Lab 8 of SP19-BL-ENGR-E599-30563

Lab 8 will instruct you to use the Intel VTune amplifier to profile your codes

Goal

- Launch and use VTune GUI (amplxe-gui)
- Use VTune GUI to profile simple c++ codes
- Learn the basic knowledge of performance metrics

Deliverables

Launch the advanced-hotspot profiling on the `vectest-vtune` binary with the following command line arguments

```
vectest-vtune 20000 20000 200 48
```

(20000×20000 matrix with 200 iterations and 48 threads)

Submit the screenshot of the summary tab of hotspot and memory consumption results.

Evaluation

Lab participation: credit for 1 point based upon a successful completion of the lab tasks

Installation of VTune

X server

A remote access to the VTune GUI tool requires a running X server when ssh connecting the remote node. For Linux desktop users, X server shall be already installed within the OS system.

For MacOS

Install the Xquartz

For Windows

Install the Xming

Set up VTune from Juliet Node

At Client Side

Enabling the X forwarding when launching the SSH Service.

For Linux and MacOS users, adding the `-X` option to your `ssh` command.

```
$ ssh -X user@juliet.futuresystems.org
```

For Windows users, start the Xming before using the SSH tool such as PuTTY.

At Server Side

Remember to add `-X` when logging into to your assigned juliet node.

```
login-1$ ssh -X j-xxx
```

VTune is located at the path `/opt/intel/vtune_amplifier_2019.0.2.570779/bin64/amplxe-gui`, which is accessible to all the juliet nodes. Add this path to your $PATH variable of `~/.bashrc` by sourcing the script

```
source /opt/intel/vtune_amplifier_2019/amplxe-vars.sh 1>>/dev/null
```

Copy the example codes

We have a C/C++ code available from previous lab session of vectorization. Here, we will use VTune to profile vectorized codes. The sample codes are located at the path `/share/jproject/lc37/vtune-tutorial`

```
cp -r /share/jproject/lc37/vtune-tutorial ~/
```

