ACRC How-To: Using Intel VTune Amplifier on BlueCrystal Phase 3

Table of Contents

Introduction

This document is *not* intended to be a comprehensive introduction to using performance analysis software, or a tutorial on using Intel's Vtune Amplifier. Rather it is a practical guide to getting started with the VTune profiler on BlueCrystal Phase 3.

Profiling applications such as VTune are key tools for any researcher who is serious about not waiting over long for their results. They help us find the poorly performing parts of our applications and so help us to identify bottlenecks in our work, which we can then fix.

A comprehensive collection of tutorials and other documentation about VTune, is at:

- http://software.intel.com/en-us/intel-vtune-amplifier-xe

In order to access VTune, you will need to load the following module (preferably in your .bashrc):

```
module add intel-cluster-studio/vtune/vtune-2013
```

Preparing your application

You should compile your code with all your usual optimisations for speed (-O3 etc.), but you should also instrument your executable with extra symbolic information using (typically) -g. This will allow you to see the source code (rather than the assembly code) when viewing the line-by-line profile.

Performing a Hotspot Analysis in Batch Mode

Much of the tutorial material for VTune focusses on the tool's graphical front-end. However, this implies profiling your code in interactive mode, which is in tension with the use of a scheduler to submit jobs to the compute nodes of BlueCrystal. Happily, however, we can easily use the command line interface to VTune. I will show you how to use VTune in batch mode and then how to subsequently inspect the results of the profile (using theVTune GUI).

VTune can perform many different analyses of your code. In this document, I will demonstrate the use of the most popular—a hotspot analysis. This will tell you which parts of your program took the longest time to run.

I will start by showing you how to profile a single-threaded executable, but will also walk through how you can profile multi-threaded and MPI programs.

Once I had built my executable, I was able to submit my profiling run to the queuing system with only a few small additions to my submission script, which are shown below.

I added **module add intel-cluster-studio/vtune/vtune-2013** , to ensure that I had access to VTune on the compute node, and then prefixed my usual launch command with **amplxe-cl -quiet -collect hotspots -result-dir r001hs**, where:

- **amplxe-cl** launches the command line interface to VTune,
- **-quiet** suppresses extraneous output,
- **-collect hotspots** specifies the kind of analysis to perform, and
- **-result-dir r001hs** indicates that the profile results will be saved in a sub-directory of the current directory called r001hs. **Note that this dir will not be overwritten by a subsequent call to VTune**, so you will need to either delete the directory are provide another, e.g. r002hs, r003hs, if you are performing a sequence of profile runs.

```
#!/bin/bash

#PBS -N D2Q9
#PBS -o OUT
#PBS -l nodes=1:ppn=1,walltime=00:01:00
#PBS -m abe

#! application name
application="d2q9-bgk.exe"
#! Run options for the application
options="input_300x200.params obstacles_300x200_circ.dat"
# load the VTune module
module add intel-cluster-studio/vtune/vtune-2013

#! change the working directory (default is home directory)
cd $PBS_O_WORKDIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo PBS job ID is $PBS_JOBID
echo This jobs runs on the following machines:
echo `cat $PBS_NODEFILE | uniq`

#! Profile the executable
amplxe-cl -quiet -collect hotspots -result-dir r001hs $application $options
```

