White Paper

Intel Xeon Phi Coprocessor
DEVELOPER'S QUICK START GUIDE FOR WINDOWS* HOST

Version 1.1

Contents

Introduction ... **4**

Goals ... 4

This document does: ... *4*

This document does not: ... *4*

Terminology ... 4

System Configuration ... **5**

Intel Xeon Phi Software ... 6

Intel Many Integrated Core Architecture Overview ... **8**

Administrative Tasks ... **9**

Preparing Your System for First Use ... 9

Steps to install the driver and start the coprocessor ... *9*

Steps to install the Software Development tools ... *14*

Regaining Access to the Intel Xeon Phi Coprocessor after Reboot ... 15

Restarting the Intel Xeon Phi Coprocessor if it Hangs ... 16

Working directly with the uOS Environment Intel Xeon Phi Coprocessor ... 16

Host File System Share ... 20

Useful Administrative Tools ... 21

Getting Started/Developing Intel Xeon Phi Coprocessor Software ... **22**

Available SW development Tools / Environments ... 22

Development Environment: Available Compilers and Libraries ... *22*

Development Environment: Available Tools ... *22*

General Development Information ... 23

Development Environment Setup ... *23*

Documentation and Sample Code ... *23*

Build-Related Information ... *24*

Compiler Switches ... *24*

Debugging Offload Activity During Runtime ... *24*

Using the Offload Compiler – Explicit Memory Copy Model ... 25

Reduction ... *25*

Creating the Offload Version ... *26*

Asynchronous Offload and Data Transfer ... *27*

Using the Offload Compiler – Implicit Memory Copy Model ... 27

Native Compilation ... 27

Parallel Programming Options on the Intel Xeon Phi coprocessor ... 28

*Parallel Programming on the Intel Xeon Phi coprocessor: OpenMP** *29*
Parallel Programming on the Intel Xeon Phi coprocessor: OpenMP + Intel Cilk Plus Extended Array Notation* *29*
Parallel Programming on the Intel Xeon Phi coprocessor: Intel Cilk Plus *30*
Parallel Programming on Intel Xeon Phi coprocessor: Intel TBB *31*
Using Intel MKL 33
SGEMM Sample *33*
Intel MKL Automatic Offload Model 34
Debugging on the Intel Xeon Phi coprocessor 35
Performance Analysis on the Intel Xeon Phi coprocessor 35
Appendix A: Basic Linux Commands 36
About the Author 38
Notices 39
Optimization Notice 40

Introduction

This document will help developers get started writing code and running applications on a system running Microsoft* Windows that includes the Intel® Xeon Phi™ coprocessor based on the Intel® Many Integrated Core Architecture (Intel® MIC Architecture). It describes the available tools and includes simple examples to show how to get C/C++ and Fortran-based programs up and running. For now, the developer will have to cut/paste the examples provided in the document to their system.

Note that there is a public document "Intel Xeon Phi Coprocessor Developer's Quick Start Guide" at http://software.intel.com/mic-developer under the **"Overview"** tab which is the official document for use with host systems running Linux*; this document is for a restricted list of customers who are early testers for Windows on the host. Most documents pertaining to early access and programming can be found here: http://software.intel.com/en-us/articles/windows-early-enabling-for-intelr-xeon-phitm-coprocessor.

Goals

This document does:

1. Walk you through the system registration and Intel® MPSS installation.
2. Introduce the build environment for Intel MIC Architecture software.
3. Give an example of how to write code for Intel Xeon Phi coprocessor and build using Intel® Composer XE 2013 SP1for Windows.
4. Demonstrate the use of Intel libraries like the Intel® Math Kernel Library (Intel® MKL).
5. Point you to information on how to debug and profile programs running on an Intel Xeon Phi coprocessor.
6. Share some best known methods (BKMs) developed by users at Intel.

This document does not:

1. Cover each tool in detail. Please refer to the user guides for the individual tools.
2. Provide in-depth training.

Terminology

Host – The Intel® Xeon® platform containing the Intel Xeon Phi coprocessor installed in a PCIe* slot. The operating systems (OS) supported and validated on the host are: Windows 7 Enterprise SP1 (64-bit), Windows 8 Enterprise (64-bit), Windows Server 2008 R2 SP1 (64-bit), Windows Server 2012 (64-bit)

Target – The Intel Xeon Phi coprocessor and corresponding runtime environment installed inside the coprocessor.

uOS – Micro Operating System – the Linux-based operating system and tools running on the Intel Xeon Phi coprocessor.

ISA – Instruction Set Architecture – part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O (Input/Output).[1]

[1] Intel acronyms dictionary, 8/6/2009, http://library.intel.com/Dictionary/Details.aspx?id=5600

VPU – Vector Processing Unit- the portion of a CPU responsible for the execution of SIMD (single instruction, multiple data) instructions.

NAcc – Native Acceleration – a mode or form of Intel MKL in which the data being processed and the MKL function processing the data reside on the Intel Xeon Phi coprocessor.

Offload Compilers – The Intel® C/C++ Compiler XE 14.0 for Windows and Intel® Visual Fortran Compiler XE 14.0 for Windows compilers, which can generate binaries for both the host system and the Intel Xeon Phi coprocessor. The offload compilers can generate binaries that will run only on the host, only on the Intel Xeon Phi coprocessor, or paired binaries that run on both the host and the Intel Xeon Phi coprocessor and communicate with each other.

SDP – Software Development Platform – the combination of the host platform and the Intel Xeon Phi coprocessor.

KNC – an abbreviation for Intel Xeon Phi Coprocessor (codename: **Knights Corner**), the first Intel Xeon Phi product.

Intel® MPSS – Intel® Manycore **P**latform **S**oftware Stack – the user- and system-level software that allows programs to run on and communicate with the Intel Xeon Phi coprocessor.

SCIF - Symmetric Communications Interface – the mechanism for inter-node communication within a single platform, where a node is a Intel Xeon Phi coprocessor or an Intel Xeon processor-based host processor complex. In particular, SCIF abstracts the details of communicating over the PCIe bus (and controlling related Intel Xeon Phi coprocessor hardware) while providing an API that is symmetric between all types of nodes

System Configuration

We tested these instructions on an Intel-Software Development Platform consisting of an Intel Workstation containing two Intel Xeon processors, one or two Intel Xeon Phi coprocessors attached to a PCIe* x16 bus, and a GPU for graphics display. A list of systems supporting Intel Xeon Phi coprocessors can be found here: http://software.intel.com/en-us/articles/which-systems-support-the-intel-xeon-phi-coprocessor.

Intel Xeon Phi Software

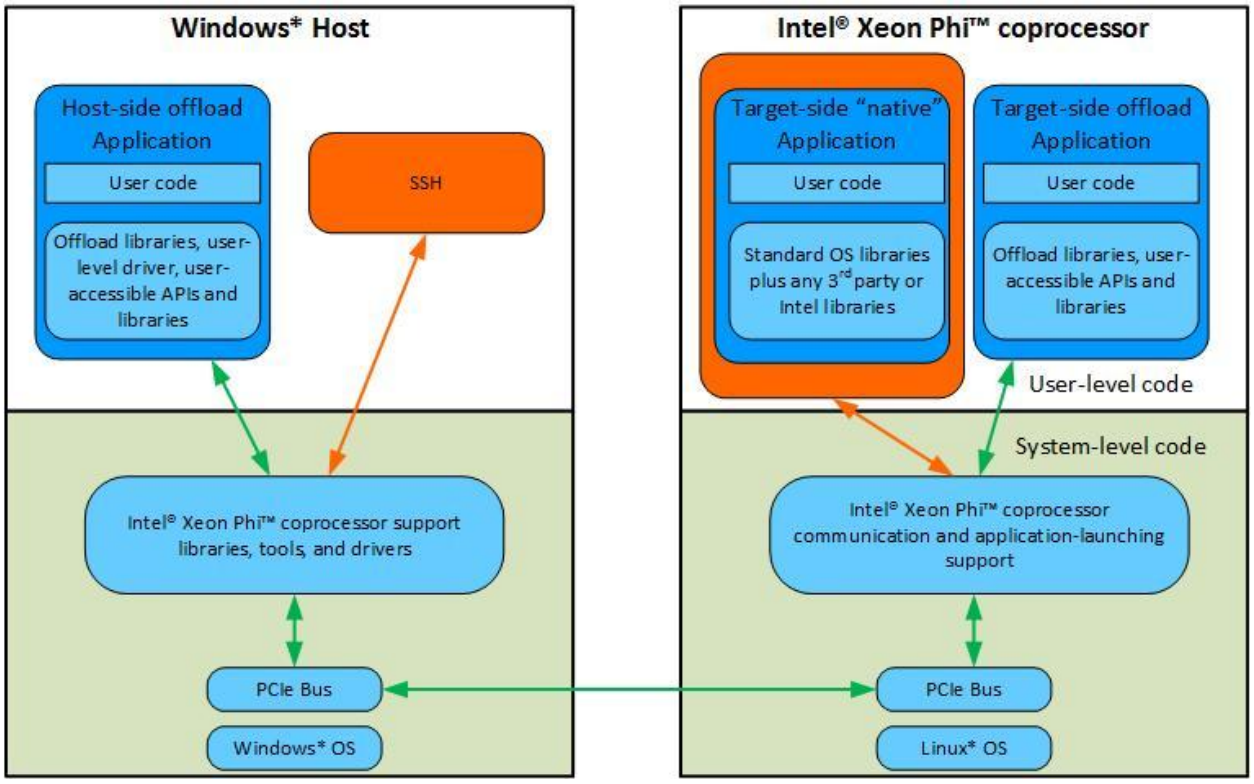

Windows* Host
Host-side offload Application
User code
Offload libraries, user-level driver, user-accessible APIs and libraries
SSH
Intel® Xeon Phi™ coprocessor support libraries, tools, and drivers
PCIe Bus
Windows* OS
Intel® Xeon Phi™ coprocessor
Target-side "native" Application
User code
Standard OS libraries plus any 3rd party or Intel libraries
Target-side offload Application
User code
Offload libraries, user-accessible APIs and libraries
User-level code
System-level code
Intel® Xeon Phi™ coprocessor communication and application-launching support
PCIe Bus
Linux* OS

Figure 1: Software Stack

The Intel Xeon Phi coprocessor software stack consists of layered software architecture as noted below and depicted in Figure 1.

Driver Stack:

The Windows software for the Intel Xeon Phi coprocessor consists of a number of components:

- **Device Driver:** At the bottom of the software stack in kernel space is the Intel Xeon Phi coprocessor device driver. The device driver is responsible for managing device initialization and communication between the host and target devices.
- **Libraries:** The libraries live on top of the device driver in user and system space. The libraries provide basic card management capabilities such as enumeration of cards in a system, buffer management, and host-to-card communication. The libraries also provide higher-level functionality such as loading and unloading the user executable onto the Intel Xeon Phi coprocessor, invoking functions from the executable on the card, and providing a two-way notification mechanism between host and card. The libraries are responsible for buffer management and communication over the PCIe* bus.
- **Tools:** Various tools that help maintain the software stack. Examples include `<MPSS-install-dir>\bin\MicInfo.exe` for querying system information, `<MPSS-install-dir>\bin\MicFlash.exe` for updating the card's flash, `<MPSS-install-dir>\bin\micctrl.exe` to help administrators

configure the card, `<MPSS-install-dir>\bin\micsmc.exe` to monitor platform status, `<MPSS-install-dir>\bin\micras.exe` to collect and log RAS events, where `<MPSS-install-dir>` is "`c:\Program Files\Intel\MPSS`" by default.

- **Card OS (uOS)**: The Linux-based operating system running on the Intel Xeon Phi coprocessor.

NOTE: Linux source for relatively recent versions of the uOS, the device driver, and the low-level SCIF library interface can be found at http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss. Some of the other low level interfaces (COI, MYO) are used only by Intel tools and are currently available for general use. These low level interfaces may be deprecated or exposed in the future.

Intel Many Integrated Core Architecture Overview

The Intel Xeon Phi coprocessor has more than 50 in-order Intel MIC Architecture processor cores running at 1GHz (up to 1.3GHz). The Intel MIC Architecture is based on the x86 ISA, extended with 64-bit addressing and new 512-bit wide SIMD vector instructions and registers. Each core supports 4 hardware threads. In addition to the cores, there are multiple on-die memory controllers and other components.

