Xeon Phi Processors with Parallel Studio XE 2017

Table of Contents

Xeon Phi Overview....................................................................................................................................................2
Using VTune Amplifier XE ...........................................................................................................................................2
Using Inspector XE ......................................................................................................................................................3
Using Advisor XE ......................................................................................................................................................4
Using Intels C++ Compiler .....................................................................................................................................7
**Xeon Phi Overview:**

We have four Xeon Phi nodes, each with Xeon Phi 7210 processors, 64 cores, and 48 GB of RAM. These Xeon Phi nodes utilize Intel’s second-generation Xeon Phi chips named Knights Landing. Knights Landing is an unprecedented processor with the ability to perform with highly parallel applications and large data sets, while being compatible with the programming models and tools common to Intel’s processors. It is Intel’s first self-boot Xeon Phi processor that is binary compatible and also provides a significant improvement in scalar and vector performance, with an innovative memory architecture for high bandwidth and capacity. Effectively, these nodes are native mode only systems, thus all programs are run in native mode and programs would not use offload programming.

**Using VTune Amplifier XE**

Overview:

Intel’s VTune Amplifier XE is an easy to use application that is made to identify the functions, loops, and files that are going to have the largest impact on your applications performance, while also being able to perform hotspot analysis, thread profiling, lock & waits analysis, and much more.

Example:

In this example, the Intel VTune Amplifier will perform a hotspot analysis on a 3D rendering application called Tachyon and generating a hotspot report from the results. First, you must load the necessary environment variables in order to use the VTune Amplifier tools and then copy the sample application into your working directory to make the program.

```
-> source /opt/intel/vtune_amplifier_xe_2017.3.0.510739/amplxe-vars.sh
-> cp /opt/intel/vtune_amplifier_xe_2017.3.0.510739/samples/en/C++/tachyon_vtune_amp_xe.tgz tachyon_vtune_amp_xe.tgz
-> tar -xzf tachyon_vtune_amp_xe.tgz
-> make
```

Then, to perform a hotspot analysis on the Tachyon application tachyon_find_hotspots.

```
-> amplxe-cl -collect hotspots -result-dir ~/XeonPhi/results ~/XeonPhi/tachyon/tachyon_find_hotspots
```

Where the flags “-collect” specifies what analysis type should be done and “-result-dir” designates the action option to save the analysis to a specified path. Running the command “amplxe-cl -help collect” shows the most updated list of available analysis types there are to be run, such as hpc-performance, memory-access, and system-overview.

To then generate a hotspots report for the results designated in the path specified by the -result-dir option.

```
-> amplxe-cl -report hotspots -result-dir ~/XeonPhi/results
```

Where “-report” specifies the type of report you want to create. Running the command “amplxe-cl -help report” shows the most updated list of available report types there are to be generated, such as summary, call stacks, and gprof-cc.
Below is the output of the report command.

You can also run an analysis type with a designated knob (configuration option). Below is the syntax to run a hotspot analysis with a designated knobValue and target application.

```
-> amplxe-cl -collect hotspots -knob <knobName=knobValue> <target>
```

To view all available knob options for any analysis type, run the command “amplxe-cl -help collect <analysis type>”.

Using Inspector XE

Overview:

Intel’s Inspector XE is used to check for memory and resource issues, such as memory leaks and invalid accesses, as well as thread correctness issues like deadlocks, race conditions, and cross stack references.

Example:

In this example, we will be using Intel’s Inspector XE to perform a Detect Deadlocks and Data Races (tii) analysis on the 3D rendering application called Tachyon. To begin, you must first load the necessary environment variables needed for use with the Inspector tools with the source command, copying the sample application into your working directory, and then compiling the program.

```
-> source /opt/intel/inspector_2017.1.3.510645/inspxe-vars.sh
-> cp /opt/intel/inspector_2017.1.3.510645/samples/en/C++/tachyon_insp_xe.tgz tachyon_insp_xe.tgz
-> tar -xzf tachyon_insp_xe.tgz
-> cd tachyon_insp_xe/
```
Then, to perform a Detect Deadlocks and Data Races (ti2) analysis on the compiled Tachyon application tachyon.find_and_fix_threading_errors.

-> inspxe-cl -collect ti2 -result-dir ~/.results ~/XeonPhi/tachyon_insp_xe/tachyon.find_and_fix_threading_errors

Below is the output of the ti2 analysis.

```
...output...
```

Where the flags “-collect” specifies what analysis to be done and the “-result-dir” specifies where the results should be saved too, followed by the path to the target application. To view a complete list of Inspectors analysis types that are available to be preformed, such as Locate Deadlocks and Data Races (ti3) or Detect Memory Problems (mi2), and the available action options to be used, run the command “inspxe-cl -help collect”.

You can then view the data collected from the directory specified by the “-result-dir” flag under the relative path summary/summary_data.xml. You can also generate a report for the specified result. Below is the Inspector command to generate a status report.

-> inspxe-cl -report status -report-output status-report.txt -result-dir results/

Where the flags “-report” specifies what report to be created, “-report-output” designates the path and name of the report to be saved, and “-result-dir” designates where the results were saved from the initial analysis collection. If the flag “-report-output” is not specified, the output will be displayed to stdout. To view a complete list of Inspectors report types that are available to be created and the available action options to be used, run the command “inspxe-cl -help report”.