Compile the source codes by

```
cd ~/vtune-tutorial
make
```

It will generate the target binary file `vectest-vtune`

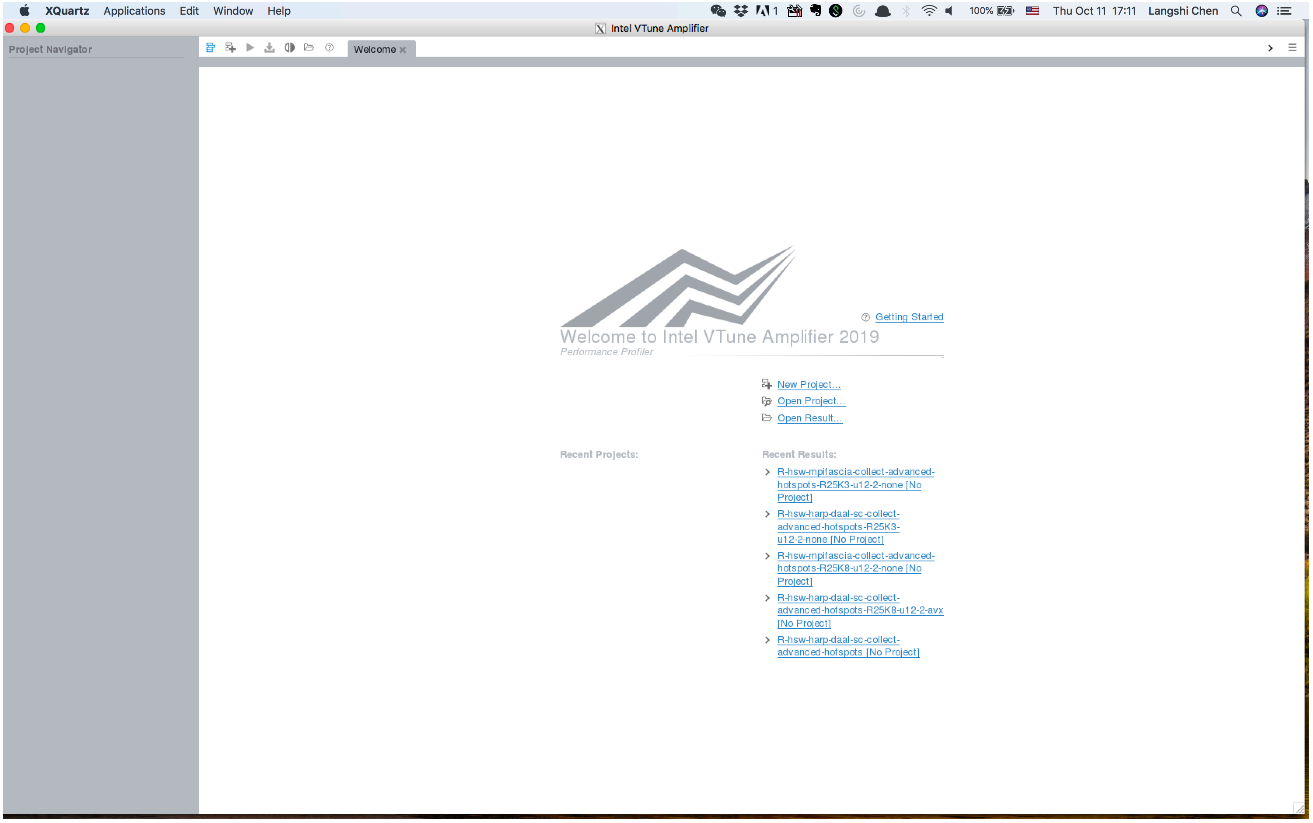
Launch VTune GUI

Note that only three copies of VTune can be run simultaneously due to the license. So, please check the schedule below (each students have two different time slots) and run it that time.

Date	Time	Student #1	Student #2	Student #3
Monday	8:00-9:00 pm	Rishab Nagaraj	Vivek Vikram Magadi	Nishant Jain
Monday	9:00-10:00 pm	Xinquan Wu	Jiayu Li	Arpit Bansal
Monday	10:00-11:00 pm	Sumeet Mishra	Srinithish K	Sahaj Singh Maini
Monday	11:00-12:00 am	Rohit Bapat	Amit Makashir	Ishneet Singh
Tuesday	11:00-12:00 pm	Saniya Ambavanekar	Vatsal Jatakia	Abishek Babuji
Tuesday	1:00-2:00 pm	Jainendra Kumar	Gattu Ramanadhan	Surya Prateek Soni
Tuesday	2:00-3:00 pm	Arpit Rajendra Shah	Shilpa Singh	Karen Sanchez
Tuesday	3:00-4:00 pm	Rishab Nagaraj	Vivek Vikram Magadi	Nishant Jain
Tuesday	4:00-5:00 pm	Xinquan Wu	Jiayu Li	Arpit Bansal
Tuesday	5:00-6:00 pm	Sumeet Mishra	Srinithish K	Sahaj Singh Maini
Tuesday	7:00-8:00 am	Rohit Bapat	Amit Makashir	Ishneet Singh
Tuesday	9:00-10:00 pm	Saniya Ambavanekar	Vatsal Jatakia	Abishek Babuji
Tuesday	10:00-11:00 pm	Jainendra Kumar	Gattu Ramanadhan	Surya Prateek Soni
Tuesday	11:00-12:00 am	Arpit Rajendra Shah	Shilpa Singh	Karen Sanchez