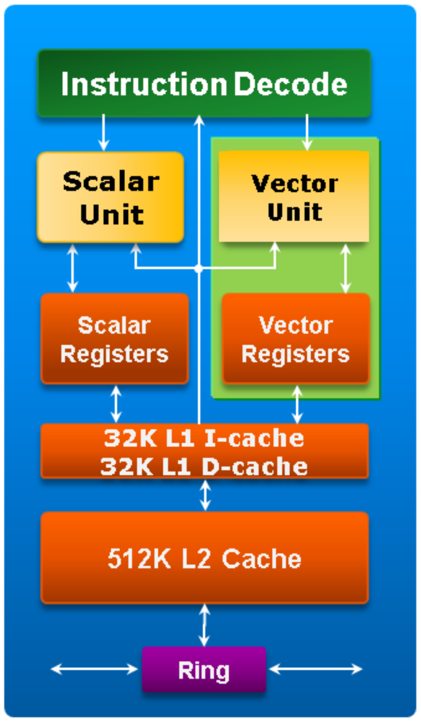


Figure 2: Architecture overview of an Intel MIC Architecture core

Each core includes a newly-designed Vector Processing Unit (VPU). Each vector unit contains 32 512-bit vector registers. To support the new vector processing model, a new 512-bit SIMD ISA was introduced.
The VPU is a key feature of the Intel MIC Architecture-based cores. Fully utilizing the vector unit is critical for the best Intel Xeon Phi coprocessor performance. It is important to note that Intel MIC Architecture cores do not support other SIMD ISAs (such as MMX™, Intel® SSE, or Intel® AVX).

Each core has a 32KB L1 data cache, a 32KB L1 instruction cache, and a 512KB L2 cache. The L2 caches of all cores are interconnected with each other and the memory controllers via a bidirectional ring bus, effectively creating a shared last-level cache of up to 32MB. The design of each core includes a short in-order pipeline. There is no latency in executing scalar operations and low latency in executing vector operations. Due to the short in-order pipeline, the overhead for branch misprediction is low.

For more details on the machine architecture, please refer to the *Intel Xeon Phi Coprocessor Software Developers Guide* posted at http://software.intel.com/mic-developer *under "Tools & Downloads" or "Programming" tabs.*

Administrative Tasks

Details on obtaining drivers and beta software development licenses are available here: http://software.intel.com/en-us/articles/beta-windows-enabled-drivers-and-development-tools-for-intelr-xeon-phitm-coprocessor.

A Composer XE 2013 SP1 serial number is provided to you as a result of registering from the URL: http://software.intel.com/en-us/articles/intelr-composer-xe-2013-beta-registration-for-intelr-xeon-phitm-coprocessor

Note that you will need to acquire your own copy and license of VTune Amplifier XE 2013 for Windows (Update 5 or later).

Refer to http://software.intel.com/en-us/articles/windows-early-enabling-for-intelr-xeon-phitm-coprocessor for step-by-step instructions on submitting Intel® Premier Support issues.

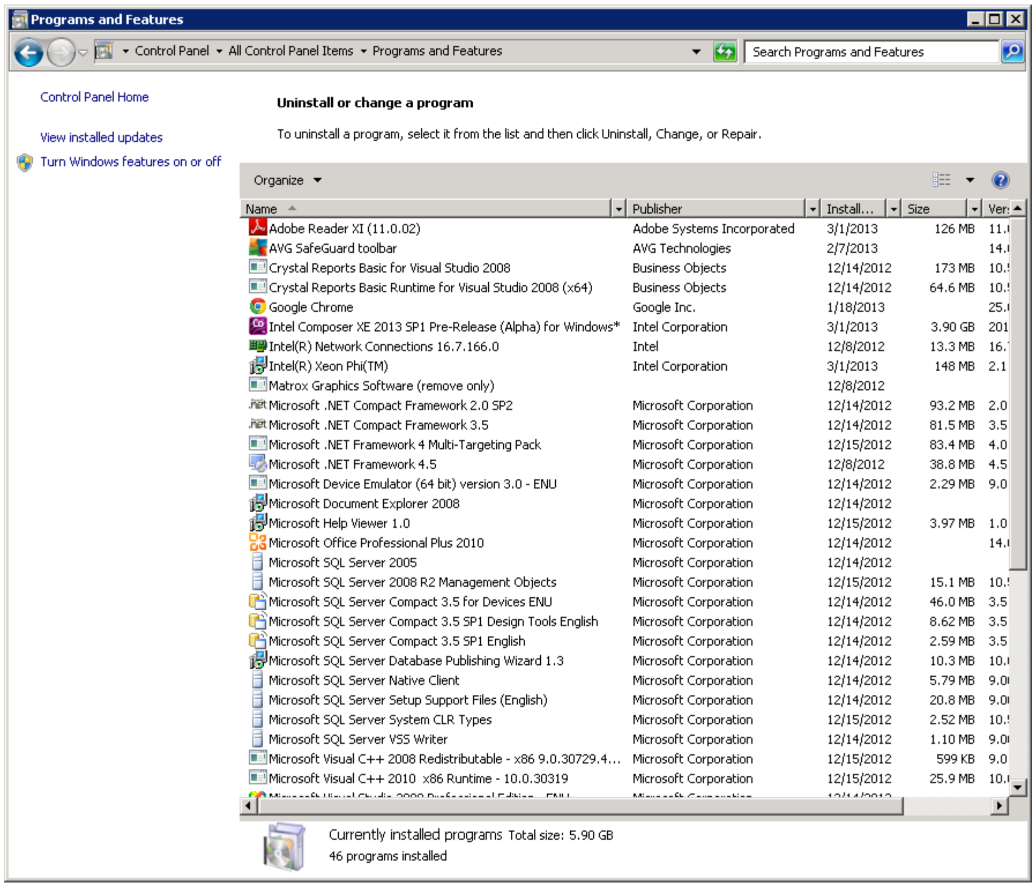
Preparing Your System for First Use

Steps to install the driver and start the coprocessor

1. Before you can download drivers and Beta compilers, you will need to register for access to the compilers
2. Download the "Readme file for the Intel MPSS release", from http://software.intel.com/en-us/articles/beta-windows-enabled-drivers-and-development-tools-for-intelr-xeon-phitm-coprocessor (look for and download Readme and Release Notes).
3. Install one of the following supported Operating Systems:
    - Microsoft* Windows 7 Enterprise SP1 (64-bit),
    - Microsoft* Windows 8 Enterprise (64-bit)
    - Microsoft* Windows Server 2008 R2 SP1 (64-bit)
    - Microsoft* Windows Server 2012 (64-bit)

4. Log in as "administrator"
5. Install .NET Framework 4.0 or higher on the system (http://www.microsoft.com/net/download) Be sure to install PuTTY* and PuTTYgen*, which is used to log in to the card's uOS (see later)
6. Execute the following sequence in a command window (select Run as Administrator)
```
prompt> bcdedit -set loadoptions DISABLE_INTEGRITY_CHECKS
prompt> bcdedit -set TESTSIGNING ON
```
7. Restart the system.
8. Download the drivers package `mpss_beta-2.1.*-windows.zip` for your Windows operating system.
9. Unzip the zip file to get the Windows Installer file (Intel Xeon Phi.msi).
10. Install the Windows Installer file "Intel Xeon Phi.msi" as detailed in section 1 of the Readme file. Note that if a previous version of the Intel Xeon Phi stack is already installed, use Windows Control Panel to

uninstall it prior to installing the current version. Confirm the new Intel MPSS stack is successfully installed by looking at Control Panel -> Program and Features: Intel Xeon Phi (see illustrations that follows). By default, Intel MPSS is installed in "`c:\Program Files\Intel\MPSS`". Select "Always trust software from Intel® VPG MIC" check box during the installation.


Programs and Features
Control Panel ▸ All Control Panel Items ▸ Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Organize

Uninstall or change a program

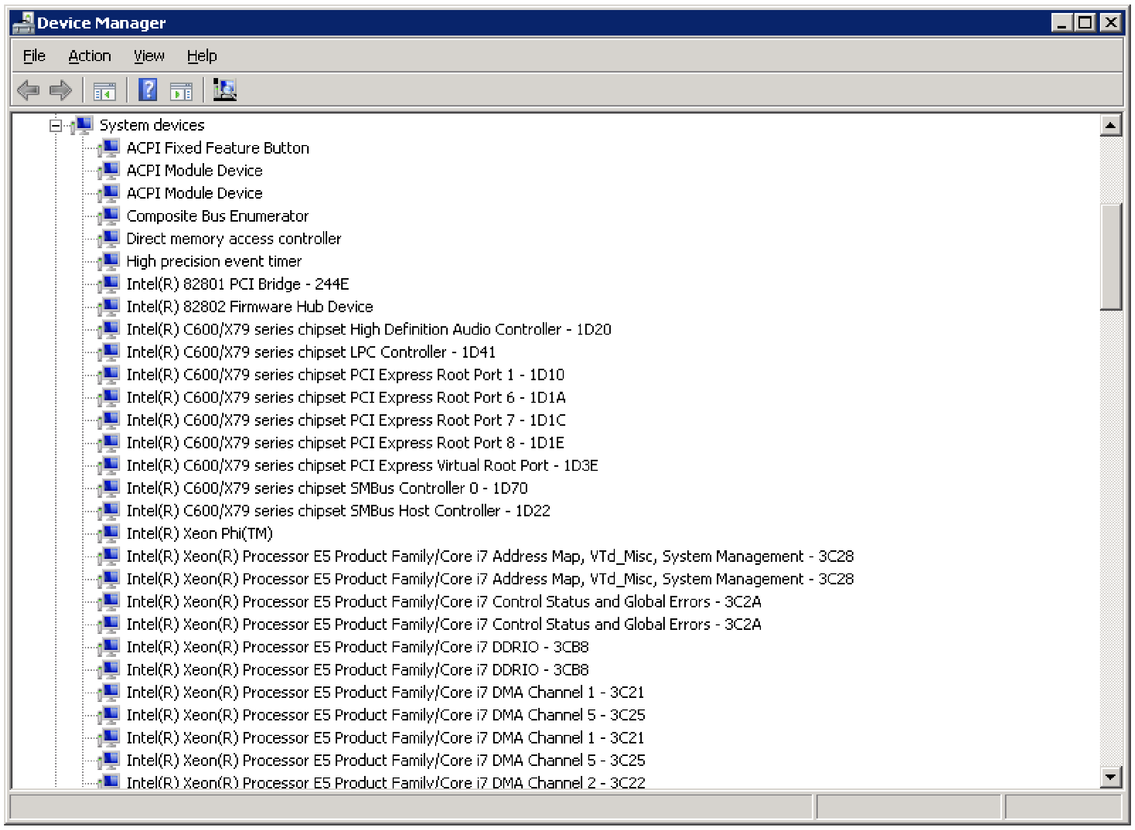
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Install...	Size	Ver:
Adobe Reader XI (11.0.02)	Adobe Systems Incorporated	3/1/2013	126 MB	11.
AVG SafeGuard toolbar	AVG Technologies	2/7/2013		14.
Crystal Reports Basic for Visual Studio 2008	Business Objects	12/14/2012	173 MB	10.
Crystal Reports Basic Runtime for Visual Studio 2008 (x64)	Business Objects	12/14/2012	64.6 MB	10.
Google Chrome	Google Inc.	1/18/2013		25.
Intel Composer XE 2013 SP1 Pre-Release (Alpha) for Windows*	Intel Corporation	3/1/2013	3.90 GB	201
Intel(R) Network Connections 16.7.166.0	Intel	12/8/2012	13.3 MB	16.
Intel(R) Xeon Phi(TM)	Intel Corporation	3/1/2013	148 MB	2.1
Matrox Graphics Software (remove only)		12/8/2012		
Microsoft .NET Compact Framework 2.0 SP2	Microsoft Corporation	12/14/2012	93.2 MB	2.0
Microsoft .NET Compact Framework 3.5	Microsoft Corporation	12/14/2012	81.5 MB	3.5
Microsoft .NET Framework 4 Multi-Targeting Pack	Microsoft Corporation	12/15/2012	83.4 MB	4.0
Microsoft .NET Framework 4.5	Microsoft Corporation	12/8/2012	38.8 MB	4.5
Microsoft Device Emulator (64 bit) version 3.0 - ENU	Microsoft Corporation	12/14/2012	2.29 MB	9.0
Microsoft Document Explorer 2008	Microsoft Corporation	12/14/2012		
Microsoft Help Viewer 1.0	Microsoft Corporation	12/15/2012	3.97 MB	1.0
Microsoft Office Professional Plus 2010	Microsoft Corporation	12/14/2012		14.
Microsoft SQL Server 2005	Microsoft Corporation	12/14/2012		
Microsoft SQL Server 2008 R2 Management Objects	Microsoft Corporation	12/15/2012	15.1 MB	10.
Microsoft SQL Server Compact 3.5 for Devices ENU	Microsoft Corporation	12/14/2012	46.0 MB	3.5
Microsoft SQL Server Compact 3.5 SP1 Design Tools English	Microsoft Corporation	12/14/2012	8.62 MB	3.5
Microsoft SQL Server Compact 3.5 SP1 English	Microsoft Corporation	12/14/2012	2.59 MB	3.5
Microsoft SQL Server Database Publishing Wizard 1.3	Microsoft Corporation	12/14/2012	10.3 MB	10.
Microsoft SQL Server Native Client	Microsoft Corporation	12/14/2012	5.79 MB	9.0
Microsoft SQL Server Setup Support Files (English)	Microsoft Corporation	12/14/2012	20.8 MB	9.0
Microsoft SQL Server System CLR Types	Microsoft Corporation	12/15/2012	2.52 MB	10.
Microsoft SQL Server VSS Writer	Microsoft Corporation	12/14/2012	1.10 MB	9.0
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.4...	Microsoft Corporation	12/15/2012	599 KB	9.0
Microsoft Visual C++ 2010 x86 Runtime - 10.0.30319	Microsoft Corporation	12/15/2012	25.9 MB	10.

