Configuring Intel® Xeon Phi™ coprocessors inside a cluster
Author: Michael Hebenstreit
Contributions: Romain Dolbeau **, Jeremy C. Siadal
Version: 0.81, 20130110

** Much of the work shown here is based on the gcc port to native Intel® Xeon Phi ™ Coprocessor done by Romain Dolbeau at CAPS Enterprise. This work would not have been possible without his initial work and the author wants to express his thanks for the effort.
Contents
Configuring Intel® Xeon Phi™ coprocessors inside a cluster ........................................ 1
Contents ......................................................................................................................... 2
Abstract ....................................................................................................................... 4
Conventions used in this guide ..................................................................................... 5
General remarks .......................................................................................................... 5
Target Configuration ................................................................................................... 6
Installation overview .................................................................................................... 8
Installation on a cluster if /opt/intel/mic is on a shared file system ......................... 11
Installation on non-standard host kernels ................................................................ 12
Modifying the uOS file system .................................................................................... 12
Building uOS from scratch .......................................................................................... 15
Native compiler for Intel Xeon Phi Coprocessor ....................................................... 16
Compiling native GNU tools ....................................................................................... 20
Compiling a kernel natively ......................................................................................... 28
Compiling Lustre ........................................................................................................ 28
Compiling Panasas* panfs* driver ........................................................................... 31
InfiniBand on Intel Xeon Phi coprocessors .............................................................. 32
Direct control of the Intel Xeon Phi coprocessor via /sys files ................................. 35
  Important caveat ....................................................................................................... 36
  Replacing micctrl -s ............................................................................................... 36
  Replacing micctrl -r ............................................................................................... 36
  Replacing micctrl -b ............................................................................................... 37
Starting rsyslogd within the Intel Xeon Phi coprocessor uOS .................................. 39
Starting in a cluster - knmicbootstart_crt.sh ......................................................... 39
Anatomy of the script knmicbootstart_crt.sh ......................................................... 40
Collecting performance statistics .............................................................................. 49
Configuring the cluster to run MPI successfully ........................................................ 49
Debugging native Intel® Xeon Phi™ coprocessor applications with ddd+gdb ............ 54
Debugging offloaded applications using gdb ............................................................ 55
  Background .............................................................................................................. 55
  Requirements .......................................................................................................... 55
Abstract
This paper is intended to provide readers a blueprint of how to set up and configure a cluster with systems containing the Intel® Xeon Phi™ Coprocessor, based on how Intel configured its own Endeavor cluster. Along the way, specific information about how to compile tools, configure filesystems, and setting up network interfaces is shared in great detail to help understand how this can be done en masse.

To satisfy current standard cluster usage models, where users expect to be able to reach every system that is part of an MPI job via a simple password-less ssh command, and find all the filesystems they expect mounted on every node, some key administrative setup must be performed.

The solution proposed in this document covers the following features:

- users access Xeon Phi coprocessors with standard privileges using direct and passwordless ssh
- the home NFS server is mounted, as well as Lustre* and Panasas* shares
- use of bridged networking to avoid routing problems
- automated detection of installed Intel Xeon Phi coprocessors via lspci
- USER accounts added to all Intel Xeon Phi coprocessor cards on the system, but no password is set
- Removal of inetd on the Intel Xeon Phi coprocessors to maximize security
- Correct MTU and NETMASK settings on the Intel Xeon Phi coprocessors Startup of coi_daemon as USER
- Enhancement of dropbear ssh environment with ulimits
- Automated startup of OFED Intel Xeon Phi Coprocessor Communication Link (CCL)
Conventions used in this guide

*Italics* indicate a file or directory.

**COURIER** denotes code.

A single `#` at the beginning of a line denotes commands entered with administrative privileges, `$` indicates standard user rights. `##` is a comment.

**uOS** refers to the Linux* micro Operating System running on an Intel Xeon Phi coprocessor.

[0123..] denotes the presence of multiple entries depending how many Intel Xeon Phi coprocessors are installed. For instance on a host with one coprocessor, `mic[0123..].image` will only expand into a single file `mic0.image`. If four coprocessors are present, correspondingly four files `mic0.image`, `mic1.image` `mic2.image` and `mic3.image` would be present.

Host or compute node denotes a system compatible with Intel64 instruction set that has at least one Intel Xeon Phi coprocessor installed.

In the Intel configuration, we use the path `/opt/crtdc` to store binaries and tools. This, of course, can be different in your environment.

**General remarks**

- The Intel Xeon Phi uOS Linux environment is using by default `busybox`, and its shell is `ash`. Please be aware of its restrictions compared to GNU* bash.
Target Configuration

The cluster configuration used in Intel’s Endeavor cluster is outlined in Figure 1 below.

Users login from their desktops to a dedicated Login server. For administration purposes, a second server (“Admin” in the figure above) can be reached from outside the cluster. This server also contains all necessary services like dhcpd, DNS, tftp, opensm, batch system... Normal users are not allowed access to this machine.

The cluster contains a number of file servers. We assume NFS and Panasas are used via Ethernet. Lustre can be both used via InfiniBand and Ethernet.

The compute nodes are identical and each one is equipped with one Intel Xeon Phi coprocessor labeled mic0. A second Intel Xeon Phi coprocessor, if installed, would be labeled mic1. The nodes are connected to the backbones via Gigabit Ethernet and InfiniBand. Each node has a hard-disk containing the OS.

In this guide we assume the following configuration:

<table>
<thead>
<tr>
<th>Interface</th>
<th>type</th>
<th>Comment</th>
<th>network</th>
<th>netmask</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth1</td>
<td>Ethernet</td>
<td>connect to corporate backbone</td>
<td>A.B.C.D</td>
<td>E.F.G.H</td>
</tr>
<tr>
<td>eth0,mic0</td>
<td>Ethernet</td>
<td>connect to Ethernet backbone of the cluster</td>
<td>10.101.0.0</td>
<td>255.255.0.0</td>
</tr>
<tr>
<td>ib0, mic0:1</td>
<td>iPoIB</td>
<td>connect to iPoIB backbone of the cluster</td>
<td>10.102.0.0</td>
<td>255.255.0.0</td>
</tr>
</tbody>
</table>

We are using a class A network for the cluster to keep the various networks separate. All Ethernet is using 10.101.0.0/16, all iPoIB (IP over InfiniBand) is using 10.102.0./16. The third octet denotes the
cabinet – that’s why all servers have IP addresses 10.X.1.Y, while the compute nodes are in cabinet 3 and use 10.X.3.Y. The complete list of IP addresses used in this example is given below.

The Intel Xeon Phi coprocessors have 2 IP addresses. The first card uses HOST-address + 40, the second HOST-address + 80 (assuming 36 nodes per cabinet). The symbolic names of any Intel Xeon Phi coprocessor are ‘hostname’-mic0 and ‘hostname’-mic1.

The whole network in the cluster is flat – so there is no need for routing at the node level. The only routing implemented in this example is used for accessing the Lustre file server from the Intel Xeon Phi coprocessor using IPoIB (advanced topic, see below).

To implement the flat model we employ the native Linux® bridging mechanism. The standard Ethernet interface is programmed to listen to 2 IP addresses. One belongs to the host system; the other is routed to the virtual Ethernet device on the uOS of the Intel Xeon Phi coprocessor.
Installation overview

To incorporate Intel Xeon Phi coprocessors into the Endeavor cluster (ranked #58 on November 2013 Top500 list, see http://www.top500.org/lists/2012/11) while maintaining security requirements and enabling different usage models, we expanded the basic capabilities of the standard startup scripts. To set up a similar environment please follow these steps:

1) Note: the MTU in this network is generally set to 9000. Please adapt this to your settings.
2) Ensure you have root access to the systems
3) Ensure the various symbolic names are known to all systems in the cluster. In the example configuration, the file /etc/hosts on all nodes contains:

```bash
$ hostname
n01
$ grep `hostname` /etc/hosts
10.101.3.1 n01
10.102.3.1 n01ib0
10.101.3.41 n01-mic0
10.101.3.81 n01-mic1
10.102.3.41 n01-mic0ib0
10.102.3.81 n01-mic1ib0
```

4) Specifically test looking up ALL systems:

```bash
$ gethostip `hostname`-mic0
$ gethostip `hostname`-mic1
```

5) If you want to use Intel Xeon Phi Coprocessor Communication Link (CCL), ensure a recent OFED version is installed. Currently only OFED 1.5.4.1 is supported.

6) Ensure you have a sufficiently recent Intel® Manycore Platform Software Stack (Intel® MPSS) stack installed. Do not forget to flash the card after installing and rebooting the system. At the time of this writing we used the following package versions:

```bash
intel-mic-2.1.4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-flash-0375-16.2.6.32-131.0.15.el6.x86_64
intel-mic-gpl-2.1.4346-16.el6.x86_64
intel-mic-knc-kmod-2.1.4346-16.2.6.32.131.el6.x86_64
intel-mic-knc-kmod-2.1.4346-16.el6
intel-mic-llapi-2.1.4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-ofed-card-4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-ofed-ibpd-4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-ofed-kmod-4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-ofed-libibscif-4346-16.2.6.32-131.0.15.el6.x86_64
intel-mic-sysmgmt-2.1.4346-16.2.6.32-131.0.15.el6.x86_64
```

7) For installation on a system with kernels not directly supported see section Installation on non-standard host kernels.

8) The default installation automatically loads the mic.ko kernel module and starts the mpss/ofed-mic services. If you, do not want this behavior, switch off the services and remove /etc/sysconfig/modules/mic.modules
# chkconfig --del ofed-mic
# chkconfig --del mpss
# /bin/rm /etc/sysconfig/modules/mic.modules

9) Be aware that kernel modules depending on OFED modules will need to be recompiled. You will need the correct intel-mic-ofed-kmod-devel-VER.x86_64.rpm. The header files are installed into /usr/src/intel-mic-ofed-kmod/. For instance Lustre build (for the HOST, configuring Lustre for the Intel Xeon Phi coprocessor uOS see section Compiling Lustre) might be invoked as:

```
./configure --with-o2ib=/usr/src/intel-mic-ofed-kmod/ --with-linux=/lib/modules/2.6.32-131.0.15.el6.x86_64.crt.1/build/ --disable-server
```

10) Create a basic configuration for the Intel Xeon Phi coprocessors: micctrl --initdefault

11) Adapt the uOS system to your needs – see section Modifying the uOS file system.

12) Prepare the host system for bridged networking. Bridged networking means that the OS creates a virtual interface – the bridge – and attaches various interfaces to the same hardware port (an example for bridged interfaces is given below). With this setup the Intel Xeon Phi coprocessor will effectively listen on the standard Ethernet port. On Redhat* 6.1:

   a. Create a file /etc/sysconfig/network-scripts/ifcfg-br0 (file attached in this document) configuring the bridge br0:
      
      ```
      # cat /etc/sysconfig/network-scripts/ifcfg-br0
      DEVICE=br0
      TYPE=Bridge
      ONBOOT=yes
      DELAY=0
      NM_CONTROLLED="no"
      MTU=9000
      BOOTPROTO=dhcp
      NOZEROCONF=yes
      ```

   b. Modify the file /etc/sysconfig/network-scripts/ifcfg-eth0 (file attached in this document) to use the bridge:
      
      ```
      # cat /etc/sysconfig/network-scripts/ifcfg-eth0
      DEVICE=eth0
      ONBOOT=yes
      BRIDGE=br0
      MTU=9000
      ```

c. Reboot the system; ensure that the old address given to eth0: is now owned by br0:

13) Configure network and bridged networking in of the Intel Xeon Phi coprocessor via /etc/sysconfig/mic/default.conf. Specifically the parameters BridgeName and Subnet have to match.

```
# cat /etc/sysconfig/mic/default.conf
...
## Bridge names starting with "mic" will be created by the MPSS daemon. Other bridges are
## assumed to already exist.
BridgeName br0
```
### Define the first 2 quads of the network address.
### Static pair configurations will fill in the second 2 quads by default. The individual MIC configuration files can override the defaults with MicIPaddress and HostIPaddress.

Subnet 10.101

14) Check /root/.ssh directory. You need a correct RSA key-pair, and /root/.ssh/authorized_keys needs to contain id_rsa.pub

15) Initialize basic configuration: 

```
# micctrl --initdefaults
```

16) Check /etc/hosts for unwarranted changes

17) Check that the native config files match your intended configuration

```
# cat /opt/intel/mic/filesystem/mic0/etc/sysconfig/network/ifcfg
IPADDR=10.101.3.41
NETMASKBITS=16
HWADDR=0a:70:45:b4:41:97
GATEWAY=10.101.1.254
# cat /opt/intel/mic/filesystem/mic1/etc/sysconfig/network/ifcfg
IPADDR=10.101.3.81
NETMASKBITS=16
HWADDR=42:93:0e:5a:ea:e8
GATEWAY=36.101.1.254
```

18) If you plan to use an imaging system, uncomment the line HWADDR (or ensure it's different on every system)

19) Start the Intel Xeon Phi coprocessor in the normal way

```
# micctrl -r
# micctrl -w
# micctrl -s
# micctrl -b
# micctrl -w
# micctrl -s
```

20) Test that you have access to the Intel Xeon Phi coprocessor as root and that you can mount NFS shares. If access fails, your /root/.ssh directory might not be set up correctly.

```
# ssh `hostname`-mic0 pwd
/root
```

21) Set up the native build environment for the Intel Xeon Phi coprocessor. This is detailed in [Compiling native GNU tools](#).

22) On the Intel Xeon Phi coprocessor uOS, mount the NFS share /opt/crdtc/micgnu containing all native tools. In this example installation the data are actually kept in /home/MIC/VERSION, and /opt/crdtc/micgnu is only a link. On the uOS execute:
mount -o vers=3,nolock 36.101.233.11:/home /home
mkdir -p /opt/crtdc
ln -s /home/MIC/3552-1 /opt/crtdc/micgnu

23) ssh to the Intel Xeon Phi coprocessor and build the complete native toolset and additional Cluster file system drivers as outlined in Compiling native GNU tools, Compiling a Kernel natively, Compiling Lustre and Compiling Panasas panfs driver.

24) Once everything is done, reboot the host system, revalidate it is fit for use. Start the mpss service and verify the Intel Xeon Phi coprocessor is working correctly. Then reset it again:

```
# service mpss start
# micctrl -s
mic0: online (mode: linux image: /lib/firmware/mic/uos.img)
mic1: online (mode: linux image: /lib/firmware/mic/uos.img)
# micctrl -r
# micctrl -w
# micctrl -s
mic0: ready
mic1: ready
```

25) Start the script as root with a USERNAME as parameter. USERNAME must be known to the system. This user will be the only one allowed to access the Intel Xeon Phi coprocessor:

```
# sh kncmicbootstart_crt.sh USERNAME
```

26) As the user, connect to the Intel Xeon Phi coprocessor via ssh (HOSTNAME is not literal):

```
$ ssh HOSTNAME-mic0
```

27) The user’s ssh setup must be configured correctly. If asked for a password when trying to login to the Intel Xeon Phi coprocessor ask the user to:

```
$ cd ~/.ssh
$ test -f id_rsa || echo "missing id_rsa, run ssh-keygen -d rsa"
$ cat id_rsa.pub >> authorized_keys
```

28) On the Intel Xeon Phi coprocessor, execute ps and check that the coi_daemon is running under the account USERNAME. This can be done as user or root.

```
$ ssh n01-mic0 ps | grep student01
6417 student01 0:00 sh -c /bin/coi_daemon
6419 student01 0:00 /bin/coi_daemon
7055 root 0:00 sshd: student01 [priv]
7057 student01 0:05 sshd: student01@pts/0
7058 student01 0:00 -sh
```

**Installation on a cluster if /opt/intel/mic is on a shared file system**

At the time of this writing, it’s not possible to relocate the Intel® MPSS. Installation will always go into the /opt/intel/mic directory. In a cluster /opt or /opt/intel often are shared, as they contain data
identical on all nodes (like the Intel compilers that are part of Intel(R) Parallel Studio XE 2013). This can create unique problems, especially when trying to install different mpss versions in parallel on the cluster.