**Using Advisor XE**

**Overview:**

Intel’s Advisor XE offers a vectorization optimization tool and threading design that is used to annotate, correct, survey, and scale the premise of your code for optimal parallel execution. With Advisor XE, you can get guidance and see scenarios of parallel deployment to visibly show where one should focus their parallelism design effort at.

**Examples:**
In this example, we will be performing a basic survey analysis on one of the Tachyon sample applications that will explore where to add efficient vectorization and/or threading into code. To begin, you must load the necessary environment variables needed for use with the Advisor XE tools with a source command as well as the Parallel Studio XE environment variables, copying the sample application into your working directory, and then compiling the program. For this example, we will also create a new project directory.

```
-> source /opt/intel/advisor_2017.1.3.510716/advice-vars.sh
-> source /opt/intel/parallel_studio_xe_2017.4.056/bin/psxevars.sh
-> cp /opt/intel/advisor_2017.1.3.510716/samples/en/C++/tachyon_Advisor.tgz tachyon_Advisor.tgz
-> tar -xzf tachyon_Advisor.tgz
-> cd tachyon_Advisor/
-> make
-> mkdir project
```

Below is the syntax to run the basic survey analysis on the Tachyon application `1_tachyon_serial_debug`

```
-> advixe-cl -collect survey -project-dir ./project -- ~/XeonPhi/tachyon_Advisor/1_tachyon_serial_debug
```

Where the flags “-collect” specifies which analysis type to be run and “-project-dir” designates the projects directory we had created, followed by the path to the target application.

Advisor can then be run to collect data with further analysis types. In this example, we will collect more data about floating-point operations and L1 memory traffic, which applies to vectorization advisor and threading advisor. Note that you must use the same project directory designated by the “-project-dir” flag for this analysis as the initial survey analysis.

```
-> advixe-cl -collect tripcounts -flops-and-masks -project-dir ./project -- ~/XeonPhi/tachyon_Advisor/1_tachyon_serial_debug
```

Below is the output of the second analysis test.
To view a complete list of Advisors analysis types that are available to be created and the available action options to be used, run the command “advixe-cl -help collect”.

To generate a survey report that shows the results of the survey analysis

-> advixe-cl -report survey -project-dir ./project/ -format=text -report-output ./project/survey_report.txt

Where in this example, we generated a .txt file to save the report too instead of generating the output to stdout. The new flags “-format=” specifies what format the report should be saved as, and “-report-output” designates the path and file name of the report. Note, you must use the same project directory to generate any reports for your projects analysis. Below is the stdout output of this example report if the “-report-output” flag is not specified.

Finally, Advisor can generate a report to display the call stacks for each function call detected during the initial example collection phase. You can use this data to explore the call sequence flow of the target application and analyze the time spent in each program unit/ For this example, we will be displaying the report in Advisors top-down mode and will be saving the report to our working directory under the specified name top_down.txt.

Note, to generate this survey report, you must be in your projects working directory.

-> cd project
-> advixe-cl -report survey -top-down -display-callstack -report-output ./top_down.txt

Where the flags “-top-down” specifies Advisor to display the report in top-down mode starting from the applications root and “-display-callstack” specifies Advisor to display the call stacks for each function call. To view a complete list of Advisors report types that are available to be created and the available action options to be used, run the command “advixe-cl -help report”.

Systems Group
Using Intels C++ Compiler

Overview:

Intels C++ Compiler compiles C and C++ source files for Intel 64 and IA-32 architectures.

Examples:

To compile C/C++ code, the syntax would be “icc source_file.c” for C source code and “icpc source_file.cpp” to compile C++ source code. To begin, you must load the necessary environment variables and Intel 64 compilers to be used for Parallel Studio XE tools.

```
-> source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
-> source /opt/intel/compilers_and_libraries_2017.4.196/linux/mpi/intel64/bin/mpivars.sh
-> source /opt/intel/parallel_studio_xe_2017.4.056/bin/psxevars.sh
```

In this example, we will be compiling a program for OpenMP and Parallel Processing while creating a native program. Native execution occurs when the application runs entirely on the Intel Xeon Phi processor. Below is the syntax for C source code that would be used to compile such a program.

```
-> icc -O3 -mmic ~/myProgram.c -o ~/myProgram-openmp
```

Where the flags “-O3” designates to optimize for maximum speed and aggressive optimizations, “-mmic” builds an application that runs natively on Intel MIC Architecture, and “openmp” that enables the compiler to generate multi-threaded code based on the OpenMP directives. If no specific optimization flag is designated, the default optimization level for Intel compilers is -O2. Note that a MIC focused application can only be run on Intel 64 architecture-based systems, targeting the Intel MIC architecture.

To view the complete list of command options and actions, run "icc -help" or "icpc -help". You can also specify a category to display specific help options such as "icpc -help openmp" to view the OpenMP and Parallel Processing action options and syntax. Valid categories are displayed at the bottom of the general "icc -help" and "icpc -help" command outputs.