Currently installed programs Total size: 5.90 GB
46 programs installed

Figure 3: Intel MPSS is installed as shown in "Program and Features" Panel

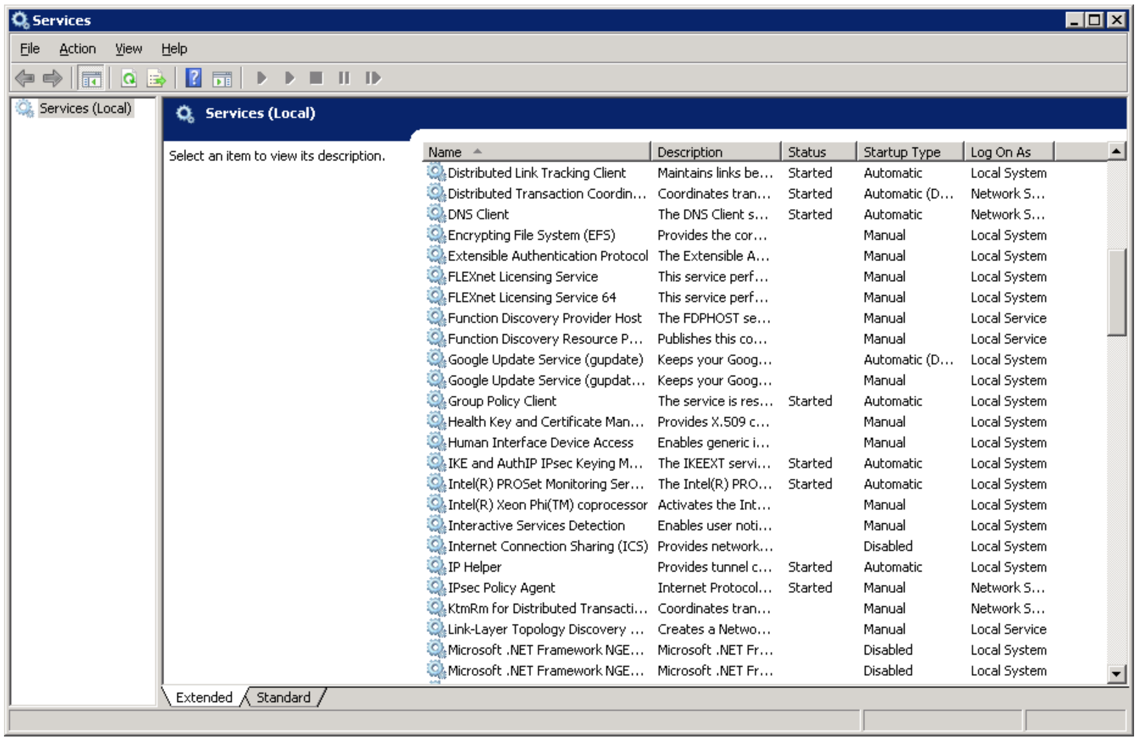
11. Update the flash according to section 3 of the Readme-windows.pdf file.
12. Reboot the system.
13. Login to the host and verify that the Intel Xeon Phi coprocessors are detected by the Device Manager (Control Panel -> Device Manager, and click on "System devices"):



System devices
ACPI Fixed Feature Button
ACPI Module Device
ACPI Module Device
Composite Bus Enumerator
Direct memory access controller
High precision event timer
Intel(R) 82801 PCI Bridge - 244E
Intel(R) 82802 Firmware Hub Device
Intel(R) C600/X79 series chipset High Definition Audio Controller - 1D20
Intel(R) C600/X79 series chipset LPC Controller - 1D41
Intel(R) C600/X79 series chipset PCI Express Root Port 1 - 1D10
Intel(R) C600/X79 series chipset PCI Express Root Port 6 - 1D1A
Intel(R) C600/X79 series chipset PCI Express Root Port 7 - 1D1C
Intel(R) C600/X79 series chipset PCI Express Root Port 8 - 1D1E
Intel(R) C600/X79 series chipset PCI Express Virtual Root Port - 1D3E
Intel(R) C600/X79 series chipset SMBus Controller 0 - 1D70
Intel(R) C600/X79 series chipset SMBus Host Controller - 1D22
Intel(R) Xeon Phi(TM)
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 Address Map, VTd_Misc, System Management - 3C28
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 Address Map, VTd_Misc, System Management - 3C28
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 Control Status and Global Errors - 3C2A
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 Control Status and Global Errors - 3C2A
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DDRIO - 3CB8
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DDRIO - 3CB8
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DMA Channel 1 - 3C21
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DMA Channel 5 - 3C25
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DMA Channel 1 - 3C21
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DMA Channel 5 - 3C25
Intel(R) Xeon(R) Processor E5 Product Family/Core i7 DMA Channel 2 - 3C22

Figure 4: Intel(R) Xeon Phi(TM) is detected by "Device Manager"

Also, verify that Intel MPSS is a Windows service (Control Panel -> System and Security -> Administration Tools ->Services):



Select an item to view its description.

Name	Description	Status	Startup Type	Log On As
Distributed Link Tracking Client	Maintains links be…	Started	Automatic	Local System
Distributed Transaction Coordin…	Coordinates tran…	Started	Automatic (D…	Network S…
DNS Client	The DNS Client s…	Started	Automatic	Network S…
Encrypting File System (EFS)	Provides the cor…		Manual	Local System
Extensible Authentication Protocol	The Extensible A…		Manual	Local System
FLEXnet Licensing Service	This service perf…		Manual	Local System
FLEXnet Licensing Service 64	This service perf…		Manual	Local System
Function Discovery Provider Host	The FDPHOST se…		Manual	Local Service
Function Discovery Resource P…	Publishes this co…		Manual	Local Service
Google Update Service (gupdate)	Keeps your Goog…		Automatic (D…	Local System
Google Update Service (gupdat…	Keeps your Goog…		Manual	Local System
Group Policy Client	The service is res…	Started	Automatic	Local System
Health Key and Certificate Man…	Provides X.509 c…		Manual	Local System
Human Interface Device Access	Enables generic i…		Manual	Local System
IKE and AuthIP IPsec Keying M…	The IKEEXT servi…	Started	Automatic	Local System
Intel(R) PROSet Monitoring Ser…	The Intel(R) PRO…	Started	Automatic	Local System
Intel(R) Xeon Phi(TM) coprocessor	Activates the Int…		Manual	Local System
Interactive Services Detection	Enables user noti…		Manual	Local System
Internet Connection Sharing (ICS)	Provides network…		Disabled	Local System
IP Helper	Provides tunnel c…	Started	Automatic	Local System
IPsec Policy Agent	Internet Protocol…	Started	Manual	Network S…
KtmRm for Distributed Transacti…	Coordinates tran…		Manual	Network S…
Link-Layer Topology Discovery …	Creates a Netwo…		Manual	Local Service
Microsoft .NET Framework NGE…	Microsoft .NET Fr…		Disabled	Local System
Microsoft .NET Framework NGE…	Microsoft .NET Fr…		Disabled	Local System

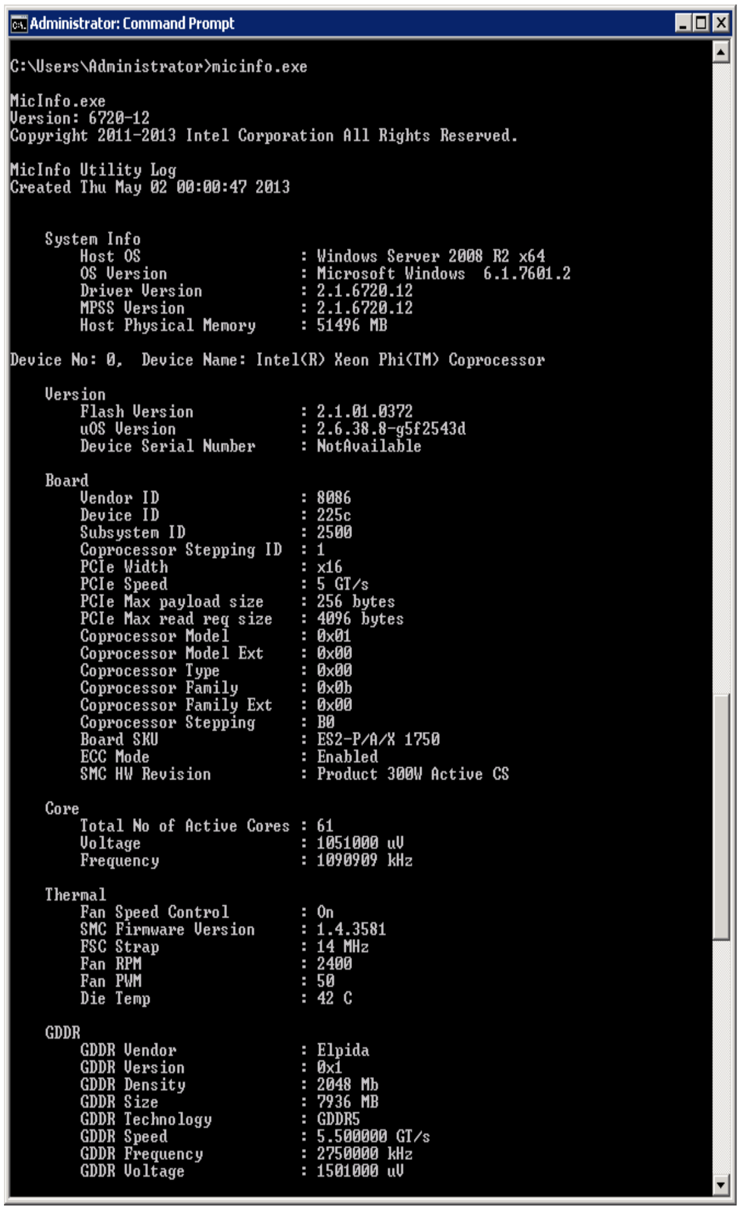
Figure 5: Intel MPSS is shown in "Services" and it is not started yet

14. Start the Intel Xeon Phi coprocessor (while you can set up the card to start with the host system, it will not do so by default). Launch a command prompt windows and start Intel MPSS stack:

```
prompt> micctrl --start
```