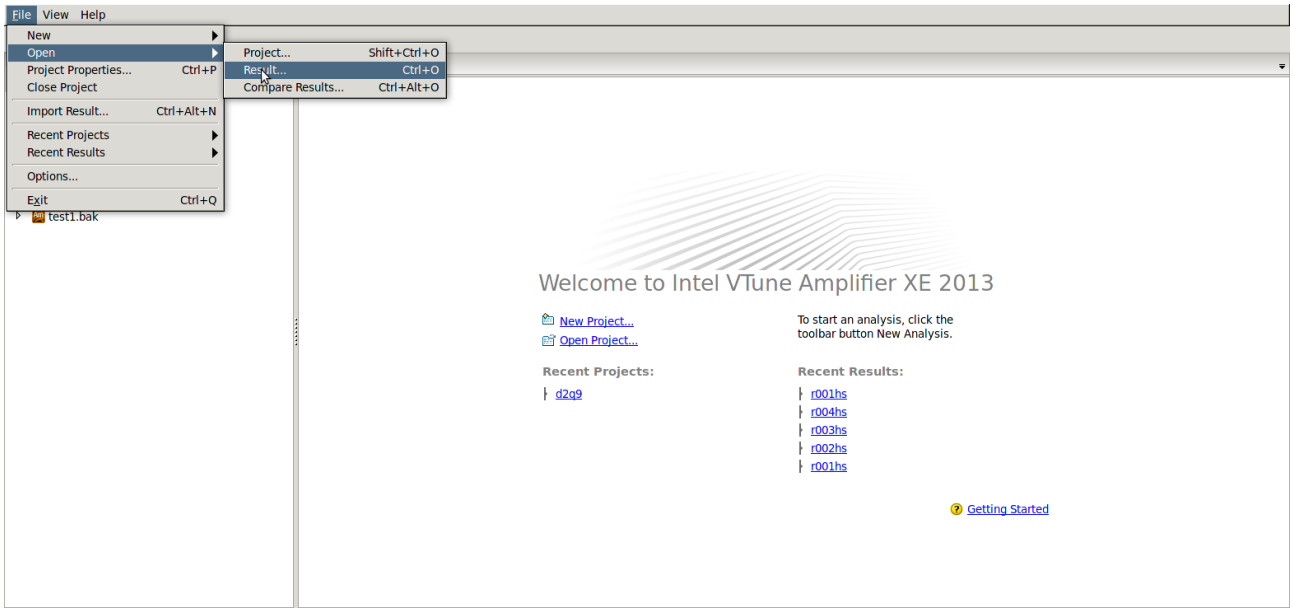
My submission script looked like:
After the job has run, you have the choice of inspecting the result using either the command line interface again, or the GUI. A simple, top-level summary report can be produced by typing:

```
amplxe-cl -report hotspots -r r001hs
```

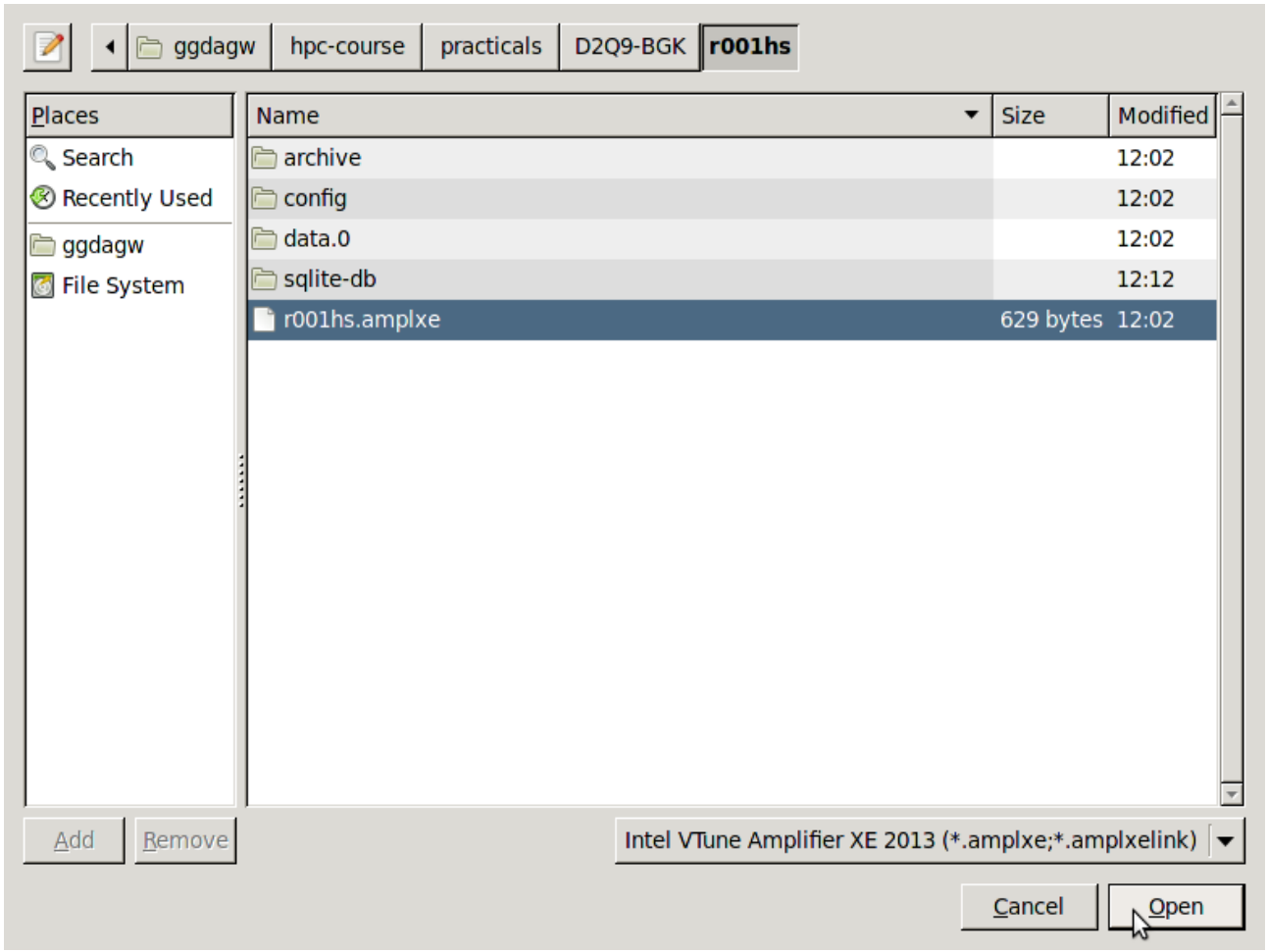
To launch the GUI, type (remember to enable X11 forwarding when connecting to the cluster):

```
amplxe-gui
```

