

Intel® Xeon Phi™ Product Family
Offload Over Fabric
User's Guide

Revision 1.6.1
January 31, 2018

Disclaimer and legal information

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: <http://www.intel.com/design/literature.htm>.

Intel technologies® features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at <http://www.intel.com/> or from the OEM or retailer.

No computer system can be absolutely secure.

Intel, Xeon, Xeon Phi and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Intel does not warrant or guarantee the performance or compatibility of third party commercial products. Reference in this site to any specific commercial product, process, or service, is for the information and convenience of the public, and does not constitute endorsement, or recommendation by Intel.

*Other names and brands may be claimed as the property of others. Copyright© 2018, Intel Corporation. All rights reserved.

Contents

1	Introduction	7
1.1	Offload over Fabric Overview	7
1.2	Terminology	8
1.3	Notational conventions	8
1.4	Reference documents	9
2	Offload over Fabric installation	10
2.1	Prerequisites	10
2.1.1	Operating system	10
2.1.2	Root access	11
2.1.3	OpenFabrics Enterprise Distribution (OFED)	11
2.2	Installation procedure	11
2.2.1	Intel® Xeon Phi™ processor software installation	11
2.2.2	Libfabric installation	12
2.2.3	Offload over Fabric installation	12
2.2.4	Uninstalling Offload over Fabric software	13
3	Offload over Fabric configuration	14
3.1	Offload host configuration	14
3.2	Offload target configuration	14
3.3	Offloading application configuration	15
3.3.1	Offload targets	15
3.3.2	User authentication	17
3.3.3	Memory usage policy	18
3.3.4	Fabric interfaces	19
4	Intel® Coprocessor Offload Infrastructure	20
4.1	Overview	20
4.2	Directories	21
4.3	Configuration	21
4.4	Building and running tutorials	22
4.5	Using <i>coitrace</i> to assist with debugging	23
4.6	<i>micnativeloadex</i> for remote execution	24
4.7	Troubleshooting	24
4.7.1	<i>COIEngineGetHandle</i> Hangs	24
4.7.2	Intel® COI API Returns an error code	24

List of Figures

1.1	Example OOF application	8
3.1	Example OOF configuration - option 1	16
3.2	Example OOF configuration - option 2	17
3.3	Example OOF configuration - option 3	17

List of Tables

1.1	Terminology	8
1.2	Notational conventions used in this paper.	8
1.3	Reference documents	9
2.1	Supported host operating systems	10
2.2	Matrix of validated fabric hardware, OFED and OFI versions.	11
3.1	Configuring authentication mechanisms	18
3.2	Configuration of target memory kind	19
4.1	Intel® COI directory listing	21
4.2	Intel® COI tutorials	22

Revision History

Revision	Date	Description
1.6.1	January 2018	Release of the L ^A T _E X version of the document
1.6.0	September 2017	Intel® Xeon Phi™ processor software 1.6.0 update
1.5.1	January 2017	Intel® Xeon Phi™ processor software 1.5.1 update
1.5	December 2016	Intel® Xeon Phi™ processor software 1.5.0 update
1.4	October 2016	Intel® Xeon Phi™ processor software 1.4.3 update
1.3	September 2016	Intel® Xeon Phi™ processor software 1.4.2 update
1.2	August 2016	Intel® Xeon Phi™ processor software 1.4.1 update
1.1	June 2016	Intel® Xeon Phi™ processor software 1.4.0 update
1.0	June 2016	Public version
1.0	February 2016	Initial official version
0.5	February 2016	Draft revision for review.

Chapter 1

Introduction

This document pertains the Offload over Fabric technology, which allows users to offload workloads to numerous interconnected compute nodes.

This guide provides an overview of the OOF technology, shows how to install and configure it, and how to make use of its features. Please note that the OOF technology was designed for systems containing Intel® Xeon Phi™ processors x200.

1.1 Offload over Fabric Overview

The Intel® Xeon Phi™ coprocessors x100 introduced the offload programming model, allowing users to offload workloads over PCIe. With the introduction of the Intel® Xeon Phi™ processor x200, this programming model is implemented as Offload over Fabric (OOF) and enables offloading to compute nodes connected within a high-speed network. Communication with the networking layer is realized by the Open Fabric Interface API (OFI). Refer to the [Programming and Compiling for Intel® Many Integrated Core Architecture](#) article for more information on the available programming models.

The Intel® C/C++ and Fortran Compilers support offloading directives in the source code. This feature allows the application developer to specify which parts of the program will be offloaded to the Intel® Xeon Phi™ processor-based nodes.

The compilers and Offload over Fabric programming model use the Intel® Coprocessor Offload Infrastructure (Intel® COI) to perform offloading. Advanced users may also use the Intel® COI API directly. Please refer to Section 5 for details.

The Offload over Fabric software is part of the Intel® Xeon Phi™ processor softwarepackage (available for download at <https://software.intel.com/en-us/articles/xeon-phi-software>). Additional information on it can be found in the Intel® Xeon Phi™ Processor Software User's Guide.