To start VTune GUI.

```
your_node$ amplxe-gui
```

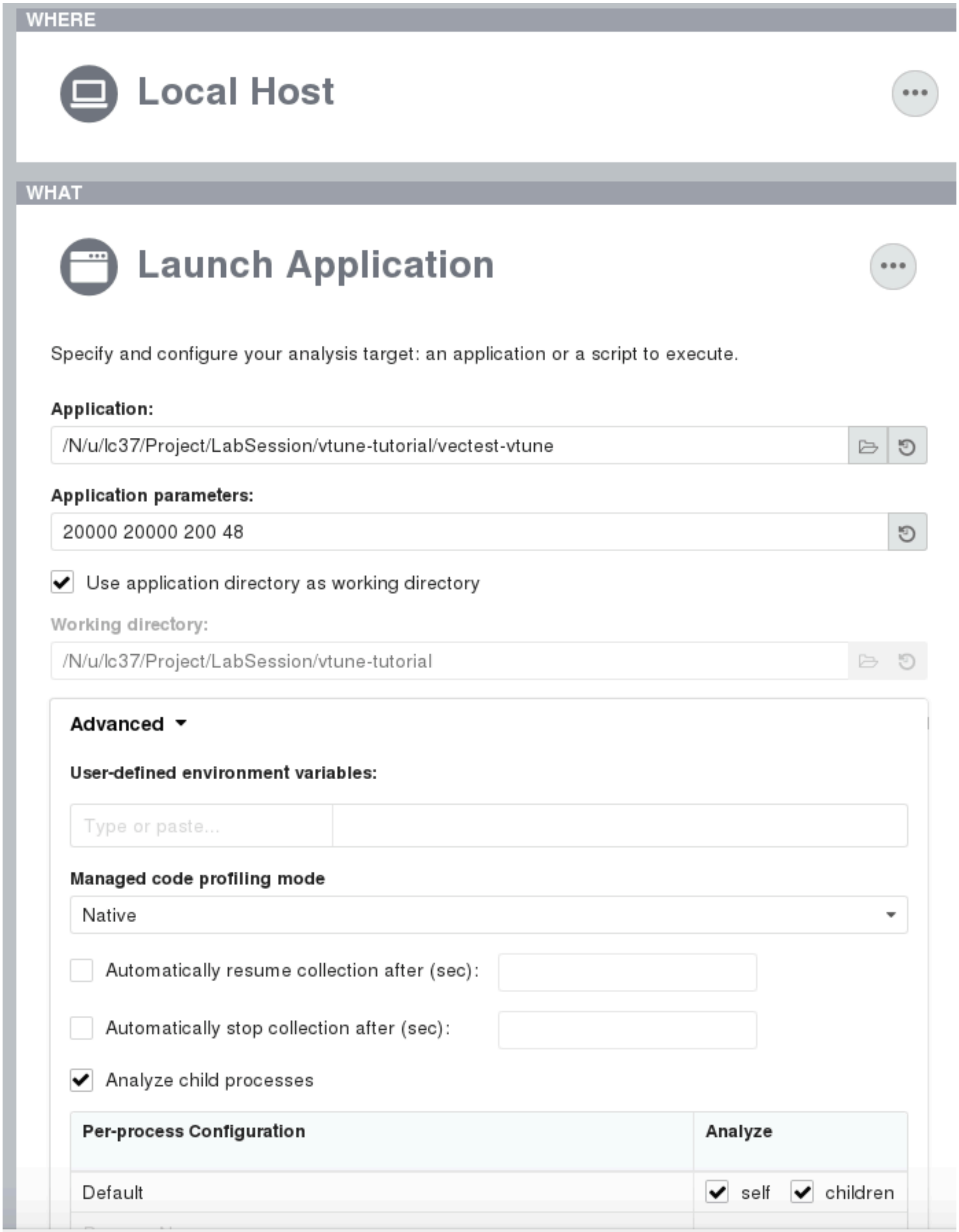


Screenshot of VTune

To set up a VTune project, simply click the `New Project` button at the Welcome panel. Type a `Project name` and set the `Location` as `/N/u/username/vtune-tutorial`. For instance, `/N/u/sakkas/vtune-tutorial`

Configure launch panel

