CHAPTER 12

Xeon Phi Application Development on Windows OS

So far we have looked at application development on the Linux OS for the Xeon Phi coprocessor. This chapter looks at what types of support are available on Windows OS for developing applications for Xeon Phi.[1] Some application domains such as computer-aided design (CAD) and other workstation applications that can benefit from the raw compute power of Intel Xeon Phi are mostly used on Windows OS. In such cases, you would be able to offload part of the computationally intensive code section to the coprocessor by using the offload programming model, such as that based on the OpenMP 4.0 standard. Most of the concepts in this chapter also apply to the Linux development environment on Xeon Phi.

The Xeon Phi development OS environment includes support for Microsoft Windows Server 2008 R2 SP1 (64 bit), Windows Server 2012 (64 bit), Windows 8 Enterprise (64 bit), and Windows 7 Enterprise SP1 (64 bit). Windows OS supports familiar tools, such as Microsoft Visual Studio IDE with Intel Compiler and tools to be used for your code development within the Windows OS. The Intel compilers and tools for Xeon Phi plug into the Microsoft Visual Studio IDE to enable such development.

The system configuration for the Windows host is similar to that for the Linux host. The card resides on a PCIe x16 bus. As discussed in Chapter 7, the coprocessor hosts a Linux-based coprocessor operating system. The same is true in the Windows environment. The Windows driver is responsible for loading the coprocessor operating system on Xeon Phi as it is for the Linux environment. However, the Windows configuration makes the available supported programming model narrower than that provided on the Linux host OS. Because the Windows platform was designed to leverage the investment made in the Linux development environment, you may see some syntax, such as compiler switches for offload compilation, that is same as that in the Linux environment. Most of the frequently used system management tools on Linux are also available on Windows to manage Xeon Phi.

The Windows OS supports both the offload and native programming models. You can use internet protocol–based tools such as the secured shell `SSH` for Windows to communicate with the coprocessor OS based on Linux. You may need to learn Linux operating system commands for application execution under native mode inside the coprocessor OS. You can also share file directories between the Windows host and the Linux coprocessor OS by virtue of its support for the standard *network file system* (NFS) protocol.

The Windows development environment consists of the Intel MPSS stack, the C/C++ and Fortran compilers that plug into Microsoft Visual Studio, the debugger, and VTune Amplifier XE for application analysis.

[1]This chapter describes the beta version of the Windows support for Xeon Phi.

MPSS

The Windows version of the MPSS installation requires administrative privilege and .Net Framework 4.0 or higher.[2] The GUI-based installation process is straightforward. If, once the driver installation is complete, the MPSS service is not started by the Windows trust manager, the administrator may need to configure the Xeon Phi coprocessor environment by using the command line tools provided as part of the MPSS installation.

The MPSS install package contains:

- Device drivers and OS software for Xeon Phi
- Tools to control Intel Xeon Phi and collect coprocessor status
- SDKs for developing coprocessor applications. The SDKs contain samples for *common offload infrastructure* (COI), SCIF, and virtual shared memory programming—*mine-your-ours* (MYO) interface. It also provides basic libraries to build applications for the coprocessor.

The user may choose either a custom-install configuration allowing selection of which components to install or the default install of all tools.

Figure 12-1 of the various system components in the Windows environment shows how the host side of the MPSS provides the Windows device driver to manage the Xeon Phi hardware and the necessary communication channel with the host applications. The coprocessor OS provides necessary supports to run native and offload applications on Xeon Phi hardware.

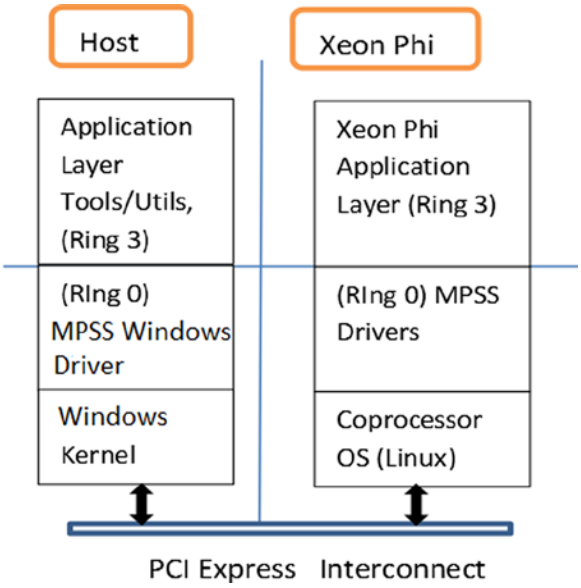

Figure 12-1. *System components in the Microsoft Windows environment*

The system software also provides important utilities for managing the coprocessor. The default install directory for MPSS is `c:\Program Files\Intel\MPSS`.