Although Offload over Fabric can coexist alongside other programming models, it was especially designed to be used in MPI applications in HPC cluster environment.

Figure 1.1 shows an example OOF application architecture and its expected execution flow. The presented system consists of several Intel® Xeon Phi™ processor-based servers hosting an MPI application (offload host servers). Massively parallel sections of the application can be offloaded to the Intel® Xeon Phi™ processor-based servers (offload target servers) using compiler offloading. This model takes advantage of the heterogeneous nature of the cluster while maintaining clear distinction in the code between the MPI-based homogeneous part and the offloaded code.

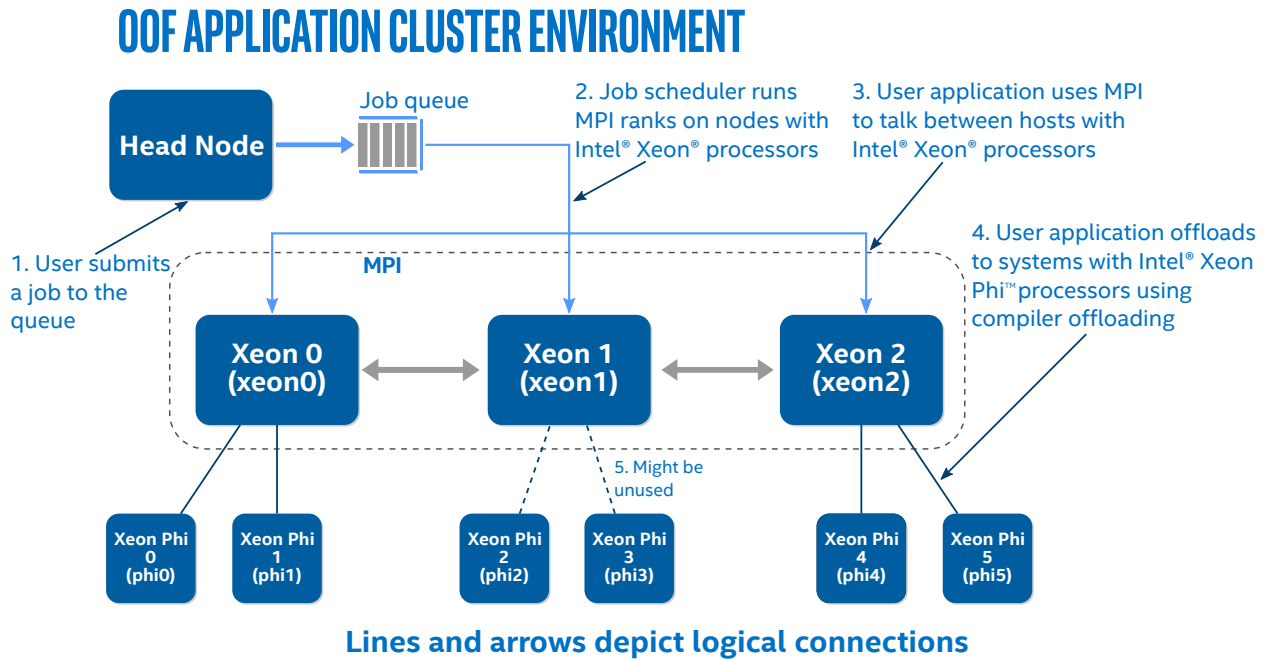


Figure 1.1 – Example OOF application

1.2 Terminology

OFED	OpenFabrics Enterprise Distribution
OFI	OpenFabrics Interfaces
OOF	Offload over Fabric
COI	Coprocessor Offload Infrastructure

Table 1.1 – Terminology

1.3 Notational conventions

<i>zypper rm <package></i>	Commands, their arguments and configuration parameters in prose sections are italicized.
<i>packages/x86_64/core</i>	Files and directories in prose sections are italicized.
<i>command</i>	Code and commands entered by the user. A backslash symbol: \ indicates that command is continued in the next line.
<i>output</i>	Terminal output by the computer.
[host]\$	Commands that do not require root privileges.
[host]#	Commands that require root privileges.
[item]	Items in square brackets are optional.
{item item}	Items to choose from are enclosed in curly braces.
...	Elipsis indicates that the preceding item can be repeated.

Table 1.2 – Notational conventions used in this paper.

1.4 Reference documents

Document	Location
Programming and Compiling for Intel® Many Integrated Cores Architecture	https://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
Intel® Xeon Phi™ Processor Software User's Guide	Available with Intel® Xeon Phi™ Processor Software https://software.intel.com/en-us/articles/xeon-phi-software
The OpenFabrics Enterprise Distribution (OFED*)	https://www.openfabrics.org/index.php/openfabrics-software.html
MUNGE documentation	https://github.com/dun/munge/wiki
COI documentation	<code>/usr/share/doc/intel-coi-<version></code>

Table 1.3 – Reference documents

Chapter 2

Offload over Fabric installation

Instructions in this section show how to install and uninstall the Offload over Fabric software.

