SGI® GPU and Coprocessor Software Guide

007-6424-001

Record of Revision

Version	Description
001	September 2015 Initial printing.

Contents

Tables

About This Guide

This guide describes the installation and management of software for NVIDIA® GPUs and Intel® Xeon Phi™ devices on SGI® systems. The SGI systems include the following:

- SGI clusters managed by SGI Management Center™ (SMC) 3.*x* or later.
- SGI UV™ systems and other standalone systems (for example, SGI Rackable™ servers)

Audience

This guide is written for the system administrators and software developers. The guide assumes the reader is familiar with Linux software installation and clusters.

Related Publications

The following SGI documents might be helpful:

- *SGI Foundation Software (SFS) User Guide* (007-6410-xxx)
- *SGI Management Center (SMC) Installation and Configuration Guide for Clusters* (007-6359-xxx)
- *SGI Management Center (SMC) Administration Guide for Clusters* (007-6358-xxx)

You can obtain SGI documentation in the following ways:

- Refer to the SGI Technical Publications Library (TPL) at http://docs.sgi.com. Various formats are available. The TPL contains the most recent and most comprehensive set of books and man pages.

  To get the latest revision of a document on the TPL, use the core publication number as your search string. For example, use `007-1234` as your search string to get the latest version of the document with part number 007-1234-xxx.
- Refer to the SGI support webpage for release notes and other documents whose access require a support contract. See "Product Support" on page xiii.

Note: For information about third-party system components, see the documentation provided by the manufacturer/supplier.

Product Support

SGI provides a comprehensive product support and maintenance program for its products. SGI also offers services to implement and integrate Linux applications in your environment.

- Refer to http://www.sgi.com/support/
- If you are in North America, contact the Technical Assistance Center at +1 800 800 4SGI or contact your authorized service provider.
- If you are outside North America, contact the SGI subsidiary or authorized distributor in your country.

Be sure to have the following information before you call Technical Support:

- Product serial number
- Product model name and number
- Applicable error messages
- Add-on boards or hardware
- Third-party hardware or software
- Operating system type and revision level

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, contact SGI. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address: techpubs@sgi.com
- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

  http://www.sgi.com/support/supportcenters.html

SGI values your comments and will respond to them promptly.

Downloading NVIDIA® Software and Building RPMs

This chapter describes how to download NVIDIA driver software and how to package that software into RPMs for easy installation.

Downloading the Software

The NVIDIA software needed to use your GPUs both for graphics and as compute accelerators consists of the following:

- A driver
- The CUDA Toolkit
- CUDA SDK samples (optional)

Downloading a Driver

You can use one of several methods to download the NVIDIA software:

- "Specifying a GPU Hardware Type" on page 2
- "Selecting a Version from an Archive" on page 2
- "Using FTP to Download a Specific Version" on page 2

In all cases, the result of the download is file `NVIDIA-Linux-x86_64-`*version*`.run`.

Specifying a GPU Hardware Type

The NVIDIA **DRIVERS DOWNLOAD** webpage allows you to select and download the current driver for a GPU hardware type:

http://www.nvidia.com/Download/index.aspx?lang=en-us

Use `Linux 64-bit` for the operating system.

Selecting a Version from an Archive

The NVIDIA **Unix Driver Archive** webpage allows you to select and download a driver version from a list of current or archived versions:

http://www.nvidia.com/object/unix.html

Choose a version from the **Linux x86_64/AMD64/EM64T** category.

Using FTP to Download a Specific Version

If you know the specific version you want to download, use the following FTP site:

ftp://download.nvidia.com/XFree86/Linux-x86_64/

Downloading the CUDA Toolkit and CUDA SDK Samples

To download the CUDA software, use the following webpage:

https://developer.nvidia.com/cuda-downloads

Click the **Linux86** tab and download a run file installer for your Linux distribution. An example download file would be `cuda_7.0.28_linux.run`.

This run file actually contains three run files: one for the toolkit, one for the samples, and one for a driver. To extract the individual run files in this example, type the following:

`./cuda_7.0.28_linux.run --extract` *target-directory*

This command creates the following three files in *target-directory*:

```
cuda-linux64-rel-7.0.28-19326674.run
cuda-samples-linux-7.0.28-19326674.run
NVIDIA-Linux-x86_64-346.46.run
```

You may discard the driver file, `NVIDIA-Linux-x86_64-346.46.run`, unless it happens to be the one you want to install.

Building RPMs

Most of the installation procedures in this guide require software packaged in RPMs, not run files. SGI provides the `sgi-package-nvidia` script to build the necessary RPMs. This script also includes an `sgi-install-nvidia` script as its post-install scriptlet. If you want, you can edit this latter script to customize the standard install procedure.

Accessing the `sgi-package-nvidia` Script

The `sgi-package-nvidia` script is part of the SGI Foundation Software, starting with release 2.13. This script will be in `/opt/sgi/bin`. You can also access this script with the following command:

```
wget ftp://oss.sgi.com/www/projects/libnuma/download/sgi-package-nvidia
```

Creating the RPMs from Run Files

The `sgi-package-nvidia` script requires that input run files for the driver and CUDA adhere to following naming conventions:

Run File Type	**Naming Convention**
Driver	`NVIDIA-Linux-x86_64-`*release*`.run`
CUDA toolkit	`cuda-linux64-rel-`*release*`.run`
CUDA samples	`cuda-samples-linux-`*release*`.run`

In the file name conventions, the *release* item represents the numerical release version.

To create an RPM file from a run file, use the following command:

`sgi-package-nvidia --distro=`*linux-distro runfile*

You can specify `rhel6`, `rhel7`, `sles11`, or `sles12` for *linux-distro*. The `--distro=` option will default to the Linux distribution on the system where you are running.

- Example build of a driver RPM for RHEL 7

`sgi-package-nvidia --distro=`*`rhel7`* `NVIDIA-Linux-x86_64-`*`346.59`*`.run`

  This command creates file `nvidia-346.59-rhel7.x86_64.rpm`.

- Example build of a CUDA toolkit RPM for SLES 11

`sgi-package-nvidia --distro=sles11 cuda-linux64-rel-`*`7.0.28`*`.run`

  This command creates file `cudatoolkit-7.0.28-sgi.sles11.x86_64.rpm`.

- Example build of a CUDA samples RPM for SLES 12

`sgi-package-nvidia --distro=sles12 cuda-samples-linux-`*`7.0.28`*`.run`

  This file `gpucomputing_sdk_samples-`*`7.0.28`*`-sgi.sles12.x86_64.rpm` is created.

Using Additional Options of the `sgi-package-nvidia` Script

The previous section described the use of the `--distro=` option of the `sgi-package-nvidia` script. The script has three other options that you may want to use. These options affect how the RPM is built and, therefore, apply to every installation of the RPM. Table 1-1 describes all of the options for the script.