The launch panel requires you to identify the machine where your codes will be running and the target binary file to profile. Click on `Configure Analysis...` and set application parameters:



Configure the launch panel

There are other parameters to configure. We will just set application parameters in this lab session.

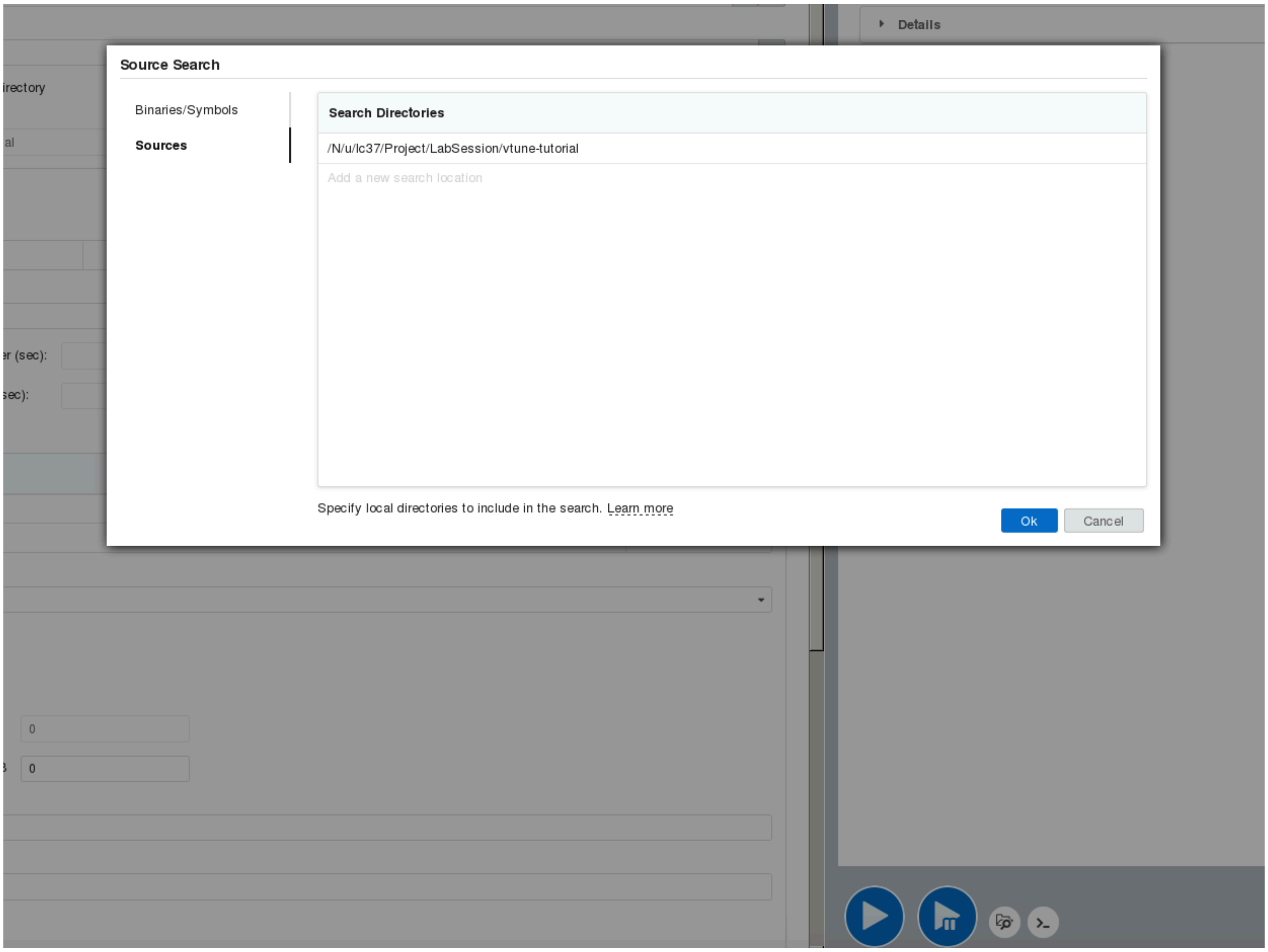
- Application Parameters: The command line arguments to your binary