Note: it is strongly recommended to read through this chapter before actually proceeding with installation to ensure that all required components and facilities are available. It is also strongly recommended that these installation steps be performed in the order they are presented.

2.1 Prerequisites

Offload over Fabric requires both Intel® Xeon® processor-based host systems and Intel® Xeon Phi™ processor-based target systems to be configured and able to communicate over fast fabric interconnection. The target system must contain at least one Intel® Xeon Phi™ processor.

2.1.1 Operating system

Offload over Fabric has been validated against specific versions of operating systems. The table below list supported versions of the operating system for the host and target systems.

Supported OS version	Kernel version
Red Hat* Enterprise Linux* 64-bit 7.4	kernel-3.10.0-693.el7.x86_64
Red Hat* Enterprise Linux* 64-bit 7.3	kernel-3.10.0-514.el7.x86_64
SUSE* Linux* Enterprise Server 12.2	kernel-default-4.4.21-69.1

Table 2.1 – Supported host operating systems

To obtain the version of the kernel running on the host, execute:

```
[host]$ uname -r
```

Note: access to standard distribution packages and repositories is required to install some of the Intel® Xeon Phi™ processor softwarepackages. Disabling any standard repository may lead to failed dependencies issues. To get more information please refer to the information provided in your Operating System documentation.

2.1.2 Root access

Many of the tasks described in this document require administrative access privileges (i.e. root access). Verify that you have such privileges to the machines which you will configure.

The use of `sudo` to acquire root privileges should be done carefully because its use may cause subtle and undesirable side effects. `Sudo` might not retain the non-root environment of the caller. This could, for example, result in use of a different `PATH` environment variable than expected, ending up with execution of the wrong code.

When `su` is used to become root, the non-root environment is (mostly) retained. (`HOME`, `SHELL`, `USER`, `LOGNAME` are reset unless the `-m` switch is given. See the `su` man page for details).

2.1.3 OpenFabrics Enterprise Distribution (OFED)

OpenFabrics Interface (OFI) is part of the OpenFabrics Enterprise Distribution (OFED). This API was first included as a part of the OpenFabrics Alliance (OFA) OFED 3.18. It is now also a part of other OFED distributions or operating systems. OFI source code can be downloaded from [Libfabric OpenFabrics website](#). A version obtained from the website, compiled, and installed on the system is referred to as OFA OFI or OFA libfabric in this document. It is assumed that OFI (libfabric) is installed on the host system and the target system. Use the [OFI test suite](#) to verify the connectivity between the two systems.

Offload over Fabric has been validated against specific versions of OFED software. The table below lists the supported versions of the fabric hardware, OFED, and OFI.

Fabric hardware	OFED	OFI
Mellanox* MCX353A-FCBT	MLNX_OFED-3.4-2.0.0.0	OFA libfabric-1.4.0
Mellanox* MCX455A-ECAT	MLNX_OFED-3.4-2.0.0.0	OFA libfabric-1.4.0
Intel® Omni-Path HFA 100 Series 1 Port PCIe 16x	Intel 10.3.0.0.81	OFA libfabric-1.4.0

Table 2.2 – Matrix of validated fabric hardware, OFED and OFI versions.

2.2 Installation procedure

Uninstall any previous version of the Intel® Xeon Phi™ processor software prior to installing a new one.

2.2.1 Intel® Xeon Phi™ processor software installation

Install Intel® Xeon Phi™ processor software on hosts with Intel® Xeon Phi™ processors x200. You can find the software on the [Intel® Developer Zone website](#). Download a package (`xpps-<version>-<os>.tar`) for your operating system, extract it and install the packages as shown in the instructions below. You can find more information on the Intel® Xeon Phi™ processor software in the *Intel® Xeon Phi™ Processor Software User's Guide*.

```
[host]$ tar -xvf xpps-<version>-<os>.tar
[host]$ cd xpps-<version>/<os>/x86_64
```

RHEL*/CentOS*:

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

SLES*

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

2.2.2 Libfabric installation

Offload over Fabric software depends on OpenFabrics Interface (OFI), a collection of libraries and applications providing fabric services. *Libfabric* is a software package that contains an OFI implementation. The Offload over Fabric requires *libfabric* to be installed both on host and target systems.

Follow instructions from the [OpenFabrics Interfaces Working Group \(OFIWG\) web page](#) to configure and install *libfabric*. Note that we recommend installing *libfabric* package with the RPM Package Manager. For instance an rpm file can be generated with the following command:

```
[host]$ rpmbuild -ta libfabric-<version>.tar.bz2 -define 'configopts \
--enable-verbs=yes'
```

Note: *Libfabric* makes use of providers to implement its functionalities for different underlying APIs and networking hardware. The offloading runtime uses verbs provider for communication. Make sure that verbs provider is enabled in your installation of *libfabric*. Use the *fi_info* application to list available providers.