15. Run the command "`micinfo`" to verify that it is set up properly:

```
prompt> micinfo.exe
```



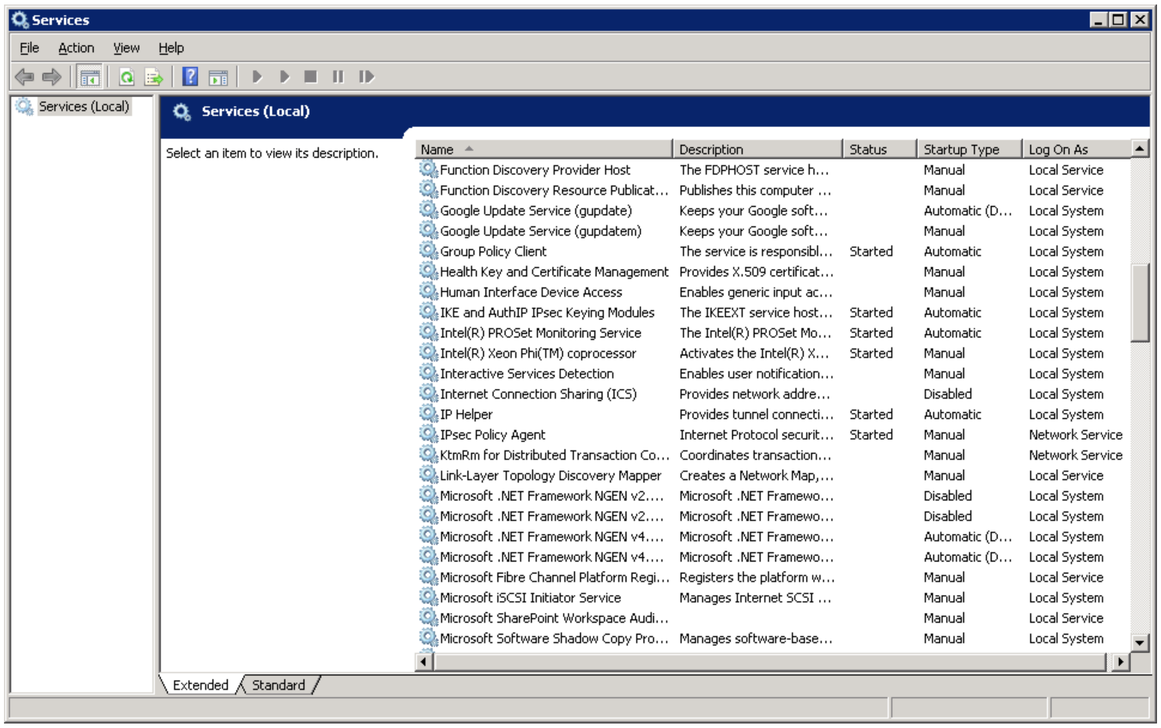
```
C:\Users\Administrator>micinfo.exe

MicInfo.exe
Version: 6720-12
Copyright 2011-2013 Intel Corporation All Rights Reserved.

MicInfo Utility Log
Created Thu May 02 00:00:47 2013


    System Info
        Host OS                  : Windows Server 2008 R2 x64
        OS Version               : Microsoft Windows  6.1.7601.2
        Driver Version           : 2.1.6720.12
        MPSS Version             : 2.1.6720.12
        Host Physical Memory     : 51496 MB

Device No: 0,  Device Name: Intel(R) Xeon Phi(TM) Coprocessor

    Version
        Flash Version            : 2.1.01.0372
        uOS Version              : 2.6.38.8-g5f2543d
        Device Serial Number     : NotAvailable

    Board
        Vendor ID                : 8086
        Device ID                : 225c
        Subsystem ID             : 2500
        Coprocessor Stepping ID  : 1
        PCIe Width               : x16
        PCIe Speed               : 5 GT/s
        PCIe Max payload size    : 256 bytes
        PCIe Max read req size   : 4096 bytes
        Coprocessor Model        : 0x01
        Coprocessor Model Ext    : 0x00
        Coprocessor Type         : 0x00
        Coprocessor Family       : 0x0b
        Coprocessor Family Ext   : 0x00
        Coprocessor Stepping     : B0
        Board SKU                : ES2-P/A/X 1750
        ECC Mode                 : Enabled
        SMC HW Revision          : Product 300W Active CS

    Core
        Total No of Active Cores : 61
        Voltage                  : 1051000 uV
        Frequency                : 1090909 kHz

    Thermal
        Fan Speed Control        : On
        SMC Firmware Version     : 1.4.3581
        FSC Strap                : 14 MHz
        Fan RPM                  : 2400
        Fan PWM                  : 50
        Die Temp                 : 42 C

    GDDR
        GDDR Vendor              : Elpida
        GDDR Version             : 0x1
        GDDR Density             : 2048 Mb
        GDDR Size                : 7936 MB
        GDDR Technology          : GDDR5
        GDDR Speed               : 5.500000 GT/s
        GDDR Frequency           : 2750000 kHz
        GDDR Voltage             : 1501000 uV
```

Figure 6: Output of the command "micinfo.exe"

- Verify that the driver version is 2.1.*
- Verify that the Intel MPSS version is 2.1.*
- Verify that the Flash Version is 2.1.*.*

16. Install "*Binutils*", the binary utilities for the Intel Xeon® Phi™ coprocessor native compiler, as detailed in section 2.2.5 of the Readme file.

We can check the Windows Services to verify that the Intel MPSS service is started:



Select an item to view its description.

Name	Description	Status	Startup Type	Log On As
Function Discovery Provider Host	The FDPHOST service h...		Manual	Local Service
Function Discovery Resource Publicat...	Publishes this computer ...		Manual	Local Service
Google Update Service (gupdate)	Keeps your Google soft...		Automatic (D...	Local System
Google Update Service (gupdatem)	Keeps your Google soft...		Manual	Local System
Group Policy Client	The service is responsibl...	Started	Automatic	Local System
Health Key and Certificate Management	Provides X.509 certificat...		Manual	Local System
Human Interface Device Access	Enables generic input ac...		Manual	Local System
IKE and AuthIP IPsec Keying Modules	The IKEEXT service host...	Started	Automatic	Local System
Intel(R) PROSet Monitoring Service	The Intel(R) PROSet Mo...	Started	Automatic	Local System
Intel(R) Xeon Phi(TM) coprocessor	Activates the Intel(R) X...	Started	Manual	Local System
Interactive Services Detection	Enables user notification...		Manual	Local System
Internet Connection Sharing (ICS)	Provides network addre...		Disabled	Local System
IP Helper	Provides tunnel connecti...	Started	Automatic	Local System
IPsec Policy Agent	Internet Protocol securit...	Started	Manual	Network Service
KtmRm for Distributed Transaction Co...	Coordinates transaction...		Manual	Network Service
Link-Layer Topology Discovery Mapper	Creates a Network Map,...		Manual	Local Service
Microsoft .NET Framework NGEN v2....	Microsoft .NET Framewo...		Disabled	Local System
Microsoft .NET Framework NGEN v2....	Microsoft .NET Framewo...		Disabled	Local System
Microsoft .NET Framework NGEN v4....	Microsoft .NET Framewo...		Automatic (D...	Local System
Microsoft .NET Framework NGEN v4....	Microsoft .NET Framewo...		Automatic (D...	Local System
Microsoft Fibre Channel Platform Regi...	Registers the platform w...		Manual	Local Service
Microsoft iSCSI Initiator Service	Manages Internet SCSI ...		Manual	Local System
Microsoft SharePoint Workspace Audi...			Manual	Local Service
Microsoft Software Shadow Copy Pro...	Manages software-base...		Manual	Local System

Figure 7: Intel MPSS was started as shown in "Services" panel

Steps to install the Software Development tools

As mentioned before, you will need to register for access to use the early builds of the Beta Intel® [C++|Fortran] Composer XE 2013 SP1 for Windows package (debugger, MKL, etc. all included).

After registering at this URL, you can download the product and you will receive an email containing the tool serial number (and a license file) for the Beta Intel [C++|Fortran] Composer XE 2013 SP1 for Windows package.

Should you need access to analysis tools to look for performance bottlenecks, you will need to acquire a copy of Intel® VTune Amplifier XE 2013 for Windows Update 5.

- If you use the Intel C++ Composer XE 2013 SP1 Beta for Windows or the Intel Visual Fortran Composer XE 2013 SP1 Beta for Windows, read the corresponding readme file to install these packages.
- For first time installations, be sure to get the product license number provided to you at registration, as it is required to activate the product, and then provide the license number during installation. Subsequent installations can select the "Use existing license" option.
- Read the readme file carefully.
- Double click on the executable file (.EXE) to begin installation. You don't need to uninstall previous versions or updates before installing a newer version of Composers. The new version can update the existing version or coexist with the older versions.

1. Install the software tools.
2. Verify that the card is working by running a sample program (located in <install-dir>\Samples\en_US\C++\mic_samples or <install-dir>\Samples\en_US\Fortran\mic_samples) with "`set OFFLOAD_REPORT=3`" to display the dialog between the Host and Intel Xeon Phi coprocessor (messages from the processor will be prefixed with "MIC:"). If you do see a dialog then everything is running fine and the system is ready for general use. For a description of these samples please refer to http://software.intel.com/en-us/articles/offload-programming-fortran-and-c-code-examples .
3. If you intend to collect and analyze performance data using SEP and Intel® Vtune Amplifier XE 2013 Update 5, after acquiring the software, unzip the package you've obtained. You should get two separate packages: "VTune_Amplifier_XE_2013_update5_setup" and a zip file "sep*_win_mic.zip". Install VTune_Amplifier_XE_2013_update5_setup to get VTune. To install SEP:
   a) Create a folder where you want to install SEP (e.g. c:\sep). Unzip the SEP install package sep*_win_mic.zip into this folder. For more information, read the document "<sep-install-dir>\docsSEP_Install_Instruction_Windows_MIC". Load the data collection driver after starting the coprocessor by going to `<sep-install-dir>\k1om\`" and running:

      ```
      prompt> .\sep_mic_install.cmd
      ```

   b) Start (or restart) the Intel MIC architecture service (this also starts the sampling driver once the files are copied in the previous step):

      ```
      prompt> micctrl --start
      prompt> micctrl -w
      ```

      The coprocessor has successfully restarted when `micctrl -w` reports "mic*x*: online"
   c) The sampling driver will now start every time the coprocessor is restarted
   d) If you ever need to uninstall the sampling driver, it can be done as follows:

      ```
      prompt> micctrl --stop
      prompt> .\sep_mic_uninstall.cmd
      prompt> micctrl --start
      prompt> micctrl -w
      ```

Regaining Access to the Intel Xeon Phi Coprocessor after Reboot

The Intel Xeon Phi coprocessor will not start when the host system reboots. So you will need to manually start the Intel Xeon Phi coprocessor, and then run "`micinfo`" to verify that it started properly. You need to have administrator permissions to run this command. First click **Start**, then click **All Programs**, and **Accessories**. In **Accessories**, right-click on **Command Prompt**, and click **Run as administrator**. After entering the password for Administrator, a command line windows pops up

```
prompt> micctrl --start

prompt> micctrl -w

prompt> micinfo
```

Restarting the Intel Xeon Phi Coprocessor if it Hangs

If a process running on the Intel Xeon Phi coprocessor hangs, but the coprocessor is otherwise responsive, log into the card via PuTTY* and kill the process like any other Linux process.