As a workaround, create local directories on each node, and replace /opt/intel/mic with a symbolic link. This has to be done before installation of the rpms. Also be aware that removal of such rpms from one system could remove the original link /opt/intel/mic and therefore make the installation on other systems unusable.

```
# mkdir /usr/local/mic
# ln -s /usr/local/mic /opt/intel/mic
```

**Installation on non-standard host kernels**

Note: VER is a shortcut for version and takes different values for the various rpms.

a) Ensure host kernel headers are installed on the build system.
b) Uninstall any previous intel-mic* rpms
c) Install source rpms:
   
   ```
   # rpm -i intel-mic-knc-kmod-VER.src.rpm intel-mic-ofed-kmod-VER.src.rpm
   ```
d) Rebuild the host driver sources:
   
   ```
   $ rpmbuild -ba intel-mic-knc-kmod.spec
   ```
   
   Note: building host-kernel modules when a intel-mic-knc-kmod-*.rpm is already installed leads to an error as the wrong header files are included.
e) Install basic packages:
   
   ```
   ```
f) Note: the package intel-mic contains the host driver, but also adds symbol and header information to the Linux source directory.
   
   `/lib/modules/`uname -r`/build/include/scif.h`
   `/lib/modules/`uname -r`/build/Module.symvers.mic`
   
   If this directory is on a cluster-wide mounted file system, parallel installation can lead to locking issues
g) Rebuild the Intel Xeon Phi CCL stack (build depends on already installed host driver):
   
   ```
   rpmbuild -ba intel-mic-ofed-kmod.spec
   ```
h) Install the Intel Xeon Phi CCL rpm:
   
   ```
   # rpm -i intel-mic-ofed-kmod-3552.1-1.el6.x86_64.rpm
   ```
i) Reboot the HOST system

**Modifying the uOS file system**

The uOS running on the Intel Xeon Phi coprocessor consists of 2 files. The kernel is found in `/lib/firmware/mic/uos.img`, and an image for every processor is located as `/opt/intel/mic/filesystem/mic[0123...].image`. 
The image files are compressed cpio archives and are created by the mpss daemon from the directories in /opt/intel/mic/filesystem/. The configuration files in /etc/sysconfig/mic list exactly what directories to include:

```
$ cat /etc/sysconfig/mic/default.conf
...
## Source for base of embedded Linux file system
BaseDir /opt/intel/mic/filesystem/base
   /opt/intel/mic/filesystem/base.filelist

## MIC card unique overlay files such as etc, etc.
CommonDir /opt/intel/mic/filesystem/common
   /opt/intel/mic/filesystem/common.filelist
...
## Include all additional functionality configuration files by default
Include "conf.d/*.conf"
```

In an installation including OFED and Intel Xeon Phi CCL this adds 2 more files

```
$ cat /etc/sysconfig/mic/conf.d/*
## COI download files
Overlay /opt/intel/mic/coi /opt/intel/mic/coi/config/coi.filelist
## OFED download files
Overlay /opt/intel/mic/ofed/card
   /opt/intel/mic/ofed/card/ofed.filelist
```

The initial configuration created by micctrl --initdefault adds one more directory and filelist for EVERY Intel Xeon Phi coprocessor present in the system. The contents of /opt/intel/mic/filesystem/ on a host with 2 coprocessors then become (directories are blue):

```
[filesystem]$ ls
   base  base.filelist  common  common.filelist  mic0  mic0.filelist
   mic0.image  mic1  mic1.filelist  mic1.image
```

The various *.filelist contain all the files and directories that are to be created on the uOS. Taking a look at

```
$ cat mic0.filelist
   file /etc/passwd etc/passwd 664 0 0
   file /etc/shadow etc/shadow 000 0 0
   dir /root 700 0 0
   file /root/.profile root/.profile 644 0 0
   ...
```

Each line corresponds to either a file or directory. In case of a file the entries read as:

```
   keyword destination origin permission UserID GroupID
```

In case of a directory:
keyword destination permission UserID GroupID

origin is relative to /opt/intel/mic/filesystem/DIRECTORY. The line

file /etc/passwd etc/passwd 664 0 0

dir /root 700 0 0

file /root/.profile root/.profile 644 0 0

file /etc/group etc/group 644 0 0

dir /root/.ssh 700 0 0

dir /root/.ssh 700 0 0

file /root/.ssh/authorized_keys root/.ssh/authorized_keys 600 0 0

dir /root/.ssh 700 0 0

dir /sbin 755 0 0

file /sbin/.profile sbin/.profile 644 0 0

dir /etc/sysconfig 775 0 0

file /etc/sysconfig/hostname etc/sysconfig/hostname 664 0 0

file /etc/hosts etc/hosts 644 0 0

dir /etc/sysconfig 775 0 0

dir /etc/sysconfig/network 775 0 0

file /etc/sysconfig/network/ifcfg-mic0 etc/sysconfig/network/ifcfg-mic0 664 0 0

When the mpssd daemon receives a boot request from an Intel Xeon Phi coprocessor, it will parse the various *.filelist files, starting from base, going to common, overlaying the coi and ofed files, and finally adding the entries unique to each coprocessor. mpssd will then go through the final list, create a file
system and store this as `/opt/intel/mic/filesystem/mic[0123..].image`. These files are gzip'ed cpio archives and are transferred to the coprocessor during boot.

Every time a `micctrl -b` is issued these image files are rebuilt.

**Building uOS from scratch**

The Intel Xeon Phi coprocessor uOS is available as open source and comes in form of a file `package-full_src-k1om.tar.bz2`. Unpacking it yields

```
ls -a
.arch .ver.linux Makefile cross host intel-mic-gpl.sh package-root ras vcons vnet .ver.build .ver.scif card dma include intel-mic-gpl.spec pm_scif trace_capture version.mk
```

Take note of the hidden files, they are necessary for a clean build. The uOS sources are in the subdirectory `card/kernel`. Certain parts are kept separate, like the SCIF driver in `card/driver`.

```
Makefile micscif.ko micscif_debug.c micscif_intr.c micscif_nodeqp.c micscif_rma.c micscif_select.c micscif_va_gen.c Module.symvers micscif.mod.c micscif_fd.c micscif_main.c micscif_ports.c micscif_rma_dma.c micscif_smpt.c micscif_va_node.c include micscif_api.c micscif_gtt.c micscif_nm.c micscif_rb.c micscif_rma_list.c micscif_sysfs.c modules.order
```

You might find it necessary to modify part of the uOS. The author for instance wanted to remove a restriction on the coi_daemon. Specifically we wanted the daemon to run with normal user privileges, but the SCI kernel modules requires root privileges to bind to a socket <1024.

The restriction was enforced in the file `micscif_api.c`, line 347ff:

```c
/*
 * SCIF ports below SCIF_ADMIN_PORT_END can only be bound by
 * system (or root) processes or by processes executed by
 * privileged users.
 */
if (!capable(CAP_SYS_ADMIN) && pn < SCIF_ADMIN_PORT_END) {
    ret = -EACCES;
    goto scif_bind_exit;
}
```

Commenting out those 4 lines was the only change necessary. The build has to happen on a system that has the intel-mic* rpms installed, as the cross compilers are needed. To check that all modules build, do a

```
make MICARCH=k1om card-kmods
```
in the top level directory. At the time of this writing the boot sequence did not support initramfs/initrd, so the modules and the kernel have to be linked together into a single image file. This is done via

```
make MICARCH=klom miclinux
```

and will create a file `card/kernel/arch/x86/boot/bzImage`. This file will be loaded by the `mic.ko` host driver after issuing a `micctrl -b` command. The location of the file is configured in `/etc/sysconfig/mic/mic[0123…].conf` via the `Osimage` flag:

```
$ grep OSimage /etc/sysconfig/mic/mic0.conf
OSimage /lib/firmware/mic/uos.img
```

So the easiest way is to copy the new image to the new location and reboot the Intel Xeon Phi coprocessor.

```
cp card/kernel/arch/x86/boot/bzImage /lib/firmware/mic/uos.img
micctrl -r
micctrl -w
micctrl -b
```

**Native compiler for Intel Xeon Phi Coprocessor**

In this section we show how to cross compile the gcc compiler and set up a native gcc version that will run on the Intel Xeon Phi coprocessor. This will allow compiling tools and kernel modules natively. There are a lot of workarounds and quick hacks to make compilation possible. This not a recipe how to create a complete Linux distribution for Intel Xeon Phi coprocessors, but rather a procedure for how to quickly set up a minimal environment that can be easily adapted to special needs.

- All steps needed to build the native gcc compiler are executed on the HOST.
- This setup uses `/opt/crtdc/micgnu` as the destination directory for a complete native environment. This directory will be mounted via NFS on the coprocessor.
- To enable a simple switch between versions we store the actual data in `/home/MIC/VERSION` and create `/opt/crtdc/micgnu` as a symbolic link.
- It’s possible to re-use the native `binutils` already installed with the Intel MPSS stack in `/opt/crtdc/micgnu/linux-k1om-4.7`. For some installations this is even required, as there are some changes to `ar` and `objdump` that cause problems. In this installation both versions are therefore installed in parallel.
- This procedure uses 2 `binutil` stacks – the one deployed with the mpss stack and part of the `package-full_src-k1om.tar.bz2` file, and `binutils-2.22.tar.gz` from [www.gnu.org](http://www.gnu.org).
- You will also need the sources for gnu-make, for instance make-3.81.tar.gz from [www.gnu.org](http://www.gnu.org).
- 3 source files from `package-full_src-k1om.tar.bz2` are needed: `binutils.tar.xz`, `gcc.tar.xz`, and `glibc.tar.xz`.
- **PITFALL**: in some instances `gcc.tar.x` and `binutils.tar.x` as distributed by Intel do NOT expand into a directory called `gcc/binutils`.  

- All 5 source files should be placed in a common directory such as $SOURCES. You also need a special patch file `gcc.patch` provided in the zip-archive that is distributed along with this document.
- You will need sources for gmp, mpc and mpfr to build gcc. This example uses the versions available from Redhat* Enterprise Linux 6.1 (gmp-4.3.1, mpc-0.8.1, mpfr-2.4.1), but you are welcome to use newer versions. These files need to be uncompressed in the $SOURCES directory.
- You need to have a standard build environment with gcc installed as well as the intel-mic-* packages installed.
- Some steps in this process only work if a single process is compiling the binaries. Parallel builds using `make -j NUMBER` will fail in these steps.

```bash
##### define version and source directory
MPSSVERSION=3552-1
SOURCES=/opt/crtdc/micgnu/src/patched_and_ready/
#####
SGNU=${SOURCES}/
SREDHAT=${SOURCES}/
SINTEL=${SOURCES}/

##### temporary working directory
WORK=`mktemp -d`
##### where is the sysroot
MICSYSROOT=/opt/crtdc/micgnu/linux-k1om-4.7/linux-k1om
##### what is the target (likely different between KNF and KNC)
MICTARGET=x86_64-k1om-linux
##### where do you want the cross & native binaries + binutils
MICCROSS=/opt/crtdc/micgnu/cross
MICNATIVE=/opt/crtdc/micgnu
STDBINUTILS=/opt/crtdc/micgnu/binutils-2.22
##### where is the GCC patch
GCCPATCH=${SOURCES}/gcc.patch
##### where are the required sources
GMPSRC=${SREDHAT}/gmp-4.3.1
MPCSRC=${SGNU}/mpc-0.8.1
MPFRSRC=${SREDHAT}/mpfr-2.4.1
ls $GMPSRC
ls $MPCSRC
ls $MPFRSRC

## copy over precompile utilities to have everything in one place
cp -r /usr/linux-k1om-4.7 /opt/crtdc/micgnu/linux-k1om-4.7

##### rebuild recent, native binutils
cd ${WORK}
tar xzf ${SGNU}/binutils-2.22.tar.gz
mkdir binutils-2.22-build
cd binutils-2.22-build
```
./binutils-2.22/configure --prefix=$STDBINUTILS --host=x86_64-redhat-linux --build=x86_64-redhat-linux --target=x86_64-redhat-linux --program-prefix=x86_64-redhat-linux

echo "XXX configure binutils-2.22-build done"
make -j 20
test $? -ne 0 && exit 0
echo "XXX build binutils-2.22-build done"
make install
test $? -ne 0 && exit 0
cd ../

##### rebuild cross-binutils
mkdir binutils-alpha9
cd binutils-alpha9
xzcat ${SINTEL}/binutils.tar.xz | tar xf -
mv binutils binutils-cross
mkdir binutils-cross-build
cd binutils-cross-build

./binutils-cross/configure --prefix=$MICCROSS --host=x86_64-redhat-linux --build=x86_64-redhat-linux --target=$MICTARGET --with-sysroot=$MICSYSROOT

echo "XXX configure binutils-cross-build done"
make -j 20
test $? -ne 0 && exit 0
echo "XXX build binutils-alpha9-build done"
make install
test $? -ne 0 && exit 0
cd ../

##### rebuild cross-gcc
xzcat ${SINTEL}/gcc.tar.xz | tar xf -
mv gcc gcc-cross
cd gcc-cross
patch -p1 < $GCCPATCH
ln -s $GMPSRC gmp
ln -s $MPFRSRC mpfr
ln -s $MPCSRC mpc
cd ../
mkdir gcc-cross-build
cd gcc-cross-build

PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin .. /gcc-cross/configure --prefix=$MICCROSS --host=x86_64-redhat-linux --build=x86_64-redhat-linux --target=$MICTARGET --enable-languages=c,c++,fortran --with-sysroot=$MICSYSROOT --disable-multilib

echo "XXX configure gcc-alpha9-build done"
# parallel build fails here for some reason...
(export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make)
test $? -ne 0 && exit 0
echo "XXX build gcc-cross-build done"
(export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make install)
test $? -ne 0 && exit 0
cd ..

##### build native-gcc
# we re-use the same sources
mkdir gcc-native-build
cd gcc-native-build
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; ..gcc- 
cross/configure --prefix=$MICNATIVE --host=$MICTARGET --build=x86_64-redhat-linux --target=$MICTARGET --enable-languages=c,c++,fortran --with-sysroot=$MICSYSROOT --disable-multilib)
echo "XXX configure gcc-native-build done"
# parallel build fails here for some reason...
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make)
test $? -ne 0 && exit 0
echo "XXX build gcc-native-build done"
(export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make install)
test $? -ne 0 && exit 0
cd ..

##### build native-binutils
## we re-use the same sources
mkdir binutils-native-build
cd binutils-native-build
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; ..binutils- 
cross/configure --prefix=$MICNATIVE --host=$MICTARGET --build=x86_64-redhat-linux --target=$MICTARGET --with-sysroot=$MICSYSROOT 
)
echo "XXX configure binutils-native-build done"
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make -j 20 )
test $? -ne 0 && exit 0
echo "XXX build binutils-native-build done"
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make install )
test $? -ne 0 && exit 0
cd ..