2.2.3 Offload over Fabric installation

The Offload over Fabric software can be downloaded from [Intel® Developer Zone website](#). It is distributed in two kinds of packages:

- *xpps-<version>-offload-host-<os>.tar* is the release package with software for the offload host.
- *xpps-<version>-offload-target-<os>.tar* is the release package with software for the Intel® Xeon Phi™ processor-based offload target.

Download and extract the *xpps-<version>-offload-host-<os>.tar* package and install the software.

```
[host]$ tar -xvf xpps-<version>-offload-host-<os>.tar
[host]$ cd xpps-<version>-offload-host/<os>/x86_64
```

RHEL*/CentOS*:

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

SLES*:

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

Download and extract the *xpps-<version>-offload-target-<os>.tar* package and install the software.

```
[target]$ tar -xvf xpps-<version>-offload-target-<os>.tar
[target]$ cd xpps-<version>-offload-target/<os>/x86_64
```

RHEL*/CentOS*:

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

SLES*:

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

2.2.4 Uninstalling Offload over Fabric software

Execute the command below to check for a previously installed version of the Offload over Fabric software:

```
[target]$ rpm -qa | grep xpps-coi  
[target]$ rpm -qa | grep intel-coi
```

Packages that correlate with Offload over Fabric will be listed and can be then uninstalled.

RHEL*/CentOS*:

```
[host]# yum remove <package-name>
```

SLES*:

```
[host]# zypper remove <package-name>
```

Chapter 3

Offload over Fabric configuration

3.1 Offload host configuration

No special system configuration is required on the offload host system. It is recommended, however, to increase the maximum number of open files to enable more complex workloads. Add the following lines to the */etc/security/limits.conf* file to set the limit for a specified *<user>*:

```
<user> hard nofile 10024
<user> soft nofile 10024
```

The changes will take effect upon *<user's>* next login.

3.2 Offload target configuration

The OOF runtime uses a virtual file system features to perform tasks related with the offload process and memory management. The runtime expects the */tmp/intel-coi* directory to be mounted, otherwise it will attempt to mount it during initialization and return an error if it is not possible. The *size* parameter of the mounted file system is treated by the offloading runtime as a limit of memory that can be allocated by offloading processes on a target device. Perform the following steps to mount the required file system:

```
[target]# mkdir -p /tmp/intel-coi
[target]# mount -t tmpfs -o size=32g tmpfs /tmp/intel-coi
[target]# chmod 1777 /tmp/intel-coi
```

Add the following line to the */etc/fstab* file to allow all users to mount the file system:

```
tmpfs /tmp/intel-coi tmpfs \
defaults,user,exec,size=32g,mode=1777 0 0
```

To allow the offloading processes to use huge pages, which usually improves performance, mount another virtual file system on the target node:

```
[target]# mkdir -p /tmp/intel-coi/COI2MB
[target]# echo 4000 > /sys/kernel/mm/hugepages/\
hugepages-2048kB/nr_overcommit_hugepages
[target]# mount none -t hugetlbfs /tmp/intel-coi/COI2MB
[target]# chmod 1777 /tmp/intel-coi/COI2MB
```

The following line can be added to the */etc/fstab* to allow all users to mount the *hugetlbfs*:

```
hugetlbfs /tmp/intel-coi/COI2MB hugetlbfs \
defaults,user,mode=1777 0 0
```

The OOF runtime will try to mount the *hugetlbfs* file system if it is not mounted and will use */tmp/intel-coi/COI2MB* as a mount point. It will return an error if the file system cannot be mounted.

It is recommended to increase the maximum number of open files available for offloading processes. Add the following lines to the */etc/security/limits.conf* file to set the limit for a specified *<user>*:

```
<user> hard nofile 10024
<user> soft nofile 10024
```

3.3 Offloading application configuration

3.3.1 Offload targets

Offload over Fabric can use the *OFFLOAD_NODES*, *OFFLOAD_DEVICES*, and *OFFLOAD_NODES_FILE* environment variables to configure nodes available for the offloading operation from the offload host. Each offload host can offload to up to 8 offload nodes (targets).

The *OFFLOAD_NODES* variable contains a comma-separated list of nodes' names (e.g. *phi0,phi1,phi2* etc.). If the targets have more than one network interface active (e.g. *eht0, eth1* and *ib0*), the hostname set in the *OFFLOAD_NODES* variable, must point the IPoIB address.