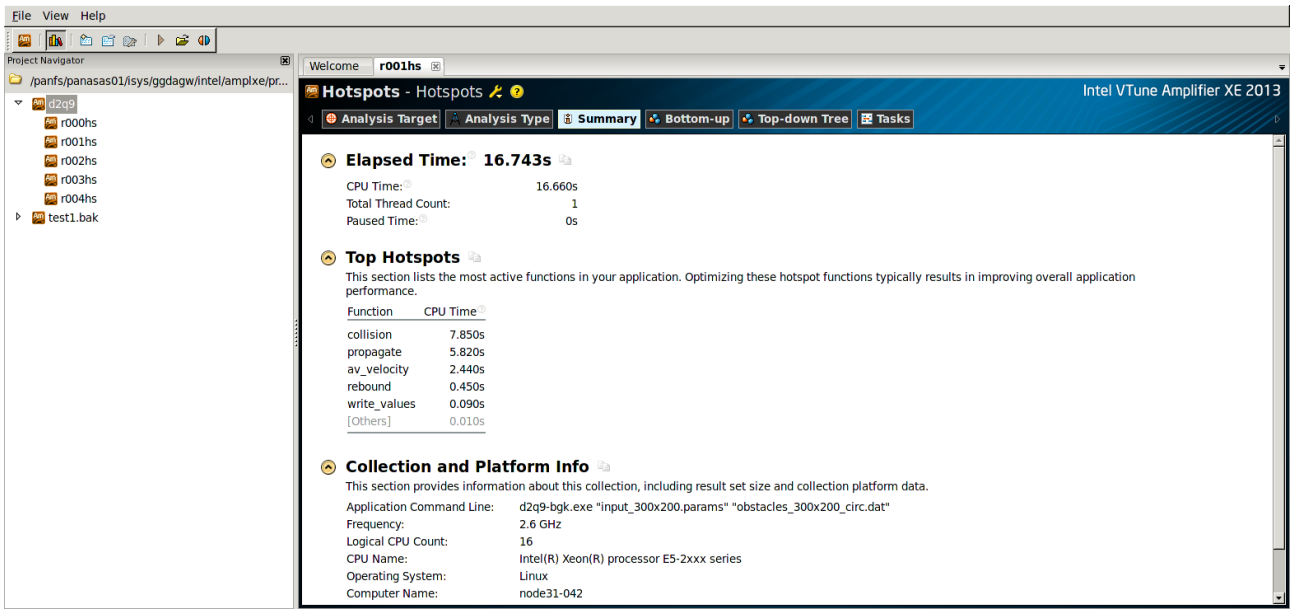
and, after the splash screen, you will see the welcome screen:



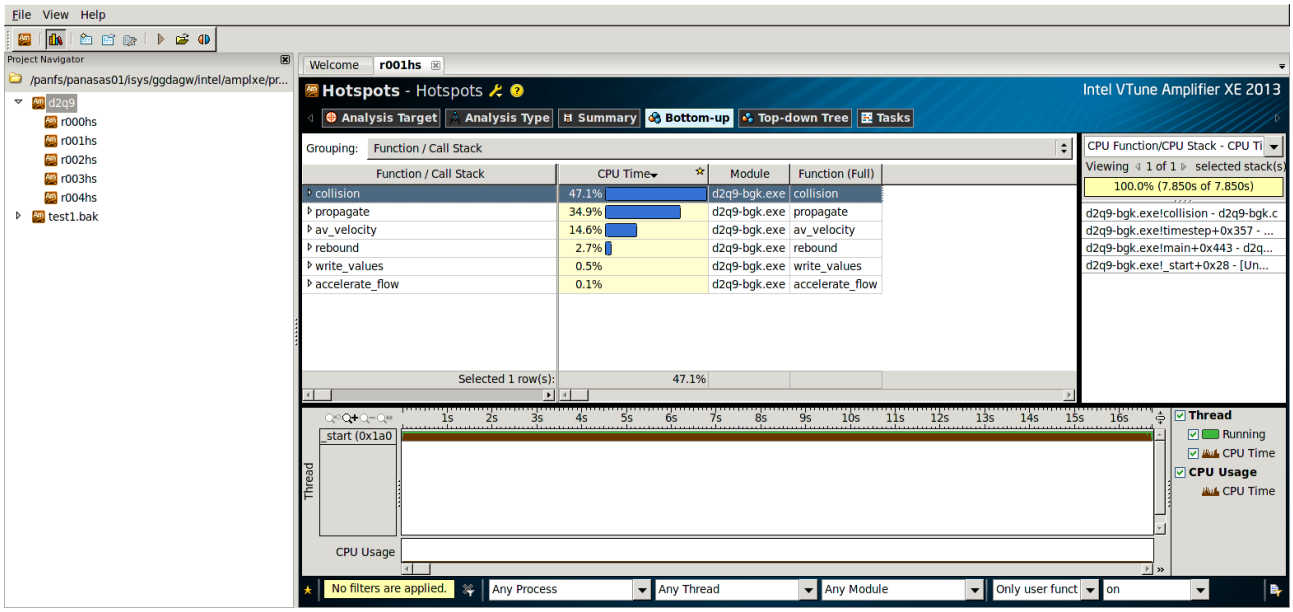
By choosing **File | Open | Result**, you can proceed to selecting the profile results that you would like to view:



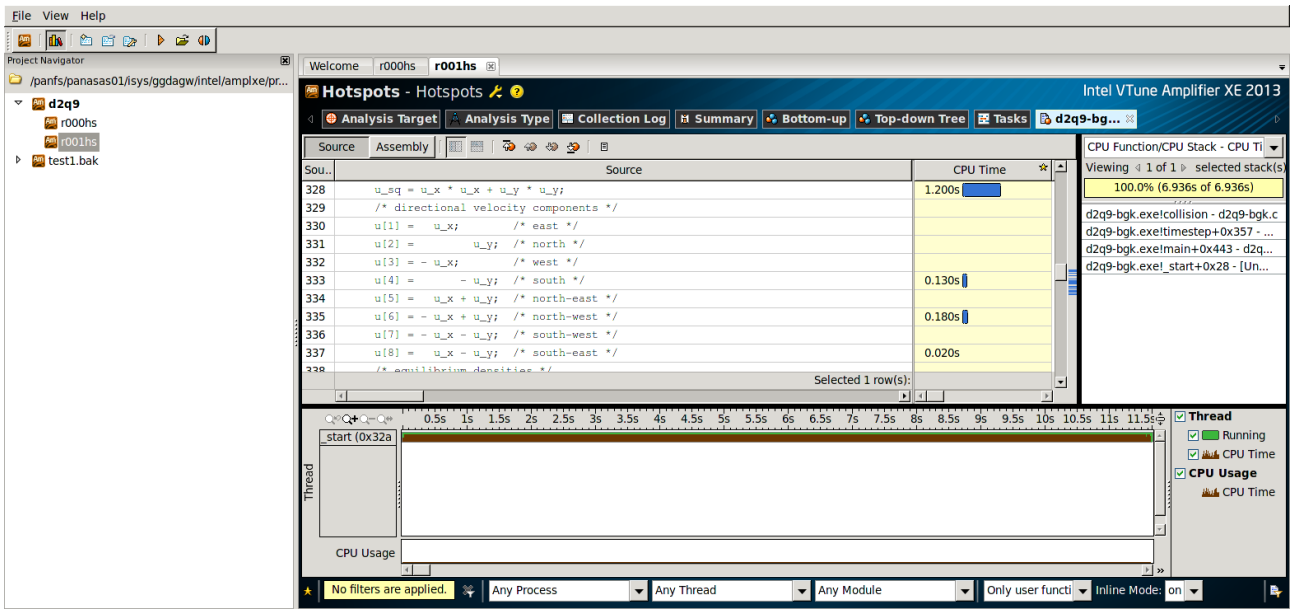
and from there to viewing the top-level results of the hotspot analysis:



The **Bottom-up** tab shows you a breakdown by function:

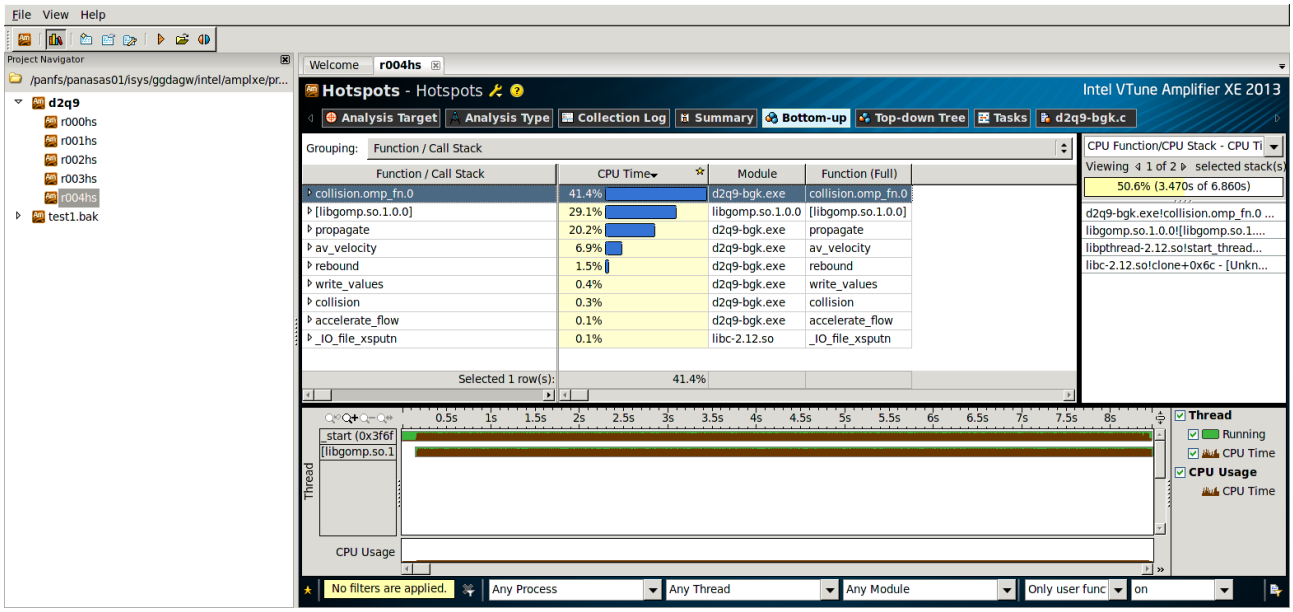


and double clicking on the blue bar will take you down to a line-by-line profile:



Profiling Multi-threaded Code

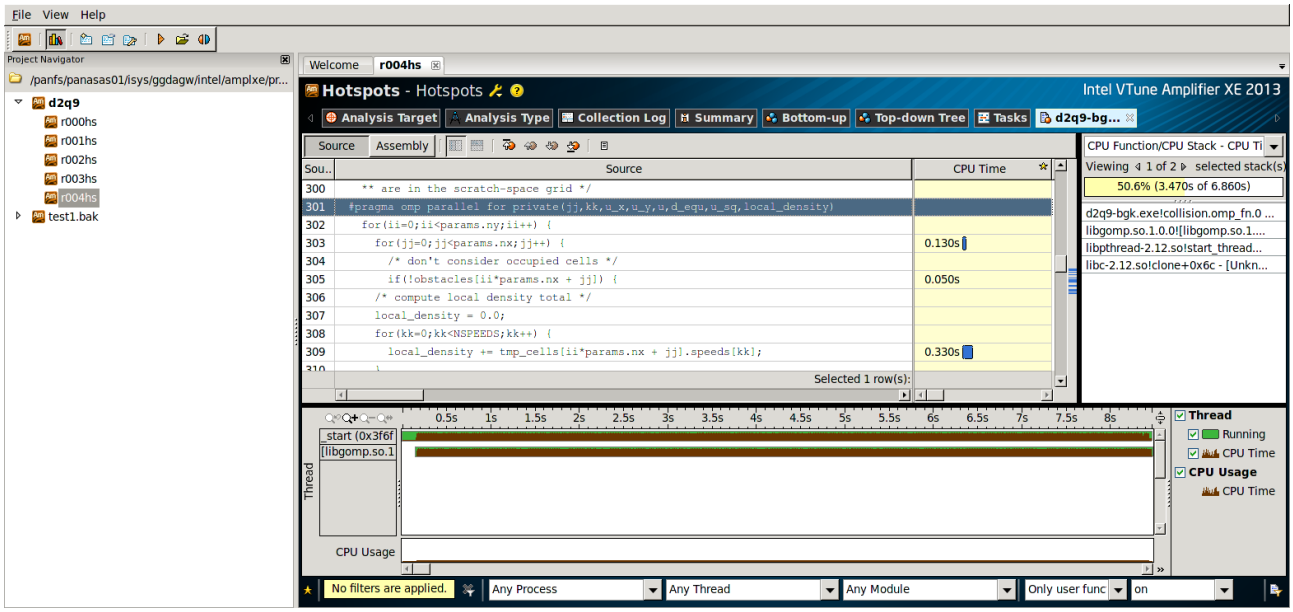
Profiling your multi-threaded code with VTune is very similar to profiling serial code. Compile your code with the appropriate flags (I'm using OpenMP example code below, so I passed -fopenmp to gcc), and run as per the batch job submission example above. Once the job is complete, you can again view the results through the VTune GUI, e.g.:



(If you are using OpenMP and would like to control the number of threads your program uses, adding **export OMP_NUM_THREADS=<num-desired-threads>**, to your submission script is useful.)

The performance, aggregate or per-thread, is seen in the top pane. The lower pane provides a per-thread trace of CPU usage. You can view the profile information for each individual thread in the top pane by clicking on that thread in the lower pane.

You can drill down and view the time spent in various OpenMP work-sharing constructs:



In addition to finding hotspots, VTune to can perform many other useful analyses of your threaded application. You may like to explore results from the **locksandwaits** and **concurrency** analyses.

Profiling MPI Code

You can also use VTune to profile MPI applications. In order to do this, you must modify your submission script so that you are running your program through VTune, which is, in turn, launched via mpirun or mpiexec. The launch line(s) in your submission script thus form a chain similar to, e.g.:

```
nnodes=`wc $PBS_NODEFILE | awk '{ print $1 }'`
mpiexec -np $nnodes amplxe-cl -quiet -collect hotspots -result-dir r001hs ./my-mpi-prog.exe
```

where, mpirun/mpiexec and amplxe-cl take their usual options.

When the job is complete, VTune will create separate results folders for each rank of the MPI job. For example, the sub-directories created by the above launch line will be:

```
r001hs.<rank>
```

where, <rank>, will be filled out accordingly. You can then use the VTune GUI to inspect the profile for any chosen rank.

For those running hybrid MPI/threaded programs, it will be apparent from the examples seen so far that profile information for all the threads associated with a given rank can be inspected.

If your MPI cohort is large and you would like to only collect statistics for a single process, then you can use the host specific syntax of various process managers so that only a subset of your processes are launched through VTune. (Look for more information on, e.g. the '--host' option for OpenMPI and the use of 'argument sets' for the HYDRA process manager, used by e.g. the MVAPICH2 and Intel MPI libraries.)

Summary

I have deliberately not attempted to document the full feature set of Intel's VTune profiler. Rather, I have focused on providing a quick-start guide to help you start using VTune on BlueCrystal Phase 3. VTune is an excellent tool and I hope I have given a flavour of its versatility. Like any new tool, starting out with VTune has an associated learning curve, but I hope the examples above help you along that path. Profiling your application (and re-profiling, as you develop it) is key to identifying inefficient portions of code and hence accelerating your research.