[2]For installation details for the Windows version of the MPSS, see the ReadMe document available with the MPSS stack.

The Windows driver is responsible for initializing the coprocessor OS and setting up the default file system layout. The coprocessor OS that is downloaded to the coprocessor during the Xeon Phi boot process resides on the Windows host file system and is by default installed in the `c:\Program Files\Intel\MPSS\drivers` folder.

The docs folder in the MPSS install directory on the host contains descriptions and usage of the tools provided with the MPSS install.

MPSS Tools

The tools installed on the host side in the default path `c:\Program Files\Intel\MPSS\bin` include:

1. `MicInfo`: Used to get information about the Xeon Phi cards installed on the system along with relevant information on the host system, the coprocessor OS, and the MPSS versions. The tool can be optionally invoked with following command line options:
   - `-version`: Displays tool version
   - `-help`: Displays command line help
   - `-listDevices`: Lists all the Xeon Phi devices detected on the host
   - `-deviceInfo <deviceNum>[-group <groupname>]`: Displays information about specific Xeon Phi devices numbered deviceNum. By default the utility dumps various groups of information about the card, OS, and MPSS together. However, you can use `<groupname>` to select only specific information to be displayed. The `groupname` may be one of the following:
   - `Versions`: Shows Flash and coprocessor OS versions
   - `Board`: Shows only coprocessor card-related information
   - `Core`: Shows Xeon Phi core-related information
   - `Thermal`: Shows fan and thermal-related data for the coprocessor card
   - `GDDR`: Shows GDDR memory-related information
2. `MicSmc`: Provides coprocessor status and utilization information in the status panel. This tool can execute in both GUI and command line mode. The GUI mode provides real-time monitoring of the coprocessor cards installed on the host. It is also used to change settings on the card, such as ECC on/off and Turbo on/off. The GUI mode is invoked by executing the `micsmc` utility without any parameters. In the GUI mode, you can see the list of all the Xeon Phi cards recognized by the driver and real-time information on core usage, core temperature, memory usage, power usage, total memory, and number of cores, either cumulatively on all cards or individually. The status panel in Figure 12-2 shows two cards on the system with a total of 122 cores and 32GB of memory. By enabling the display of mic0 and mic1, you can see the status and power consumption of the individual cards. In this case, I disabled power management to be able to use `time stamp counter` without being affected by the power management feature of the core processor, which might affect the timing procedure. You can also choose between core histogram view and historical utilization view by selecting icons in the top right corners of the individual coprocessor windows. On the Advanced menu on the GUI of the tool, you can look at the error log, change card settings such as ECC and power management support, and get information about individual cards, such as their device driver versions.

Figure 12-2. *MicSmc GUI mode execution with two cards available on the system*

ECC Mode Changes: Through `MicSmc` GUI, you can change the ECC settings of individual cards by using the Advanced ➤ Settings menu. This will put the card in maintenance mode, change the settings, and restart the coprocessor to make the changes effective. If an error condition is encountered, an error dialog will appear detailing the information. It will also be logged in the error log that can be viewed by the Advanced ➤ Error Log menu option. Similar setting changes can be done for Turbo and power management states. In addition, the LED mode setting is used to blink the LEDs on individual cards to identify the cards visually.

3. `MicRas`: A utility used on the host system to collect and log *reliability*, *availability*, and *serviceability* (RAS) events produced by Xeon Phi coprocessor. This utility is run as a Windows service and monitors any RAS event generated by the card. If it detects a fatal RAS event, it puts the card or cards in maintenance mode. `MicRas` logs messages into the file micras.log under the binary directory where it is installed. `MicRas` service depends on MPSS service. The command line invocation of `MicRas` can take optional parameters and can be as follows:

   - `-help`: Displays help information
   - `-daemon`: This option starts `MicRas` in the daemon mode and runs in the background. In this mode it can handle RAS events and logs the messages silently.
   - `-loglevel [level]`: Sets the category of messages. The `level` is a bitfield value with least 4 bits used to define the categories as follows:
     - Bit 0: Logs informational messages
     - Bit 1: Logs warning messages
     - Bit 2: Logs error messages
     - Bit 3: Logs critical messages

The levels are defined to follow IETF RFC 5424 Syslog protocol standard.