Table 1-1 Options for the `sgi-package-nvidia` Script

Option	Description
`--distro=`	Specifies the Linux distribution for your installation environment. Use `rhel6`, `rhel7`, `sles11`, or `sles12`. The `--distro=` option will default to Linux distribution on the system where you are running.
`--world`	Opens up world read/write on the NVIDIA device. Installing the RPM appends the following to the file `/etc/modprobe.d/nvidia-0666.conf`: `options nvidia NVreg_DeviceFileMode=0666`
`--nodisplay`	Informs X that there are no monitors connected to the GPUs. Installing the RPM produces the same effect as the following: `nvidia-xconfig --use-display-device=NONE`

`--nopersist`	Starting the `nvidia-persistenced` daemon will not set the GPUs into persistence mode; that is, the `nvidia-persistenced` daemon will not be started with the `--persistence-mode` option on its command line. If you want to add the option later, edit the following file:
	
	RHEL 6 `upstart`:
	`/etc/init/nvidia-persistenced.conf`
	RHEL 7, SLES 12 `systemd`:
	`/usr/lib/systemd/system/nvidia-persistenced.service`
	SLES 11 System V:
	`/etc/init.d/nvidia-persistenced`

Installing NVIDIA Software in a Cluster Image

This chapter describes the installation of NVIDIA software in a compute image of a cluster managed by SGI Management Center™ (SMC) 3.*x* or later. Specifically, this chapter describes the installation of a GPU driver, the CUDA toolkit, and CUDA sample code.

The following are the general steps:

1. “Select a Compute Image as a Base Image.” on page 8
2. “Clone the Base Image.” on page 9
3. “Install Linux Distribution Packages into the Image.” on page 9
4. “Create RPMs from NVIDIA Run Files.” on page 10
5. “Install the NVIDIA Software from the RPMs.” on page 11
6. “Make Any Custom Modifications to the New Image.” on page 12
7. “Push the Image to the Desired Compute Nodes.” on page 12

Note: This chapter describes the installation of the software via RPMs. If you want to manually install the NVIDIA run files, see Appendix A, “Manually Installing NVIDIA Software (RHEL)”, and Appendix B, “Manually Installing NVIDIA Software (SLES)”.

Select a Compute Image as a Base Image.

On the admin node, display the existing images and their associated kernel(s):

```
# cinstallman --show-images
Image Name                                  BT VCS   Compat_Distro
rhel7.1                                     0  1     rhel7
        3.10.0-229.el7.x86_64
        0-rescue-f80b1858d9f44f1b8cc667b2004fcb00
ice-rhel7.1                                 0  1     rhel7
        3.10.0-229.el7.x86_64
        0-rescue-62c9c0ed8f5747bfb49fa8b76efa69ef
lead-rhel7.1                                0  1     rhel7
        3.10.0-229.el7.x86_64
        0-rescue-c1f4b3df21ed4bb2b0af2d7382729fe7
```

The preceding example shows images for an SGI ICE X cluster, where the `rhel7.1` image is for a flat compute node (legacy *service* node), the `ice-rhel7.1` image is for an SGI ICE compute node, and `lead-rhel7.1` is for a leader node.

If you have a large repository of images and want to view them in a sorted manner, the following sequence of command might be helpful:

```
cd /var/lib/systemimager/images
find . -maxdepth 3 | grep sgi-compute-node
find . -maxdepth 3 | grep sgi-lead-node
find . -maxdepth 3 | grep sgi-service-node
```

For an SGI Rackable cluster (flat cluster), select an image for a flat compute node (legacy service node). For an SGI ICE X cluster, select an image for an SGI ICE compute node.

Clone the Base Image.

To create a clone of a base image, use the `cinstallman` command. For example, the following clones a base image named `rhel6.6` to a test image named `nvidia1` for a flat compute node:

```
BASE=rhel6.6
IMG=nvidia1
cinstallman --create-image --clone --source $BASE --image $IMG
```

Note: For convenience, this section will use these environment variable definitions going forward.

Install Linux Distribution Packages into the Image.

You need to ensure that the new image contains required source from your Linux distribution like kernel software and miscellaneous packages needed for booting images and building the NVIDIA software. If required files are missing, use the `cinstallman` command to insert them.

Kernel Software

Ensure that the new image contains the following:

RHEL 6 and RHEL 7:

`kernel-`*version*
`kernel-devel-`*version*

SLES 11:

`kernel-default-base-`*version*
`kernel-default-`*version*
`kernel-source-`*version*
`kernel-default-devel-`*version* (optional)

SLES 12:

`kernel-default-`*version*
`kernel-devel-`*version*
`kernel-source-`*version*
`kernel-default-devel-`*version* (optional)

The following example ensures that the RHEL kernel source files are present:

```
# cinstallman --yum-image --image $IMG install kernel-devel
```

Miscellaneous Packages

The build of the NVIDIA driver and CUDA sample codes or the boot of the compute image may fail if certain prerequisite packages are not installed. Ensure that the following prerequisite packages are installed:

`device-mapper-libs` (RHEL 6, for building SDK samples)
`gcc`
`glibc`
`freeglut-devel` (for building SDK samples)

To install a package, use the following command:

```
# cinstallman --yum-image --image $IMG install package
```

Create RPMs from NVIDIA Run Files.

The `sgi-package-nvidia` script packages the NVIDIA run files as RPMs. For a description of how to build the RPMs with this script and file naming conventions for the RPMs, see “Building RPMs” on page 3.

Install the NVIDIA Software from the RPMs.

You install the driver, CUDA toolkit, and CUDA samples from the RPMs just as you would install any other product from an RPM:

1. On the admin node, copy the newly created RPMs into a directory dedicated to the RPMs.

   For example, if your chosen repository for the RPMs is `/tftpboot/nvidia`, use the following commands:

   ```
   # cp driver-rpm /tftpboot/nvidia
   # cp cudatookit-rpm /tftpboot/nvidia
   # cp cuda-samples-rpm /tftpboot/nvidia
   ```

2. Formally register your chosen directory in the installation repository with the `crepo` command and select it:

   ```
   # crepo --add /tftpboot/nvidia --custom repo-name
   # crepo --select repo-name
   ```

   For *repo-name*, use some descriptor for the directory of RPMs you are adding–e.g., `nvidia-upgrade`.

3. Install the NVIDIA software into the image with the following commands:

```
# cinstallman --yum-image --image $IMG install nvidia
# cinstallman --yum-image --image $IMG install cudatoolkit
# cinstallman --yum-image --image $IMG install gpucomputing_sdk_samples
```

   Note that the package names match the first part of the names of the RPMs. See "Creating the RPMs from Run Files" on page 4.

Make Any Custom Modifications to the New Image.