- Managed code profiling mode
    - Native: for c/c++, Fortran codes
    - Managed: for languages such as Java and Python.
- Analyze child processes
- Duration time estimation
- Limit collected data

In order to trace the source codes of your program, you must set up the directories to search the binary symbols and the source files. Click the button you see in the red square.

open the set up page for search path

Set search directory `/N/u/username/vtune-tutorial`.

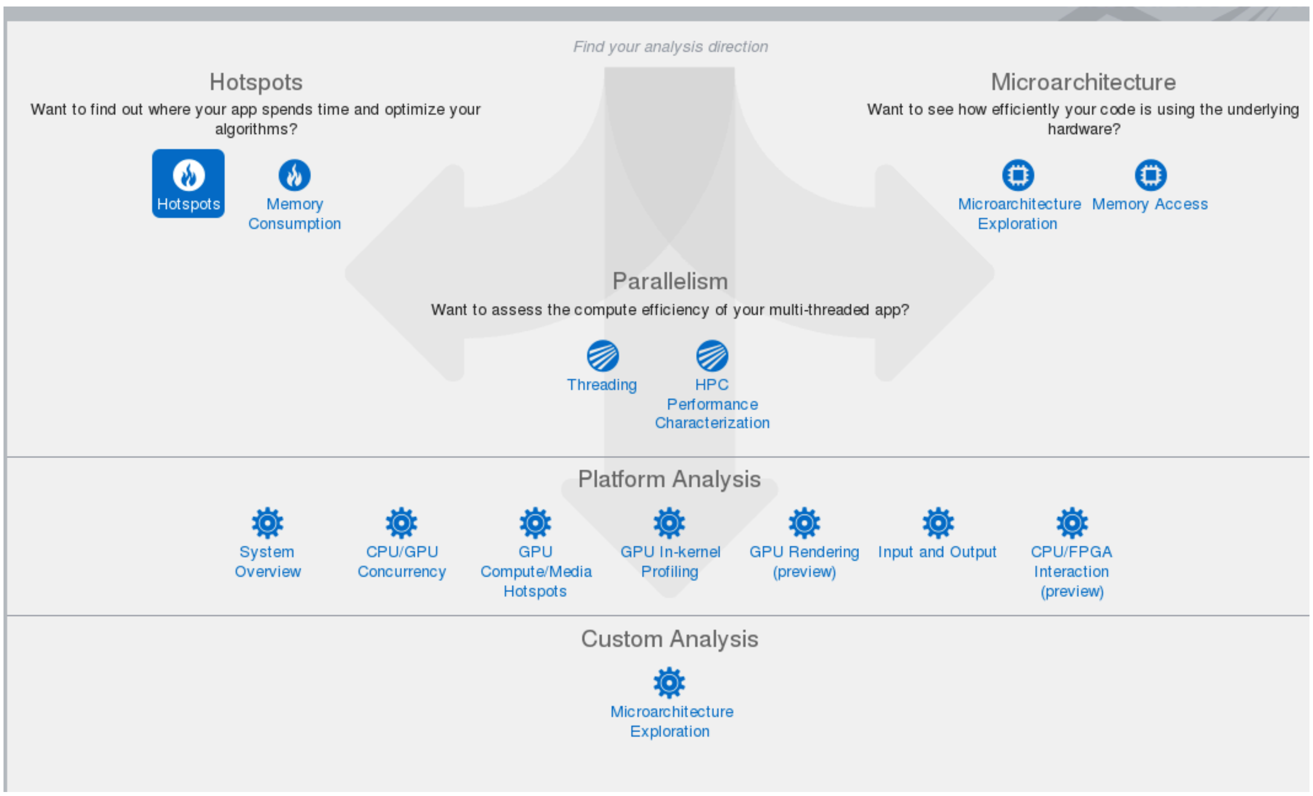


Set up the search path for source files

Also, you should add the `-g` option when you compiling your source codes. This part is already included in `MakeFIle`.

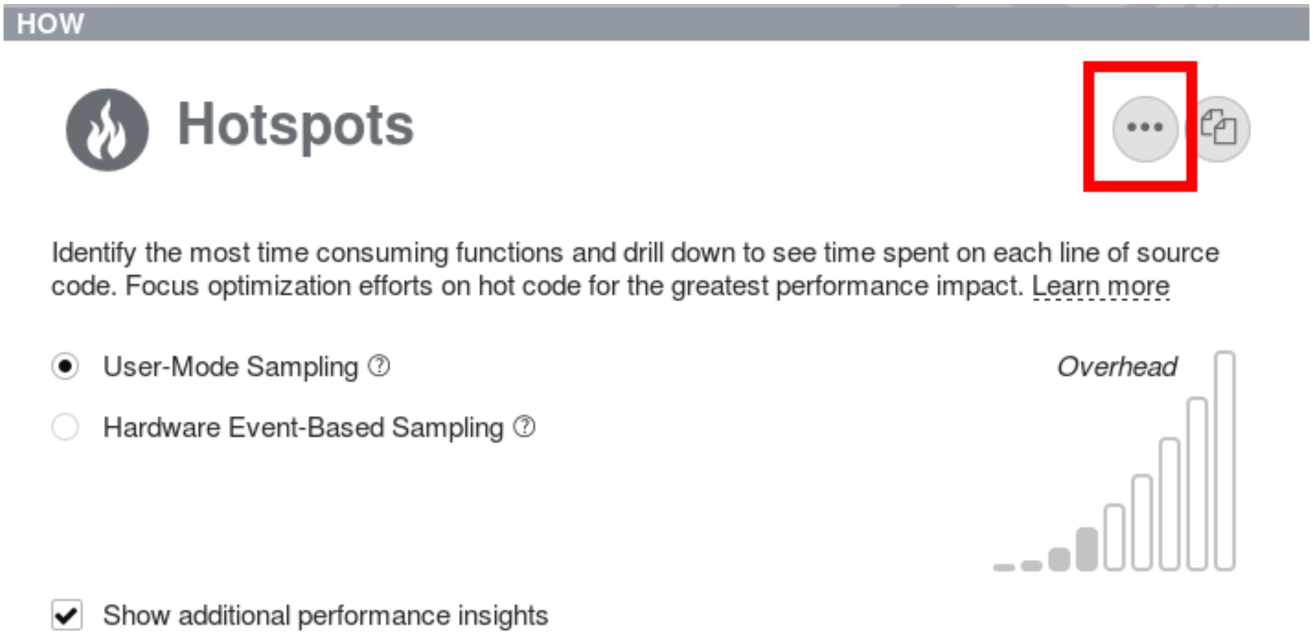
```
## compile
CXXFLAGS=-g -O3
CC=g++

all:
	${CC} ${CXXFLAGS} -fopenmp -o vectest-vtune main.cpp
```

The launch panel includes a variety of profiling type:



The profiling types in VTune

To access the profiling types, click the three dotted button on the top right. (See the image below)