##### cross-build native GNU make
tar xzf ${SOURCES}/make-3.81.tar.gz
cd make-3.81
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; ./configure -- 
prefix=$MICNATIVE --build=x86_64-redhat-linux --host=$MICTARGET )
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make )
( export PATH=$MICCROSS/bin:$PATH:$STDBINUTILS/bin; make install )

As a final step we consolidate the two directories /opt/crtdc/micgnu/lib and /opt/crtdc/micgnu/lib64.
All files in the lib directory were moved to the lib64, and lib replaced by a link.

ls -l /opt/crtdc/micgnu/
total 100
drw-r-xr-x 2 mic mic 12288 Aug 10 10:12 bin
drw-rwxr-xr-x 8 mic mic 4096 Jul 5 13:09 binutils-2.22
Compiling native GNU tools

In this section we demonstrate how to create a minimal GNU tools environment that allows native applications to be built on the Intel Xeon Phi coprocessor uOS. Currently this includes the following software:

Python autoconf bison byacc elfutils expat file findutils flex gawk gdbm gettext glib libffi ltrace m4 ncurses ntp openssh openssl perl pkg-config readline rsyslog strace tcl8.5.7 zlib

- All steps needed to build the native environment are executed on the uOS of the Intel Xeon Phi coprocessor.
- This setup uses /opt/crtdc/micgnu as the destination directory for a complete native environment. This directory has to be mounted via NFS inside the uOS.
- To enable a simple switch between versions we store the actual data in /home/MIC/VERSION and create /opt/crtdc/micgnu as a symbolic link.
- There are a lot of workarounds and quick hacks to make compilation possible. This not a recipe on how to create a complete Linux distribution for Intel Xeon Phi coprocessors, but rather a process on how to quickly set up a minimal environment that can be easily adapted to special needs.
- For Intel MPSS version 3552-* it is necessary to re-compile ncurses and zlib. Both are part of a standard mpss stack, but had been compiled without PIC support.
- The current selection of software was mostly influenced by the goal to be able to compile native kernel modules, specifically Lustre and Panasas panfs.
- A lot of workarounds could have better solutions. As the author is not very familiar with autoconf/automake, any feedback how to improve this procedure is welcome.
- The changes are usually implemented via a sed or patch command. In the latter case a comment describes what is happening, and it should not be necessary to do the changes described manually.
- In some cases you might get problems with ar, ranlib and similar tools. In such cases it helped to specifically use the binutils provided by Intel using a configure line like:

```
./configure RANLIB=/opt/crtdc/micgnu/x86_64-k1om-
    linux/bin/ranlib AR=/opt/crtdc/micgnu/x86_64-k1om-
    linux/bin/ar --prefix=/opt/crtdc/micgnu/ --build=x86_64-
    k1om-linux ...
```

export PATH=/opt/crtdc/micgnu/bin:$PATH
export LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib64/
SRC=/opt/crtdc/micgnu/src/patched_and_ready
echo $SRC
ls $SRC
umask 022

tar xzf $SRC/pkg-config-0.23.tar.gz
cd pkg-config-0.23/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd ..

tar xzf $SRC/gettext-0.17.tar.gz
cd gettext-0.17/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
   without-included-gettext --enable-nls --disable-static --enable-shared
   --with-pic=yes --disable-csharp --disable-java --disable-native-java
   --disable-rpath --without-libncurses-prefix
## edit Makefile LDFLAGS to: "LDFLAGS = ../intl/.libs/printf.o"
sed -e 's,LDFLAGS =,LDFLAGS = ../intl/.libs/printf.o,' -i gettext-
   tools/tests/Makefile
make
make install
cd ..

tar xzf $SRC/libffi-3.0.5.tar.gz
cd libffi-3.0.5/
cp $SRC/unix64.S src/x86/unix64.S
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
   CCASFLAGS=-DHAVE_AS_X86_PCREL
make
make install
cd ..

tar xzf $SRC/m4-1.4.13.tar.gz
cd m4-1.4.13/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd ..

ls $SRC
tar xzf $SRC/zlib-1.2.3.tar.gz
cd zlib-1.2.3/
./configure --shared
## remove @ around #if HAVE_UNISTD_H   file: zconf.h
patch -p0 < $SRC/zlib.patch
make test
make install prefix=/opt/crtdc/micgnu/
cd ..

ls $SRC
tar xzf $SRC/ncurses-5.7.tar.gz
cd ncurses-5.7/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux --with-shared --without-ada --with-ospeed=unsigned --enable-hardtabs --enable-xmc-glitch --enable-overwrite --enable-pc-files --with-chtype=long --with-termlib=tinfo
make
make install
cd ..

ts $SRC
tar xzjf $SRC/readline-6.0.tar.gz
cd readline-6.0/
  ./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
cd ..

ts $SRC
tar xzjf $SRC/flex-2.5.35.tar.gz
cd flex-2.5.35/
  ./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
cd ..

ts $SRC
tar xzjf $SRC/bison-2.4.1.tar.gz
cd bison-2.4.1/
  ./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
cd ..

ts $SRC
tar xzjf $SRC/file-5.04.tar.gz
cd file-5.04/
  ./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
cd ..

ts $SRC
tar xzjf $SRC/glib-2.22.5.tar.gz
cd glib-2.22.5/
  ./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
cd ..

## note - removed patches 2 and 13 from Redhat spec file
##%patch2 -p1 -b .droproot
##%patch13 -p1 -b .editline
tar xzjf $SRC/ntp-4.2.4p8.tar.gz
cd ntp-4.2.4p8/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
mkdir /opt/crtdc/micgnu/etc
cp -r $SRC/etc/ntp $SRC/etc/ntp.conf /opt/crtdc/micgnu/etc
echo "YOU NEED TO CONFIGURE $SRC/ntp $SRC/ntp.conf!"
cd ..

tar xzf $SRC/gdbm-1.8.0.tgz
cd gdbm-1.8.0/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd ..

ls $SRC
tar xzf $SRC/findutils-4.4.2.tgz
cd findutils-4.4.2
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd ..

tar xzf $SRC/strace-4.5.19.tar.gz
cd strace-4.5.19/
patch -p0 < $SRC/strace.patch
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd..

tar xzf $SRC/elfutils-0.152.tgz
cd elfutils-0.152/
## fix in configure - incompatibility to ash
patch -p0 < $SRC/elfutils.patch
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
CFLAGS="-fexceptions -I/opt/crtdc/micgnu/include/
LDFLAGS="-L/opt/crtdc/micgnu/lib64"
make
make install
cd ..

tar xzf $SRC/ltrace-0.5.tgz
cd ltrace-0.5/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
CFLAGS="-D_LARGEFILE64_SOURCE -I/opt/crtdc/micgnu/include/
LDFLAGS="-L/opt/crtdc/micgnu/lib64"
cd sysdeps/linux-gnu
ln -s x86_64 k1om
cd ..
cp $SRC/ltrace-k1om_arch.h sysdeps/linux-gnu/k1om/arch.h
make
make install
cd ..

tar xzf $SRC/gawk-3.1.7.tgz
cd gawk-3.1.7/
./configure AR=/opt/crtdc/micgnu/x86_64-k1om-linux/bin/ar --
   prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make check
make
install
cd ..

tar xzf $SRC/tcl-8.5.7.tar.gz
cd tcl8.5.7/
cd unix/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make --disable-threads --enable-symbols --enable-shared
make
make install
cd ..../

tar xzf $SRC/byacc-1.9.20070509.tar.gz
cd byacc-20070509/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux --
   host=x86_64-k1om-linux
make
make install
cd ..

tar xzf $SRC/expat-2.0.1.tar.gz
cd expat-2.0.1/
./configure AR=/opt/crtdc/micgnu/x86_64-k1om-linux/bin/ar --
   prefix=/opt/crtdc/micgnu/ --build=x86_64-k1om-linux
make
make install
cd ..

tar xzf $SRC/perl-5.10.1.tgz
cd perl-5.10.1
cp $SRC/perl/*.sh .
patch -p1 < $SRC/perl/perl.patch
./Configure -d -Dgcc -Dld=gcc -Doptimize="-O2" -DDEBUGGING=-g -
   Dversion=5.10.1 -Dmyhostname=localhost -Dperladmin=root@localhost
   -Duseshrplib -Dusethreads -Duseithreads -Duselargefiles -
   Dd_dosuid -Dd_semctl1_semun -Dd_db -Dud_ndbm -Dd_gdbm -Dd_shadow -
   Dd_syslog -Dman3ext=3pm -Duseperlio -Dinstallusrbinperl=n -
   Ubicompatability=5005 -Dversiononly -Dpager="/usr/bin/less -I" -
   Dd_gethostent_r_proto -Dd_endservent_r_proto -
   Dd_setprotoent_r_proto -Dd_endprotoent_r_proto -
   Dd_sethostent_r_proto -Dd_endhostent_r_proto -
   Dd_setservent_r_proto -Dd_endservent_r_proto -
   Dd_setprotoent_r_proto -Dd_endprotoent_r_proto -

## perl changes

### in config.h

"#define MULTIPLICITY" is commented out, change it to:

```c
ifndef MULTIPLICITY
#define MULTIPLICITY / **/
endif
```

change CLDFLAGS in Makefile

```make
CLDFLAGS = --fstack-protector -lm -lrt
```

### Makefile: remove "lib/auto/IO/Compress/Compress.so" from "dynamic_ext="

### change ext/Compress-Raw-Zlib/Makefile

```make
CCFLAGS = -I/opt/crtdc/micgnu/include/gdbm - I/opt/crtdc/micgnu/include -D_REENTRANT ....
```

### change ext/GDBM_File/Makefile

```make
CCFLAGS = -I/opt/crtdc/micgnu/include/gdbm - I/opt/crtdc/micgnu/include -D_REENTRANT ....
```

```
patch -p1 < $SRC/perl/perl2.patch
make
make test
make install
cd ..
```

### note: removed patch 0 from Redhat spec file

```
%patch0 -p1 -b .erlang
tar xzf $SRC/autoconf-2.63.tar.gz
cd autoconf-2.63/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-kлом-linux
make
make install
cd ..
```

```
tar xzf $SRC/bash-4.1.tar.gz
cd bash-4.1/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-kлом-linux -- with-bash-malloc=no
make
make install
cd ..
```

```
tar xzf $SRC/openssl-1.0.0.tgz
cd openssl-1.0.0/
which perl
## edit path to perl
sed -e "s,#!/usr/bin/perl.*,#!`which perl`," -i Configure
./Configure --prefix=/opt/crtdc/micgnu/ zlib enable-camellia
   enable-seed enable-tlsext enable-rfc3779 enable-cms enable-md2
```
## comment out lines 96/97 to disable SSE
##  //#define COMPILE_HW_AESNI
##  //static ENGINE *ENGINE_aesni (void);
## in crypto/engine/eng_aesni.c
patch -p0 < $SRC/openssl_1.patch

# in Makefile add -I/opt/crtdc/micgnu/include to CFLAG
sed -e 's,^CFLAG= -fPIC,CFLAG= -I/opt/crtdc/micgnu/include -fPIC,,' -i Makefile

make depend
make
make install
cd ..

tar xzf $SRC/openssh-5.3p1.tgz
cd openssh-5.3p1
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux --with-zlib=/opt/crtdc/micgnu/
make
make install
cd ..

##start a new shell for python compile
bash << EOD
tar xzf $SRC/Python-2.6.6.tar.gz
cd Python-2.6.6
topdir=`pwd`
export CFLAGS="-D_GNU_SOURCE -fPIC -fwrapv"
export CXXFLAGS="-D_GNU_SOURCE -fPIC -fwrapv"
export CPPFLAGS="pkg-config --cflags-only-I libffi`"
export OPT="-D_GNU_SOURCE -fPIC -fwrapv"
export LINKCC="gcc"
export CFLAGS="$CFLAGS `pkg-config --cflags openssl`"
export LDFLAGS="$LDFLAGS `pkg-config --libs-only-L openssl`"
export CC=gcc
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux --enable-shared --with-system-ffi
patch -p1 < $SRC/python.patch
make
mkdir ./Lib/test/NOT
mv ./Lib/test/test_file.py ./Lib/test/NOT
make test
make install
EOD
cd ..
tar xzf $SRC/rsyslog-5.8.10.tgz
cd rsyslog-5.8.10/
export CFLAGS=-I/opt/crtdc/micgnu/include
export LDFLAGS=-L/opt/crtdc/micgnu/lib64
./configure AR=/opt/crtdc/micgnu/x86_64
make
make install
cd ..
ls $SRC
tar xzf $SRC/libhugetlbfs-2.8.tar.gz
cd libhugetlbfs-2.8/
patch -p0 < $SRC/libhugetlbs.patch
sed -e 's,^PREFIX = /usr/local,PREFIX = /opt/crtdc/micgnu,\' -e 's,^EXEDIR = /bin,EXEDIR = /opt/crtdc/micgnu/bin,\' -i Makefile
make CC=gcc
sed -e "s,#!/usr/bin/env python.*,#!/\`which python`,\" -i ./tests/run_tests.py
echo "run as root on the MIC card:
echo " export PATH=/opt/crtdc/micgnu/bin:$PATH"
echo " export LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib64/"
echo " `pwd`/../obj/hugeadm --pool-pages-min 2MB:0"
echo " `pwd`/../obj/hugeadm --pool-pages-max 2MB:4096"
echo " mkdir /tmp/hugepages"
echo " mount none /tmp/hugepages -t hugetlbfs -o mode=0777"
echo "waiting, press ENTER to continue"
read A
make check
make install
cd ..
tar xzf $SRC/tcsh-6.17.00.tgz
cd tcsh-6.17.00
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
patch -p0 < $SRC/tcsh.patch
make
make install
cd ..
tar xzf $SRC/pdksh-5.2.14.tgz
cd pdksh-5.2.14/
./configure --prefix=/opt/crtdc/micgnu/ --build=x86_64-klom-linux
make
make install
 ( cd /opt/crtdc/micgnu//bin; mv ksh pdksh; ln -s pdksh ksh )
 ( cd /opt/crtdc/micgnu//man/man1/; mv ksh.1 pdksh.1; ln -s pdksh.1 ksh.1 )
cd ..
Compiling a kernel natively
This step is necessary to compile kernel modules like Lustre. We need the header files, and the module
`crc32.ko` which is not compiled into the uOS. First check to see if the native environment works as expected.

All commands are executed within the Intel Xeon Phi coprocessor uOS.

```
export PATH=/opt/crtdc/micgnu/bin:$PATH
export LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib64/

## untar the package_full_src to /opt/crtdc/micgnu/src
cd /opt/crtdc/micgnu/src
export MICARCH=k1om
make defconfig-

## login to KNC card
cd /opt/crtdc/micgnu/src/card/kernel
export PATH=/opt/crtdc/micgnu/x86_64-k1om-linux/bin/:$PATH
cd arch
ln -s x86 k1om
cd ..
make
##Fix create include files links compatible with Lustre
cd include/linux
ln -s ../generated/autoconf.h .
ln -s ../generated/utsrelease.h .
```

Compiling Lustre
Starting from a standard Lustre 1.8.5, building it was fairly straightforward. At the moment it’s not possible to use Lustre over InfiniBand, as the uOS Intel Xeon Phi Coprocessor Communication Link (CCL) kernel driver has no support for `rdm_cm`. Therefore Lustre is only supported using TCP/IP.