When a coprocessor hangs, and is inaccessible or unresponsive via PuTTY*, there are two ways to restart it. But first, see if you can tell what is happening:

```
prompt> micctrl -s <micx>
```

Assuming that the Intel MPSS service is still functioning properly, you can try to restart the coprocessor without affecting any attached coprocessors as follows:

```
prompt> micctrl –r <micx>
prompt> micctrl -w
prompt> micctrl –b <micx>
prompt> micctrl -w
prompt> micinfo
```

If the Intel MPSS service is not running properly, then we need to restart the driver and all connected coprocessors:

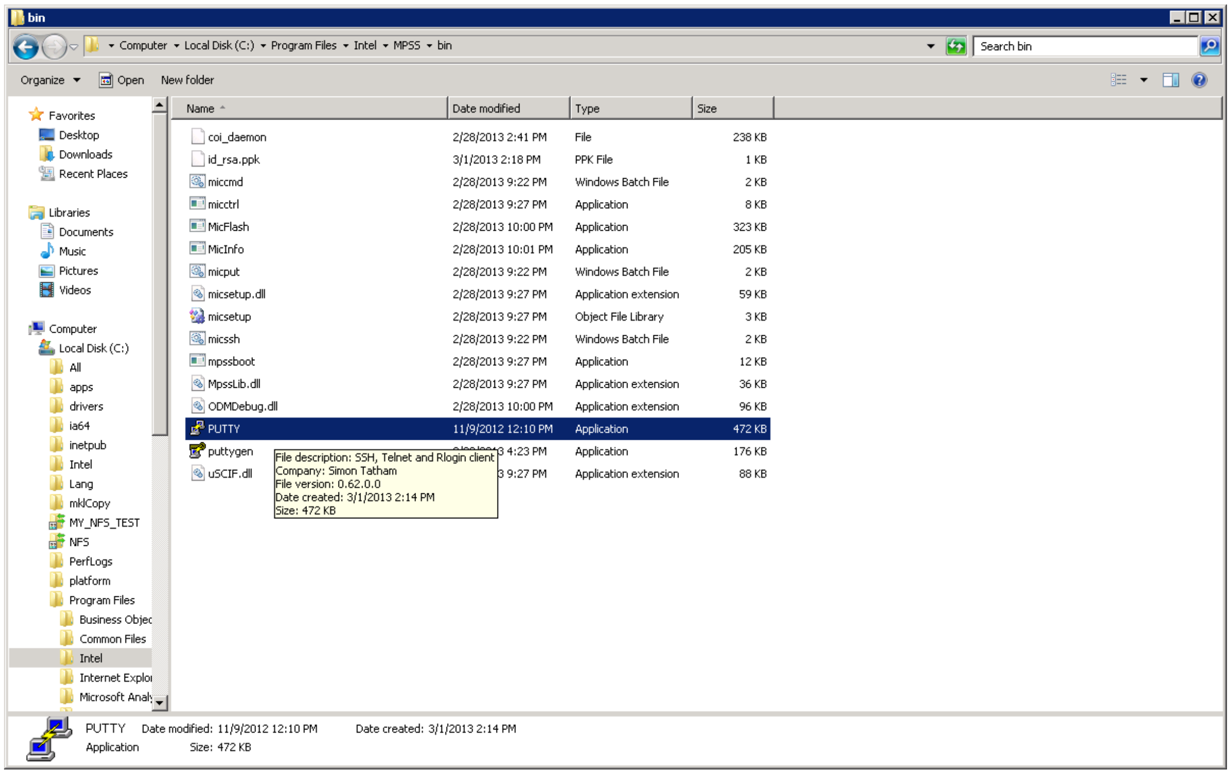
```
prompt> micctrl --stop
prompt> micctrl --start
prompt> micctrl -w
prompt> micinfo
```

Working directly with the uOS Environment Intel Xeon Phi Coprocessor

The default IP address for the coprocessor as seen from the host is `192.168.<coprocessor>.100`, while the coprocessor sees the host at `192.168.<coprocessor>.99` by default. The coprocessor can also be referred to from the host by the alias `mic<coprocessor>`. For example, the first coprocessor you install in your system is called "`mic0`" and is located at `192.168.1.100`. It sees the host at `192.168.1.99`. The second is called "`mic1`" and is located at `192.168.2.100`, seeing the host at `192.168.2.99`.

Since the coprocessor is running Linux and is effectively a separate network node, root or non-root users can log into it via PuTTY* and issue many common Linux commands. Files are transferred to/from the coprocessor using "`pscp.exe`" or other means.

- From http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html download the third party software PuTTY*. After downloading this tool, place it under the host folder "`<MPSS-install-dir>\bin`". To create SSH keys, download the PuTTYgen* utility from the same link and also place it under the host folder "`<MPSS-install-dir>\bin`":



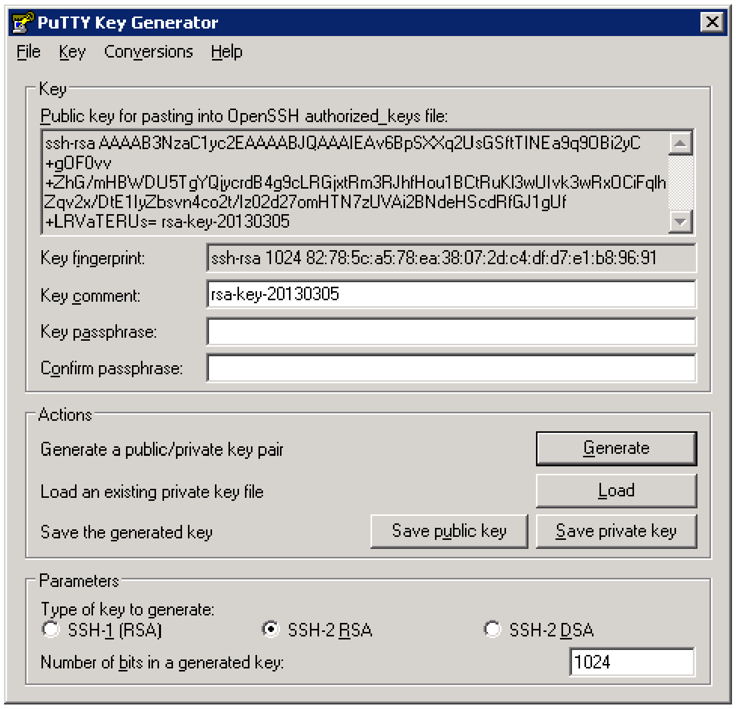
Name	Date modified	Type	Size
coi_daemon	2/28/2013 2:41 PM	File	238 KB
id_rsa.ppk	3/1/2013 2:18 PM	PPK File	1 KB
miccmd	2/28/2013 9:22 PM	Windows Batch File	2 KB
micctrl	2/28/2013 9:27 PM	Application	8 KB
MicFlash	2/28/2013 10:00 PM	Application	323 KB
MicInfo	2/28/2013 10:01 PM	Application	205 KB
micput	2/28/2013 9:22 PM	Windows Batch File	2 KB
micsetup.dll	2/28/2013 9:27 PM	Application extension	59 KB
micsetup	2/28/2013 9:27 PM	Object File Library	3 KB
micssh	2/28/2013 9:22 PM	Windows Batch File	2 KB
mpssboot	2/28/2013 9:27 PM	Application	12 KB
MpssLib.dll	2/28/2013 9:27 PM	Application extension	36 KB
ODMDebug.dll	2/28/2013 10:00 PM	Application extension	96 KB
PUTTY	11/9/2012 12:10 PM	Application	472 KB
puttygen	3 4:23 PM	Application	176 KB
uSCIF.dll	3 9:27 PM	Application extension	88 KB

File description: SSH, Telnet and Rlogin client
Company: Simon Tatham
File version: 0.62.0.0
Date created: 3/1/2013 2:14 PM
Size: 472 KB

PUTTY Date modified: 11/9/2012 12:10 PM Date created: 3/1/2013 2:14 PM
Application Size: 472 KB

Figure 8: Placing PuTTY* and PuTTgen* in "`<MPSS-install-dir>\bin`"

- Launch PuTTYgen* and click the button "Generate" to generate the SSH public and private keys. Follow the instructions to generate the public key as displayed in the following screen



Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAIEAv6BpSXXq2UsGSftTINEa9q9OBi2yC
+gOF0vv
+ZhG/mHBWDU5TgYQjycrdB4g9cLRGjxtRm3RJhfHou1BCtRuKl3wUIvk3wRxOCiFqlh
Zqv2x/DtE1lyZbsvn4co2t/Iz02d27omHTN7zUVAi2BNdeHScdRfGJ1gUf
+LRVaTERUs= rsa-key-20130305
```

Key fingerprint: ssh-rsa 1024 82:78:5c:a5:78:ea:38:07:2d:c4:df:d7:e1:b8:96:91

Key comment: rsa-key-20130305

Key passphrase:

Confirm passphrase:

Generate a public/private key pair

Load an existing private key file

Save the generated key

Type of key to generate:
SSH-1 (RSA) [✓ SSH-2 RSA] SSH-2 DSA

Number of bits in a generated key: 1024

Figure 9: PuTTYgen*

- Using Notepad, create a file named `authorized_keys` (NOTE: with no .txt file extension) and place it in the folder `"<MPSS-install-dir>\bin"`.
- Copy the text appearing in "Public key pasting into OpenSSH authorized_keys file" and paste it into the file `authorized_keys`.
- Click the button "Save private key" to save the private key to the file named `id_rsa.ppk`. Move this file `id_rsa.ppk` to `"<MPSS-install-dir>\bin"`.
- Open a command prompt windows (using **Run as Administrator**), and change directory into `"<MPSS-install-dir>\bin"`, then execute the following command:

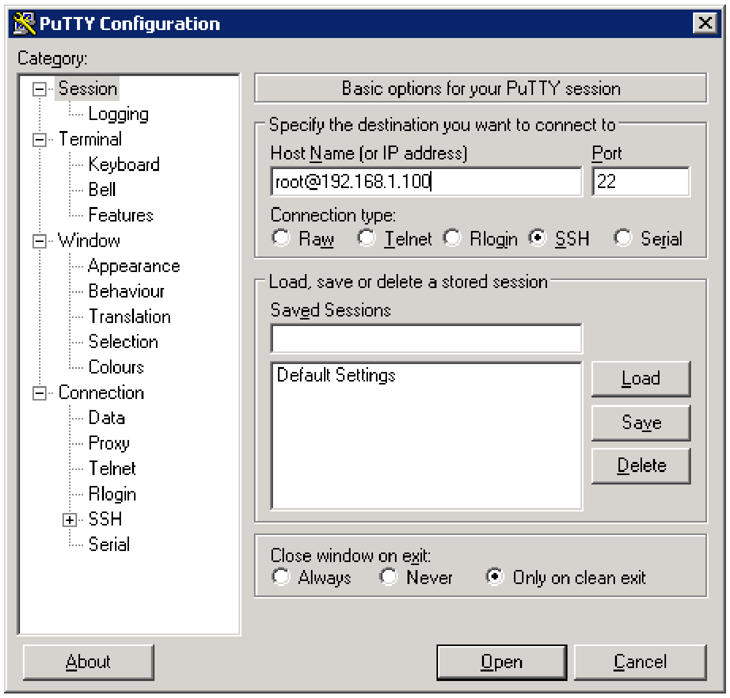
```
prompt> micctrl --addssh root -f "<MPSS-install-dir>\bin\authorized_keys"
```

- Restart the coprocessor by typing "`micctrl --stop`" and then "`micctrl--start`"