If you need to make some custom modifications to the new image, you must do so in a `chroot` environment. The location of the image is `/var/lib/systemimager/images/$IMG`.

To make the modifications, do the following:

1. Go to the chroot environment:

```
# chroot /var/lib/systemimager/images/$IMG
```

2. Make the desired modifications to the files in that environment.

3. Exit that environment:

```
# exit
```

Push the Image to the Desired Compute Nodes.

Push the new image to the desired compute nodes as you would with any new image, as follows:

For a flat compute node image:

1. (optional) Be sure the miniroot builds successfully:

```
# cinstallman --update-miniroot --image $IMG
```

2. Blacklist any nouveau driver in the image (from the boot command line):

```
# cadmin --image $IMG --set-kernel-extra-params rdblacklist=nouveau
# cinstallman --refresh-netboot --node target-nodes
```

3. Associate the new image and boot kernel to the nodes:

```
# cinstallman --assign-image --node target-nodes --image $IMG \
--kernel 2.6.32-504.el6.x86_64
```

   This examples uses 2.6.32-504.el6.x86_64 for the boot kernel.

4. Finish the push by setting up for network boot:

```
# cinstallman --next-boot image --node target-nodes --transport udpcast
# cpower node cycle target-nodes
```

5. To watch boot progress on a node:

```
# console target-node
```

For an SGI ICE compute node image:

1. (optional) Be sure the miniroot builds successfully:

cinstallman --update-miniroot --image $IMG

2. Blacklist any nouveau driver in the image (from the boot command line):

cadmin --image $IMG --set-kernel-extra-params rdblacklist=nouveau
cinstallman --refresh-netboot --node *target-nodes*

3. Push the image to the affected leader nodes:

cimage --push-rack $IMG "r*"

4. Assign the image and boot kernel to the desired nodes:

cimage --set $IMG 2.6.32-504.el6.x86_64 *target-nodes*

   This examples uses 2.6.32-504.el6.x86_64 for the boot kernel.

5. Finish the push by setting up for network boot:

cpower node cycle *target-nodes*

6. To watch boot progress on a node:

console *target-node*

Installing NVIDIA Software on a Standalone System

This chapter describes how to install an NVIDIA driver, the CUDA toolkit, and CUDA code samples on SGI standalone (non-clustered) systems. The standalone systems include the following:

- SGI UV™ 2000 systems
- SGI UV™ 3000 systems
- SGI UV™ 300 systems
- SGI UV™ 20 systems
- SGI UV™ 30 systems
- SGI Rackable™ servers

The chapter includes the following sections:

Installing NVIDIA Software from SGI RPMs

Perform the following steps to install the NVIDIA software:

1. Create RPMs from NVIDIA run files.

   The `sgi-package-nvidia` script packages the NVIDIA run files as RPMs. For a description of how to build the RPMs with this script and file naming conventions for the RPMs, see "Building RPMs" on page 3.

2. Use `yum` or `zypper` to install the RPMs:

   ```
   # yum install driver-rpm
   # yum install cudatoolkit-rpm
   # yum install cuda-samples-rpm
   ```

   The driver RPM uses the `sgi-install-nvidia` script as its post-install scriptlet.

   The CUDA samples are optional but are useful for testing the system.

Customizing the Installation

This section describes two alternatives to installing the NVIDIA software from RPMs:

- Executing NVIDIA run files
- Customizing the install script

Executing NVIDIA Run Files

An NVIDIA run file can be executed directly, and it will install the driver in your system. However, the run file will not create weak modules for multiple kernels, nor will it install the `nvidia.persistenced` daemon as does the `sgi-install-nvidia` script.

Customizing the Install Script

You can install the run files yourself in a custom manner and still retain the benefits just cited of the script. You can do so by tweaking and executing the `sgi-install-nvidia` script:

1. Extract the `sgi-install-nvidia` script from the `sgi-package-nvidia` script in the following manner:

   ```
   # ./sgi-package-nvidia -s
   ```

2. Customize the install script.

   Note that the `/opt/sgi/Factory-install/nvidia` is the default location of the run file. The install script will look for the run file there.

3. Execute `sgi-install-nvidia` in this way:

   ```
   # ./sgi-install-nvidia --pkg nnn.nn install
   ```

   The *nnn.nn* is the version of the NVIDIA driver, which is part of the name of the run file.

Using the `sgi-update-x` Script

The `sgi-update-x` script, which is typically in `/opt/sgi/bin`, can do any of the following:

- Update the X configuration for your GPUs.
- Query and download a driver.
- Install or upgrade a driver.
  For installing a driver, this script is superceded by the `sgi-package-nvidia` tool. For an upgrade of a driver, remove the old driver and install the new one.

The `sgi-update-x` script also assumes that the NVIDIA run file has been installed by an SGI RPM into directory `/opt/sgi/Factory-Install/nvidia/` as `NVIDIA-Linux-x86_64-<VER>.run.`

A typical usage:

```
sgi-update-x -u -n -l :0
```

Option	Description
`-u`	Update the xorg configuration file (modifies `/etc/X11/xorg.conf`).
`-n`	There is no display attached on all GPUs.
`-l :0`	Use ServerLayout name `:0`. `:0.`*n* to GPU *n*.

Use `sgi-update-x -h` to see all the options.

Managing Your X Configuration

You can use the `sgi-update-x` options in Table 3-1 to manage your X configuration.

Table 3-1 `sgi-update-x` Options–Managing X Configurations

Option	Description
`-I --new-install`	Performs initial installation of the driver and X (implies `-d -i -u`).
`-u --update`	Updates NVIDIA xorg config file. Causes current options in `xorg.conf` to be added/modified/removed back to their defaults.
`--display=:`*N*	Specifies X Server (as in :*N*.*x*).
`-l --layoutname=`*X*	Specifies ServerLayout name. Typical entry is `:0`. Can also be set with environment variable `SGI_UPDATE_X_LAYOUT_NAME`.
`--layout=`*X*	Do update operation only on Layout *X*.
`-d --device=`*X*	Do update operation only on Device*X*.
`--screen=`*X*	Do update operation only on Screen*X*.
`--mode=`	Specifies display mode.
`--modedebug`	Turns on mode debug flag.
`--virtual=`*W*`x`*H*	Specifies desktop virtual size.
`--separate`	Specifies separate X screens for each GPU. Duplicates each device and screen entry to (2nd) entry.
`--twinview=`*X*	Specifies twinview type: `RightOf`, `LeftOf`, `Above`, `Below`, or `Clone.`
`--twc`	Shortcut for `--twinview=clone`
`-n --nodisplay`	Indicates there is no display attached on all GPUs. Use `--device=` or `--screen=` for setting only single GPU.
`--nv_opt=option`	Adds a specific NVIDIA option to `xconfig`. Can occur multiple times.
`--newconf`	Creates a new `/etc/X11/xorg.conf` instead of using the current one.