- This setup uses `/opt/crtdc/micgnu` as destination directory for a complete native environment.
- To enable a simple switch between versions we store the actual data in `/home/MIC/VERSION` and create `/opt/crtdc/micgnu` as a symbolic link.
- There are a lot of workarounds and quick hacks to make compilation possible. This not a recipe on how to create a complete Linux distribution for Intel Xeon Phi coprocessors, but rather a procedure on how to quickly set up a minimal environment that can be easily adapted to special needs.
- A lot of workarounds probably have better solutions. As the author is not very familiar with `autoconf/automake`, any feedback how to improve this procedure is welcome.
All commands are executed within the Intel Xeon Phi coprocessor uOS.

```bash
export PATH=/opt/crtdc/micgnu/x86_64-kom-linux/bin:/$PATH
export LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib64/
SRC=/opt/crtdc/micgnu/src/patched_and_ready
mkdir -p /opt/crtdc/micgnu/lib/modules/
cp /opt/crtdc/micgnu/src/card/kernel/lib/crc32.ko
  /opt/crtdc/micgnu/lib/modules/
tar xzf $SRC/lustre-1.8.5.tar.gz
cd lustre-1.8.5
patch -p0 < $SRC/lustre_A.patch
sed -e 's,-Werror,' -i .lustre/autoconf/lustre-core.m4
sed -e 's,-Werror"","" -i configure
sed -e 's,-Werror,"" -i .lustre.spec.in
sed -e 's,-Werror,"" -i ./libsysio/configure
sed -e 's,-Werror"" -i ./lustre.spec
./configure
CFLAGS="-g -O2 -I/opt/crtdc/micgnu/include/" LDFLAGS=-lpthread
--with-linux=/opt/crtdc/micgnu/src/card/kernel/ --build=x86_64-kom-linux
--without-o2ib --disable-server
build_alias=x86_64-kom-linux

sed -e 's,AR=/usr/bin/ar,AR=/opt/crtdc/micgnu/x86_64-kom-linux/bin/ar," -i ./lustre/liblustre/genlib.sh
sed -e 's,AR=/usr/bin/ar,AR=/opt/crtdc/micgnu/x86_64-kom-linux/bin/ar," -i ./lnet/utils/genlib.sh
sed -e 's,LD=/usr/bin/ld,LD=/opt/crtdc/micgnu/x86_64-kom-linux/lib/ld," -i ./lnet/utils/genlib.sh
sed -e 's,RANLIB=/usr/bin/ranlib,RANLIB=/opt/crtdc/micgnu/x86_64-kom-linux/bin/ranlib," -i ./lustre/liblustre/genlib.sh
sed -e 's,RANLIB=/usr/bin/ranlib,RANLIB=/opt/crtdc/micgnu/x86_64-kom-linux/bin/ranlib," -i ./lnet/utils/genlib.sh
make

for I in llite_lloop.ko lov.ko lquota.ko lustre.ko lvfs.ko mdc.ko
  mgc.ko obdclass.ko obdecho.ko osc.ko ptlrpc.ko ko2ib1nd.ko
  ksocklnd.ko libcfs.ko lnet.ko lnet_selftest.ko
  do
  find . -name "$I"
    find . -name "$I" -exec cp {} /opt/crtdc/micgnu/lib/modules/ \;
  done

for I in liblustre.a liblustre.so liblustreapi.a libpt1ctl.a
  do
```
find . -name "$I" -exec cp {} /opt/crtdc/micgnu/lib64 \;
done

mkdir -p /opt/crtdc/micgnu/lib64/lustre/snmp
find . -name lustresnmp.so -exec cp {} /opt/crtdc/micgnu/lib64/lustre/snmp \;

mkdir /opt/crtdc/micgnu/libexec/lustre/
find . -name lc_common -exec cp {} /opt/crtdc/micgnu/libexec/lustre/ \;

for I in mount.lustre config.sh gather_stats_everywhere.sh ior-survey lfs lfs_migrate libecho llobdstat llstat lstats.sh lustre_req_history mcreate munlink obdfilter-survey ost-survey parse-ior plot-llstat plot-obdfilter plot-ost plot-sgpd l sgpd- survey debugctl 1_getgroups lc_cluman lc_hb lc_lvm lc_md lc_modprobe lc_net lc_servip lctl ll_decode_filter_fid ll_recover_lost_found_objs llog_reader llverdev llverfs loadgen lr_reader lrun lshowmount lst lstclient ltrack_stats lustre_config lustre_createcsv lustre_rmmmod lustre_start lustre_up14 mkfs.lustre ptmlctl routerstat tunefs.lustre wirecheck wiretest
do
find . -name "$I" -exec cp {} /opt/crtdc/micgnu/bin \;
done

cd ..
Compiling Panasas* panfs* driver

This module does not come from Open source. You’ll need to get sources directly from Panasas, and then the author can provide you with the cons program necessary to compile it. The author is also in contact with Panasas to get this new architecture directly supported.

All commands are executed within the Intel Xeon Phi coprocessor uOS.

tar xzf $SRC/panfs_client_4.1.1-641366.223.tar.gz
cp -r $SRC/panfs_client-overide/* panfs_client
cp $SRC/cons .
cd panfs_client
chmod -R u+w .
mkdir -p import/linux/rhel_6_k1om/kernel/
ln -s /opt/crtdc/micgnu/src/card/kernel/
import/linux/rhel_6_k1om/kernel/`uname -r`

## note: ensure you have write privs to /etc/ on the Intel Xeon Phi
echo "Red Hat Enterprise Linux Server release 6.1 (Santiago)" >
/etc/redhat-release
## ensure you have a correct /etc/hosts
grep `hostname` /etc/hosts

sed -e 's,awk -f roken.awk,/opt/crtdc/micgnu/bin/awk -f roken.awk,,'
-i ./src/heimdal/Conscript
sed -e "s,my \$nm = 'nm';,my \$nm = '/opt/crtdc/micgnu/x86_64-k1om-
linux/bin/nm';," -i
src/panfs/build/pan_gen_linux_inline_config.pl
sed -e 's,ld -m elf_k1om,/opt/crtdc/micgnu/x86_64-k1om-linux/bin/ld
-m elf_k1om,' -i Construct

../cons -cd LINUX-WEAK-MODULE=off
rhel_6_k1om/release/releng/spool/panfs-`uname -r`.rpm

for I in mount.panfs panfs_quota check_panfs config_panfs kcollect
  panfs_stat panfs_trace panfs_trace_manage trpost
do
  echo $I
  find ./rhel_6_k1om/release/export -name $I
  find ./rhel_6_k1om/release/export -name $I -exec cp {}
  /opt/crtdc/micgnu/bin \;
done

mkdir /opt/crtdc/micgnu/lib/modules/
for I in panfs.ko
do
  find ./rhel_6_k1om/release/export -name $I
  find ./rhel_6_k1om/release/export -name $I -exec cp {} /opt/crtdc/micgnu/lib/modules/ \;
done

cd ..
InfiniBand on Intel Xeon Phi coprocessors

Intel Xeon Phi CCL implementation currently only supports Mellanox* HCA adaptor. It also requires OFED 1.5.4.1 to be installed, other installations MIGHT work but are not tested. After installing the RPMs first ensure that InfiniBand is running and the `ibstat` is showing “Physical State: Linkup”:

```
# ibstat
CA 'mlx4_0'
   CA type: MT4099
   Number of ports: 1
   Firmware version: 2.10.700
   Hardware version: 0
   Node GUID: 0x0002c903002f0ae0
   System image GUID: 0x0002c903002f0ae3
   Port 1:
       State: Active
       Physical state: LinkUp
       Rate: 56
       Base lid: 506
       LMC: 0
       SM lid: 1
       Capability mask: 0x02514868
       Port GUID: 0x0002c903002f0ae1
       Link layer: InfiniBand
```

Next, ensure the Intel Xeon Phi coprocessor is running and start the `ofed-mic` service:

```
# micctrl -s
mic0: online (mode: linux image: /lib/firmware/mic/uos.img)
# service ofed-mic start
Starting OFED/MIC Stack:
esg019-mic0 [ OK ]
host [ OK ]
```

On the Intel Xeon Phi coprocessor two InfiniBand devices will be set up. The scif0 interface is used for HOST to uOS communication, while the mlx4_0 interface is a virtual representation of the InfiniBand HCA installed on the HOST system.

```
# ibv_devinfo  # executed on the Intel Xeon Phi coprocessor
hca_id: mlx4_0
    transport: InfiniBand (0)
    fw_ver: 2.10.700
    node_guid: 0002:c903:002f:18e0
    sys_image_guid: 0002:c903:002f:18e3
    vendor_id: 0x02c9
    vendor_part_id: 4099
    hw_ver: 0x0
    phys_port_cnt: 1
        port: 1
            state: PORT_ACTIVE (4)
```
To test InfiniBand speed between 2 separate Intel Xeon Phi coprocessors use the tools `ib_read_bw`, `ib_read_lat`, `ib_write_bw` and `ib_write_lat`. These tools measure bandwidth and latency for RDMA read and write calls.

All 4 programs operate in a similar way. You start on one system the program without any options except for the device to be used, and it will act as server. From the second Intel Xeon Phi coprocessor you issue the command again, this time giving the name or IP address of the first Intel Xeon Phi coprocessor as parameter.

Both invocations require the option “-d mlx4_0” to ensure the correct device is used. Let’s demonstrate an example measuring read-bandwidth using the corresponding program `ib_read_bw`:

1. Start on first Intel Xeon Phi Coprocessor:

   ```
   # hostname
esg019-mic0
   # ib_read_bw -d mlx4_0
   ```

   RDMA_Read BW Test
   -----------------------------
   Number of qps : 1
   Connection type : RC
   CQ Moderation : 50
   Mtu : 2048B
   Link type : IB
   Outstand reads : 16
2. **Start on second Intel Xeon Phi Coprocessor:**

   
   ```
   # hostname
esg020-mic0
   # ib_read_bw -d mlx4_0 esg019-mic0
   ```

   
   **RDMA_Read BW Test**
   
<table>
<thead>
<tr>
<th>Number of qps</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection type</td>
<td>RC</td>
</tr>
<tr>
<td>TX depth</td>
<td>300</td>
</tr>
<tr>
<td>CQ Moderation</td>
<td>50</td>
</tr>
<tr>
<td>Mtu</td>
<td>2048B</td>
</tr>
<tr>
<td>Link type</td>
<td>IB</td>
</tr>
<tr>
<td>Outstand reads</td>
<td>16</td>
</tr>
<tr>
<td>rdma_cm QPs</td>
<td>OFF</td>
</tr>
<tr>
<td>Data ex. method</td>
<td>Ethernet</td>
</tr>
</tbody>
</table>

   
   local address: LID 0x154 QPN 0x8005f PSN 0xb7fa4e OUT 0x10 RKey 0xb80030e9 VAddr 0x007fe40bce7000
   
   remote address: LID 0x15c QPN 0x005f PSN 0x9238e6 OUT 0x10 RKey 0xb80030e9 VAddr 0x007fe40bce7000

   
   **#bytes   #iterations   BW peak[MB/sec]   BW average[MB/sec]**
   | 65536    | 1000         | 970.71          | 970.58          |

3. **The command on the first Intel Xeon Phi Coprocessor has now finished as well adding information about the remote system:**

   
   remote address: LID 0x154 QPN 0x8005f PSN 0xb7fa4e OUT 0x10 RKey 0x500030c8 VAddr 0x007fe40bce7000
Direct control of the Intel Xeon Phi coprocessor via /sys files
State, reset and boot of the Intel Xeon Phi coprocessor can be obtained and modified from the Linux /sys file system. For each coprocessor installed the mic.ko host kernel driver creates a directory /sys/class/mic/mic[0123...]

These directories contain a number of files:

# ls -l
total 0
-r--r--r-- 1 root root 4096 Jul 24 10:37 active_cores
-rw-r--r-- 1 root root 4096 Jul 20 10:20 cmdline
-r--r--r-- 1 root root 4096 Jul 24 10:37 dev
-rw-r--r-- 1 root root 4096 Jul 24 10:37 dpc3_enabled
-r--r--r-- 1 root root 4096 Jul 24 10:37 extended_family
-rw-r--r-- 1 root root 4096 Jul 24 10:37 extended_model
-r--r--r-- 1 root root 4096 Jul 20 10:20 family
-r--r--r-- 1 root root 4096 Jul 24 10:37 family_data
-r--r--r-- 1 root root 4096 Jul 24 10:37 flashversion
-r--r--r-- 1 root root 4096 Jul 24 10:37 fuse_config_rev
-r--r--r-- 1 root root 4096 Jul 23 10:38 image
-rw-r--r-- 1 root root 4096 Jul 20 10:20 initramfs
-r--r--r-- 1 root root 4096 Jul 24 10:37 kernel_cmdline
-r--r--r-- 1 root root 4096 Jul 24 10:37 meminfo
-r--r--r-- 1 root root 4096 Jul 24 10:37 memoryfrequency
-r--r--r-- 1 root root 4096 Jul 24 10:37 memoryvoltage
-r--r--r-- 1 root root 4096 Jul 24 10:37 memsize
-r--r--r-- 1 root root 4096 Jul 23 10:38 mode
-r--r--r-- 1 root root 4096 Jul 24 10:37 model
lrwxrwxrwx 1 root root 0 Jul 24 10:37 pci_82:00.00 -> ../../../pci0000:80/0000:80:02.0/0000:82:00.0
-r--r--r-- 1 root root 4096 Jul 24 10:37 platform
-r--r--r-- 1 root root 4096 Jul 24 10:37 post_code
drwxr-xr-x 2 root root 0 Jul 24 10:37 power
-r--r--r-- 1 root root 4096 Jul 24 10:37 processor
-r--r--r-- 1 root root 4096 Jul 24 10:37 scif_status
-r--r--r-- 1 root root 4096 Jul 24 10:37 sku
-rw-r--r-- 1 root root 4096 Jul 20 10:20 state
-r--r--r-- 1 root root 4096 Jul 24 10:37 stepping
-rw-r--r-- 1 root root 4096 Jul 24 10:37 stepping_data
-r--r--r-- 1 root root 4096 Jul 24 10:37 substepping_data
lrwxrwxrwx 1 root root 0 Jul 20 10:20 subsystem -> ../../../../class/mic
-rw-r--r-- 1 root root 4096 Jul 20 10:20 uevent

These provide hardware details of the Intel Xeon Phi coprocessor: family, extended_family, extended_model, active_cores, model, meminfo, memoryfrequency, memoryvoltage, memsize, sku, stepping, stepping_data, substepping_data
*flashversion* lists the revision of the firmware installed on the Intel Xeon Phi coprocessor. *dev* and *uevent* are the major and minor device numbers. A link like *pci_82:00.00* points to the location in the PCI device tree and helps identify cards if multiple Intel Xeon Phi coprocessors are installed.

*initramfs* is writable and contains the location of the initial ramdisk (support not yet implemented). *image* shows the location of the system image booted by the uOS.

*post_code* contains the Power-On-Self-Test result from the Intel Xeon Phi coprocessor. It should read FF after a successful reset.

*cmdline* is writable and contains boot options to the uOS that can be changed. The HOST-side kernel driver will add automatically some options like memory addresses used for HOST <-> uOS communication.

*state* contains the current state of the Intel Xeon Phi coprocessor. It also allows resetting or booting the Intel Xeon Phi Coprocessor by writing to it.

*kernel_cmdline* contains the command line actually used to boot the uOS. Some parts can be modified via *cmdline*, some are automatically added.

*scif_status* gives information on the state of the SCI subsystem, the main interface to exchange data between the HOST and the uOS.

Using *cmdline*, *state* and *image* it is possible to control the Intel Xeon Phi coprocessor without resorting to the *micctrl* utility.

**Important caveat**
While you can replace the micctrl utility with direct commands, it is necessary that the Intel MPSS daemon is running! Without the daemon running the Intel Xeon Phi coprocessor will fail.