4. `MicFlash`: A utility is used to program firmware on the coprocessor cards. This is done by updating the SMC boot loader as well as the flash on the card. The firmware and flash images are installed by default in the `c:\Program Files\Intel\MPSS` directory. Make sure you select the correct flash image to match the revision of Xeon Phi card you have. The flash update is done by putting the card in the ready state and then burning the new flash image by executing the following command (assuming default install directory):

   ```
   Command_prompt > MicFlash –update <path to flash file> -device all
   ```

5. `micctrl`: An important utility used to start, stop, and restart the coprocessor. You can use `micctrl --start` to start the Xeon Phi coprocessor, which performs the device initialization together with uploading the coprocessor OS and getting the coprocessor card to a usable stage. To stop the driver run, you can issue the command `micctlr –-stop`. If you want to just reboot the coprocessor, a useful command is `micctrl –b`.

SDK (Binutils)

SDKs installed with the MPSS are a collection of tools and libraries that include files necessary to build and cross-compile Xeon Phi applications on Windows. These components are used by Intel Composer XE tools to build Xeon Phi applications. The utility install program comes with the MPSS install package and may be installed explicitly by using the provided `binutil` install batch file.

Development Tools

The development tools for Xeon Phi on Windows include the C/C++ and Fortran compilers, debuggers, and VTune Amplifier XE application analyzers. The debugger and compiler are part of the Intel Composer XE 2013 for Windows package. These tools can recognize the code blocks to be offloaded to Xeon Phi and provide runtime support for executing offloaded code sections on Xeon Phi coprocessor(s). The language extensions for Xeon Phi in Windows are very similar to those of the Linux host OS.

The Composer XE installation comes with documentation and samples to get you started on programming the Xeon Phi coprocessor.

Language Extensions for the Xeon Phi Coprocessor

Composer XE for Windows provides two methods of offloading computation and sharing data between the host and the coprocessor: the nonshared memory model (*data marshaling*) and the virtual shared memory model, which is available only for C/C++ programs. Most of the features and language extensions for Xeon Phi that are available on Linux are still available on Windows. There are certain restrictions on the Windows offload model, which are valid in host-based Windows code and must be delineated by `__MIC__` or `__INTEL_OFFLOAD` macros and may need to be placed in a separate file in certain cases, as follow:

- No support for `long and long double` data types
- No `bitfield` support in data structure
- No runtime type information (RTTI) support

- No `structured exceptions handling in offload` code
  - For example, avoid code such as the following in the offload code:

```
void f()
{
  try {<statements>}
  catch[<error type>] { <statements> }
  finally {<statements>}
}
```

- No support for `#pragma pack`
- No `dll_import/dll_export` such as `__declspec(dllexport) void f();` in the offload code
- No pointer fields and only bitwise copyable object for nonshared memory model
- You may not use Windows calling convention `__cdecl` to declare an external callable function called from offload code
- Do not include Windows header `#include <windows.h>` in the offload code sections

Nonshared Memory Offload Extensions

Intel Composer XE 2013 for Xeon Phi supports proprietary extensions for nonshared memory program execution from the Windows host to the Xeon Phi coprocessor. The compiler also supports standard OpenMP 4.0 extensions for performing nonshared memory programming. Because the OpenMP 4.0 standard is still in progress, this chapter discusses only the proprietary Intel extension. The concept of the nonshared memory programming model is, however, very similar in the OpenMP 4.0 and Intel proprietary extensions.

These extensions provide the high-level syntax and runtime environment in C/C++ and Fortran to execute portions of the code on Xeon Phi. These methods use data marshaling techniques for transferring data between the host and the coprocessor and as such are appropriate for dealing with flat-data structures such as bitwise copyable structures, arrays, and scalars. On invocation of the offload code, the data are copied between the host and the card as indicated by the programmer. The data exchanged include data explicitly listed in the offload clause as well as implicitly defined data objects that are lexically referenced inside the offload code.

Virtual Shared Memory Offload Extensions (C/C++ Language Only)

The offload keywords `_Cilk_shared`, `_Cilk_offload`, and `_Cilk_offload_to` are defined for virtual shared memory programming support in Composer XE for Windows. These are useful when dealing with complex data structures such as linked lists, which need to be shared between the host and the Xeon Phi card.