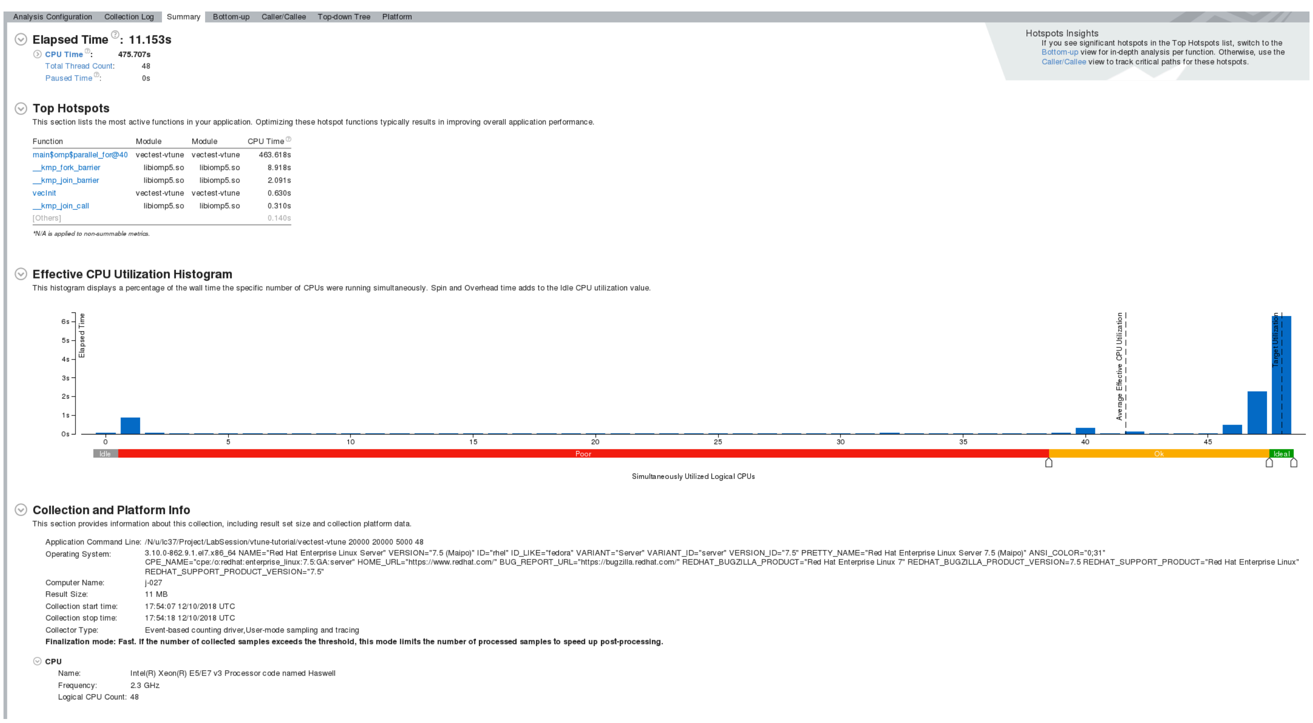


Accessing the different profile types

Hotspot Analysis

Hotspot analysis allows you to identify the bottleneck of your codes, i.e., which function takes most of the running time. It also reports the average CPU usage.

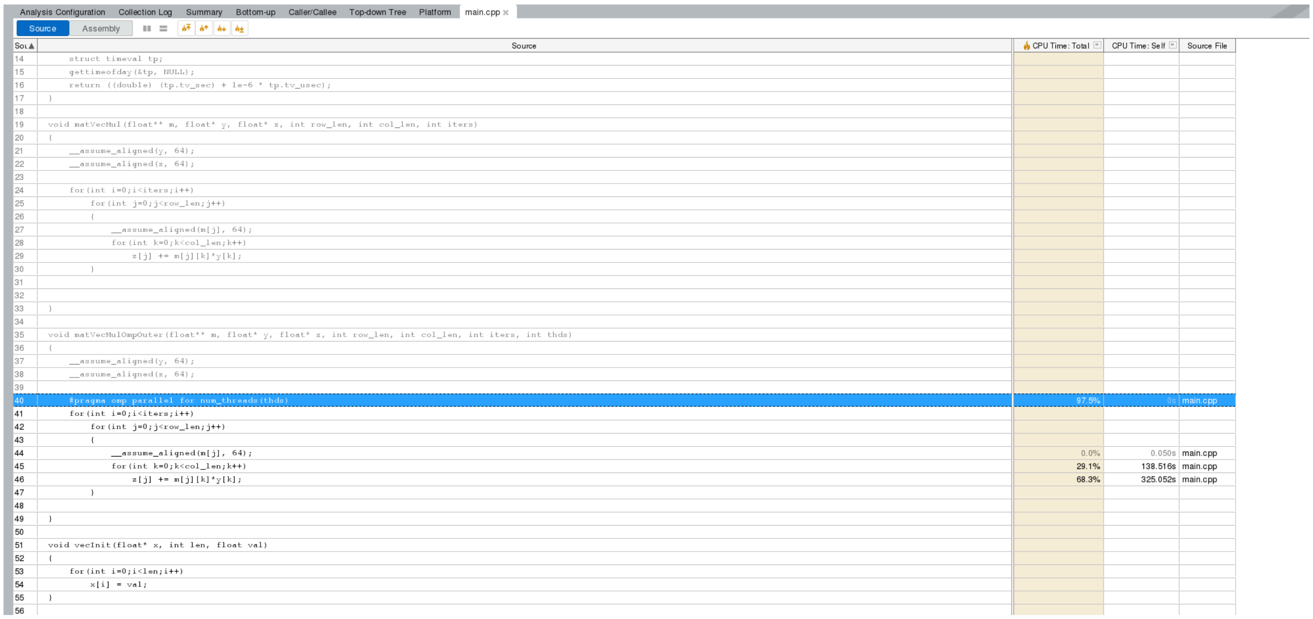
The Summary Report



The Summary tab for hotspot analysis

- It lists out the top hotspots (functions)
- Effective CPU utilization histogram about the distribution of concurrent threads running time
- Other platform information

Click the function name in the top hotspot link, it will open the source code panel and highlight the lines of the hotspot function.