**Replacing micctrl -s**

Status information can be found from the state file.

```bash
# cat /sys/class/mic/mic0/state
online
```

Currently 4 states are possible.

- **resetting**: Intel Xeon Phi processor is currently undergoing a reset (cause: *micctrl -r*)
- **ready**: Intel Xeon Phi processor has undergone a reset and is ready to boot
- **booting**: Intel Xeon Phi processor is currently undergoing a boot (cause: *micctrl -b*)
- **online**: Intel Xeon Phi processor has undergone a boot and is ready for use

**Replacing micctrl -r**

At any time an Intel Xeon Phi coprocessor can be forced to reset by writing “reset” to the *state* file. The *state* file will then change to “resetting”.
# echo reset > /sys/class/mic/mic0/state
# cat /sys/class/mic/mic0/state
resetting

Replacing micctrl -b

If the Intel Xeon Phi coprocessor is in state “ready” it can be booted. If you want to change any boot options first set up the cmdline (and once it is implemented initramfs).

1) Check state and reset the Intel Xeon Phi coprocessor if necessary:

```bash
cat /sys/class/mic/mic0/state
online
# echo reset > /sys/class/mic/mic0/state
# cat /sys/class/mic/mic0/state
resetting
```

2) Wait some time (normally less than 30s, but can take several minutes; check dmesg for more information on the progress)

```bash
[root@esg019 mic0]# cat /sys/class/mic/mic0/state
ready
```

3) Display and modify current cmdline settings:

```bash
# cat /sys/class/mic/mic0/cmdline
quiet console=hvc0 highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_on
# echo "quiet console=hvc0 highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_on
icache_snoop_off" > /sys/class/mic/mic0/cmdline
# cat /sys/class/mic/mic0/cmdline
quiet console=hvc0 highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_on
icache_snoop_off
```

4) Boot the Intel Xeon Phi coprocessor:

```bash
# echo "boot:linux:/lib/firmware/mic/uos.img" >
/sys/class/mic/mic0/state
# cat /sys/class/mic/mic0/state
booting
```

5) Wait some time (normally less than 90s, but can take several minutes; check dmesg for more information on the progress)

```bash
# cat /sys/class/mic/mic0/state
online
```

6) Connect to the Intel Xeon Phi coprocessor and check the kernel command line:

```bash
# ssh `hostname`-mic0
```
esg019-mic0# cat /proc/cmdline
 card=0 vnet=dma scif_id=1 scif_addr=0x90194ebe00
 vcons_hdr_addr=0x901b78eec0 mem=8192M p2p=1 reg_cache=1
 huge_page=1 quiet console=hvc0 highres=off
 micpm=cpufreq_on;corec6_off;pc3_on;pc6_on
 icache_snoop_off

7) Compare with the output from the `kernel_cmdline` file:

    # cat /sys/class/mic/mic0/kernel_cmdline
    card=0 vnet=dma scif_id=1 scif_addr=0x90194ebe00
    vcons_hdr_addr=0x901b78eec0 mem=8192M p2p=1 reg_cache=1
    huge_page=1 quiet console=hvc0 highres=off
    micpm=cpufreq_on;corec6_off;pc3_on;pc6_on
    icache_snoop_off

Note how the HOST mic.ko driver expanded the original kernel command line. The entries “card=0” to “mem=8192M” are automatically generated. The options “p2p=1”, “reg_cache=1” and “huge_page=1” are configured as parameters of the HOST mic.ko driver via /etc/modprobe.d/mic.conf. The remaining entries are each taken from the /sys/class/mic/mic0/cmdline file.

$ cat /etc/modprobe.d/mic.conf
## Options for the Intel Many Integrated Core Co-processor card driver
##
## p2p enables the use of the SCIF interface peer to peer communication
## 1 to enable or 0 to disable
## reg_cache enables SCIF Registration Caching
## 1 to enable or 0 to disable
## huge_page enables SCIF Huge Page Support
## 1 to enable or 0 to disable
## watchdog enables the SCIF watchdog for Lost Node detection.
## 1 to enable or 0 to disable

options mic reg_cache=1 huge_page=1 watchdog=1 p2p=1
Starting rsyslogd within the Intel Xeon Phi coprocessor uOS
Configuration is done in /opt/crtdc/micgnu/etc/rsyslog.conf.

    # mkdir /var/run/
    # /opt/crtdc/micgnu/sbin/rsyslogd -f
    # /opt/crtdc/micgnu/etc/rsyslog.conf
    # klogd

Starting in a cluster - kncmicbootstart_crt.sh
To start Intel Xeon Phi coprocessors inside a clustered environment we developed a script
kncmicbootstart_crt.sh. This script should be executed from a batch system (LSF, PBS...) during
the prologue phase of a job on all nodes allocated to a job which have Intel Xeon Phi coprocessors
installed.

kncmicbootstart_crt.sh takes at minimum one parameter, the owner of the job. The script will
correctly start the Intel Xeon Phi coprocessors, mount file systems inside the uOS and ensure only the
owner has access to Intel Xeon Phi coprocessors. By putting all necessary changes into this script, we
ensured a re-installation of the drivers would have no impact.

The script can also be started under the name kncmicbootstop_crt.sh, which should be done
during the epilogue phase of a job. In this case the script will only reset the Intel Xeon Phi coprocessors.

NOTE: this is only ONE option how to setup, prepare and start Intel Xeon Phi coprocessors within a
cluster. We found this procedure to be adequate to our requirements and hope it is helpful to someone
in a similar situation.
Anatomy of the script knmicbootstart_crt.sh

Note that at some points a comment

    ## inform the batch system something is wrong

is a placeholder for actual code: in this example, the comment should be replaced with code to tell the batch system about errors, as this will be different from cluster to cluster. The complete script is also available below. For better understanding some code has been removed or re-ordered.

Configuration variables

```
#!/bin/bash
DEBUG=1            # script provides some debug infos
verbose="-v"      # verbose="-v" for ssh verbosity
SYSFS=/sys/class/mic  # location of sysfs files
MICIFNAME=mic      # name of network interfaces
HOSTROOT=/usr/local/mic  # where are the local MIC files
SHELL=/bin/sh   # standard shell for users
MICOS="2.6.34-g0483f82"  # uOS version
BRIDGE=br0        # name of Ethernet bridge
MTU=9000          # network MTU
NETMASK=255.255.0.0 # netmask used
IPoIBNW="36.102.0.0/16"  # IP over InfiniBand network
NTPSERVER=36.101.201.1  # NTP server
```

Automatic detect hostname and number of Intel Xeon Phi coprocessors

```
HOST=`/bin/hostname`
NUMMICCARDS=`/sbin/lspci | /bin/grep -c "Coprocessor: Intel Corporation Device"`
```

Default settings for the user environment on the Intel Xeon Phi coprocessor

```
STACK_SIZE=8192
CORE_FILE_SIZE=0
LOCKED_MEMORY=4194304
PROCESSES=61314
FILE_DESCRIPTOR=1024
COI=y
```

```
MIC_ISNOOPOFF=0  # 0/1
MIC_ECCOFF=0     # 0/1
MIC_HUGEPAGEOFF=0 # 0/1
MIC_REGCACHEOFF=0 # 0/1
MIC_PSM="N"      # N/Y
MIC_P2P=0        # 0/1
```
• Parsing of input parameters. After removing all known switches the first parameter is the user name who'll be allowed to access the Intel Xeon Phi coprocessor:

```bash
while [ -n "$1" ]
  do
      case $1 in
        "-s" ) STACK_SIZE=$2;;
        "-c" ) CORE_FILE_SIZE=$2;;
        "-l" ) LOCKED_MEMORY=$2;;
        "-p" ) PROCESSES=$2;;
        "-n" ) FILE_DESCRIPTOR=$2;;
        *     ) break ;;
      esac
      shift 2
  done
USER=$1
```

• Define check-state – a function to read the state of an Intel Xeon Phi coprocessor. The function is called as:

```
Check_State XeonPhiProcessor Timeout Error ExpectedState
```

If the target processor does not reach the expected state after `Timeout` seconds, the script will terminate with exit code `Error`.

```
Check_State()
{
  Counter=0
  MyCard=$1
  shift
  Timeout=$1
  shift
  MyError=$1
  shift
  ExpectedState=""$*"

  while test "$Counter" -lt "$Timeout"
    do
        echo $ExpectedState | grep -q -w `cat ${SYSFS}/${MyCard}/state`
        && return 0
        Counter=`expr "$Counter" + 1`
        echo -n "$Counter "
        sleep 1
    done
  echo "Card $MyCard not reacting quickly enough, code $MyError, exiting"
  exit $MyError
}
```

• Run basic checks. The script must be executed by root

```
if [ ! `whoami` = "root" ]
```
then
    echo "Error: must be running as root. Exiting..."
    exit 1
fi

if [ "${NUMMICCARDS}" = 0 ]
then
    echo "no Intel Xeon Phi card found, exiting"
    # inform the batch system something is wrong
    exit 2
fi

- check that the mpss service is running – if not, start it

service mpss status || service mpss start
service mpss status
if [ $? -gt 0 ]
then
    echo "Error: mpss not running. Exiting ... "
    # inform the batch system something is wrong
    exit 7
fi

- get list of Intel Xeon Phi coprocessors – the number of devices is already available. The device names will be mic0, mic1... . For each device an IP-address mapped to 'hostname'-mic[01234..] has to be configured in the system. The utility gethostip utilizes all configured name resolution techniques (/etc/hosts, NIS, DNS...) and will find the correct IP-address. If no IP-address can be found, the scripts exit with error code 7. A list of device names is stored in ${MICCARDS}.

MICCARDS=""
I=0
for J in `seq ${NUMMICCARDS}`
do
    MICCARDS="${MICCARDS} ${MICIFNAME}$I"
    MICIPNAME="${HOST}-${MICIFNAME}$I"
    MICIP=`gethostip -d ${MICIPNAME}`
    if [ -z "${MICIP}" ]; then echo "no IP address for ${MICIPNAME} found in /etc/hosts, exiting";exit 7; fi
    I=$(($I + 1 ))
done
echo "Intel Xeon Phi cards:${MICCARDS}"

- Ensure the Intel® Xeon Phi™ Coprocessor Communication Link (CCL) stack is stopped (If MIC OFED is not installed, comment it out):

/usr/sbin/micibstop

- check Ethernet BRIDGE interface is ready

if [ ! -d /sys/class/net/${BRIDGE} ]
then
  echo "Bridge-device /sys/class/net/${BRIDGE} not found, exiting"
  # inform the batch system something is wrong
  exit 1
fi

- Ensure the Intel Xeon Phi coprocessor(s) is /are in a clean state, otherwise issue a reset command. Configure the virtual Ethernet interface of the Intel Xeon Phi coprocessor with the IP-address found earlier. Also ensure the hardware address is commented out – on a cluster build with an imaging system you might otherwise end up with duplicate MAC addresses.

  for CARD in ${MICCARDS}
  do
    Check_State $CARD 60 180 ready online
  done
  for CARD in ${MICCARDS}
  do
    test `cat ${SYSFS}/${CARD}/state` = ready || micctrl -r ${CARD}
    MICIP=`gethostip -d ${HOST}-${CARD}`
    sed -e "s,IPADDR=.*,IPADDR=${MICIP}," -i
    ${HOSTROOT}/filesystem/${CARD}/etc/sysconfig/network/ifcfg-mic0
    sed -e "s,`HWADDR=,#HWADDR=," -i
    ${HOSTROOT}/filesystem/${CARD}/etc/sysconfig/network/ifcfg-mic0
    echo "HOSTNAME=${HOST}-${CARD}" >
    /opt/intel/mic/filesystem/${CARD}/etc/sysconfig/hostname
  done

- Check the state of the Intel Xeon Phi coprocessor again, as the previous step might have issued a reset command.

  for CARD in ${MICCARDS}
  do
    Check_State $CARD 180 101 ready
  done

  ERROR=0
  for CARD in ${MICCARDS}
  do
    test `cat ${SYSFS}/${CARD}/state` = ready || ERROR=1
  done
  if [ "${ERROR}" -gt 0 ]
  then
    # inform the batch system something is wrong
    exit 1
  fi

- Is this the stop script called in epilogue of a batch job? If yes, exit

  [ `basename $0` = "kncmicbootstop_crt.sh" ] && exit 0

- Otherwise more checks are necessary, e.g. the user must be known to the system
if [ -z "${USER}" ]
then
  echo "usage: $0 USER"
  exit 3
fi

## check the user is known to the host system
/usr/bin/id $USER >/dev/null 2>/dev/null
if [ $? != 0 ]
then
  echo "error: no user $USER known to system" && exit 3
fi

- Create an entry for the /etc/passwd file of the uOS.

  MUID=`/usr/bin/id -u $USER`
  MGID=`/usr/bin/id -g $USER`

  ##this is the password entry created on the MIC
  PASS="${USER}:x:${MUID}:${MGID}:/:home/$USER:${SHELL}"

- Ensure we have routing/forwarding enabled

  sysctl -w net.ipv4.ip_forward=1

- Now boot the cards

  for CARD in ${MICCARDS}
  do
    micctrl -b ${CARD}
  done

  for CARD in ${MICCARDS}
  do
    Check_State $CARD 90 102 online
  done

- report host driver version info

  echo "$0: host:"
  echo " `cat ${SYSFS}/ctrl/version`"
  echo " SCIF: `cat ${SYSFS}/ctrl/host_revision`"
  echo " Linux: `cat ${SYSFS}/ctrl/uos_revision` (`uname -r`)"

- test network connection to the card is working, then test the correct uOS is running, and finally
  print version of the Intel Xeon Phi coprocessor

  for CARD in ${MICCARDS}
  do
    # testing access to card
    MICIP=`gethostip -d ${HOST}-${CARD}`
declare -i lapse=0
while ! ping -q -c 1 ${MICIP} >/dev/null; do
  printf "\r%d: " ${lapse}
  if [ "$((lapse++))" -ge 50 ]; then
    echo card ${HOST}-${CARD} is not ready after $lapse seconds.
    exit 6
  fi
  sleep 1
done

declare -i lapse=0
while [ ! "$\(\$MICCMD\) $\(\$MICIP\) \(\text{uname -r}\)" = "\$MICOS" ]
do
  printf "\r%d: " ${lapse}
  if [ "$((lapse++))" -ge 50 ]; then
    echo card ${HOST}-${CARD} is not ready after $lapse seconds.
    exit 6
  fi
  sleep 1
done

# print card version
echo -n " ${CARD}:
${MICCMD} $\\$verbose$MICIP /\text{etc/mic-version | sed 's,^, ,'
echo
done

• start the Intel Xeon Phi CCL OFED stack (if MIC OFED is not installed, comment it out) and test network again:

/usr/sbin/micibstart
echo $\\$MICCARDS
for CARD in $\\$MICCARDS
do
  Check_State $\\$CARD 60 104 online
  declare -i lapse=0
  while [ ! "$\(\$MICCMD\) $\(\$MICIP\) \(\text{uname -r}\)" = "\$MICOS" ]
do
    printf "\r%d: " ${lapse}
    if [ "$((lapse++))" -ge 50 ]; then
      echo card ${HOST}-${CARD} is not ready after $lapse seconds.
      exit 7
    fi
  done
  sleep 1
done
done
At that point the card has completed boot and we are modifying the uOS according to our need in the cluster. Most work is done within a single ssh command. Quoting it within <<EOD/EOD allows easy access to all previously defined variables, and uses only a single ssh connection speeding up the process. The following steps are taken