For example:

The */etc/hosts* contains the IP addresses for every network interface active in the targets:

```
# Node Entry for node: phi01 (ID=363)

192.168.90.111  phi01.localdomain phi01 phi01-eth0.localdomain phi01-eth0
192.168.1.111  phi01-ib0.localdomain phi01-ib0

# Node Entry for node: phi02 (ID=364)

192.168.90.112  phi02.localdomain phi02 phi02-eth0.localdomain phi02-eth0
192.168.1.112  phi02-ib0.localdomain phi02-ib0
```

In this case, the value for the *OFFLOAD_NODES* variable should be: *OFFLOAD_NODES=phi01-ib0,phi02-ib0*

OFFLOAD_DEVICES can be used to configure which nodes (listed in the *OFFLOAD_NODES* variable) will be available for offloading from a particular user process. It contains a comma-separated list of up to 8 indexes of nodes specified in the *OFFLOAD_NODES* variable. If *OFFLOAD_DEVICES* is not set, OOF runtime will try to use all the nodes from *OFFLOAD_NODES*, but it will return an error if the number of nodes is greater than 8.

The *OFFLOAD_NODES_FILE* environment variable points to a file containing description of the desired offload topology. The file should be available on all nodes (e.g. via network file system or by copying it to each machine). Each line in this file corresponds to a system consisting of an offload host and up to 8 targets. The offloading runtime expects a hostname of the offload host at the start of the line followed by a space-separated list of offload target hostnames.

OFFLOAD_NODES_FILE variable has lower priority than *OFFLOAD_NODES* variable and it is not parsed by the offloading runtime if *OFFLOAD_NODES* is set.

For example:

To establish a configuration shown in figure 1.1 user can choose between one of three options. Those options are equivalent, user should choose one that better suits the needs of their particular application.

- Option 1 (figure 3.1) uses the same value of *OFFLOAD_NODES* for every process and specifies offload targets using the *OFFLOAD_DEVICES* variables.
- Option 2 (figure 3.2) uses only *OFFLOAD_NODES* to achieve the same goal (*OFFLOAD_DEVICES* is not set). The *OFFLOAD_NODES* variable is set to a different value for each process.
- Option 3 (figure 3.3) uses *OFFLOAD_NODES_FILE* pointing to */mnt/nfs/topology.txt*, which contains the following lines:

```
xeon0 phi0 phi1
xeon1 phi2 phi3
xeon2 phi4 phi5
```