Querying Drivers

You can use the `sgi-update-x` options in Table 3-2 to query the present driver or obtain a new driver package. However, use the procedure in "Downloading a Driver" on page 1 to select a new driver.

Table 3-2 `sgi-update-x` Options–Querying Drivers

Option	Description
`-q --query`	Shows currently downloaded NVIDIA package versions (installed, latest and downloaded).
`--query-inst`	Shows currently installed NVIDIA package.
`-Q --ftpquery`	Shows latest available NVIDIA version.
`-d --download`	Downloads latest version from NVIDIA ftp site.
`--file=`*X*	Downloads file *X* instead of the latest from NVIDIA ftp site. Supported transports: `ftp`: `http`: `user@host:[scp]`, `cp.` Can be specified as `ftp`:*version*.

Miscellaneous Options

Table 3-3 describes some miscellaneous `sgi-update-x` options.

Table 3-3 `sgi-update-x` Options–Miscellaneous

Option	Description
`-x --extract-only`	Extract contents of package *XXXXXX*.run file to directory *XXXXXX*.
`--blacklist`	Blacklists the nouveau driver in `/etc/modprobe.d/blacklist-nouveau.conf` and in `/etc/sysconfig/uvconfig`.
`--clean`	Clean up and restore modified files using command `remove_sysconf_files`.
`-D --debug`	Use verbose debug messages.

Installation Options (deprecated)

Use the `sgi-package-nvidia` tool to create RPMs for use in installing or uninstalling an NVIDIA driver. Table 3-4 describes the old `sgi-update-x` alternatives to the new tool.

Table 3-4 `sgi-update-x` Options–Installation

Option	Description
`-i --install`	Installs downloaded NVIDIA package.
`--pkg=`	Specifies NVIDIA package name.
`-A --accept`	Automatically accepts NVIDIA licensing terms during install.
`-k`	Updates kernel module for current kernel.
`--kernel=`*kernel*	Updates kernel module *kernel*.
`--kip=`*PATH*	Uses *PATH* as kernel install path.
`--keep`	Keeps current version of NVIDIA driver.
`-f --force`	Forces update of kernel module even if no GPUs are present.
`-S --standard`	Installs standard (non-patched) version.
`--patch=`	Specifies patch for kernel driver.

Using the `nvidia-xconfig` Tool

Systems with NVIDIA GPUs but also with built-in VGA adapters may fail to start the X server when X is initially configured with `nvidia-xconfig` or `SaX`. In log file `/var/log/Xorg.0.log`, you might see the following entry:

```
Fatal server error:
no screens found
```

X attempts to start with the driver specified (`nvidia`) but tries to start on the built-in VGA adapter. To prevent this behavior, X must use a specific BusID for each card. To find the BusID, enter the following command:

```
# nvidia-xconfig --query-gpu-info
GPU #0: Name : Quadro K6000 UUID :
GPU-94b0aece-7bf4-3e1a-d138-5ce52986bba7 PCI BusID : PCI:2@3:0:0
```

You instruct X to use the PCI BusID with the following command:

```
# nvidia-xconfig --allow-empty-initial-configuration --busid=PCI:2@3:0:0
```

This will set up a new `xorg.conf` with a single screen section. It should enable you to start X even with no monitor attached (`allow-empty-initial-configuration`). Once started, you can use the `nvidia-settings` command to configure the running X server for additional options, screens, monitors, etc.

Defining More Complicated X Configurations

You can use the `nvidia-settings` command to define more complicated X configurations such as for Xinerama or Mosaic. The configuration file is `/etc/X11/xorg.conf`. The `nvidia-settings` command provides a convenient GUI to modify this file. The following command shows the GUI on your local display:

```
# nvidia-settings -c :0
```

The command has a `-help` option that displays all available options.

Figure 3-1 shows the opening screen for the `nvidia-settings` GUI.

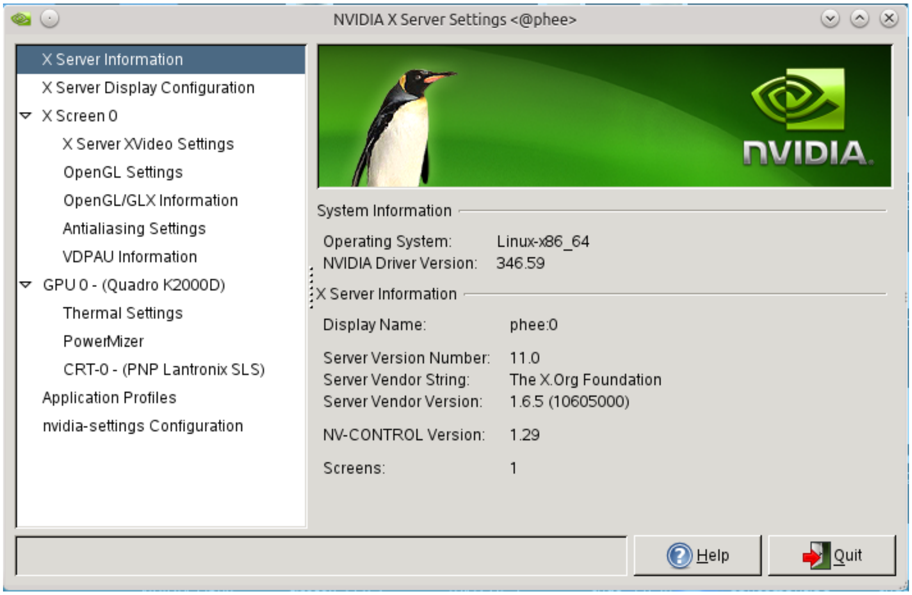

Figure 3-1 Opening Screen for `nvidia-settings` GUI

After you make a change to the configuration file, you must restart X in order to effect the changes. To restart X, perform the following steps:

1. Kill the X program:

`killall X`

2. If X persists, stop `gdm` or `xdm`. For example:

`kill -9` *pid*

The pid is the process ID for `gdm` or `xdm`.

3. Restart X.

`startx &`

SLES 11 Workaround for a Problem When Starting X

If you have permission problems with the file `/var/lib/gdm/.ICEAuthority` when starting X, you can try the following:

```
# chown gdm:gdm /var/lib/gdm
```

Upgrades and Tuning

This chapter describes the following topics:

- "Upgrading to New NVIDIA Software" on page 25
- "Upgrading New NVIDIA Software after Upgrading to a New Kernel" on page 26
- "Performance Tuning" on page 28

Upgrading to New NVIDIA Software

This section describes how to upgrade to new NVIDIA software on a cluster manged by SGI Management Center (SMC) 3.x or later, on an SGI UV system, or on another SGI standalone system.

Note: This section assumes that there has been no change to the kernel.