- Copy /etc/localtime and /etc/hosts from host to Intel Xeon Phi coprocessor
- Set the system date on the Intel Xeon Phi coprocessor
- Change permission of /bin/busybox, removing the standard setuid root
- Configure the MTU on the virtual network interface
- Mount the /home file system
- /usr/lib64 is not used, so create a link pointing to /opt/crtdc/micgnu/lib64
- Create a link structure mimicking the file systems on the host, even though only /home contains all data
- Load Lustre modules and mount the Lustre file system in the background
- Permission changes in the uOS file system to meet cluster standards
- Create a password file containing only root, sshd and the USER
- Replace initab with custom configuration, tell init about the change
- Create a custom sshd daemon to enable custom ulimits settings
- Kill all processes that are not required in this configuration
- Create a correct /etc/hosts file that knows about names in the cluster
- Configure and start ntp
- Start the coi_demon with USER privileges (instead of root)
- Wait for Lustre mount to be done. Disable Lustre checksums to optimize for speed
- Load the panfs driver and mount the Panasas volumes
- Setup a termination of the original ssh daemon dropbear

for CARD in ${MICCARDS}
do

MICIP=`gethostip -d ${HOST}-${CARD}`
MICIB=`gethostip -d ${HOST}-${CARD}ib0`
IPoIP=`gethostip -d ${HOST}ib0`

# Copy /etc/localtime and /etc/hosts from host
scp $verbose /etc/localtime ${MICIP}:/etc/localtime
scp $verbose /etc/hosts ${MICIP}:/tmp

echo -n "$0: ${CARD}: set system date to 
${MICCMD} $verbose ${MICIP} date -u `date -u "+%m%d%H%M%S"`

${MICCMD} $verbose ${MICIP} <<EOD

#change permission of /bin/busybox, removing setuid root
chmod 755 /bin/busybox

# Configuring the MTU on the virtual network interface
/sbin/ifconfig mic0 mtu ${MTU}

# mount the /home file system
${MOUNTCMD}
#!/usr/lib64 is not used by default on the uOS

ln -s /opt/crtdc/micgnu/lib64 /usr/lib64
LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib/:/opt/crtdc/micgnu/lib64
export LD_LIBRARY_PATH
PATH=/sbin:/usr/sbin:/usr/bin:/bin:/opt/crtdc/micgnu/bin

# Create a link structure mimicking the host file systems
umask 022
mkdir -p /opt/crtdc
mkdir -p /opt/intel
mkdir -p /lfs/lfs7
ln -s /home/MIC/3126-1 /opt/crtdc/micgnu
ln -s /home/MIC/xeonphi /opt/intel/xeonphi
ln -s /home/MIC/licenses /opt/intel/licenses

# Load Lustre modules; background mounting Lustre file system
/sbin/insmod /opt/crtdc/micgnu/lib/modules/libcfs.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/lvfs.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/inet.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/obdclass.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/crc32.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/ksocklnd.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/ptlrpc.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/osc.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/lquota.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/mdc.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/lov.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/lustre.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/mgc.ko
mount.lustre 36.101.223.1@tcp:/lfs7 /lfs/lfs7 &

# Fix some minor problems in the uOS filesystem
chmod 755 /var
chmod 755 /var/log
touch /var/log/messages
chmod 644 /var/log/messages /etc/localtime
chmod 777 /tmp
mkdir /var/empty
chmod 700 /var/empty

# Create a password file containing only root, sshd, USER
echo "$PASS >> /etc/passwd
echo "sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin"   >> /etc/passwd
echo 'mic:x:6015:${USER}'> /etc/group
echo 'sshd:x:73:'>> /etc/group

# Custom inittab
cp /opt/crtdc/micgnu/etc/inittab /etc

# Create a custom sshd daemon to enable custom ulimits settings
echo '#/bin/sh'                      > /etc/mydropbear
echo 'ulimit -s ${STACK_SIZE}'        >> /etc/mydropbear
echo 'ulimit -l ${LOCKED_MEMORY}'    >> /etc/mydropbear
echo 'ulimit -p ${PROCESSES}'        >> /etc/mydropbear
echo 'ulimit -n ${FILE_DESCRIPTOR}' >> /etc/mydropbear
echo '/opt/crtdc/micgnu/sbin/sshd -D' >> /etc/mydropbear
chmod 755 /etc/mydropbear

# Kill all processes that are not required in this configuration
killall -HUP init
killall inetd
killall getty
killall coi_daemon

# Create /etc/hosts
/bin/cat /tmp/hosts | grep -v 192.168 >> /etc/hosts
/bin/rm /tmp/hosts

# Configure and start ntp
echo 'ntp             123/tcp' >> /etc/services
echo 'ntp             123/udp' >> /etc/services
ntpd ${NTPSERVER}
ln -s /opt/crtdc/micgnu/etc/ntp       /etc/ntp
ln -s /opt/crtdc/micgnu/etc/ntp.conf  /etc/ntp.conf
/opt/crtdc/micgnu/bin/ntpd

# Start the coi_daemon with USER privileges (instead of root)
[ '${COI}' ] && ( nohup su ${USER} -c /bin/coi_daemon
2>/dev/null 1>/dev/null & )

# Wait for Lustre mount; disable Lustre check sums
wait
for I in /proc/fs/lustre/*/*/checksums; do echo 0 > $I; done
for I in /proc/fs/lustre/*/llite/*/checksum_pages; do echo 0 > $I; done

# Load the panfs driver and mount the Panasas volumes
/sbin/insmod /opt/crtdc/micgnu/lib/modules/panfs.ko
mkdir -p /panfs/home
/opt/crtdc/micgnu/bin/mount.panfs -o	nosuid,rw,noauto,callback-network-allow=36.101.0.0/255.255.0.0	panfs://36.107.212.1/admin/home /panfs/home

# Setup a termination of the original ssh daemon dropbear
(nohtup /bin/sh -c 'sleep 2; killall dropbear;' 2>/dev/null
1>/dev/null )&
• Ensure normal users can access the SCI driver

/bin/chmod 666 /dev/mic/scif

**Collecting performance statistics**

To obtain performance statistics, including hardware event data, install Intel® VTune™ Amplifier XE 2013. Additional information can be obtained from “VTune Amplifier XE 2011 installation on clusters” and “Analyzing MPI programs with Intel® VTune™ Amplifier XE and Intel® Inspector XE tools” on [http://software.intel.com](http://software.intel.com).

**Configuring the cluster to run MPI successfully**

While Intel Xeon Phi coprocessors are supported by the Intel® MPI Library (iMPI), some workarounds are still needed. For this example we assume iMPI is installed into /opt/intel/mpi/4.1.0.018. This method assumes the installation directory is identical on host and Intel Xeon Phi coprocessor (aka mounted as shared cluster file system).

• To run iMPI applications natively on the Intel Xeon Phi coprocessor an additional bootstrap program is needed to set up the MPI environment for each MPI process correctly. The bootstrap is a simple shell program and can be put into ${I_MPI_ROOT} – in this example this would be /opt/intel/mpi/4.1.0.018/bootstrap.sh:

```bash
#!/bin/sh

cmd=ssh

hostMPI=/opt/intel/mpi/4.1.0.018/intel64/bin
micMPI=/opt/intel/mpi/4.1.0.018/mic/bin

for arg in $@
do
  {
    if [ "${arg}" = "pmi_proxy" ]
    then
      if [[ `echo ${node} | grep "-mic"` ]]
      then
        echo "this is mic"
        cmd="${cmd} env PATH="${micMPI}:${PATH}"
      else
        echo "this is host"
        cmd="${cmd} env PATH="${hostMPI}:${PATH}""
      fi
    fi

    node=${arg}
    cmd="${cmd} ${arg}" 
  }
done
```
echo "new cmd = $cmd"
exec ${cmd}

- The process to compile native binaries for both host and Intel Xeon Phi coprocessor is straightforward.
  - Source the environment for the Intel compiler.
  - Source the environment for the Intel MPI library.
  - Use mpiicc to compile your MPI program for the host architecture.
  - Use mpiicc -mmic to compile for the Intel Xeon Phi coprocessor.

$ . /PATH_TO_INSTALL/bin/compilervars.sh intel64
$ . /PATH_TO_INSTALL/intel64/bin/mpivars.sh
$ mpiicc -o mpibench.x86_64 mpibench.c
$ mpiicc -mmic -o mpibench.mic mpibench.c

- When porting MPI applications it’s often sufficient to follow these steps (replace PROGRAM with the actual name of the resulting executable):

  $ export CC=mpiicc
  $ export LD=mpiicc
  $ make
  $ mv PROGRAM PROGRAM.x86_64
  $ export CC=mpiicc -mmic
  $ export LD=mpiicc -mmic
  $ make
  $ mv PROGRAM PROGRAM.mic

- Start the mpi program using mpiexec.hydra. The basic command structure is

  $ mpiexec.hydra DEBUGOPTIONS INTERCONNECTOPTIONS -bootstrap-
  exec PATH/boottstrap.sh HOSTINFO1 : HOSTINFO2 : HOSTINFO3....

- For each host and each Intel Xeon Phi coprocessor you need to add one HOSTINFO entry. All HOSTINFOs are separated by single ".". Each HOSTINFO has the form:

  -host NODE-NAME -n NUMBER_OF_PROCESSES/NODE PATH/BINARY
  PROGRAM_ARGUMENTS

- A more complete example might look like:

  $ ICCINSTALL=PATH_TO_COMPILER_INSTALLATION
  $ MPIINSTALL=PATH_TO_IMPI_INSTALLATION
  $ HOSTBIN=SOMEPATH/PROGRAM.x86_64
  $ MIC_BIN=SOMEPATH/PROGRAM.mic
  $ ARG_COM=COMMON_ARGUMENTS_OF_PROGRAMM
  $ HOST_NP=NUMBER_OF_MPI_PROCESSES_PER_HOST
  $ MIC_NP=NUMBER_OF_MPI_PROCESSES_PER_XEON_PHI
The command switch -genv I_MPI_DEBUG=50 can be used to get detailed output on the MPI startup.

Intel MPI by default selects the optimum backbone found. To force the application to use TCP add the option: -genv I_MPI_FABRICS shm:tcp

To enforce use of InfiniBand use: -genv I_MPI_OFA_ADAPTER_NAME mlx4_0 -genv I_MPI_FABRICS shm:ofa

We conclude with a real life example, deploying an application to 2 nodes with one Intel Xeon Phi coprocessor each. The application detects the interconnect automatically in the first run, is then forced to use Gigabit Ethernet/TCP; and finally InfiniBand is selected again.

Minimum averaged time to complete 5 send/recv using 1000000 doubles:

2 ms

Table showing delay vs. minimum time in ms

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>-&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
esg020       Avg:0.0041  Min:0.0012740  Max:0.0058420
esg019       Avg:0.0052  Min:0.00019980  Max:0.0134050
esg020-mic0  Avg:0.0072  Min:0.0050010  Max:0.0161220
esg019-mic0  Avg:0.0072  Min:0.0049980  Max:0.0161220

Minimal time to complete 5 send/recv using 1000000 doubles in s:
  0.0012740

$ # now running over TCP
$ mpiexec.hydra -bootstrap-exec $I_MPI_ROOT/bootstrap.sh \ 
  -genv I_MPI_FABRICS shm:tcp \ 
  -host esg019 -n $HOST_NP $HOSTBIN $ARG_COM : \ 
  -host esg020 -n $HOST_NP $HOSTBIN $ARG_COM : \ 
  -host esg019-mic0 -n $MIC__NP $MIC_BIN $ARG_COM : \ 
  -host esg020-mic0 -n $MIC__NP $MIC_BIN $ARG_COM 
...
Received: rank 0 of 4 running on esg019
Received: rank 1 of 4 running on esg020
Received: rank 2 of 4 running on esg019-mic0
Received: rank 3 of 4 running on esg020-mic0
...

Minimum averaged time to complete 5 send/recv using 1000000 doubles:
  53 ms

Table showing delay vs. minimum time in ms
Sender   Receiver ->
    v
    0  1  2  3
 0      13  0  15
 1     15  53
 2     23
 3
esg019       Avg:0.0626  Min:0.0509680  Max:0.0755150
esg019-mic0  Avg:0.0658  Min:0.0509680  Max:0.0979900
esg020       Avg:0.0802  Min:0.0508270  Max:0.3203040
esg020-mic0  Avg:0.0836  Min:0.0508270  Max:0.3203040

Minimal time to complete 5 send/recv using 1000000 doubles in s:
  0.0508270

$ # now running over InfiniBand with options
$ mpiexec.hydra -bootstrap-exec $I_MPI_ROOT/bootstrap.sh \ 
  > -genv I_MPI_OFA_ADAPTER_NAME mlx4_0 -genv I_MPI_FABRICS shm:ofa \

Minimum averaged time to complete 5 send/recev using 1000000 doubles:
  2 ms

Table showing delay vs. minimum time in ms
Sender   Receiver ->
  v
    0   1   2   3
0       0   6   6
1       4   4   
2       9
3

esg020  Avg: 0.0050  Min: 0.0012860  Max: 0.0074520
esg019  Avg: 0.0062  Min: 0.0012860  Max: 0.0142080
esg019-mic0 Avg: 0.0086  Min: 0.0059900  Max: 0.0167260
esg020-mic0 Avg: 0.0086  Min: 0.0060110  Max: 0.0167260

Minimal time to complete 5 send/recev using 1000000 doubles in s:
  0.0012860
Debugging native Intel® Xeon Phi™ coprocessor applications with ddd+gdb

Intel provides a gdb debugger that allows debugging native applications for the Intel Xeon Phi coprocessor. The ddd graphical interface is able to work with a gdb over a remote connection. To enable this on the current Intel MPSS stack follow this procedure:

- Current versions of ddd show a small problem with the initialization file. For some reasons it's always overwritten when you exit ddd.
- Start the ddd with the following options:

  ```
  --host IPADDRESS
  
  IPADDRESS is the IP address of the Intel Xeon Phi coprocessor.
  
  --no-exec-window
  --debugger PATH/gdb
  
  Use the absolute path to the gdb running natively on the Intel Xeon Phi coprocessor.
  
  The complete command might look like:
  
  ```
  $ echo "Ddd*rshCommand: ssh" >> ~/.ddd/init
  $ ddd --host mic0 --no-exec-window --debugger
  /opt/crtdc/micgnu/bin/gdb /home/USER/test.mic
  ```
Debugging offloaded applications using gdb

Background
The coi_daemon runs on the Intel Xeon Phi coprocessor’s uOS and takes care of initializing the communication between the application running on the HOST and the offloaded part executed on the Intel Xeon Phi coprocessor. After the initial fork+exec, the device process has a direct communication channel to the host process.

- During compilation of the binary the compiler creates for functions and variables declared as “shared” codeblocks both for intel64 and mic architectures.
- The HOST part of the binary than tries to communicate with the coi_daemon on the Xeon Phi coprocessor.
- If initialization of the offload process was successful, the infrastructure extracts the mic-binaries and stores them in /tmp/coi_procs/…/ on the HOST.
- If communication is successful, the files are copied onto the Intel Xeon Phi coprocessor and stored in identical path-names /tmp/coi_procs/<device number>/<pid>.
- Finally the program /tmp/coi_procs/<device number>/<pid>/offload_main is executed by the coi_daemon, and the offloaded execution begins.

Requirements
- A user can connect to the Intel Xeon Phi coprocessor with normal (aka non root) privileges.
- The file system used (aka the location of binaries and sources) are identical on Intel Xeon Phi coprocessor and the host.
- A native gdb is available on the Intel Xeon Phi coprocessor.
- You have access to a coi_daemon recent enough. Otherwise you might run into unforeseen issues. It is possible to use a coi_daemon completely separate from the standard /bin/coi_daemon file. This example assumes you have a debug capable version of coi_daemon installed in /opt/crtdc/micgnu/lib/coi_debug/.

Preparation of the Intel® Xeon Phi™ coprocessor’s uOS
A necessity when debugging offloaded application is the ability to start the coi_daemon. Initial communication happens over SCI ports, which are similar to normal BSD sockets. For security reasons the default coi-port is below 1024. This is unfortunate for debugging. In future mpss versions it will be possible to change this port – at the moment as workaround the security check needs to be disabled.

In the default configuration only root can execute the coi_daemon correctly. If a normal user runs