```
prompt> micctrl --stop

prompt> micctrl --start
```

At this step, we can login to the coprocessor using PuTTY*

- Launch the PuTTY* tool.
- Set the box "Host Name (or IP address)" to root@192.168.1.100 for mic0 (and root@192.168.2.100 for mic1 if available).



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): root@192.168.1.100

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

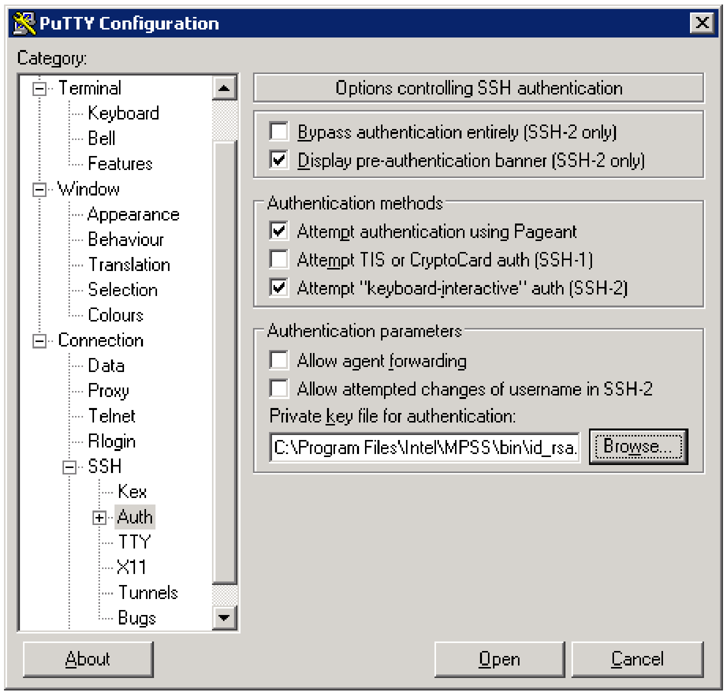
Saved Sessions

Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

Figure 10: Login onto the coprocessor using PuTTY*

- Expand Connection->SSH and click on Auth. In the box "Private key file for authentication", use Browse to select the private key file `id_rsa.ppk`.



Options controlling SSH authentication

☐ Bypass authentication entirely (SSH-2 only)

☑ Display pre-authentication banner (SSH-2 only)

Authentication methods

☑ Attempt authentication using Pageant

☐ Attempt TIS or CryptoCard auth (SSH-1)

☑ Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

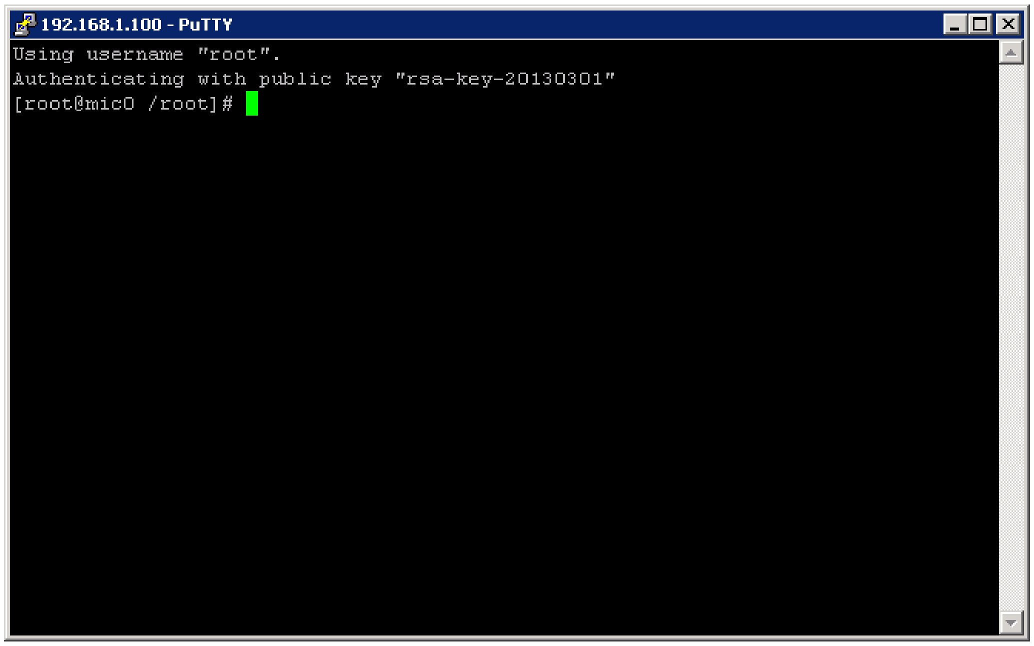
☐ Allow agent forwarding

☐ Allow attempted changes of username in SSH-2

Private key file for authentication: C:\Program Files\Intel\MPSS\bin\id_rsa.

Figure 11: Selecting the private key file

- Click Open to connect to the coprocessor. A window pops up that allows the user to login into the coprocessor (without being challenged for a password).



```
Using username "root".
Authenticating with public key "rsa-key-20130301"
[root@mic0 /root]#
```

Figure 12: Root directory of mic0

- The utility "pscp" allows users to copy files from the host to the coprocessor(s). To use this third-party utility, users need to download the command-line secure file copy PSCP application from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html to "`<MPSS-install-dir>\bin`"

  For example, you can copy a file compiled for the coprocessor (called `application.mic`) from the host to the mic0 coprocessor by typing the following command from the host:

  ```
  prompt>pscp -i id_rsa.ppk "application.mic" root@192.168.1.100:/tmp/
  application.mic
  ```

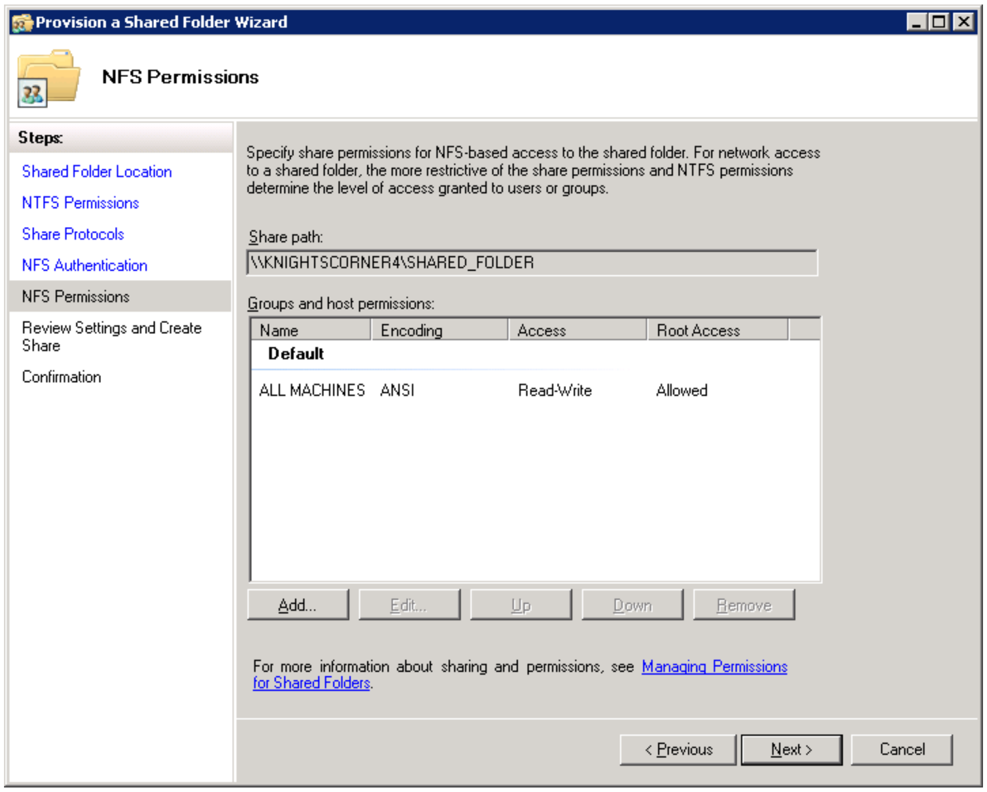
  You can verify that the file is successfully transferred into the `/tmp` directory on the coprocessor by log in on the coprocessor and issue the command "`ls /tmp`".

Host File System Share

Instead of transferring your native Intel Xeon Phi coprocessor executable to a coprocessor, you can set up a shared directory between the host and a coprocessor. After completing this setup, any file in the shared directory can be accessed by both the host and the coprocessor.

For detailed information on mounting an NFS file system exported by the host for use on the Intel Xeon Phi coprocessor, please see section 5 of Readme-windows.pdf.

It is worth noting that if you want to share a folder across the host and all available coprocessors, then from the **Provision a Shared Folder Wizard** window, click **Edit** to change **Access** field to Read-Write and **Root Access** field to Allowed.



Specify share permissions for NFS-based access to the shared folder. For network access to a shared folder, the more restrictive of the share permissions and NTFS permissions determine the level of access granted to users or groups.

Share path:
\\KNIGHTSCORNER4\SHARED_FOLDER

Groups and host permissions:

Name	Encoding	Access	Root Access
Default			
ALL MACHINES	ANSI	Read-Write	Allowed

For more information about sharing and permissions, see Managing Permissions for Shared Folders.

Figure 13: Sharing a folder with all coprocessors

Useful Administrative Tools

This product ships the following administrative tools, which are found in the "`<MPSS-install-dir>\bin`" folder. Root, and users needing to use these tools, should add this folder to their default path:

- **micinfo** - provides information about host and coprocessor system configuration.
- **micflash** - updates the flash on the coprocessor; saves and retrieves the version and other information for each section of the flash.
- **micctrl** – a tool to help the system administrator configure and restart the coprocessor.
- **micras** – this tool runs on the host, collects and logs RAS events generated by the Intel Xeon Phi coprocessor. This tool is also responsible for handling test and repair by kicking the coprocessor into Maintenance mode upon detecting a fatal RAS event.
- **micsmc** – a tool used to monitor core utilization, temperature, memory usage, power usage statistics, and error logs. It can function in two modes: graphical user interface or command-line interface mode.

Please see section 6 in *Readme-windows.pdf* for details on these tools and their arguments.

Getting Started/Developing Intel Xeon Phi Coprocessor Software

You develop applications for the Intel MIC Architecture using your existing knowledge of multi-core and SIMD programming. The offload language extensions allow you to port sections of your code (written in C/C++ or FORTRAN) to run on the Intel Xeon Phi coprocessor, or you can port your entire application to the Intel MIC Architecture. Best performance will only be attained with highly parallel applications that also use SIMD operations (generated by the compiler or using compiler intrinsics) for most of their execution.

Available SW development Tools / Environments

You can start programming for the Intel Xeon Phi coprocessor using your existing parallel programming knowledge and the same techniques you use to develop parallel applications on the host. New tools were not created to support development directly on the Intel Xeon Phi coprocessor; rather, the familiar host-based Intel tools have been extended to add support for the Intel MIC Architecture via a few additions to standard languages and APIs. However, to make best use of the development tools and to get best performance from the Intel Xeon Phi coprocessor, it is important to understand the Intel MIC Architecture.

Development Environment: Available Compilers and Libraries

- Compilers
    - Beta Intel C++ Composer XE 2013 SP1 for Windows for building applications that run on Intel 64 Architecture and Intel MIC Architecture
    - Beta Intel® Visual Fortran Composer XE 2013 SP1 for Windows for building applications that run on Intel 64 Architecture and Intel MIC Architecture
- Libraries packaged with the compilers include:
    - Intel® Math Kernel Library (Intel MKL) optimized for the Intel MIC Architecture
    - Intel® Threading Building Blocks (Intel® TBB)
    - Intel® Integrated Performance Primitive (Intel® IPP)

Development Environment: Available Tools

In addition to the standard compilers and Intel libraries, the following tools are available to help you debug and optimize software running on the Intel Xeon Phi coprocessor.

- Debugger
    - GDB for applications running on the Intel 64 and Intel MIC Architecture
- Profiling
    - Intel VTune Amplifier XE 2013 for Windows Update 5, which is used on the host Windows OS to collect and view performance data collected on the Intel Xeon Phi coprocessor

General Development Information

Development Environment Setup

- In this beta version, Microsoft * Visual Studio integration is supported for Microsoft* Visual Studio 2008 and above versions. Our instructions show the compiler command line interface to compile for offload. To set up your development environment for use with the Intel tools, you need to source the following script (the default install locations are assumed):
    - **Intel® C++ and Visual Fortran Composer XE 2013 SP1 Beta for Windows**:

      Open a command prompt window: Start > All Programs > Intel Parallel Studio XE 2013 > Command Prompt > Parallel Studio XE with Intel Compiler XE v14.0 > Intel 64 Visual Studio XXXX mode.

Documentation and Sample Code

- The most useful documentation can be found in **`<install-dir>\Documentation\en_US\`** including:
    - **`compiler_c\cl\compiler_ug_c`** and **`compiler_f\cl\compiler_ug_f`** - complete documentation for Intel® C++ Compiler XE 14.0 Beta for Windows and the Intel® Visual Fortran Compiler XE 14.0 Beta for Windows (or simply Start > All Programs > Intel Parallel Studio XE 2013 > Documentation)
        - Most information on how to build for the Intel MIC Architecture can be found in the "Key Features" section under "Intel MIC Architecture / Programming for the Intel MIC Architecture"
        - Information on Intel MIC Architecture intrinsics can be found in the "Compiler Reference/Intrinsics" section under "Intrinsics for Intel MIC Architecture"
    - **`Release_Notes-C-2013SP1_W_EN.pdf`** and **`Release_Notes-F-2013SP1_W_EN.pdf`** - please read these carefully for known issues and their workarounds, plus installation instructions, for all the tools with Intel MIC Architecture support. You'll find Intel MIC Architecture-specific information primarily in section 3.
        - **Note**: For various reasons, this document can miss some last-minute updates. The **`Release_Notes-*-2013SP1_W_EN.pdf`** documents from the C++/Fortran compiler can be downloaded from the IRC and will always have the most recent version of this document (see Section "Installation").
    - **`<install-dir>\Documentation\en_US\debugger\gdb\gdb.pdf`** – Information on how to use the debugger. In the beta release, Windows Visual Studio can be used to debug offload on Intel Xeon Phi coprocessor with the underlying GDB..
- Other documentation that includes sections on using the Intel Xeon Phi coprocessor:
    - The Intel MKL User's Guide, which can be accessed via **`mkl_documentation.htm`** found in **`<install-dir>\Documentation\en_US\mkl`**, contains a section called "Using the Intel Math Kernel Library on Intel MIC Core Architecture Coprocessors" which describes both "Automatic Offload" and "Compiler Assisted Offload" of Intel MKL functions (or simply Start > All Programs > Intel Parallel Studio XE 2013 > Documentation > Math Kernel Library)
    - Information on collecting performance data on the Intel Xeon Phi coprocessor using SEP can be found in **`sep_spec_mic`**, located in **`<sep-install-dir>\docs`**.

- Useful documentation on the Web (for Linux host but still useful):
  - On the website http://software.intel.com/mic-developer you will find a wide range of documentation that can be downloaded, most notably the ***Intel Xeon Phi Software Developers Guide*** under ***"Software Development"*** tab. Here you will also find user forums, tools, and case studies.
  - Navigating down from http://software.intel.com/en-us/blogs/2012/06/05/knights-corner-open-source-software-stack/, you will find the source code for the Intel Xeon Phi coprocessor's uOS, a native Intel MIC Architecture version of gdb, and documentation including the ***Intel Xeon Phi Coprocessor Instruction Set Reference Manual***, ***ABI document System V Application Binary Interface K1OM Architecture Processor Supplement***, and ***Intel Xeon Phi Performance Monitor Units*** under the "Resources" link.
- Some sample offload code using the explicit memory copy model can be found in:
  - **Intel C++**: `<install-dir>\Samples\en_US\C++\mic_samples`
  - **Intel Fortran**: `<install-dir>\Samples\en_US\Fortran\mic_samples\`
- Some sample offload code using the implicit memory copy model can be found in:
  - **C**: `<install-dir>\Samples\en_US\C++\mic_sample\LEO_tutorial`

Build-Related Information

- The offload compiler produces "fat" binaries and `.dll` files that contain code for both host and the Intel Xeon Phi coprocessor.
- The offload compiler produces code that examines the runtime execution environment for the presence of an Intel Xeon Phi coprocessor. If a coprocessor is not present and the offload fails, the program will exit with an error message.
- A number of workarounds and hints can be found in *release-notes-*-2013-w-en.pdf*.

Compiler Switches

When building applications that offload some of their code to the Intel Xeon Phi coprocessor, it is possible to cause the offloaded code to be built with different compiler options from the host code. The method of passing these options to the compiler is documented in the compiler documentation under the "Compiler Reference/Compiler Options/Compiler Option Categories and Descriptions" section. Look for the `/Qoffload-option` compiler switch. In that same section, also look up the `/Qoffload-attribute-target` compiler switch, which provides an alternative to editing your source files in some situations (applies to the pragma-based offload methods). Finally, `/Qoffload-` (or `/Qoffload:none`) provides a way to make the compiler ignore the Language Extensions for Offload (which cause it by default to build a heterogeneous binary).

Debugging Offload Activity During Runtime

To debug offload activity, the OFFLOAD_REPORT environment variable is available:

- To learn whether offload potions of the program are running on the host or coprocessor and receive debug information