Compiling and Running the Offload Programs

You can use either the command line option or Microsoft Visual Studio to build the offload programs for Windows. By default the C/C++ and Fortran compiler builds code for both the host and the Xeon Phi coprocessor. To compile and run in command line mode, start a command prompt with the proper environment variables set. You can start the command prompt as shown in Figure 12-3. The compiler detects the offload keywords and performs the offload compilation when encountered. The compiled binary is placed as designated by the Makefile and can be executed from the command line.

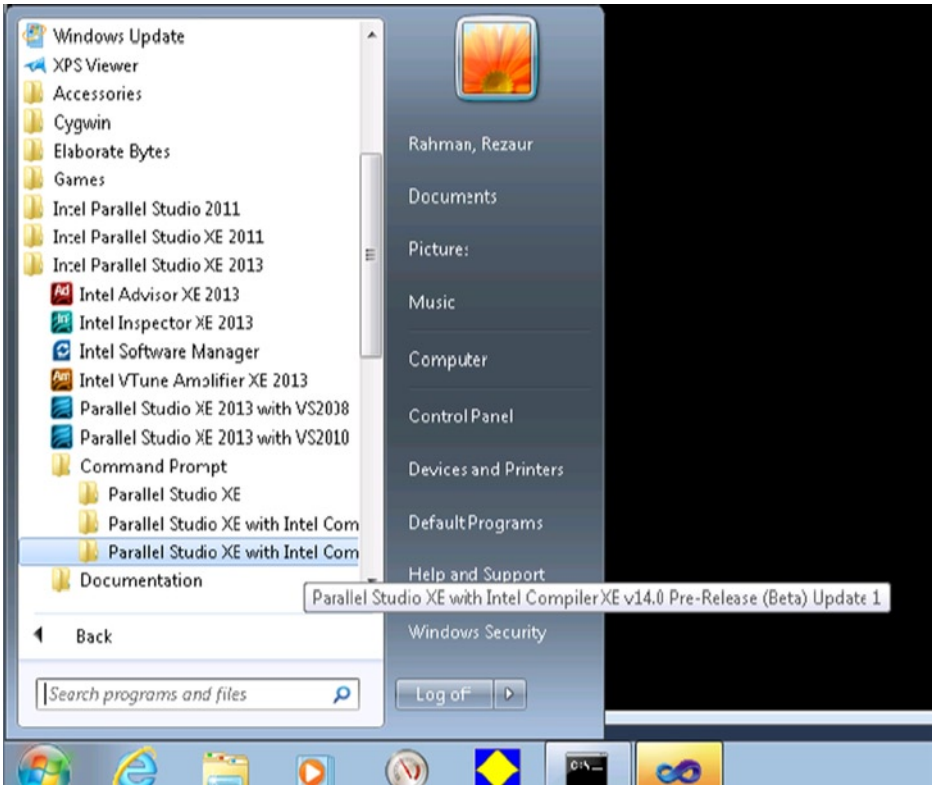

Figure 12-3. *Starting command prompt on Windows to build Xeon Phi applications*

Note that offload code generation may not imply offload code execution. Offload code execution will depend on the presence and availability of the Xeon Phi coprocessor in the system. The offload may be designated as mandatory or optional by either of the following alternatives:

1. Using the `mandatory`, `optional`, and `status` clauses with offload pragmas or directives.
2. Using `/QOffload [none|optional|mandatory]` command line options. The default is `mandatory`.

The clauses overwrite the compile time options.

For `optional` offload, if the offload request can be satisfied, the code is executed on the coprocessor; otherwise the code is executed on the host. If the `status` clause is used, the status variable can be queried for details of the offload. The `status (var)` clause may be used with `offload`, `offload_transfer`, and `offload_wait` directives. It sets the variable of type `__Offload_status` defined in `offload.h` in C/C++ or of type `Offload_status` for Fortran defined in `mic_lib.f90`. The `status` variable has a predefined member that can be queried for the success of the `offload` clauses during program execution.

In `mandatory` offload, if the offload clause can be satisfied, the code is executed on the coprocessor, otherwise the code is not automatically executed on the host. If the `status` clause is specified, the program continues and the user may examine the status code to take appropriate option. If the status variable is not specified, the application is terminated for mandatory offload case.

Using Visual Studio Integrated Development Environment

Intel Composer XE integrates into Microsoft Visual Studio Integrated Development Environment (IDE) by default during installation. Offload applications can use the same project and solution types as nonoffload applications except that they are limited to x64 release and debug configurations.

To compile and run an application, launch Microsoft Visual Studio and create appropriate projects and add source files to it. Alternatively, you can open an existing Visual Studio project. You can then use the Build Solutions or Projects page to build or run the application.

