Debugging Intel Xeon Phi KNC Tutorial

Overview: The Intel Xeon Phi Coprocessor 2
  Debug Library Requirements 2

Debugging Host-Side Applications that Use the Intel Offload Directives (LEO) 4

Debugging Native Applications 11
  Intel Xeon Phi Native Debugging 11
    Using a Standard Single Server Launch String 11
    Using the Xeon Phi Native Launch String 12

Debugging Native and Symmetric MPI Parallel Applications 14

Memory Debugging on Xeon Phi 16

Xeon Phi-Specific TotalView Options and State Variables 17
  Xeon Phi-Specific Command Line Option 17
  Xeon Phi-Specific State Variables 17

Known Issues 18
Overview: The Intel Xeon Phi Coprocessor

The Intel Xeon Phi coprocessor, based on Intel's Many Integrated Core (MIC) architecture, is a major advancement in the performance and speed of parallel processing. Xeon Phi is designed for highly-parallel workloads and contains more than 60 individual cores.

With its 8.13 release, TotalView® for HPC provides developers the ability to view, control, and debug codes running on both the host processor and the Intel Xeon Phi coprocessor.

With the introduction of its next generation of Xeon Phi architecture (codenamed Knight's Landing, or KNL), Intel unified the underlying architecture of Xeon and Xeon Phi. KNL is compatible with TotalView's linux-x86-64 platform, thus obviating the need for the special instructions in this tutorial. For Knight's Corner, or KNC, this tutorial is still valid.

NOTE >> The following tutorial is valid for KNC and KNC architecture only. For KNL architecture, please refer to the regular TotalView documentation.

Table 1 lists the major supported modes for both KNC and KNL.

Table 1: KNC and KNL Support in TotalView for HPC

<table>
<thead>
<tr>
<th>Architecture/Model</th>
<th>Applications running natively on Xeon Phi</th>
<th>Host-side applications with Intel offload directives (LEO)</th>
<th>Scalable MPI applications, launched on host and running on card</th>
<th>Symmetric MPI applications, running on both host and card</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>KNL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

TotalView also supports memory debugging with MemoryScape in all programming modes except LEO. TotalView does not yet support reverse debugging with ReplayEngine for the Intel Xeon Phi coprocessor.

This tutorial assumes that users know how to use TotalView to debug their parallel, multiprocessing or MPI application, running in a cluster environment or similar setup. The user is expected to be familiar with many of the advanced topics introduced in the TotalView for HPC User Guide, Part IV: Advanced Tools, Configuration and Customization.

Debug Library Requirements

Before starting a debug session, ensure that your coprocessor card can access the GNU `libpthread.so.0` and `libthread_db.so.1` libraries with full debug information because newer versions provided by the MPSS (Manycore Platform Software Stack) driver do not store the debug information.
You can either copy these libraries to the /lib64/.debug/ directory on each card, or make them accessible from the card. For example, you need the following files:

/opt/mpss/3.4/sysroots/kлом-mpss-linux/lib64/.debug/libpthread.so.0
/opt/mpss/3.4/sysroots/kлом-mpss-linux/lib64/.debug/libthread_db.so.1

If TotalView still can't find them automatically, manually set the gnu_debuglink variable in TotalView, like so:

dset TV::gnu_debuglink_search_path :%D:%D.debug:%G%/%D:/opt/mpss/3.4/sysroots/kлом-mpss-linux/lib64/.debug/

Alternatively, you can provide the path to the libraries when you start TotalView, for instance:

totalview -e "dset TV:gnu_debuglink_search_path :%D:%D.debug:%G%/%D:/opt/mpss/3.4/sysroots/kлом-mpss-linux/lib64/.debug/" <target name>
Debugging Host-Side Applications that Use the Intel Offload Directives (LEO)

The Intel Offload directives are OpenMP-like pragmas added to C/C++ and Fortran code to mark sections of code to offload onto the Intel Xeon Phi Coprocessor(s). TotalView automatically detects these offload runtime events and attaches the debugger to offload processes running on all Xeon Phi Coprocessor cards.

