share/doc/intel-coi-[version number]\tutorials\	-Full tutorials source and Makefiles
/usr/share/man/man3	-man pages

22.3 SCIF documentation

SCIF provides a mechanism for communication between the components of a distributed application. It is intended for tools developers and application developers.

22.3.1 SCIF User Guide

~/mpss-[version number]/docs/SCIF_UserGuide.pdf

22.3.2 SCIF Tutorials Location

/usr/share/doc/scif/tutorials

SCIF Tutorials Source:
~/mpss-[version number]/mpss-sciftutorials-doc-*.rpm

Pre-built SCIF Tutorials Binary:
~/mpss-[version number]/mpss-sciftutorials-3.*.rpm

Instructions to Build and Run the SCIF Tutorials:
/usr/share/doc/scif/tutorials/README.txt

22.3.3 SCIF Man Page Locations

/usr/share/man/man3

/usr/share/man/man9

22.4 Intel® Xeon Phi™ Coprocessor Collateral

Intel® Xeon Phi™ Coprocessor Product Brief:
https://www-ssl.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html

Intel® Xeon Phi™ Coprocessor Specification Update:
https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-specification-update.html

Intel® Xeon Phi™ Coprocessor Safety and Compliance Guide:
https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-safety-compliance-guide.html

Intel® Xeon Phi™ Coprocessor Datasheet:
https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-datasheet.html

Intel® Xeon Phi™ Coprocessor Software Users Guide:
https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-software-users-guide.html

Intel® Xeon Phi™ Coprocessor System Software Developers Guide:

https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-system-software-developers-guide.html

Intel® Xeon Phi™ Coprocessor Developers Quick Start Guide:
http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide

Intel® Xeon Phi™ Coprocessor System Administration Guide:
http://software.intel.com/en-us/articles/system-administration-for-the-intel-xeon-phi-coprocessor
Intel® Xeon Phi™ Coprocessor Instruction Set Architecture Reference Manual:
http://software.intel.com/sites/default/files/forum/278102/327364001en.pdf