```
mic0 $ /bin/coi_daemon --coiuser=`whoami`
```

the daemon would terminate due to an “EACCES” access denied error.

**WARNING:** this change opens a security hole. It exposes vulnerabilities in SCI, especially a variety of DOS attacks. It has not been tested against possible exploits via privilege escalation.

You need the src package package_full_src .tgz as provided by Intel. On a system with the mpss stack installed extract it into a working directory of the HOST.
## untar the package_full_src to /tmp/work

```
cd /tmp/work
```

Now change in file `card/driver/micscif_api.c` the function `scif_bind()` (line ~367) from

```c
if (pn) {
    /*
    * Modeled on
    http://www.ietf.org/rfc/rfc1700.txt?number=1700
    * SCIF ports below SCIF_ADMIN_PORT_END can only be
    * bound by
    * system (or root) processes or by processes
    * executed by
    * privileged users.
    */
    if (!capable(CAP_SYS_ADMIN) && pn < SCIF_ADMIN_PORT_END) {
        ret = -EACCES;
        goto scif_bind_exit;
    }
    if ((tmp = rsrv_scif_port(pn)) != pn) {
        ret = -EINVAL;
        goto scif_bind_exit;
    }
} else {
```

to

```c
if (pn) {
    /*
    * Modeled on
    http://www.ietf.org/rfc/rfc1700.txt?number=1700
    * SCIF ports below SCIF_ADMIN_PORT_END can only be
    * bound by
    * system (or root) processes or by processes
    * executed by
    * privileged users.
    */
    if (pn < SCIF_COI_PORT_0 || pn > SCIF_COI_PORT_9) {
        if (!capable(CAP_SYS_ADMIN) && pn < SCIF_ADMIN_PORT_END) {
            ret = -EACCES;
            goto scif_bind_exit;
        } 
        if ((tmp = rsrv_scif_port(pn)) != pn) {
            ret = -EINVAL;
            goto scif_bind_exit;
        }
    } else {
```

```c
```
Adding the 3 lines marked green simply removes the test for the ports used by COI and allows a normal user to bind the coi_daemon to the default port.

The next step creates a new uOS for the Intel Xeon Phi coprocessor and replaces the old image file.

$ make ARCH=k1om card
$ mv /lib/firmware/mic/uos.img /lib/firmware/mic/uos.img.orig
$ cp card/kernel/arch/x86/boot/bzImage /lib/firmware/mic/uos.img

Reboot the Intel Xeon Phi coprocessor and connect as normal user, then execute again:

mic0 $ /bin/coi_daemon --coiuser=`whoami`

The daemon should now continue to operate in the foreground. Ctrl-z and a bg command should work as expected at this point as well.

**Execution**

For the following procedure one will need 2 parallel connections to the HOST. Session A is used to control the coi_daemon and the offloaded part, Session B used to run the binary.

- **Session A**: connect to the Intel Xeon Phi coprocessor as normal user and execute the coi_daemon in the background.

  $ ssh mic0
  mic0 $ /bin/coi_daemon --coiuser=`whoami`

- **Session B**: do a test run: source the compiler, compile your program using the –g switch and execute it. The program has to employ an offload model, illustrated in this example using a #pragma offload command.

  $ . /opt/intel/Compilers/2013.0.079/bin/compilervars.sh intel64
  $ head -12 pi_openmp_mic_sol.cpp
  #include "pi_openmp_mic_sol.h"

  #define SIZE 1000

  __declspec(target(mic))
  double *rect, *midPt, *area;

  int compute(long int num_steps)
  {
    double pi = 0;
    #pragma offload target(mic) inout(pi) in(num_steps)
    in(rect:length(num_steps)) in(midPt:length(num_steps))
    in(area:length(num_steps))
  }

  $ icpc -g -o pi_mic pi_openmp_mic_sol.cpp
  $ ./pi_mic
  counts:100
  preparation starting
allocated 3 times 0.000762939 MBs
preparation done
PI:3.14257
succeeded in 1.05035 seconds.
starting cleanup

- **Session B: note** – the offload happens within the function `compute()`

$ head -12 pi_openmp_mic_sol.cpp
#include "pi_openmp_mic_sol.h"

#define SIZE 100l
__declspec(target(mic)) double *rect, *midPt, *area;

int compute(long int num_steps)
{
    double pi = 0;
    #pragma offload target(mic) inout(pi) in(num_steps) 
        in(rect:length(num_steps)) in(midPt:length(num_steps))
        in(area:length(num_steps))
    
- **Session A: start the gdb debugger**

$ /opt/crtdc/micgnu/gdb

- **Session A - gdb: set the executable (path to the debug version of coi_daemon) using the file command**

    file /opt/crtdc/micgnu/lib/coi_debug/device-linux-debug/bin/coi_daemon

- **Session A - gdb: point the coi_daemon to the correct library; otherwise a default path would be used**

    set environment COI_LIBRARY_PATH =
    /opt/crtdc/micgnu/lib/coi_debug/device-linux-debug/lib/libcoi_device.so.0

- **Session A - gdb: prevent coi_daemon to fork a thread measuring CPU utilization; prevents issues with following the right path on forks**

    set environment COI_CPU_UTIL = 0

- **Session A - gdb: kill all existing coi_daemons to free up the port**

    shell killall coi_daemon
• Session A - gdb: set the follow mode to child (aka on a call to clone() syscall gdb will follow the child process)

        set follow-fork-mode child

• Session A - gdb: set a breakpoint on the function compute(); this breakpoint will happen in a binary that HAS NOT BEEN LOADED YET

        set breakpoint pending on b compute

• Session A - gdb: run the coi_daemon (replace USERNAME with your actual userid)

        run --coiuser=USERNAME

• Session A - gdb: example output

        Starting program: /opt/crtdc/micgnu/lib/coi_debug/device-linux-debug/bin/coi_daemon
        [Thread debugging using libthread_db enabled]
        Using host libthread_db library "/lib64/libthread_db.so.1".
        [New Thread 0x7ffff66e6700 (LWP 8578)]
        [Thread 0x7ffff66e6700 (LWP 8578) exited]
        [New process 8579]

• Session B: ensure that the matching coi-libraries are loaded by setting the LD_LIBRARY_PATH, then start your program

        $ COIHOST=/opt/crtdc/micgnu/lib/coi_debug/host-linux-debug/
        $ export LD_LIBRARY_PATH=$COIHOST/lib:$LD_LIBRARY_PATH
        $ ./pi_mic

• Session A: program has (hopefully) stopped at the breakpoint defined above; example output

        [Switching to Thread 0x7fffff3ab7700 (LWP 8583)]

        Breakpoint 1,
        __offload_entry_pi_openmp_mic_sol_cpp_11_Z7computel () at pi_openmp_mic_sol.cpp:11
        11      #pragma offload target(mic) inout(pi) in(num_steps)
            in(rect:length(num_steps)) in(midPt:length(num_steps))
            in(area:length(num_steps))
        (gdb)

• Session A: at this point you can use standard gdb techniques to debug your code
• Session A: to make things easier one can pack all the preparation gdb commands into a command file and start gdb with a --x switch:

        $ cat gdb.commands
        file /opt/crtdc/micgnu/lib/coi_debug/device-linux-debug/bin/coi_daemon
set environment COI_LIBRARY_PATH = /opt/crtdc/micgnu/lib/coi_debug/device-linux-debug/lib/libcoi_device.so.0
set environment COI_CPU_UTIL = 0
shell killall coi_daemon
set follow-fork-mode child
set breakpoint pending on b compute
run --coiuser=mhebenst
$ gdb -x gdb.commands

Addendum: Configuration details
The remainder of this document contains examples of specific configuration files used to configure the cluster. Please note that some of these scripts contain lines commented out. This is either for debugging purposes or due to a feature currently unavailable in our live cluster.

IP addresses used in example configuration

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Description</th>
<th>Interfaces</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>server for administration</td>
<td>eth0, ib0</td>
<td>10.101.1.1, 10.102.1.1</td>
</tr>
<tr>
<td>login</td>
<td>login servers for users: access files, compile programs, submit jobs</td>
<td>eth0, ib0</td>
<td>10.101.1.2, 10.102.1.2</td>
</tr>
<tr>
<td>nfs1</td>
<td>NFS server, delivering /home</td>
<td>eth0</td>
<td>10.101.1.3</td>
</tr>
<tr>
<td>panfs1</td>
<td>Panasas server, delivering /opt</td>
<td>eth0</td>
<td>10.101.1.4</td>
</tr>
<tr>
<td>lfs1</td>
<td>Lustre server, delivering /scratch</td>
<td>eth0, ib0</td>
<td>10.101.1.5, 10.102.1.5</td>
</tr>
<tr>
<td>n01</td>
<td>compute node</td>
<td>eth0, ib0</td>
<td>10.101.3.1, 10.102.3.1</td>
</tr>
<tr>
<td>n01-mic0</td>
<td>first Intel Xeon Phi card on node n01</td>
<td>mic0, mic0:1</td>
<td>10.101.3.41, 10.102.3.41</td>
</tr>
<tr>
<td>n01-mic1</td>
<td>second Intel Xeon Phi card on node n01</td>
<td>mic0, mic0:1</td>
<td>10.101.3.81, 10.102.3.81</td>
</tr>
<tr>
<td>n02</td>
<td>compute node</td>
<td>eth0, ib0</td>
<td>10.101.3.2, 10.102.3.2</td>
</tr>
<tr>
<td>n02-mic0</td>
<td>first Intel Xeon Phi card on node n02</td>
<td>mic0, mic0:1</td>
<td>10.101.3.42, 10.102.3.42</td>
</tr>
<tr>
<td>n02-mic1</td>
<td>second Intel Xeon Phi card on node n02</td>
<td>mic0, mic0:1</td>
<td>10.101.3.82, 10.102.3.82</td>
</tr>
<tr>
<td>n03</td>
<td>compute node</td>
<td>eth0, ib0</td>
<td>10.101.3.3, 10.102.3.3</td>
</tr>
<tr>
<td>n03-mic0</td>
<td>first Intel Xeon Phi card on node n03</td>
<td>mic0, mic0:1</td>
<td>10.101.3.43, 10.102.3.43</td>
</tr>
</tbody>
</table>
Example of bridged interface

Before installing the bridge `br0`:

```
$ ifconfig

eth0      Link encap:Ethernet  HWaddr 00:1E:67:0C:B7:4C
inet addr:36.101.15.36  Bcast:36.101.255.255  Mask:255.255.0.0
UP  BROADCAST  RUNNING  MULTICAST  MTU:9000  Metric:1
RX packets:200890212 errors:0 dropped:0 overruns:0 frame:0
TX packets:1408892368 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:9209030076798 (8.3 TiB)  TX bytes:6094872504068 (5.5 TiB)
Memory:b1a20000-b1a40000

lo       Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP  LOOPBACK  RUNNING  MTU:16436  Metric:1
RX packets:1181150 errors:0 dropped:0 overruns:0 frame:0
TX packets:1181150 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:7515759257 (6.9 GiB)  TX bytes:7515759257 (6.9 GiB)
```

After installing the network bridge, `eth0` now does not have an IP address anymore, and the address has moved to `br0`. `br0` and `eth0` share a hardware address:

```
$ ifconfig

br0       Link encap:Ethernet  HWaddr 00:1E:67:0C:B7:4C
inet addr:36.101.15.36  Bcast:36.101.255.255  Mask:255.255.0.0
UP  BROADCAST  RUNNING  MULTICAST  MTU:9000  Metric:1
RX packets:636853 errors:0 dropped:0 overruns:0 frame:0
TX packets:37529 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:65024686 (62.0 MiB)  TX bytes:65690500 (62.6 MiB)