In order to debug on the coprocessor(s), TotalView launches a debug server, using Intel’s `micnativeloadex` tool (installed in your system’s `/opt` directory) on each Xeon Phi coprocessor card as specified by the TotalView Xeon-Phi-specific state variable `micnativeloadex_server_launch_string`.

**NOTE >>** This should work in most cases, but if it fails, you can edit the `TV::micnativeloadex_server_launch_string` variable, as discussed in “Using a Standard Single Server Launch String” on page 11.

1. Set the environment variable `AMPLXE_COI_DEBUG_SUPPORT=TRUE`.
2. Start TotalView on the host system as usual. For example:

   ```
totalview tx_mic_basic
   ```
TotalView loads the program called `tx_mic_basic` but does not yet start it:

3. Set some breakpoints. You can set breakpoints in both host and offload code.

4. For example, if you set a breakpoint at **line 25** in offloaded code, this breakpoint will be hit either after offload occurs or, if the host has no available Xeon Phi coprocessors, when the program running on the host reaches the breakpoint. Breakpoints outside offload programs are hit only by host processes running on the Xeon host.
After you set initial breakpoints, run your program.

5. Press the Go button.

Note that the console reports:

```
Launching TotalView Debugger Server with command:
micnativeloadex /nfs/toolworks/totalview.8.13.0-0/linux-x86-64/bin/
tvdsrvmain_mic -d 1 -a "-callback 172.28.29.11:16381 -set_pw 755f8bfe:7512e5cc -verbosity info -cuda"
```

where \(-d 1\) indicates it’s running on Xeon Phi coprocessor 1(e.g., \texttt{mic0}).

If Xeon Phi coprocessors are available on the system, the debugging process is similar to parallel debugging. After hitting the first offload directive, the Intel runtime launches a special program, \texttt{offload\_main}, on the Xeon Phi and TotalView attaches to it.
If you have two coprocessors on the system, select **Continue** and then **Go**. This dialog again launches when `offload_main` is started on the second coprocessor.

In this case, the debugger attaches to all coprocessors and continues running to the first breakpoint.
You can avoid these dialogs by setting **Preferences > Parallel** to attach automatically:

![Preference settings](image)

You can also change the preference to attach to all processes and stop the group after each offload attach by selecting the **Stop the group** checkbox under “When a job goes parallel or calls exec().”
6. Hit **Go** again. TotalView runs until it hits the first breakpoint, Breakpoint 1 on line 25, in this case.
Since this is a multi-process, multi-threaded application, you can switch between different threads and processes using the usual controls in the Process or Root windows and debug in the same way as any other parallel and or multi-process applications.
Debugging Native Applications

Intel Xeon Phi Native Debugging

You can debug applications running natively on the Intel Xeon Phi coprocessor. As with the offload directives mode, TotalView launches its debug server on the coprocessor that will start the remote application. A review of the chapter “Setting Up Remote Debugging Sessions” in the TotalView for HPC might be useful.

There are two options for the launch:

- **Using a Standard Single Server Launch String**
- **Using the Xeon Phi Native Launch String**

**Using a Standard Single Server Launch String**

This method can be useful if TotalView is installed on a file system that is accessible from both the host and all Xeon Phi coprocessors, such as on /opt or /nfs file systems. In this case, you can use the standard server launch string, controlled by the `TV::server_launch_string` variable, which you can also set on the TotalView Preferences page’s Launch Strings tab. Assuming that your program executable is also visible to both the coprocessor and the host, you can start debugging by just running

```
totalview -r XeonPhi-hostname mic_native_hello
```

where `XeonPhi-hostname` is the Xeon Phi coprocessor and `mic_native_hello` is the program to debug.

TotalView starts its debug server and displays the following message in the console:

```
Launching TotalView Debugger Server with command:
ssh host-mic0 -n "/opt/toolworks/totalview.8.13.0-0/linux-x86-64/bin/tvdsvr -working_directory <Current_dir> -callback 172.28.29.11:16381 -set_pw 755f8bfe:7512e5cc -verbosity info -cuda"
```

Note that

- The default remote launch command is `ssh`, so you may need to change this to your system’s launch command either in the Single Launch Strings UI preferences or by setting `env TVDSVRLAUNCHCOMMAND=<your_command>`.