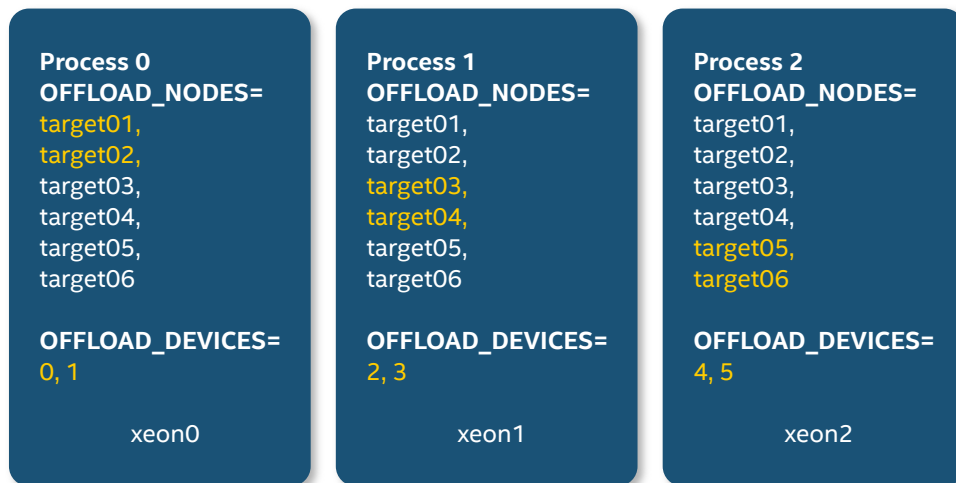


Figure 3.1 – Example OOF configuration - option 1



Figure 3.2 – Example OOF configuration - option 2

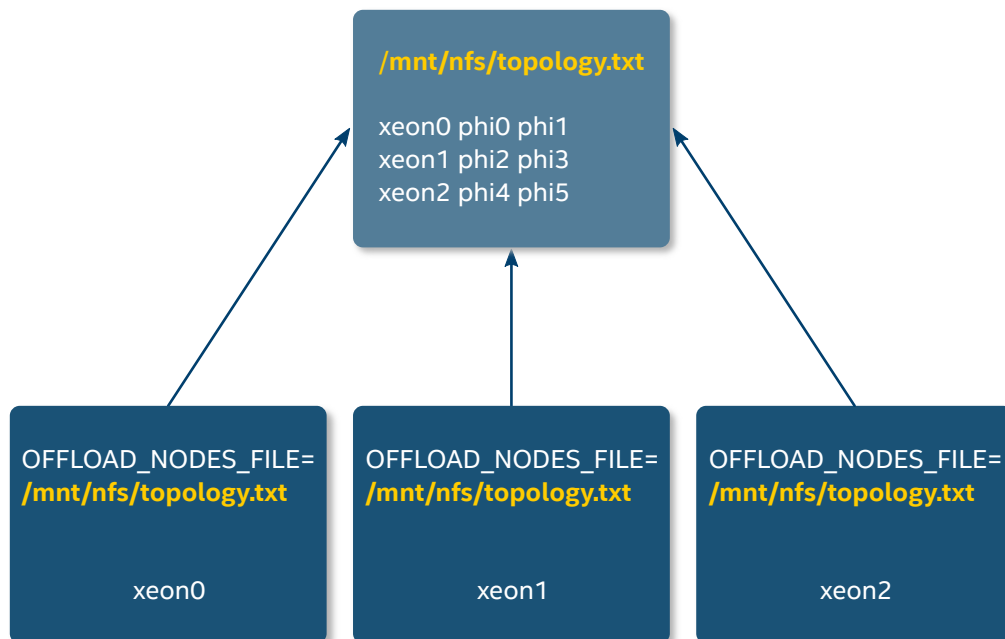


Figure 3.3 – Example OOF configuration - option 3

3.3.2 User authentication

The offloading runtime creates many network connections between the offload host and the offload targets during application lifetime. User authentication mechanisms are used to make sure that only authorized users can execute offloading. The offloading runtime can use one of three mechanisms to authenticate connections between the offload host and the offload targets:

- SSH (Secure Shell),
- MUNGE (MUNGE Uid 'N' Gid Emporium),
- NOAUTH (no authentication).

Note that the data transfer between the offload host and the offload targets is not secured with a cyphering algorithm. Authentication means that the runtime checks if the offloading user is authorized to perform the offloading process.

The SSH authentication mode is the default one and requires no additional configuration. In particular, the offloading runtime automatically starts and stops its services (called daemons) and selects network ports to be used for network connections.

Additional configuration is needed to select and use other authentication mechanisms. The table below contains details about configuration of the offloading runtime in different authentication modes.

Mode name	Additional configuration	Service startup
ssh	Not needed	Automatic
munge	Port	Manual
noauth	Port	Manual

Table 3.1 – Configuring authentication mechanisms

In *MUNGE* or *NOATH* modes user must manually start one daemon on each of the offload targets and configure it with a TCP/IP port number that is free on all offloading targets. The daemons will wait on the configured port for the connections from the offload host. Use the following command line to start the offload daemon:

```
[target]$ coi_daemon --auth-mode=<mode name> --coiport=<port number>
```

On the offload host the offloading runtime must also be configured to use selected authentication mechanism and port. Set the *COI_AUTH_MODE* environment variable to mode name to select an authentication mechanism. If *COI_AUTH_MODE* is not set, *SSH* mode is used. Use the *COI_DAEMON_PORT* environment variable to configure the offloading runtime on the offload host to use selected TCP/IP port number.

MUNGE authentication mechanism requires additional services and libraries to be installed on the offload host and the offload target systems. Information on how to install and configure *MUNGE* is available on the [project's website](#). The offloading runtime will return an error if *MUNGE* mechanism is selected and the required software is not installed.

For example:

Running the commands below will cause the offloading runtime to use the *noauth* authentication mechanism, use port 1338 and run offloading application with offload target *<target>*:

```
[target]$ coi_daemon --auth-mode=noauth --coiport=1338
[host]$ export COI_AUTH_MODE=noauth
[host]$ export COI_DAEMON_PORT=1338
[host]$ export OFFLOAD_NODES=<target>
[host]$ ./offload_app
```

3.3.3 Memory usage policy

The Intel® Xeon Phi™ processor is equipped with 16GB of high-bandwidth memory (MCDRAM). MCDRAM can operate in one of three modes:

- Cache mode – MCDRAM serves as a cache for DDR memory.
- Flat mode – MCDRAM extends the regular address space of DDR memory.

- Hybrid mode – MCDRAM is divided with one part working in cache mode and the other part in flat mode.

Usually, the best performance can be achieved when using multiple threads accessing MCDRAM simultaneously. Go to <http://colfaxresearch.com/knl-mcdram/> to learn more about MCDRAM and different ways to take advantage of its features.

Offloading runtime can be configured to use either MCDRAM or DDR memory for processes on target devices. This is done using the `OFFLOAD_MEM_KIND` environmental variable, which is also used to set fallback mechanism which defines what happens when the selected type of memory is exhausted. The format of the variable is `OFFLOAD_MEM_KIND=<mem_kind>,<fallback>`, where `<mem_kind>` is either `hbw` or `ddr`, and `<fallback>` is the other memory type or `abort`. Table 5 contains possible combinations of options used to configure the offloading runtime.

<code>OFFLOAD_MEM_KIND</code>	Primary memory	Fallback mechanism
Variable not set	MCDRAM	DDR
<code>hbw,ddr</code>	MCDRAM	DDR
<code>ddr,hbw</code>	DDR	MCDRAM
<code>hbw,abort</code>	MCDRAM	Fail when out of MCDRAM
<code>ddr,abort</code>	DDR	Fail when out of DDR
All other combinations fail during initialization of the offload runtime		

Table 3.2 – Configuration of target memory kind

3.3.4 Fabric interfaces

It is possible to set up more than one fabric interface on the offload host or on offload targets. The offloading runtime will automatically select first available interface from a list returned by the operating system kernel. It is possible to configure the offloading runtime to use a specific fabric interface using the `COI_IB_LISTENING_IF_NAME` environmental variable. The variable should be set separately on the offload host and on offload targets.