To control the behavior of offload compilation, you can set the configuration properties of a specific project. You can also set the behavior of offload constructs from the configuration properties. In Figure 12-4, the default offload behavior of the program is set to mandatory, as shown by the highlighted offload construct field of the property table. This tells the runtime environment to execute on the coprocessor or fail if the coprocessor is not available. Setting the option to None instructs the compiler to ignore all the offload constructs.

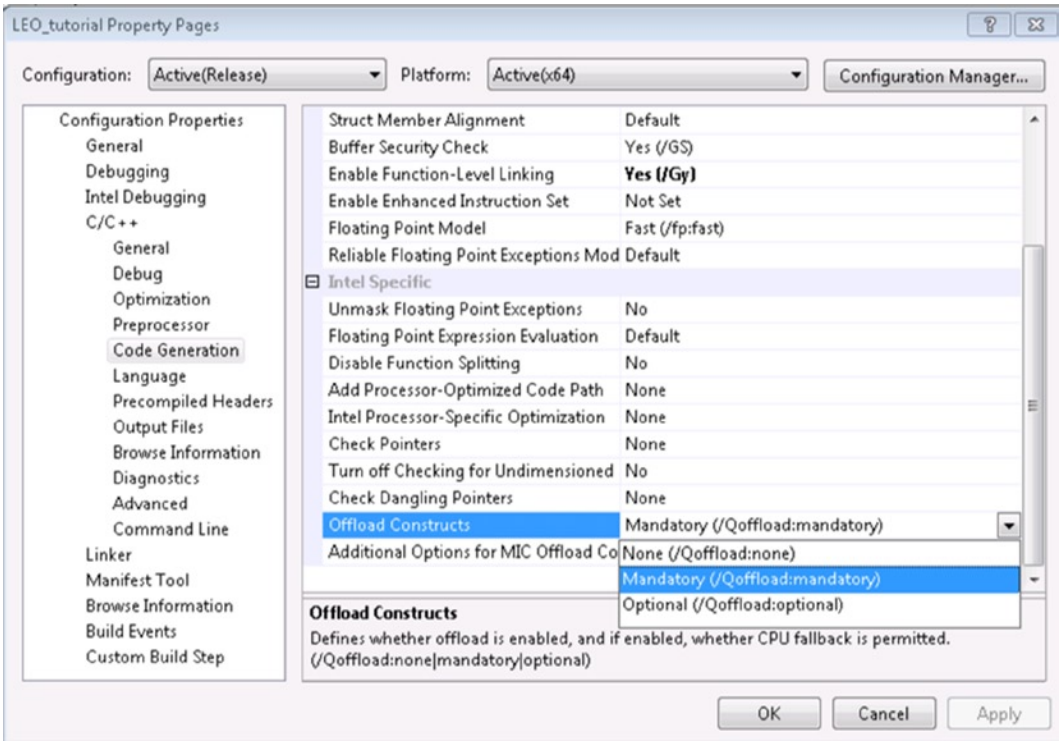

Figure 12-4. *Setting the behavior of offload constructs*

You can also provide additional options for the offload compiler by entering Linux-like command line options in the field "Additional Options for MIC Offload Compiler," shown in the next field below the highlighted field in Figure 12-4. For example, you can set the optimization level differently from your host compilation level.

Similarly options can be set specifically for the offload linker by entering the offload-specific link option in the dialog reached through the following navigation path: Projects ➤ Properties ➤ Configuration Properties ➤ Linker ➤ General ➤ Additional Options for MIC Offload Linker.

Compile Time Messages and Detecting Missing Symbols During Link Time

To differentiate between the compilation for the host and the coprocessor, the compiler prints out compile time messages with the special tag *MIC*, such as:

```
test.c(5): warning #120: *MIC* return value type does not match the function type
```

In the offload compilation model, the binaries are generated as Linux dynamic link libraries (files with the .so extension). Because the dynamic link libraries do not need to resolve the symbols during link time, the offload code may encounter runtime failure if the missing symbols are encountered during the runtime. You can use the following linker option for offload compilation to detect any missing symbols during compile time: `/Qoffload-option,mic,link,"-Wl,-no-undefined"`.

Offload Environment Variables

As with the Linux host OS execution environment, Windows also allows you to control offload execution using specific environment variables. The variable can be used to set the coprocessor OS execution environment from the Windows host. The environment variable MIC_ENV_PREFIX is used to distinguish variables that control the execution environment on the coprocessor. The variables are common between the Linux and Windows execution environment, except that you need to follow Windows conventions when specifying file system paths in Windows.