```
prompt> set OFFLOAD_REPORT=3
```

- A value of 1 reports just the time the offload took measured by the host, and the amount of computation time done by the coprocessor. A value of 2 adds information on how much data was transferred in either direction.

```
prompt> set OFFLOAD_REPORT=<1 or 2>
```

Details can be found in the compiler documentation in the "Compilation/Setting Environment Variables" section.

Using the Offload Compiler – Explicit Memory Copy Model

In this section, a reduction is used as an example to show a step-by-step approach for developing applications for the Intel Xeon Phi coprocessor using the offload compiler. The offload compiler is a heterogeneous[2] compiler, with both host CPU and target compilation environments. Code for both the host CPU and Intel Xeon Phi coprocessor is compiled within the host environment, and offloaded code is automatically run within the target environment. The offload behavior is controlled by compiler directives: pragmas in C/C++, and directives in Fortran.

Some common libraries, such as the Intel Math Kernel Library (Intel MKL), are available in host versions as well as target versions. When an application executes its first offload and the target is available, the runtime loads the target executable onto the Intel Xeon Phi coprocessor. At this time, it also initializes the libraries linked with the target code. The loaded target executable remains in the target memory until the host program terminates. Thus, any global state maintained by the library is maintained across offload instances.

Note: Although, the user may specify the region of code to run on the target, there is no guarantee of execution on the Intel Xeon Phi coprocessor. Depending on the presence of the target hardware or the availability of resources on the Intel Xeon Phi coprocessor when execution reaches the region of code marked for offload, the code can run on the Intel Xeon Phi coprocessor or may fall back to executing on the host.

The following code samples show several versions of porting reduction code to the Intel Xeon Phi coprocessor using the offload pragma directive.

Reduction

The operation refers to computing the expression:

```
ans = a[0] + a[1] + … + a[n-1]
```

Host Version:

The following sample code shows the C code to implement this version of the reduction.

```
float reduction_serial(float *data, int size)
{
    float ret = 0.f;
    for (int i=0; i<size; ++i)
    {
```

[2] http://dictionary.reference.com/browse/heterogeneous

```
        ret += data[i];
    }
    return ret;
}
```

Code Example 1: Implementing Reduction Code in C/C++

Creating the Offload Version

Serial Reduction with Offload

The programmer uses **#pragma offload target(mic)** (as shown in the example below) to mark statements (offload constructs) that should execute on the Intel Xeon Phi coprocessor. The offloaded region is defined as the offload construct plus the additional regions of code that run on the target as the result of function calls. Execution of the statements on the host will resume once the statements on the target have executed and the results are available on the host (i.e. the offload will block, although there is a version of this pragma that allows asynchronous execution). The **in**, **out**, and **inout** clauses specify the direction of data to be transferred between the host and the target.

Variables used within an offloaded construct that are declared outside the scope of the construct (including the file-scope) are copied (by default) to the target before execution on the target begins and copied back to the host on completion.

For example, in the code below, the variable **ret** is automatically copied to the target before execution on the target and copied back to the host on completion. The offloaded code below is executed by a single thread on a single Intel MIC Architecture core.

```
float reduction_offload(float *data, int size)
{
    float ret = 0.f;
    #pragma offload target(mic) in(data:length(size))
    for (int i=0; i<size; ++i)
    {
        ret += data[i];
    }
    return ret;
}
```

Code Example 2: Serial Reduction with Offload

Vector Reduction with Offload

Each core on the Intel Xeon Phi coprocessor has a VPU. The auto vectorization option is enabled by default on the offload compiler. Alternately, as seen in the example below, the programmer can use the Intel® Cilk™ Plus Extended Array Notation to maximize vectorization and take advantage of the Intel MIC Architecture core's 32

512-bit registers. The offloaded code is executed by a single thread on a single core. The thread uses the built-in reduction function *__sec_reduce_add()* to use the core's 32 512-bit vector registers to reduce the elements in the array sixteen at a time.

```
float reduction_vectorreduction(float *data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(data:length(size))
    ret = __sec_reduce_add(data[0:size]); //Intel Cilk Plus
                                          //Extended Array Notation
    return ret;
}
```

Code Example 3: Vector Reduction with Offload in C/C++

Asynchronous Offload and Data Transfer

Asynchronous offload and data transfer between the host and the Intel Xeon Phi coprocessor is available. For details see the "About Asynchronous Computation" and "About Asynchronous Data Transfer" sections in the Intel® C++ Compiler User and Reference Guide (under "Key Features/Programming for the Intel MIC Architecture").

For an example showing the use of asynchronous offload and transfer, refer to

```
c:\Program Files (x86)\Intel\Composer XE\Samples\en_US\C++\mic_samples\
intro_sampleC\sampleC13.c
```

Note that when using the Explicit Memory Copy Model in C/C++, arrays are supported provided the array element type is scalar or a bitwise copyable struct or class. Arrays of pointers are not supported. For C/C++ complex data structure, use the Implicit Memory Copy Model. Please consult the section "Restrictions on Offload Code Using a Pragma" in the document "Intel® C++ Compiler XE 13.0 User and Reference Guide" for more information.

Using the Offload Compiler – Implicit Memory Copy Model

The Implicit Memory Copy model is not currently available. It will be available in a future release.

Native Compilation

Applications can also be run natively on the Intel Xeon Phi coprocessor, in which case the coprocessor will be treated as a standalone multicore computer. Once the binary is built on the host system, copy the binary and other related binaries or data to the Intel Xeon Phi coprocessor's filesystem (or make them visible on the coprocessor via NFS).

Example:

1. Unzip `<install-dir>\Samples\en_US\C++\openmp_samples.zip` and copy `<install-dir>\Samples\en_US\C++\openmp_samples\openmp_sample.c` to your home directory.

2. Build the application with the `/Qmic` flag:

   ```
   icl /Qmic -openmp openmp_sample.c
   ```

3. Upload the binary to the coprocessor `mic0`:

   ```
   pscp -i id_rsa.ppk a.out root@192.168.1.100:/tmp/a.out
   ```

4. Copy over any shared libraries required by your application, in this case the OpenMP* runtime library:

   ```
   pscp -i id_rsa.ppk "c:\Program Files (x86)\Common Files\Intel\Shared
   Libraries\compiler\lib\mic\lib\libiomp5.so" root@192.168.1.100:/tmp/
   libiomp5.so
   ```

5. Connect to the coprocessor with `PuTTY*` and export the local directory so that the application can find any shared libraries it uses (in this case the OpenMP* runtime library):

   ```
   export LD_LIBRARY_PATH=/tmp
   ```

6. This application may generate a segmentation fault if the stacksize is not set correctly. To modify the stacksize use:

   ```
   ulimit -s unlimited
   ```

7. Go to `/tmp` and run `a.out`:

   ```
   cd /tmp
   ./a.out
   ```

Parallel Programming Options on the Intel Xeon Phi coprocessor

Most of the parallel programming options available on the host systems are available for the Intel Xeon Phi coprocessor. These include the following:

1. Intel Threading Building Blocks (Intel TBB)
2. OpenMP*
3. Intel® Cilk Plus
4. pthreads*

The following sections will discuss the use of these parallel programming models in code using the offload extensions. Code that runs natively on the Intel Xeon Phi coprocessor can use these parallel programming models just as they would on the host, with no unusual complications beyond the larger number of threads.

Parallel Programming on the Intel Xeon Phi coprocessor: OpenMP*

There is no correspondence between OpenMP threads on the host CPU and on the Intel Xeon Phi coprocessor. Because an OpenMP parallel region within an offload/pragma is offloaded as a unit, the offload compiler creates a team of threads based on the available resources on Intel Xeon Phi coprocessor. Since the entire OpenMP construct is executed on the Intel Xeon Phi coprocessor, within the construct the usual OpenMP semantics of shared and private data apply.

Multiple host CPU threads can offload to the Intel Xeon Phi coprocessor at any time. If a CPU thread attempts to offload to the Intel Xeon Phi coprocessor and resources are not available on the coprocessor, the code meant to be offloaded may be executed on the host. When a thread on the coprocessor reaches the "omp parallel" directive, it creates a team of threads based on the resources available on the coprocessor. The theoretical maximum number of hardware threads that can be created is 4 times the number of cores in your Intel Xeon Phi coprocessor. The practical limit is four less than this (for offloaded code) because a core is reserved for the uOS and its services.

The code shown below is an example of a single host CPU thread attempting to offload the reduction code to the Intel Xeon Phi coprocessor using OpenMP in the offload construct.

```
float OMP_reduction_OMP(float *data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(size) in(data:length(size))
    {
        #pragma omp parallel for reduction(+:ret)
        for (int i=0; i<size; ++i)
        {
            ret += data[i];
        }
    }
    return ret;
}
```

Code Example 4: C/C++: Using OpenMP* in Offloaded Reduction Code

For a Fortran example showing the use of OpenMP* in Offload Reduction Code, please refer to <install-dir>`\Samples\en_US\Fortran\mic_samples\LEO_Fortran_intro`

Parallel Programming on the Intel Xeon Phi coprocessor: OpenMP* + Intel Cilk Plus Extended Array Notation

The following code sample further extends the OpenMP example to use Intel Cilk Plus Extended Array Notation. In the following code sample, each thread uses the Intel Cilk Plus Extended Array Notation

__sec_reduce_add() built-in reduction function to use all 32 of the Intel MIC Architecture's 512-bit vector registers to reduce the elements in the array.