eth0      Link encap:Ethernet  HWaddr 00:1E:67:0C:B7:4C
UP  BROADCAST  RUNNING  MULTICAST  MTU:9000  Metric:1
RX packets:1036186 errors:0 dropped:0 overruns:0 frame:0
TX packets:1633683 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:146064064 (139.2 MiB)  TX bytes:13274916653 (12.3 GiB)
```
Memory: d0960000-d0980000

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueue len:0
RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

ifcfg-eth0:
  DEVICE=eth0
  ONBOOT=yes
  BRIDGE=br0
  MTU=9000

Ifcfg-br0:
  DEVICE=br0
  TYPE=Bridge
  ONBOOT=yes
  DELAY=0
  NM_CONTROLLED="no"
  MTU=9000
  BOOTPROTO= dhcp
  NOZEROCONF=yes

Complete example of script kncmicbootstart_crt.sh
$ cat /opt/admin/bin/kncmicbootstart_crt.sh
#!/bin/bash
#set -x
DEBUG=1
#verbose="-v"
verbose=""

SYSFS=/sys/class/mic
MICIFNAME=mic
MICCMD=/usr/sbin/miccmd
MICPUT=/usr/sbin/micput
HOSTROOT=/usr/local/mic
SSHPUBID=/usr/local/mic/id_rsa.pub
SSHID=/usr/local/mic/id_rsa
SHELL=/bin/sh

MICOS="2.6.34-g0483f82"

MTU=9000
BRIDGE=br0
NETMASK=255.255.0.0
IPoIBNW="36.102.0.0/16"
MOUNTCMD="mount -o vers=3,nolock 36.101.233.11:/home /home"
NTPSERVER=36.101.201.1
# options for uOS cmdline
MODOPTIONS="huge_page=1 reg_cache=1 quiet hostname=${HOST}" -
mic${card} hosts='MODOPTIONS="${MODOPTIONS} pubkey="/bin/cat ${SSHPUBID} | /bin/cut -f2 -d' " mce=off console=hvc0"'
MODOPTIONS="${MODOPTIONS}${HOSTIP}/16,host"
HOST="/bin/hostname` NUMMICCARDS="/sbin/lspci | /bin/grep -c "Coprocessor: Intel Corporation Device"
STACK_SIZE=8192
CORE_FILE_SIZE=0
LOCKED_MEMORY=4194304
PROCESSES=61314
FILE_DESCRIPTOR=1024
COI=y
MIC_ISNOOPOFF=0  # 0/1
MIC_ECCOFF=0    # 0/1
MIC_HUGEPAGEOFF=0  # 0/1
MIC_REGCACHEOFF=0  # 0/1
MIC_PSM="N"     # N/Y
MIC_P2P=0       # 0/1

# Parsing input parameters:
while [ -n "$1" ]
do
    case $1 in
        -s ) STACK_SIZE=$2;;
            # -c ) CORE_FILE_SIZE=$2;;
        -l ) LOCKED_MEMORY=$2;;
        -p ) PROCESSES=$2;;
        -n ) FILE_DESCRIPTOR=$2;;
        -isnoopoff) MIC_ISNOOPOFF=$2;;
        -eccoff) MIC_ECCOFF=$2;;
        -HUGEPAGEOFF) MIC_HUGEPAGEOFF=$2;;
        -REGCACHEOFF) MIC_REGCACHEOFF=$2;;
        -psm) MIC_PSM=$2;;
        -p2p) MIC_P2P=$2;;
        * ) break ;;
    esac
    shift 2
done

USER=$1

Check_State()
{
    Counter=0
while test "$Counter" -lt "$Timeout"
do
  echo $ExpectedState | grep -q -w `cat /sys/devices/virtual/mic/mic0/state` && return 0
  Counter=`expr "$Counter" + 1`
  echo -n "$Counter"
  sleep 1
done

echo "Card $MyCard not reacting quickly enough, code $MyError, exiting"
exit $MyError

}
badmin hclose -C "mpss not running" $HOST
exit 7
fi

# get list of MIC cards
MICCARDS=""
I=0
for J in `seq ${NUMMICCARDS}`
do
    MICCARDS="${MICCARDS} ${MICIFNAME}${I}"  
    MICIPNAME="${HOST}-${MICIFNAME}${I}"
    MICIP=`gethostip -d ${MICIPNAME}`  
    if [ -z "${MICIP}" ]; then echo "no IP address for ${MICIPNAME} found in /etc/hosts, exiting";exit 7; fi
    I=$(( $I + 1 ))
done

echo "MIC cards: ${MICCARDS}"
HOSTIP=`gethostip -d ${HOST}-host`

[ "$DEBUG" ] && echo "XX micibstop `date +%s`"
# stopping OFED
/usr/sbin/micibstop
[ "$DEBUG" ] && echo "XX after_micibstop `date +%s`"

# check BRIDGE interface is ready
if [ ! -d /sys/class/net/${BRIDGE} ]
then
    echo "Bridge-device /sys/class/net/${BRIDGE} not found, exiting"
    badmin hclose -C "bridge not found" $HOST
    exit 1
fi

####################################################################
######################################
# cleaning up previous config
#
####################################################################

[ "$DEBUG" ] && echo "XX reset `date +%s`"
for CARD in ${MICCARDS}
do
    # pdsh -u 60 -w localhost micctrl -w $CARD
    Check_State $CARD 60 180 ready online
done
for CARD in ${MICCARDS}
do
    test `cat /sys/class/mic/${CARD}/state` = ready || micctrl -r $CARD
    MICIP=`gethostip -d ${HOST}-${CARD}`
sed -e "s,IPADDR=.*IPADDR=/${MICIP}," -i
/opt/intel/mic/filesystem/${CARD}/etc/sysconfig/network/ifcfg-mic0
sed -e "s,^HWADDR=,#HWADDR=," -i
/opt/intel/mic/filesystem/${CARD}/etc/sysconfig/network/ifcfg-mic0
done
    echo "HOSTNAME=${HOST}-${CARD}" >
    /opt/intel/mic/filesystem/${CARD}/etc/sysconfig/hostname
for CARD in ${MICCARDS}
do
    # pdsh -u 60 -w localhost micctrl -w $CARD
    Check_State $CARD 180 101 ready
done
[ "$DEBUG" ] && echo XX after_reset `date +%s`
# XXX check card is down
ERROR=0
for CARD in ${MICCARDS}
do
    test `cat /sys/class/mic/${CARD}/state` = ready || ERROR=1
done
if [ "${ERROR}" -gt 0 ]
then
    badmin hclose -C "reset not working" $HOST
    exit 1
fi

#############################
# is this kncstop script?
#
[ `basename $0` = "kncmicbootstop_crt.sh" ] && exit 0

#############################
# more checks
#
if [ -z "${USER}" ]
then
    echo "usage: $0 USER"
exit 3
fi

# check the user is known to the host system
/usr/bin/id $USER >/dev/null 2>/dev/null
if [ $? != 0 ]
then
echo "error: no user $USER known to system" && exit 3
fi

MUID=~/usr/bin/id -u $USER`
MGID=~/usr/bin/id -g $USER`

#this is the password entry created on the MIC
PASS="$\{USER\}:x:\{MUID\}:\{MGID\}:/:home/\{USER\}:\{SHELL\}"

# ensure we have routing/forwarding present
sysctl -w net.ipv4.ip_forward=1

####################################################################
####################################
# starting the driver
#
####################################################################
cat /root/.ssh/id_rsa.pub >
/usr/local/mic/filesystem/mic0/root/.ssh/authorized_keys
cat /root/.ssh/id_rsa.pub >
/usr/local/mic/filesystem/mic1/root/.ssh/authorized_keys

[ "$DEBUG" ] && echo "XX boot `date +%s`"
for CARD in \{MICCARDS\}
do
  micctrl -b \{CARD\}
done
for CARD in \{MICCARDS\}
do
  # pdsh -u 60 -w localhost micctrl -w \{CARD\}
  Check_State \$CARD 90 102 online
done

[ "$DEBUG" ] && echo "XX after_boot `date +%s`"

# report host driver version info
echo "$0: host:"
echo "  `cat \{SYSFS\}/ctrl/version`"
echo "  SCIF: `cat \{SYSFS\}/ctrl/host_revision`"
echo "  Linux: `cat \{SYSFS\}/ctrl/uos_revision` (`uname -r`)

for CARD in ${MICCARDS}
do
    # testing access to card
    MICIP=`gethostip -d ${HOST}-${CARD}`

declare -i lapse=0
while ! ping -q -c 1 ${MICIP} >/dev/null; do
    printf "\r%d: " ${lapse}
    if [ "$(lapse++)" -ge 50 ]; then
        echo card ${HOST}-${CARD} is not ready after $lapse seconds.
        exit 6
    fi
    sleep 1
done

declare -i lapse=0
while [ ! "`${MICCMD} ${MICIP} uname -r`" = "${MICOS}" ]
do
    printf "\r%d: " ${lapse}
    if [ "$(lapse++)" -ge 50 ]; then
        echo card ${HOST}-${CARD} is not ready after $lapse seconds.
        exit 6
    fi
    sleep 1
done

# print card version
echo -n "  ${CARD}:
${MICCMD} $verbose ${MICIP} /etc/mic-version | sed 's,^,    ,'
echo

done

####################################################################
################################
#####
# starting OFED
#####
####################################################################

echo ${MICCARDS}
for CARD in ${MICCARDS}
do
    #  pdsh -u 60 -w localhost micctrl -w ${CARD}
    Check_State $CARD 60 103 online
    declare -i lapse=0
    MICIP=`gethostip -d ${HOST}-${CARD}`
while [ ! "`${MICCMD} ${MICIP} uname -r`" = "`${MICOS}`" ]
  do
    printf "\r%d: " ${lapse}
    if [ "$((lapse++))" -ge 50 ]; then
      echo card ${HOST}-${CARD} is not ready after $lapse seconds.
      exit 7
    fi
    sleep 1
  done
done
[ "$DEBUG" ] && echo XX micibstart `date +%s`
/usr/sbin/micibstart
[ "$DEBUG" ] && echo XX after_micibstart `date +%s`

echo ${MICCARDS}
for CARD in ${MICCARDS}
do
  # pdsh -u 60 -w localhost micctrl -w ${CARD}
  Check_State $CARD 60 104 online
  declare -i lapse=0
  while [ ! "`${MICCMD} ${MICIP} uname -r`" = "`${MICOS}`" ]
    do
      printf "\r%d: " ${lapse}
      if [ "$((lapse++))" -ge 50 ]; then
        echo card ${HOST}-${CARD} is not ready after $lapse seconds.
        exit 7
      fi
      sleep 1
    done
done

####################################################################
####################################
# more checks
#
# for CARD in ${MICCARDS}
do
  MICIP=`gethostip -d ${HOST}-${CARD}`
  MICIB=`gethostip -d ${HOST}-${CARD}ib0`
  IPoIP=`gethostip -d ${HOST}ib0`
route del -host ${MICIB} ${BRIDGE} 2>/dev/null
route add -host ${MICIB} ${BRIDGE}

${MICPUT} $verbose ${MICIP} /etc/localtime /etc/localtime
echo -n "$0: ${CARD}: set system date to "
${MICCMD} $verbose ${MICIP} date -u `date -u "+%m%d%H%M%S"`

${MICPUT} $verbose ${MICIP} /etc/hosts /tmp

${MICCMD} $verbose ${MICIP} <<EOD
    chmod 755 /bin/busybox
    /sbin/ifconfig mic0 mtu ${MTU}
    ${MOUNTCMD}
    echo 45056 > /proc/sys/vm/min_free_kbytes
    echo 800000 > /proc/sys/vm/min_free_kbytes
    export PATH
    ln -s /opt/crtdc/micgnu/lib64 /usr/lib64
    LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib:/opt/crtdc/micgnu/lib64
    export LD_LIBRARY_PATH
    umask 022
    #XX chmod 644 /etc/dat.conf /etc/protocols
    mkdir -p /opt/crtdc
    mkdir -p /opt/intel
    mkdir -p /lfs/lfs4
    mkdir -p /lfs/lfs5
    mkdir -p /lfs/lfs6
    mkdir -p /lfs/lfs7
    ln -s /home/MIC/3126-1 /opt/crtdc/micgnu
    ln -s /home/MIC/xeonphi /opt/intel/xeonphi
    ln -s /home/MIC/licenses /opt/intel/licenses
    /sbin/ifconfig mic0:1 ${MICIB} netmask 255.255.0.0
    /sbin/route del -net 36.102.0.0 netmask 255.255.0.0
    /sbin/route del -net default
    /sbin/route add default mic0
    ip route add ${IPoIBNW} via ${IPoIP} dev mic0
    /sbin/route del default
    /sbin/route add default gw 36.101.181.109
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/libcfs.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/lvfs.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/lnet.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/obdclass.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/crc32.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/ksocklnd.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/ptlrpc.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/osc.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/lquota.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/mdc.ko
    /sbin/insmod /opt/crtdc/micgnu/lib/modules/lov.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/lustre.ko
/sbin/insmod /opt/crtdc/micgnu/lib/modules/mgc.ko

export
PATH=/sbin:/usr/sbin:/usr/bin:/bin:/opt/crtdc/micgnu/bin
export
LD_LIBRARY_PATH=/opt/crtdc/micgnu/lib:/opt/crtdc/micgnu/lib64

# mount.lustre 36.102.225.1@tcp:/lfs5 /lfs/lfs5 &
# mount.lustre 36.101.226.111@tcp:36.101.226.112@tcp:/tsintel /lfs/lfs6 &
mount.lustre 36.101.223.1@tcp:/lfs7 /lfs/lfs7 &

chmod 755 /var
chmod 755 /var/log
touch /var/log/messages
chmod 644 /var/log/messages /etc/localtime
chmod 777 /tmp

# chmod 755 /lib64/libcoi_device.so
# ln -fs /lib64/libcoi_device.so /lib64/libcoi_device.so.0
# chmod 755 /bin/coi_daemon

# echo 'root:***not-salt*not-a-password*:0:0:root://:/:/bin/sh' > /etc/passwd
echo $PASS >> /etc/passwd
echo "sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin" >> /etc/passwd
echo 'mic:x:6015:${USER}'>>/etc/group
echo 'sshd:x:73:'>>/etc/group

# disable inetd service
cp /opt/crtdc/micgnu/etc/inittab /etc
mkdir /var/empty
chmod 700 /var/empty

echo '##/bin/sh' > /etc/mydropbear
echo 'ulimit -s ${STACK_SIZE}'    >> /etc/mydropbear
echo 'ulimit -l ${LOCKED_MEMORY}' >> /etc/mydropbear
echo 'ulimit -p ${PROCESSES}'    >> /etc/mydropbear
echo 'ulimit -n ${FILE_DESCRIPTOR}' >> /etc/mydropbear
echo '/opt/crtdc/micgnu/sbin/sshd -D' >> /etc/mydropbear
# echo '##/usr/sbin/dropbear -F'    >> /etc/mydropbear
chmod 755 /etc/mydropbear

sed -e 's,RTLDDLST=/lib64/ld-linux-llom.so.2,RTLDDLST=/lib64/ld-linux-klom.so.2,' -i /usr/bin/ldd
killall -HUP init
killall inetd
killall getty
killall coi_daemon

# add host file
echo '127.0.0.1 localhost.localdomain localhost' > /etc/hosts
/bin/cat /tmp/hosts | grep -v 192.168 >> /etc/hosts
/bin/rm /tmp/hosts

echo 'ntp 123/tcp' >> /etc/services
echo 'ntp 123/udp' >> /etc/services
ntpdate ${NTPSERVER}
ln -s /opt/crtdc/micgnu/etc/ntp /etc/ntp
ln -s /opt/crtdc/micgnu/etc/ntp.conf /etc/ntp.conf
/opt/crtdc/micgnu/bin/ntpd
echo tsc > /sys/devices/system/clocksource/clocksource0/current_clocksource

for I in mapper mic; do chmod o+rx /dev/\${I}; done
for I in scif full null ptmx random tty urandom zero; do chmod o+rwx /dev/\${I}; done
[ '${COI}' ] && ( noshd su ${USER} -c /bin/coi_daemon 2>/dev/null 1>/dev/null & )

/sbin/insmod /opt/crtdc/micgnu/lib/modules/panfs.ko
mkdir -p /panfs/home
mkdir -p /panfs/panfs2/home1
mkdir -p /panfs/panfs2/home2
mkdir -p /panfs/panfs2/archive
/opt/crtdc/micgnu/bin/mount.panfs -o nosuid,ro,nw auto, callback-network=allow=36.101.0.0/255.255.0.0 panfs://36.107.212.1/admin/home /panfs/home
/opt/crtdc/micgnu/bin/mount.panfs -o nosuid,ro,nw auto, callback-network=allow=36.101.0.0/255.255.0.0 panfs://36.107.212.1/home1 /panfs/panfs2/home1
/opt/crtdc/micgnu/bin/mount.panfs -o nosuid,ro,nw auto, callback-network=allow=36.101.0.0/255.255.0.0 panfs://36.107.212.1/home2 /panfs/panfs2/home2
/opt/crtdc/micgnu/bin/mount.panfs -o nosuid,ro,nw auto, callback-network=allow=36.101.0.0/255.255.0.0 panfs://36.107.212.1/archive /panfs/panfs2/archive
(nohup /bin/sh -c 'sleep 2; killall dropbear;' 2>/dev/null 1>/dev/null )&
EOD
done

/bin/chmod 666 /dev/mic/scif

[ "$DEBUG" ] && echo XX done `date +%s`
About the Authors

Michael Hebenstreit (michael.hebenstreit@intel.com) is a Senior Cluster Architect and tech lead for the Endeavor HPC benchmarking datacenter. In his 7 years at Intel he helped to make Endeavor a prime HPC benchmarking datacenter in the world and has been essential in integrating the Intel® Xeon Phi™ coprocessors into the cluster. Using only 140 servers equipped with Xeon Phi coprocessors the CRT-DC team reached #150 on the June 2012 top 500 list under the name Discovery.

Romain Dolbeau is a co-founder of CAPS entreprise (http://www.caps-entreprise.com/) . He joined the company in 2004 and is now an HPC fellow. Specializing in understanding and exploiting hardware architectures for high-performance computing, his areas of specialties include conventional CPUs, GPUs and novel multi-cores architectures. Prior to joining CAPS, he was a student at ENS Cachan, University Paris XI and University Rennes 1, where he teaches a class in parallel computing.
Additional Resources

- Intel Xeon Phi Coprocessor Developer site and community:  http://software.intel.com/mic-developer

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

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

*Other names and brands may be claimed as the property of others

Copyright© 2013 Intel Corporation. All rights reserved.
Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Performance Notice

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

This sample source code is released under the Intel Sample Source Code License Agreement