SMC Cluster

To upgrade your NVIDIA software on an SMC cluster, perform the following steps:

1. Remove the old packages from the repository. For example:

```
# cinstallman --yum-image --image $IMG remove nvidia
# cinstallman --yum-image --image $IMG remove cudatoolkit
# cinstallman --yum-image --image $IMG remove gpucomputing_sdk_samples
```

2. Create new RPMs for the new software.

   See "Building RPMs" on page 3.

3. Install the new software. For example:

```
# cinstallman --yum-image --image $IMG install nvidia
# cinstallman --yum-image --image $IMG install cudatoolkit
# cinstallman --yum-image --image $IMG install gpucomputing_sdk_samples
```

SGI UV and Other Standalone Systems

To upgrade the NVIDIA software on an SGI UV or other standalone system using the `sgi-nvidia-install` tool, you use the same procedure as with an initial installation, except you must first remove the old driver software.

Perform the following steps to upgrade:

1. Remove the old packages with `yum` (or `zypper`)–for example:

   ```
   # yum remove old-driver-rpm
   # yum remove old-cudatoolkit-rpm
   # yum remove old-cuda-samples-rpm
   ```

2. Install the new packages `yum` (or `zypper`)–for example:

   ```
   # yum install new-driver-rpm
   # yum install new-cudatoolkit-rpm
   # yum install new-cuda-samples-rpm
   ```

Upgrading New NVIDIA Software after Upgrading to a New Kernel

As a general rule, you will probably want to use the latest supported version of the NVIDIA software after upgrading to a new kernel.

Latest Supported NVIDIA Driver Software

For the latest NVIDIA driver, use the instructions from section "Selecting a Version from an Archive" on page 2. Select the version labeled as follows:

Latest Long Lived Branch version

Download and package that version as described in Chapter 1, "Downloading NVIDIA® Software and Building RPMs."

Latest Supported CUDA Toolkit and Samples

Confirm that the latest CUDA software is supported by your Linux distribution. See "Downloading the CUDA Toolkit and CUDA SDK Samples" on page 3.

If the desired CUDA software is supported, download and package that version as described in Chapter 1, "Downloading NVIDIA® Software and Building RPMs."

Upgrading on an SMC Cluster

To upgrade your NVIDIA software on an SMC cluster, use the same procedure as with an initial installation. See Chapter 2, "Installing NVIDIA Software in a Cluster Image."

Upgrading on an SGI UV or Other Standalone System

To upgrade the NVIDIA software on an SGI UV or other standalone system using the `sgi-nvidia-install` tool, perform the following steps:

1. Remove the old packages with `yum` (or `zypper`)–for example:

   # **yum remove** *old-driver-rpm*
   # **yum remove** *old-cudatoolkit-rpm*
   # **yum remove** *old-cuda-samples-rpm*

2. Upgrade to the new kernel

3. Install the new packages `yum` (or `zypper`)–for example:

   # **yum install** *new-driver-rpm*
   # **yum install** *new-cudatoolkit-rpm*
   # **yum install** *new-cuda-samples-rpm*

Performance Tuning

This section describes performance tuning topics pertinent to all SGI systems and topics pertinent to only specific SGI systems.

Error-Correcting Code (ECC) Memory

You can run both Tesla™ and Quadro™ GPUs with ECC enabled or disabled. By default, Tesla GPUs are configured with ECC enabled to accommodate CUDA applications, where accuracy is vital. By default, Quadro GPUs are configured with ECC disabled to accommodate graphics, where a few wrong pixels may be inconsequential. Since ECC consumes about 6% of a GPU's memory, performance can be enhanced with ECC disabled.

If there is reason to suspect that your GPUs are not in the desired ECC state, you can query their state using the following command:

```
# nvidia-smi -q
...
    Ecc Mode
     Current      :Enabled
     Pending      :Enabled
```

You can enable/disable ECC with the following command:

```
# nvidia-smi -e n
```

A value of 0 for *n* disables ECC mode and a value of 1 enables ECC mode. When ECC mode is toggled, a reboot is required to put the new state into effect. The `nvidia-smi -e` *n* command will note that a reboot is required. Once set, the ECC mode will persist across reboots.

Tuning on SGI UV Systems

This section describes how you can affect system performance on SGI UV systems in the following areas:

- "Avoiding Long Startup Delays" on page 29
- "Addressing Large Memories" on page 29
- "Maximum PCIe Bandwidth" on page 30

Avoiding Long Startup Delays

To avoid long startup delays, you must ensure that persistence mode is turned on. You can turn on persistence mode at installation time with the following command:

```
# nvidia-smi -pm 1
```

The installation of the NVIDIA driver also installs the `nvidia-persistenced` daemon, which will enable persistence mode but only after a reboot.

Addressing Large Memories

If an SGI UV system has an address space larger than 1TB the IOMMU must be enabled. This is because the NVIDIA GPUs have only a 40-bit maximum address. Without the IOMMU, programs will fail or hang randomly as the GPUs attempt to address buffers above the 1TB limit. The GPUs will be addressing the wrong memory.

To query the address space size, enter the following commands:

```
# sudo topology -v -s | grep Partition
```

The following subsections show how to alter the boot loader command line to enable the IOMMU. In each case, the machine must be rebooted after making the adjustment.

Automatically Enabling the IOMMU on Any Distribution

Use the `uvconfig` or `nvidia_config` file to set up the boot line to turn on the IOMMU automatically. In file `/etc/sysconfig/uvconfig`, use the following setting to enable the IOMMU automatically:

```
UV_UPDATE_IOMMU=”yes”
```

Use the following setting in that file to disable the IOMMU: (This is not recommended for use in production.)

```
UV_UPDATE_IOMMU=”no”
```

Manually Enabling the IOMMU on SLES 11 sp3

To ensure that the IOMMU is enabled at boot time on SLES 11 sp3, add the options `intel_iommu=on iommu=pt` to the boot line in file `/etc/elilo.conf`.

Manually Enabling the IOMMU on RHEL 6

To ensure that the IOMMU is enabled at boot time on RHEL 6, add the options `intel_iommu=on,forcedac,pt64` to the boot line in file `/boot/grub/menu.lst`.

Manually Enabling the IOMMU on RHEL 7 or SLES 12

To ensure that the IOMMU is enabled at boot time on RHEL 7 and SLES 12, add the options `intel_iommu=on iommu=pt` to the boot line in file `/etc/grub.d/42_sgi.last` and enter the following command:

```
# grub2-mkconfig
```

Maximum PCIe Bandwidth

On SGI UV systems, each processor's C1E bit in `MSR_POWER_CTL` must be cleared to achieve maximum PCIe bandwidth. This bit is cleared automatically in SLES 12 and RHEL 7 and can be cleared by service `sgi-perf` in SLES 11 and RHEL 6. See the man page for `sgi-perf`.