- The `tvdsvr` location(`/opt/toolworks/totalview.8.13.0-0/linux-x86-64/bin/` in this case) should be accessible from the coprocessor.

- The `<Current_dir>` path to your executable should be accessible as well. If it is not accessible, you need to change it or define it in your environment.

If your file system is different from that described, you can copy your executable and `tvdsvrmain_mic` to the location `/tmp/` on both the host and coprocessor and modify the launch string from the default

```
%C %R -n "%B/tvdsvr%K -working_directory %D -callback %L -set_pw %P %F"
```
Using the Xeon Phi Native Launch String

Note: This is the recommended method if the TotalView installation is not accessible from the Xeon Phi coprocessor because it affects only the TotalView Xeon Phi-specific settings and avoids having to explicitly change general environment settings and preferences.

This method is useful in two primary use cases:

- To perform debugging on both the host processor and Xeon Phi coprocessor while maintaining separate server launch strings
- To run TotalView in an environment where TotalView servers are not installed on an accessible, shared file system.

Start TotalView and your remote debugging session using this launch string, for example:

```
totalview -mmic -r minnie-mic0 mic_native_hello
```

where `minnie-mic0` is the XeonPhi-hostname of the Xeon Phi coprocessor and `mic_native_hello` is the program to debug.

The command line option `-mmic` sets `TV::mic_native_launch` to `true` and selects the `TV::mic_native_server_launch_string` string to launch the `tvdsvr`. 

```bash
ssh %R -n " /tmp/tvdsvrmain_mic -working_directory /tmp/ -callback %L -set_pw %P %F"
```
Note that even though you started TotalView on the host, the debugged process is running on the coprocessor side (in this example minnie-mic0, viewable above). Debug the executable in the same way you would debug an ordinary program on a remote host (CPU).

If the Xeon Phi coprocessor cannot access the same file system as the host processor, you will need to customize the \texttt{TV::mic\_native\_launch\_string} in the global or your personal .\texttt{tvdrc} file and start TotalView with the \texttt{-mmic} flag for debugging on a Xeon Phi coprocessor. For example:

\begin{verbatim}
dset TV::mic\_native\_server\_launch\_string{ //1
  ssh -n %R "/bin/rm -f /tmp/tvdsvrmain%K"; //2
  scp %B/tvdsvrmain%K %R:/tmp/tvdsvrmain_mic; //3
  ssh -n %R -n "/tmp/tvdsvrmain%K -callback %L -set_pw %P
         -verbosity %V \%F"
}
\end{verbatim}

- //1 Removes your previous tvdsvrmain_mic.
- //2 Copies it from the installation directory to the /tmp/ directory on the coprocessor
//3 Starts the server on the Xeon Phi coprocessor.

Now you can launch TotalView and start your remote debugging session using this launch string:

totalview -mmic -r XeonPhi-hostname mic_native_hello

Debugging Native and Symmetric MPI Parallel Applications

Debugging MPI parallel applications on Xeon Phi coprocessors is no different than debugging parallel applications on a CPU-based cluster environment. You can also use two mechanisms to launch debug servers on the coprocessor, as described in “Intel Xeon Phi Native Debugging” on page 11.

However, you must use TotalView's classic launch (that is, totalview -args mpiexec). By default, Intel MPI is installed on an accessible shared file system and you don't need to modify it. If it is installed separately for host and cards, and the standard MPI libraries were manually copied to the Xeon Phi coprocessors, you need to copy the libmpi_dbg.so and possibly the libmpi_dbg_mt.so libraries as well.