Example

The offload host and the offload target have two fabric interfaces: `ib0` and `ib1`. The operating system kernel returns `ib0` as the first available fabric interface causing the offloading runtime to select `ib0` by default. To configure the offloading runtime to use `ib1` interface, export `COI_IB_LISTENING_IF_NAME=ib1` on the offload host and offload target.

Note: To make the `COI_IB_LISTENING_IF_NAME` variable available to the offloaded processes on target devices in `ssh` authentication mode, it should be exported in the global shell configuration (e.g. in the `.bashrc` file) of the user who performs offloading.

Chapter 4

Intel® Coprocessor Offload Infrastructure

4.1 Overview

Intel® Coprocessor Offload Infrastructure (Intel® COI) is a module that provides the following functionalities:

- Enumeration of the offloading engine.
- Management of user processes and their dependencies (shared libraries).
- Memory management and data transfers between offload target and host.
- Execution of user (offloaded) code on the target host.
- Management of dependencies between functions executed on the target, buffers used by those functions, and implicit and explicit buffer data transfers.

Intel® COI plays a critical role in the offloading process, as it abstracts most of the details of interconnectivity between the host and target. It implements reach API that can be directly used to perform offloading by the user or by the compiler runtime.

4.2 Directories

By default, Intel® COI is installed in multiple locations on the offload host. These locations include:

Directory	Contents
<i>/usr/share/doc/intel-coi-<version></i>	Documentation, including this document, the Intel® COI API Reference Manual, and release notes.
<i>/usr/include/intel-coi</i>	Include files needed to build Intel® COI applications.
<i>/usr/share/doc/ intel-coi-<version>/tutorials</i>	Simple code samples that can be helpful in learning how to write Intel® COI applications.
<i>/usr/bin</i>	Intel® COI tools to assist in development.
<i>/usr/lib64</i>	Intel® COI shared libraries needed to build Intel® COI applications. Four different versions of each of these libraries are provided with all combinations of host or device and debug or release.
<i>/opt/mpss/x200/minsdk</i>	Compatibility libraries used for building and compiling code that should be executed on target devices. Those libraries have no dependencies and user applications should use them for linking only. It is assumed that full versions of libraries are installed on target systems.

Table 4.1 – Intel® COI directory listing

4.3 Configuration

Intel® COI can be configured using direct calls to the Intel® COI API and a set of environment variables. It uses *COI_OFFLOAD_NODES* and *COI_OFFLOAD_DEVICES* environment variables to configure offload target nodes.

The format and the usage of the *COI_OFFLOAD_NODES* and *COI_OFFLOAD_DEVICES* is exactly the same as the format of *OFFLOAD_NODES* and *OFFLOAD_DEVICES* variables described in section 3.3.1. Check Intel® COI documentation to learn more the configuration details.

4.4 Building and running tutorials

The Intel® COI release comes with a number of simple code tutorials, including the following:

Tutorial	Description
<i>hello_world</i>	Shows an application with no pipelines that uses the Intel® COI I/O proxy to print output on the source. This type of usage can be suitable for users who would like to run a remote application.
<i>coi_simple</i>	Shows the use of a single pipeline and run function.
<i>buffers_with_pipeline_function</i>	Shows simple buffer operations.
<i>multiple_pipeline_implicit</i>	Shows how to use implicit buffer dependencies to coordinate between multiple pipelines.
<i>multiple_pipeline_explicit</i>	Shows how to use explicit dependencies to coordinate between multiple pipelines.
<i>user_event</i>	Shows how to utilize user-created barriers for synchronization between sink and source.
<i>buffer_references</i>	Illustrates the use of buffer reference counting to implement out-of-order asynchronous operations.

Table 4.2 – Intel® COI tutorials

Each tutorial directory contains the source code and makefiles needed to build binaries. The *makefiles* are configured to use *gcc*.