The highlight of the hotspot function

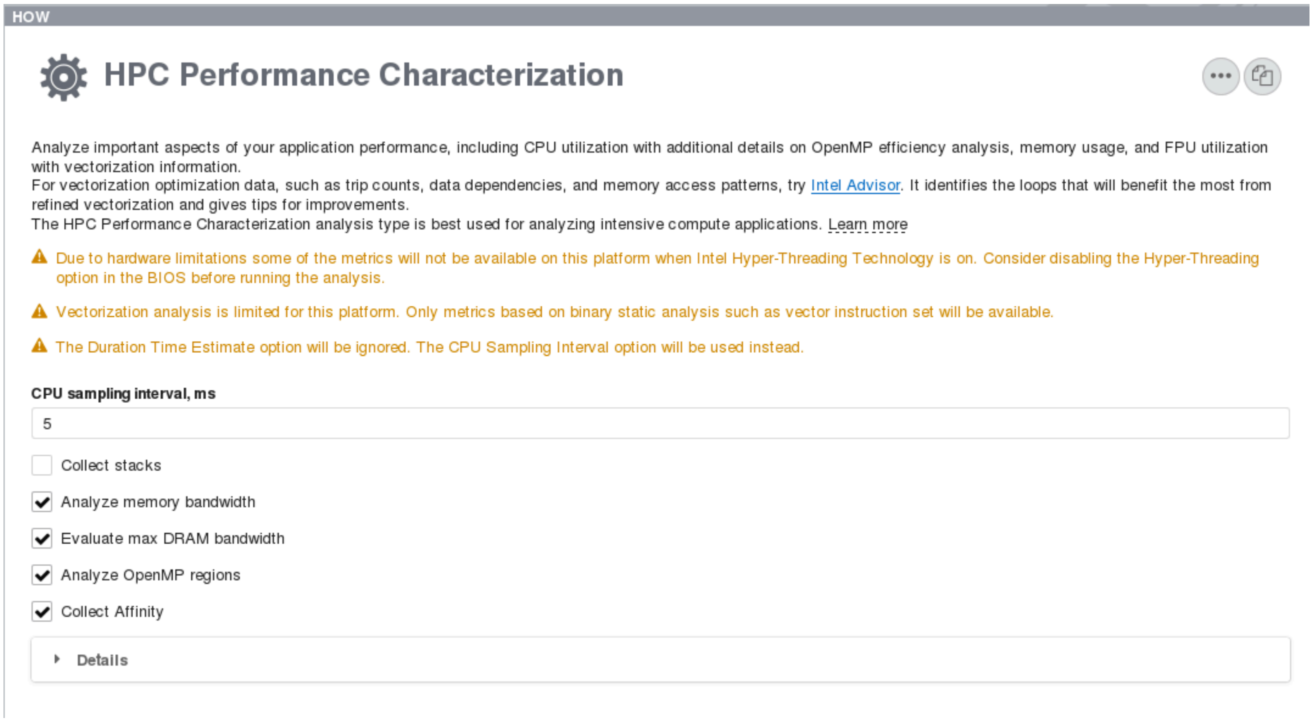
We know that it is the paralleled for loop in computing matrix vector multiplication that consumes 97% of the execution time.

The bottom-up View

The bottom-up view allows you to trace the caller/callee functions and trace the multi-thread concurrency in the timeline