The following command should show bit 1 as `0` (2nd bit from the right):

```
# rdmsr --hexadecimal 0x1fc
2504005a
```

In this example, the hex `a` is 1010. So, C1E is on.

Managing Intel® Xeon Phi™ Software

This chapter describes the following topics:

Components of the Intel Xeon Phi Software Stack

The Intel Xeon Phi software stack consists of the following software:

- Intel Manycore Platform Software Stack (MPSS)
- Intel compilers with the following:
  - Native Xeon Phi code targeting
  - Support for Intel Language Extensions for Offload (LEO)
  - Support for accelerators via OpenMP language constructs
- Intel MPI with Xeon Phi support

The Intel MPSS software can be freely downloaded, but Intel compilers and Intel MPI must be purchased from SGI or from Intel directly.

Downloading Intel MPSS Software

Note: The MPSS software can be freely downloaded, but software support entitlement is through the Intel compiler support license.

You can download the software from the following website:

https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss

Download the following MPSS components:

- MPSS tar file for Linux
- MPSS readme file
- MPSS Software for Coprocessor OS
- `releasenotes-linux.txt`
- MPSS user guide in PDF form
- Intel MPSS Licensing Agreement (text file)

Installing MPSS on Standalone Systems with Intel Xeon Phi Devices

The MPSS software can be installed and configured in one of two general configurations:

- MPSS basic configuration
  - Supports LEO offload via compiler directives.
  - Supports OpenMP offload constructs.
  - Supports direct login to Xeon Phi devices.
  - Omits the MPSS OFED RPMs and is, therefore, compatible with any version of OFED.
- MPSS software-rich configuration

  This configuration supports what is available in the basic configuration plus the following features:
  - Provides DAPL provider layers on Xeon Phi devices optimized for InfiniBand.
  - Provides local network access to Xeon Phi devices on other nodes.
  - Includes some RPMs that replace components of OFED.
  - Is compatible with specified OFED versions —community OFED and Mellanox® OFED.

The detailed information about installing MPSS is described in the readme document that is downloaded with MPSS. The MPSS user guide describes the other aspects of system configuration you will need.

Installing MPSS on Standalone Systems with Intel Xeon Phi Devices

The basic MPSS installation instructions in the MPSS readme file and MPSS user guide work only on a system that has Xeon Phi devices attached. In an SGI Management Center `chroot` image, the Xeon Phi devices will not be found. Therefore, you must take some special steps to delay the Xeon Phi configuration until after boot time. You must place those configuration steps described in the readme file in a system script that executes on every boot. This will ensure that the Xeon Phi devices will automatically be discovered and configured on every system boot.

Upgrading Intel MPSS Software

Intel MPSS software is upgraded by performing an uninstall of the current version of MPSS followed by a fresh install of the new version of MPSS. See the MPSS readme file for the uninstall procedure.

Performance-Tuning Xeon Phi Devices

This sections describes performance tuning for Xeon Phi devices pertinent for all systems and topics for specific systems.

Local Device Assignment (all systems)

Data transfers between the host system and Xeon Phi devices can be more efficient if care is taken to co-locate the host threads and the Xeon Phi devices. This is important on all systems but is crucial on SGI UV300 and UV2000 systems. The `nt_mic_local_device`(3) function in library `libnumat` in the Numatools RPM identifies the Xeon Phi device that is connected to the CPU socket to which the calling thread is pinned. See the `nt_mic_local_device`(3) and `nt_node_i_am_pinned_to`(3) man pages for details. Numatools is provided with SGI Performance Suite.

Local Device Assignment (UV300 and UV2000 systems only)

On SGI ccNUMA systems, it is very advantageous to run the host processes doing offloads on the same CPU sockets that are directly attached via PCIe links to the targeted Xeon Phi devices. One way to discover the NUMA nodes associated with Xeon Phi devices in the system is to use the SGI `topology` command as follows:

```
# topology -cops -v
```

Manually Installing NVIDIA Software (RHEL)

Note: You do not need to perform the procedures described in this appendix. They are performed automatically by the installation of the RPMs built with the `sgi-package-nvidia` tool. This appendix describes the procedures found in script `sgi-install-nvidia`, which you can extract in the following manner:

```
# sgi-package-nvidia -s
```

This appendix describes the installation of NVIDIA software in the following environment:

- In a compute image of a cluster managed by SGI Management Center (SMC) 3.*x* or later
- Using the NVIDIA run files versus RPMs provided by the `sgi-package-nvidia` tool
- Using a RHEL Linux distribution

Specifically, this chapter describes the installation of a GPU driver, the CUDA toolkit, and CUDA sample code.

This appendix describes the following steps:

1. “Perform the General Setup.” on page 36
2. “Make the NVIDIA Modules for the Kernel Source.” on page 36
3. “Make Symbol and Weak Module Updates.” on page 37
4. “Install Non-kernel Parts of the Driver.” on page 38
5. “Install the CUDA Toolkit and Sample Code” on page 40
6. “Set File Permissions for Use of CUDA” on page 41

Perform the General Setup.

Use the following steps:

1. Create a base image and modify it as necessary.

   See the first three steps in Chapter 2, "Installing NVIDIA Software in a Cluster Image."

2. Ensure that **`gcc`** is installed in the image.
3. Disable X.

   Use `init 3` to go to run level 3.

4. Ensure that the nouveau driver is not installed.
5. Remove any running NVIDIA module (e.g., `nvidia` or `nvidia-uvm`).

   A running `nvidia-persistenced` can be using the module.

6. Copy the driver run file to the image and `chroot` to the image:

```
# cp ./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}.run \
/var/lib/systemimager/images/image
```

```
# chroot /var/lib/systemimager/images/image
```

7. Expand the run file:

```
# sh ./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}.run -x
```

This creates directory `./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}.`

Make the NVIDIA Modules for the Kernel Source.

Follow these steps to make the NVIDIA modules for the kernel source.

1. Ensure that package `kernel-devel` is installed before proceeding.

   Check for the following files:

```
/usr/src/kernels/version/.config
/usr/src/kernels/version/include/linux/version.h
```

   If they do not exist, enter the following:

```
# cd /usr/src/kernels/version
# make oldconfig
# make init
```

2. Make the NVIDIA modules for kernel version *version*:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}/kernel
# export IGNORE_CC_MISMATCH=1
# export SYSSRC=/usr/src/kernels/version
# export SYSOUT=/usr/src/kernels/version
# make module
# cd uvm
# make module
```

3. Copy the modules into place:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}
# mkdir -p /lib/modules/version/video
# cp kernel/nvidia.ko /lib/modules/version/video/nvidia.ko
# cp kernel/uvm/nvidia-uvm.ko /lib/modules/version/video/nvidia-uvm.ko
```

Make Symbol and Weak Module Updates.