```
float OMPnthreads_CilkPlusEAN_reduction(float *data, int size)
{
    float ret=0;
    #pragma offload target(mic) in(data:length(size))
    {
        int nthreads = omp_get_max_threads();
        int ElementsPerThread = size/nthreads;
        #pragma omp parallel for reduction(+:ret)
        for(int i=0;i<nthreads;i++)
        {
            ret =__sec_reduce_add(
                   data[i*ElementsPerThread:ElementsPerThread]);
        }
        //rest of the array
        for(int i=nthreads*ElementsPerThread; i<size; i++)
        {
            ret+=data[i];
        }
    }
    return ret;
}
```

Code Example 5: Array Reduction Using Open MP and Intel Cilk Plus in C/C++

Parallel Programming on the Intel Xeon Phi coprocessor: Intel Cilk Plus

Intel Cilk Plus header files are not available on the target environment by default. To make the header files available to an application built for the Intel MIC Architecture using Intel Cilk Plus, wrap the header files with *#pragma offload_attribute(push,target(mic))* and *#pragma offload_attribute(pop)* as follows:

```
#pragma offload_attribute(push,target(mic))
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
#pragma offload_attribute(pop)
```

Code Example 6: Wrapping the Header Files in C/C++

In the following example, the compiler converts the *cilk_for* loop into a recursively called function using an efficient divide-and-conquer strategy.

```
float ReduceCilk(float*data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(data:length(size))
    {
        cilk::reducer_opadd<int> total;
        cilk_for (int i=0; i<size; ++i)
        {
            total += data[i];
        }
        ret = total.get_value();
    }
    return ret;
}
```

Code Example 7: Creating a Recursively Called Function by Converting the "*cilk_for*" Loop

Parallel Programming on Intel Xeon Phi coprocessor: Intel TBB

Like Intel Cilk Plus, the Intel TBB header files are not available on the target environment by default. They are made available to the Intel MIC Architecture target environment using similar techniques:

```
#pragma offload_attribute (push,target(mic))
#include "tbb/task_scheduler_init.h"
#include "tbb/blocked_range.h"
#include "tbb/parallel_reduce.h"
#include "tbb/task.h"
#pragma offload_attribute (pop)

using namespace tbb;
```

Code Example 8: Wrapping the Intel TBB Header Files in C/C++

Functions called from within the offloaded construct and global data required on the Intel Xeon Phi coprocessor should be prepended by the special function `__declspec(target(mic))` .

As an example, *parallel_reduce* recursively splits an array into subranges for each thread to work on. The *parallel_reduce* uses a splitting constructor to make one or more copies for each thread. For each split, the method join is invoked to accumulate the results.

1. Prefix the class by the macro `__MIC__` and the class name by `__declspec(target(mic))` if you want them to be generated for the coprocessor.

```
#ifdef __MIC__
class __declspec(target(mic)) ReduceTBB
{
private:
    float *my_data;
public:
    float sum;
    void operator()( const blocked_range<size_t>& r )
    {
        float *data = my_data;
        for( size_t i=r.begin(); i!=r.end(); ++i)
        {
            sum += data[i];
        }
    }

      ReduceTBB( ReduceTBB& x, split) : my_data(x.my_data), sum(0) {}

      void join( const ReduceTBB& y) { sum += y.sum; }

      ReduceTBB( float data[] ) : my_data(data), sum(0) {}
};
#endif
```

Code Example 9: Prefixing an Intel TBB Class for Intel MIC Architecture code generation in C/C++

2. Prefix the function to be offloaded to the Intel Xeon Phi coprocessor by `__declspec(target(mic))`

```
__declspec(target(mic))
float MICReductionTBB(float *data, int size)
{
    ReduceTBB redc(data);
    // initializing the library
    task_scheduler_init init;
    parallel_reduce(blocked_range<size_t>(0, size), redc);
    return redc.sum;
}
```

Code Example 10: Prefixing an Intel TBB Function for Intel MIC Architecture code generation in C/C++

3. Use #pragma offload target(mic) to offload the parallel code using Intel TBB to the coprocessor

```
float MICReductionTBB(float *data, int size)
{
    float ret(0.f);
    #pragma offload target(mic) in(size) in(data:length(size)) out(ret)
    ret = _MICReductionTBB(data, size);
    return ret;
}
```

Code Example 11: Offloading Intel TBB Code to the coprocessor in C/C++

NOTE: Codes using Intel TBB with an offload should be compiled with `/Qtbb` flag

Using Intel MKL

For offload users, Intel MKL is most commonly used in Native Acceleration (NAcc) mode on the Intel Xeon Phi coprocessor. In NAcc, all data and binaries reside on the Intel Xeon Phi coprocessor. Data is transferred by the programmer through offload compiler pragmas and semantics to be used by Intel MKL calls within an offloaded region or function. NAcc functionality contains BLAS, LAPACK, FFT, VML, VSL, (Sparse Matrix Vector), and required Intel MKL Service functions. Please see the Intel MKL release documents for details on which functions are optimized and which are not supported.

The Native Acceleration Mode can also be used in native Intel MIC Architecture code – in this case the Intel MKL shared libraries must be copied to the Intel Xeon Phi coprocessor before execution.

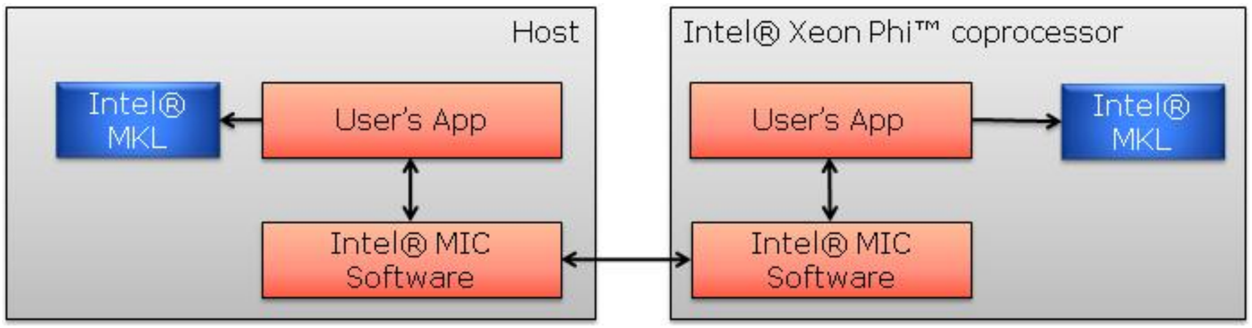


Figure 6: Using MKL Native Acceleration with Offload

SGEMM Sample

Using SGEMM routine from BLAS library

Sample Code – sgemm

Step 1: Initialize the matrices, which in this example need to be global variables to make use of data persistence.

Step 2: Send the data over to the Intel Xeon Phi coprocessor using `#pragma offload`. In this example, the `free_if(0)` qualifier is used to make the data persistent on the Intel Xeon Phi coprocessor.

```
#define PHI_DEV 0
#pragma offload_transfer target(mic:PHI_DEV) \
    in(A:length(matrix_elements) free_if(0)) \
    in(B:length(matrix_elements) free_if(0)) \
    in(C:length(matrix_elements) free_if(0))
```

Code Example 12: Sending the Data to the Intel Xeon Phi coprocessor

Step 3: Call sgemm inside the offload section to use the "Native Acceleration" version of Intel MKL on the Intel Xeon Phi coprocessor. The `nocopy()` qualifier causes the data copied to the card in step 2 to be reused.

```
#pragma offload target(mic:PHI_DEV) \
    in(transa, transb, N, alpha, beta) \
    nocopy(A: alloc_if(0) free_if(0)) nocopy(B: alloc_if(0) free_if(0)) \
    out(C:length(matrix_elements) alloc_if(0) free_if(0)) // output data
    {
        sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
            &beta, C, &N);
    }
```

Code Example 13: Calling sgemm Inside the Offload Section

Step 4: Free the memory you copied to the card in step 2. The `alloc_if(0)` qualifier is used to reuse the data on the card on entering the offload section, and the `free_if(1)` qualifier is used to free the data on the card on exit.

```
#pragma offload_transfer target(mic:PHI_DEV) \
        nocopy(A:length(matrix_elements) alloc_if(0) free_if(1)) \
        nocopy(B:length(matrix_elements) alloc_if(0) free_if(1)) \
        nocopy(C:length(matrix_elements) alloc_if(0) free_if(1))
```

Code Example 14: Set the Copied Memory Free

As with Intel MKL on any platform, it is possible to limit the number of threads it uses by setting the number of allowed OpenMP threads before executing the MKL function within the offloaded code.

```
#pragma offload target(mic:PHI_DEV) \
    in(transa, transb, N, alpha, beta) \
    nocopy(A: alloc_if(0) free_if(0)) nocopy(B: alloc_if(0) free_if(0))
    out(C:length(matrix_elements) alloc_if(0) free_if(0)) // output data
    {
        omp_set_num_threads(64); // set num threads in openmp
        sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
            &beta, C, &N);
    }
```

Code Example 15: Controlling Threads on the Intel Xeon Phi coprocessor Using omp_set_num_threads()

Intel MKL Automatic Offload Model

MKL Automatic Offload Model is not available, but it will be supported in a near future.

Debugging on the Intel Xeon Phi coprocessor

You will find information specific to debugging Intel MIC Architecture applications in **`<install-dir>\Documentation\en_US\debugger\gdb\gdb.pdf`**

Performance Analysis on the Intel Xeon Phi coprocessor

Users can refer to the document "Optimization – Part 2: Hardware Events" for optimizing applications on the Intel Xeon Phi coprocessor using VTune™ Amplifier XE 2013 for Windows. This document can be found at http://software.intel.com/mic-developer under "**Programming**" tab, and "**Optimization**" section.

Appendix A: Basic Linux Commands

This appendix shows some common Linux commands for use on the Intel Xeon Phi coprocessor.

1. Logout of the coprocessor:

   User can log out by using "`exit`" command in coprocessor terminal. User is logged out and the terminal disappears.

   ```
   > exit
   ```

2. List files and directories in the current directory: "`ls`"

   "`ls`" can be used to list all files and directories

   "`ls -l`" is used to list all files and directories with all attributes.

   ```
   > ls
   a.out
   libioomp5.so
   .........
   ```

3. Retrieve the path of the current directory: "`pwd`"

   ```
   > pwd
   /root
   ```

4. To Navigate to a certain directory: "`cd <path>`"

   ```
   > cd /tmp
   ```

5. List all current running processes: "`ps`"

   ```
   > ps
   5847 root    0:00 /sbin/sshd
   5914 micuser 2:47 /bin/coi_daemon -coiuser=micuser
   .........
   ```

6. Kill a process by its process identification: "`kill -9 <pid>`"

   ```
   > kill -9 4555
   ```

7. List the processes which consume the most CPU: "`top`"

   ```
   > top
   ```

8. Copy a file: "`cp <sourcefilename> <destinationfilename>`"

   ```
   > cp file1 file2
   ```

9. Remove (delete) a file: "`rm <filename>`"

   ```
   > rm file1
   ```

10. Review the content of a file: "`less` *`<filename>`*"
    ```
    > less file2
    ```

11. Search a line that matches a pattern: "`grep` *`<pattern>`*". For example, to search any process that have name with "`coi`", you can combine `grep` and `ps`.
    ```
    > ps | grep coi
    5914 micuser 2:47 /bin/coi_daemon -coiuser=micuser
    ```

12. Use "`export`" to set a specific environment variable

    ```
    > export LD_LIBRARY_PATH=/tmp
    ```

13. Use "ctrl key" and "C" to terminate the current running task.

About the Author

Loc Q Nguyen received an MBA from University of Dallas, a master's degree in Electrical Engineering from McGill University, and a bachelor's degree in Electrical Engineering from École Polytechnique de Montréal. He is currently a software engineer with Intel Corporation's Software and Services Group. His areas of interest include computer networking, computer graphics, and parallel processing.

Notices

Optimization Notice