To start debugging with Intel MPI, first verify that you can run your program without TotalView:

mpiexec -np 40 -host XeonPhi-hostname -wdir /tmp/ ./tx_basic_mpi

Then just add totalview -args before mpiexec, like this:

totalview -args mpiexec -np 40 -host XeonPhi-hostname -wdir /tmp/ \
./tx_basic_mpi

or, to use the mic_native_server_launch_string, like so:

totalview -mmic -args mpiexec -np 40 -host XeonPhi-hostname \
-wdir /tmp/ ./tx_basic_mpi
You can also debug a symmetric multi-host, multi-card MPI job in the same way:

```
totalview -args mpiexec -np 5 -host host1 -wdir /tmp/ \
./tx_basic_mpi : -np 5 -host host1-mic1 -wdir /tmp/ ./tx_basic_mpi.MIC
```

**NOTE**

If you are going to debug a symmetric MPI application, you need to have two separate programs (images), one for the hosts and one for the Xeon Phi coprocessors. In this case you are going to debug an MPMD parallel job. Also in this case, your breakpoints and all TotalView group controls (Go, Stop, etc.) will be shared only within a given image, so you will need to set breakpoints and use TotalView group controls for each image individually.

In the case of pure native mode:

```
totalview -mmic -args mpiexec -np 250 -hosts host1-mic0, \
host1-mic1,host2-mic0,host2-mic1 ./tx_basic_mpi
```
If you are debugging a multi-host MPI application on Xeon Phi coprocessors, you need to satisfy the following conditions:

- Each Xeon Phi coprocessor must have its own IP address and be accessible from the front host node, running TotalView.
- TotalView must be installed in a global area and be accessible from each coprocessor in allocation, so that you can start `tvdsvr_mic` on each coprocessor from the partition, or you can copy `tvdsvr` using the `mic_native_server_launch_string`.

Note the Root Window when running on two Xeon Phi Coprocessors:

![Root Window](image)

### Memory Debugging on Xeon Phi

Memory debugging on a Xeon Phi coprocessor is no different than memory debugging on the usual linux-x86-64 platform, except for the following:

- Instead of the `libtvheap_64.so` library, it uses `libtvheap_mic.so`.
- If TotalView is not installed on an accessible shared file system and you need to copy the debug server, `tvdsvrmain_mic`, and you also need to copy `libtvheap_mic.so` to `/lib64` or some other location on each card.
- If you want to pre-link your executable with the HIA library, you need to use `libtvheap_mic.so`.

**NOTE >>** Enabling memory debugging on an Intel MPI job (native or symmetric) enables memory debugging of all ranked processes on hosts and cards, but not on `mpiexec` itself.
Xeon Phi-Specific TotalView Options and State Variables

Xeon Phi-Specific Command Line Option

A new option has been added to the `totalview` command to support Xeon Phi.

`-mmic`

- Sets the remote system to Xeon Phi
- Uses `mic_native_server_launch_string` instead of the single launch string

The option `-mmic` sets `TV::mic_native_launch` to `true` and selects the `TV::mic_native_server_launch_string` string to launch the `tvdsvr`.

Warning: This option takes precedence over the `TV::micnativeloadex_server_launch_string`, so if you use offload and this option, then the `mic_native_server_launch_string` will be used instead of `micnativeloadex_server_launch`, and TotalView may not start.

Xeon Phi-Specific State Variables

These new state variables support the use of Xeon Phi on TotalView.

- `TV::mic_native_launch` *(true|false)*
  
  Default: `false`
  
  When set to `true`, this is the same as using the `-mmic` option when starting TotalView.

- `TV::mic_native_server_launch_string`

  Default: `{%C %R "-%B/tvdsvr%K -working_directory %D -callback %L -set_pw %P -verbosity %V %F"}

  If `TV::mic_native_launch` is set to `true` or the `-mmic` command option is used, this variable defines the remote debug servers where TotalView will launch.

- `TV::micnativeloadex_server_launch_string`

  Default: `(micnativeloadex %B/tvdsvrmic -d %d -a "-callback %L -set_pw %P -verbosity %V %F")`

  Used for a Xeon Phi LEO (offload) launch, this launches the debug server using Intel's `micnativeloadex` tool, on the Xeon Phi coprocessor specified by `-d %d`, where `%d` refers to the coprocessor index number +1.

See the chapter “The tvdsvr Command and Its Options” in the *TotalView for HPC Reference Guide* for more information.
Known Issues

- ZMM registers are shown as YMM and are not supported.