Debugging Offload Execution

Offload and native application debugging is supported only under Visual Studio (VS) 2012. This allows for seamless debugging under IDE for C++/Fortran code on multiple coprocessors. The debugger framework includes a debug engine and VS plugin that integrates GNU Project Debugger (GDB) for Xeon Phi into Microsoft Visual Studio 2012. These extensions are part of the Intel Composer XE 2013 SP1 and later products.

The offload debugging is started by a VS plugin when it detects a breakpoint inside offload region. The plugin spawns the debug engine which automatically connects to `offload_main` process running on the coprocessor. Debugging of offload applications requires specific PuTTY utilities, plink and pscp, and private key file id_rsa.ppk be installed on MPSS bin directory whose default path is `c:\Program Files\Intel\MPSS\bin`. You need to set up PuTTY so that you can access the Xeon Phi card from the host as explained in the next section.

Set up the debugger by the following steps:

1. Install required PuTTY utilities.
    a. Install it under `c:\Program Files\Intel\MPSS\bin` for default installs or appropriately if the MPSS path is different.
    b. If PuTTY is installed in a separate directory, copy plink.exe and pscp.exe utilities from the PuTTY install folder to the `c:\Program Files\Intel\MPSS\bin` directory.
2. Create key files private, public, authorized_keys under `c:\Program Files\Intel\MPSS\bin`.
3. The private key must correspond to the `authorized_keys` uploaded to Xeon Phi coprocessor. The private key must be renamed to `id_rsa.ppk` for recognition by Intel GNU GDB extension for Windows.

Logging into Xeon Phi Console using PuTTY

Because the Xeon Phi card hosts a Linux-based operating system, you can log in to the coprocessor OS as if it were a separate Linux node. After the MPSS is installed and running, you can directly log in to the Xeon Phi coprocessor OS for executing applications natively or for other purposes. To do so, you need to use standard `SSH` tools such as PuTTY.[3] To use SSH to log into the coprocessor, you need to have been added by the administrator as a user in the `micuser` group, and the admin needs to set up a public key for you on the Xeon Phi coprocessor OS to give you access, through `micctrl –addssh <username> <public-key>`, where `<username>` is your username and `<public-key>` is the public portion of the key pairs generated by you using a utility such as `PuTTY Key Generator (PuTTYgen)`. PuTTYgen can be used to generate authorized_keys, public keys, and private keys.

Once this is set up, you can log in to the coprocessor using the IP address of the Xeon Phi coprocessor card you are interested in. The default IP address is set to 192.168.1.100. You can use WinSCP to transfer files between the host and the coprocessor.

Using VTune Amplifier XE to Profile Offload Code on Windows

The VTune Amplifier XE tool can be used to understand performance issues on Xeon Phi execution on a Windows host, just as it is in the Linux environment. To be able profile and locate where in the source the issue is, the code to be analyzed needs to have debug symbols. Use the compile and link time option `/debug:` full to enable symbols.

Set up the VTune configuration so that the source and binary search path point to the relevant files. The VTune comes with predefined analysis types such as `Lightweight Hotspots`, `General Exploration`, and `Bandwidth` to help you get started with the profiling session. In many cases, these profiles point you to possible hotspots. You can add your own custom profile to enhance your VTune analysis.

Building and Running Xeon Phi Native Applications from the Windows Host

You can cross-compile code for Xeon Phi using Intel Composer XE from a Windows command prompt. To compile, start the Intel Parallel Studio XE 2013 command-prompt window for Intel 64 for a given VS configuration. Use the compile switch `/Qmic` to cross-compile the source for Xeon Phi native execution.

Because native programs build applications to run on the Xeon Phi Linux coprocessor OS, the binary code is a Linux application and the compiler takes Linux switches.

Once built, the binary and relevant dynamic libraries (.so) files need to be copied to the Xeon Phi coprocessor using pscp or similar secured copy tools. The mic runtime libraries such as libiomp5.so for OpenMP runtime are installed under Composer XE install directories `<install_dir>\compiler\lib\mic`.

You can debug the native applications from VS using procedures similar to those used for a remote Linux target.[4]

Summary

This chapter looked at the development, debug, and application profiling supports on the Windows platform for the Xeon Phi coprocessor. These procedures closely follow those for Linux-based development, and many of the concepts presented throughout this book are equally applicable for Windows-based application development and profiling.

[3] http://www.putty.org/

[4] For debugging details, see the Intel Composer XE user guide.