The tutorials can be built by copying a tutorial's entire directory into a user's directory, and issuing the make command:

```
[host]$ cp -r /usr/share/doc/ intel-coi-<version>/tutorials/hello_world .
[host]$ cd hello_world
[host]$ make
```

After building the tutorial, it can be executed to perform offloading to node named target by running the host executable:

```
[host]$ cd debug
[host]$ export COI_OFFLOAD_NODES=target
[host]$ ./hello_world_source_host
```

```
1 engines available
Hello from the sink!
Press enter to kill the sink process
```

```
[host]$
```

4.5 Using *coitrace* to assist with debugging

The *coitrace* tool is included in the installation package. This trace utility functions similarly to Unix*-style tools like *strace* and shows all of the Intel® COI API invocations and input parameters. This can be helpful in identifying what Intel® COI commands are being executed for tracing and debugging. To see the complete list of options run:

```
[host]$ coitrace -h
```

To use it run:

```
[host]$ cointrace <application>
```

For example executing the *hello_world* through *coitrace* will produce the following output:

```
[host]$ coitrace ./hello_world_source_host
```

```
COIEngineGetCount [ThID:0x7f905bcf17c0] = COI_SUCCESS
    in_DeviceType = COI_DEVICE_MIC
    out_pNumEngines = 0x7fff5ae1f180 0x00000001 (hex) : 1 (dec)

1 engines available
COIEngineGetHandle [ThID:0x7f905bcf17c0] = COI_SUCCESS
    in_DeviceType = COI_DEVICE_MIC
    in_EngineIndex = 0x00000000 (hex) : 0 (dec)
    out_pEngineHandle = 0x7fff5ae1f1b0 0x7f905b8632c0

Got engine handle
COIProcessCreateFromMemory [ThID:0x7f905bcf17c0] = COI_SUCCESS
    in_Engine = 0x7f905b8632c0
    in_pBinaryName = hello_world_sink_mic
    in_pBinaryBuffer = 0x7f905bcf9000
    in_BinaryBufferLength = 0x00000000000021d4 (hex) : 8660 (dec)
    in_Argc = 0
    in_ppArgv = 0
    (bool) in_DupEnv = false
    in_ppAdditionalEnv = 0
    (bool) in_ProxyActive = true
    in_Reserved = (nil)
    in_BufferSpace = 0x0000000000000000 (hex) : 0 (dec)
    in_LibrarySearchPath = (nil)
    in_FileOfOrigin = hello_world_sink_mic
    in_FileOfOriginOffset = 0x0000000000000000 (hex) : 0 (dec)
    out_pProcess = 0x7fff5ae1f1a0 0x27af290

COIProcessCreateFromFile [ThID:0x7f905bcf17c0] = COI_SUCCESS
    in_Engine = 0x7f905b8632c0
    in_pBinaryName = hello_world_sink_mic
    in_Argc = 0
    in_ppArgv = 0
    (bool) in_DupEnv = false
    in_ppAdditionalEnv = 0
    (bool) in_ProxyActive = true
    in_Reserved = (nil)
    in_BufferSpace = 0x0000000000000000 (hex) : 0 (dec)
    in_LibrarySearchPath = (nil)
    out_pProcess = 0x7fff5ae1f1a0 0x27af290
```

Sink process created, press enter to destroy it.

Hello from the sink!

```
COIProcessDestroy [ThID:0x7f905bcf17c0] = COI_SUCCESS
    in_Process = 0x27af290
    in_WaitForMainTimeout = -1
    (bool) in_ForceDestroy = false
    out_pProcessReturn = 0x7fff5ae1f170
    Sink process returned 0
    Sink exit reason SHUTDOWN OK
```

4.6 *micnativeloadex* for remote execution

The *micnativeloadex* utility included in the package can be used to remotely execute native code from a host console. This tool works similarly to *ssh*, which can be used to start a remote process, but does not require logging in. *Micnativeloadex* will automatically transfer dependent libraries, and will redirect console IO back to the host console. Internally *micnativeloadex* uses Intel® COI so it follows the same library loading rules and requirements for the *SINK_LD_LIBRARY_PATH* environment variable. *COI_OFFLOAD_NODES* and *COI_OFFLOAD_DEVICES* variables should be used to choose target machine as described in chapter 3.

4.7 Troubleshooting

This section presents several techniques that can be performed to fix or mitigate problems with the Offload over Fabric software. If for some reason following these steps does not resolve the occurring problems, or if some of these steps need to be done consistently, please file a defect or contact your Intel® support representative.

4.7.1 *COIEngineGetHandle* Hangs

COIProcessCreate call may hang for a number of reasons. Initially check whether the Linux* OS on the target system is active and accessible. *SSH* can be used to verify this:

```
[host]$ ssh <target_IP>
```

The user should be able to log into the target system without using password. If this operation was successful, use the *ssh* session to validate whether the virtual file systems required for Intel® COI to run are mounted and accessible to the user (see section 3.2).

You can also verify the connectivity and configuration correctness using OFI tests, which can be downloaded from <https://github.com/ofiwg/fabtests>.

4.7.2 Intel® COI API Returns an error code

Sometimes providing an accurate error code does not clarify the source of a problem. For example, if *COIProcessCreateFromFile* returns *COI_MISSING_DEPENDENCY*, it indicates that a dynamic library needed by the executable could not be found in the host or target file systems. However, if the debug version of the Intel® COI library is used, more information can be learned by looking at the automatically generated log file. This file is named *<executable>.coilog*, where *<executable>* is the name of the source executable. The log file is located in the directory the user was in when the application was launched.