In directory `NVIDIA-Linux-x86_64-${NVIDIA_VERSION}`, the `/boot/symvers-xxx.gz` file for each target kernel needs `nvUvmInterface` symbols or else `nvidia-uvm.ko` is falsely seen as incompatible with compatible kernels.

1. Ensure that `kernel/uvm/Module.symvers` is built.

   For each kernel in `/lib/modules`, do enter the following in directory `NVIDIA-Linux-x86_64-${NVIDIA_VERSION}`:

```
# cp /boot/symvers-${KERN}.gz bootsymvers.gz
# gunzip bootsymvers.gz
# sort -u bootsymvers kernel/uvm/Module.symvers > /boot/symvers-${KERN}
# gzip -f /boot/symvers-${KERN}
```

2. Run `weak-modules` on `/lib/modules/nvidia.modules`:

```
# export WM=$WM:/sbin/weak-modules
# $WM --verbose --add-modules <<EOF
/lib/modules/version/video/nvidia-uvm.ko
/lib/modules/version/video/nvidia.ko
<EOF>
```

3. Make new module dependencies:

```
# depmod
```

Install Non-kernel Parts of the Driver.

To install the non-kernel parts of the driver, do the following:

1. Go to the directory with the NVIDIA driver source:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}
```

2. Invoke the NVIDIA installer:

```
#./nvidia-installer --ui=none --no-questions --run-nvidia-xconfig \
--accept-license --no-kernel-module
```

   The `NVIDIA-Linux-x86_64-${NVIDIA_VERSION}` directory can be removed.

3. Install the `nvidia-persistenced` daemon:

```
# cd /usr/share/doc/NVIDIA_GLX-1.0/sample
# tar xfj nvidia-persistenced-init.tar.bz2
# cd nvidia-persistenced-init
# ./install.sh
```

4. Adjust the `nvidia-persistenced` startup:

   - For systems using `systemd`:

     – Add `--persistence-mode` to the startup command line of `/usr/lib/systemd/system/nvidia-persistenced.service.`

     – Start the daemon:
     `#` **`service start nvidia-persistenced`**

   - For systems using System V:

     – Add `--persistence-mode` to the startup command line of `/etc/init.d/nvidia-persistenced.`

     – Start the daemon:
     `#` **`/etc/init.d/nvidia-persistenced start`**

   - For systems using `upstart`:

     – Add `--persistence-mode` to the startup command line of `/etc/init/nvidia-persistenced.conf.`

     – Start the daemon:
     `#` **`initctl start nvidia-persistenced`**

5. Exit from the `chroot:`

   `#` **`exit`**

Install the CUDA Toolkit and Sample Code

To install the CUDA tooklit and CUDA samples, do the following:

1. Copy `cuda-linux64-rel-`*cuda-version*`.run` to the image and `chroot` to the image:q

```
# cp cuda-linux64-rel-cuda-version.run \
/var/lib/systemimager/images/image
```

```
# chroot /var/lib/systemimager/images/image
```

2. Install `/usr/local/cuda` in the image:

```
# sh cuda-linux64-cuda-version.run -noprompt --nox11 \
-cudaprefix=/usr/local/cuda
```

The `-noprompt` option silences all prompts during the install and implies acceptance of the EULA. The `--nox11` option suppresses the creation of an xterm.

If you wish to install and make the CUDA sample codes, do the following:

1. Ensure that the image contain packages `freeglut-devel`, `gcc` and `gcc-c++`.
2. Copy `cuda-samples-linux-`*cuda-version*`.run` to your image.
3. Install the sample code:

```
# chroot /var/lib/systemimager/images/image
# sh cuda-samples-linux-cuda-version.run -noprompt --nox11 \
-cudaprefix=/usr/local/cuda
```

```
# export PATH=$PATH:/usr/local/cuda/bin
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64:/lib64:\
/usr/local/cuda/lib64:/usr/local/cuda/nvvm/lib64
# cd /usr/local/cuda/samples
# make -j 8
# exit
```

Set File Permissions for Use of CUDA

To use CUDA, users need to belong to Unix file group `video`.

To add the current user to that group, enter the following:

```
# usermod -A video $USER
```

Alternatively, you can make the device available to everyone by ensuring the file `/etc/modprobe.d/50-nvidia.conf` contains this entry:

```
options nvidia NVreg_DeviceFileUID=0 NVreg_DeviceFileGID=33 \
NVreg_DeviceFileMode=0666
```

This entry sets permissions on `/dev/fb*`. The GID of 33 is the `video` group. (see `/etc/group`). The mode specification of `666` allows all users.

Manually Installing NVIDIA Software (SLES)

Note: You do not need to perform the procedures described in this appendix. They are performed automatically by the installation of the RPMs built with the `sgi-package-nvidia` tool. This appendix describes the procedures found in script `sgi-install-nvidia`, which you can extract in the following manner:

```
# sgi-package-nvidia -s
```

This appendix describes the installation of NVIDIA software in the following environment:

- In a compute image of a cluster managed by SGI Management Center (SMC) 3.*x* or later
- Using the NVIDIA run files versus RPMs provided by the `sgi-package-nvidia` tool
- Using a SLES Linux distribution

Specifically, this chapter describes the installation of a GPU driver, the CUDA toolkit, and CUDA sample code.

This appendix describes the following steps:

1. "Perform the General Setup." on page 44
2. "Make the NVIDIA Modules for the Kernel Source." on page 45
3. "Make Symbol and Weak Module Updates." on page 46
4. "Install Non-kernel Parts of the Driver." on page 47
5. "Install the CUDA Toolkit and Sample Code" on page 49
6. "Set File Permissions for Use of CUDA" on page 50

Perform the General Setup.

Use the following steps:

1. Create a base image and modify it as necessary.

   See the first three steps in Chapter 2, "Installing NVIDIA Software in a Cluster Image."

2. Ensure that **gcc** is installed in the image.

3. Disable X.

   Use `init 3` to go to run level 3.

4. Ensure that the nouveau driver is not installed.

5. Remove any running NVIDIA module (e.g., `nvidia` or `nvidia-uvm`).

   A running `nvidia-persistenced` can be using the module.

6. Copy the driver run file to the image and `chroot` to the image:

```
# cp ./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}.run \
/var/lib/systemimager/images/image
```

```
# chroot /var/lib/systemimager/images/image
```

7. Expand the run file:

```
# sh ./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}.run -x
```

This creates directory `./NVIDIA-Linux-x86_64-${NVIDIA_VERSION}`.

Make the NVIDIA Modules for the Kernel Source.

Follow these steps to make the NVIDIA modules for the kernel source.

1. Ascertain the version of the kernel source to which the directory `/usr/src/linux` points.

   The `kernel-source` package creates directory `/usr/src/linux` as a symlink to the installed source tree. This will be the kernel version, *version*, used in subsequent steps.

2. Ensure that the following directories all contain entries for that version:

   ```
   /boot/symvers*
   /lib/modules/*
   /boot/config-version
   ```

   If not, remove the old packages and install the proper kernel and `kernel-source` packages.

   The `/usr/src/`*version*`-obj` directory may be present because of other administrator procedures on the directory. That is okay.

3. If the `/usr/src/`*version*`/.config` file does not exist, create it:

   ```
   # cd /usr/src/version
   # make oldconfig
   ```

4. Make enough of the kernel to provide the basic header files:

   ```
   # make init
   ```

5. Ensure that file `utsrelease.h` exists:

   The directory may be `/usr/src/`*version* or `/usr/src/`*version*`-obj`. One of the following should exist:

   ```
   x86_64/default/include/generated/utsrelease.h
   include/generated/utsrelease.h
   ```

   If neither exists, enter the following:

   ```
   # cd /usr/src/linux
   # make oldconfig
   # make init
   ```

6. Create directory `/lib/modules/`*version*`/build` if it does not exist by making a symlink to `/usr/src/`*version*:

   ```
   # ln -s /usr/src/version /lib/modules/version/build
   ```

   The module make uses this build directory.

7. If there is no `Module.symvers` entry in directory `/usr/src/`*version*, create it by doing the following:

```
# cp /boot/symvers-version.gz /usr/src/version/Module.symvers.gz
# gunzip /usr/src/version/Module.symvers.gz
```

8. Make the modules for kernel version *version* by entering the following:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}/kernel
# export IGNORE_CC_MISMATCH=1
# export SYSSRC=/usr/src/kernels/version
# export SYSOUT=/usr/src/kernels/version
# make module
# cd uvm
# make module
```

8. Copy the modules into place by entering the following:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}
# mkdir -p /lib/modules/version/kernel/drivers/video/nvidia
# cp kernel/nvidia.ko \
/lib/modules/version/kernel/drivers/video/nvidia/nvidia.ko
# cp kernel/uvm/nvidia-uvm.ko \
/lib/modules/versionID/kernel/drivers/video/nvidia/nvidia-uvm.ko
```

Make Symbol and Weak Module Updates.

In directory `NVIDIA-Linux-x86_64-${NVIDIA_VERSION`, the `/boot/symvers-xxx.gz` file for each target kernel needs `nvUvmInterface` symbols or else `nvidia-uvm.ko` is falsely seen as incompatible with compatible kernels.

1. Ensure that `kernel/uvm/Module.symvers` is built.

   For each kernel in `/lib/modules`, do enter the following in directory `NVIDIA-Linux-x86_64-${NVIDIA_VERSION`:

```
# cp /boot/symvers-${KERN}.gz bootsymvers.gz
# gunzip bootsymvers.gz
# sort -u bootsymvers kernel/uvm/Module.symvers > /boot/symvers-${KERN}
# gzip -f /boot/symvers-${KERN}
```

2. Run `weak-modules` on `/lib/modules/nvidia.modules`:

```
# export WM=$WM:/usr/lib/module-init-tools/weak-modules
# $WM --verbose --add-modules <<EOF
/lib/modules/version/kernel/drivers/video/nvidia/nvidia.ko
/lib/modules/version/kernel/drivers/video/nvidia/nvidia-uvm.ko
<EOF>
```

3. Make new module dependencies:

```
# depmod
```

Install Non-kernel Parts of the Driver.

To install the non-kernel parts of the driver, do the following:

1. Go to the directory with the NVIDIA driver source:

```
# cd NVIDIA-Linux-x86_64-${NVIDIA_VERSION}
```

2. Invoke the NVIDIA installer:

```
#./nvidia-installer --ui=none --no-questions --run-nvidia-xconfig \
--accept-license --no-kernel-module
```

The `NVIDIA-Linux-x86_64-${NVIDIA_VERSION}` directory can be removed.

3. Install the `nvidia-persistenced` daemon:

```
# cd /usr/share/doc/NVIDIA_GLX-1.0/sample
# tar xfj nvidia-persistenced-init.tar.bz2
# cd nvidia-persistenced-init
# ./install.sh
```

4. Adjust the `nvidia-persistenced` startup:

   - For systems using `systemd`:
     - Add `--persistence-mode` to the startup command line of `/usr/lib/systemd/system/nvidia-persistenced.service.`
     - Start the daemon:
       # **`service start nvidia-persistenced`**
   - For systems using System V:
     - Add `--persistence-mode` to the startup command line of `/etc/init.d/nvidia-persistenced.`
     - Start the daemon:
       # **`/etc/init.d/nvidia-persistenced start`**
   - For systems using `upstart`:
     - Add `--persistence-mode` to the startup command line of `/etc/init/nvidia-persistenced.conf.`
     - Start the daemon:
       # **`initctl start nvidia-persistenced`**

5. Exit from the `chroot:`

   ```
   # exit
   ```

Install the CUDA Toolkit and Sample Code

To install the CUDA tooklit and CUDA samples, do the following:

1. Copy `cuda-linux64-rel-`*cuda-version*`.run` to the image and `chroot` to the image:q

```
# cp cuda-linux64-rel-cuda-version.run \
/var/lib/systemimager/images/image
```

```
# chroot /var/lib/systemimager/images/image
```

2. Install `/usr/local/cuda` in the image:

```
# sh cuda-linux64-cuda-version.run -noprompt --nox11 \
-cudaprefix=/usr/local/cuda
```

The `-noprompt` option silences all prompts during the install and implies acceptance of the EULA. The `--nox11` option suppresses the creation of an xterm.

If you wish to install and make the CUDA sample codes, do the following:

1. Ensure that the image contain packages `freeglut-devel`, `gcc` and `gcc-c++`.
2. Copy `cuda-samples-linux-`*cuda-version*`.run` to your image.
3. Install the sample code:

```
# chroot /var/lib/systemimager/images/image
# sh cuda-samples-linux-cuda-version.run -noprompt --nox11 \
-cudaprefix=/usr/local/cuda
```

```
# export PATH=$PATH:/usr/local/cuda/bin
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64:/lib64:\
/usr/local/cuda/lib64:/usr/local/cuda/nvvm/lib64
# cd /usr/local/cuda/samples
# make -j 8
# exit
```

Set File Permissions for Use of CUDA

To use CUDA, users need to belong to Unix file group `video`.

To add the current user to that group, enter the following:

```
# usermod -A video $USER
```

Alternatively, you can make the device available to everyone by ensuring the file `/etc/modprobe.d/50-nvidia.conf` contains this entry:

```
options nvidia NVreg_DeviceFileUID=0 NVreg_DeviceFileGID=33 \
NVreg_DeviceFileMode=0666
```

This entry sets permissions on `/dev/fb*`. The GID of 33 is the `video` group. (see `/etc/group`). The mode specification of `666` allows all users.