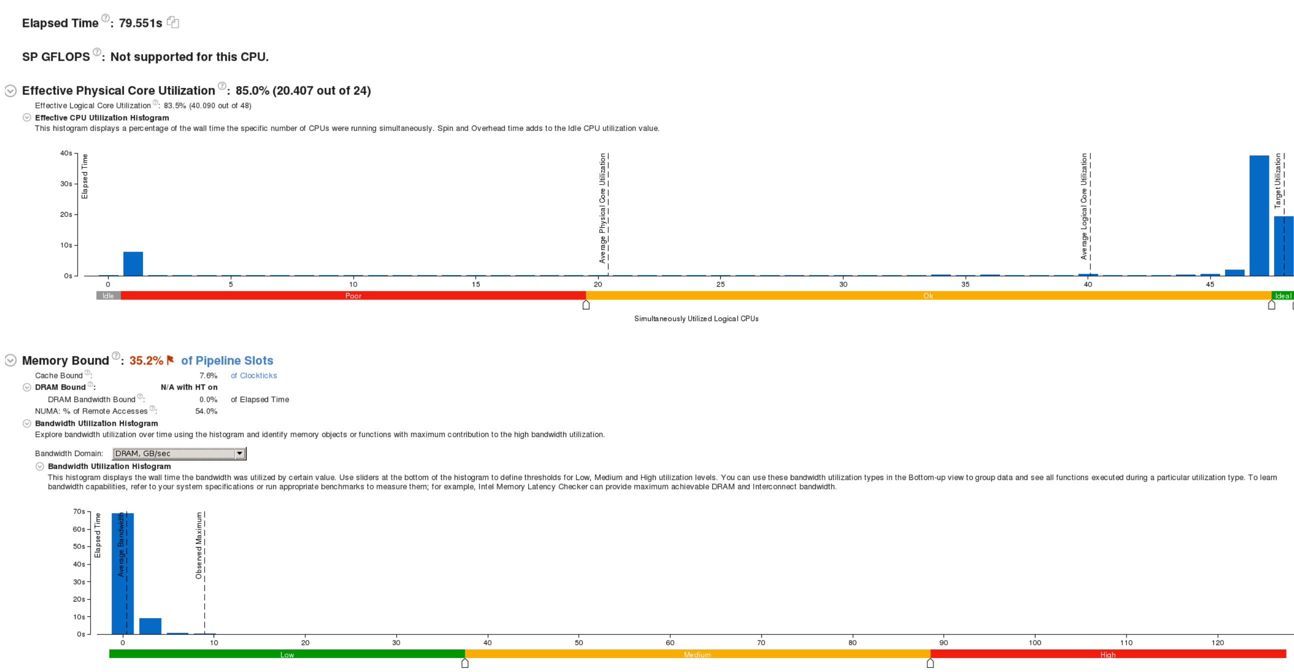
HPC Performance Characterization Analysis

Use the HPC Performance Characterization analysis to identify how effectively your compute-intensive application uses CPU, memory, and floating-point operation hardware resources.



Launch the HPC Characterization profiling

The Summary

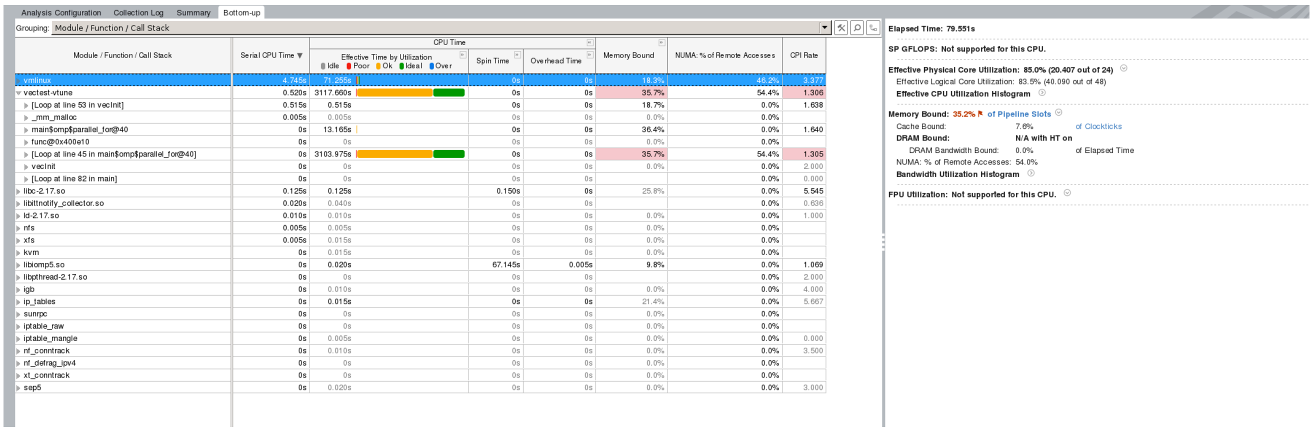


The summary on CPU utilization and memory utilization

The CPU histogram is the same with that in hotspot analysis. We focus on the memory utilization. The memory bound metric is 35%, which has the following definition according to Intel

> This metric shows how memory subsystem issues affect the performance. Memory Bound measures a fraction of slots where pipeline could be stalled due to demand load or store instructions. This accounts mainly for incomplete in-flight memory demand loads that coincide with execution starvation in addition to less common cases where stores could imply back-pressure on the pipeline.

The histogram gives out the DRAM bandwidth usage that is similar to the CPU time histogram. This program only uses less than 2 GB/s of DRAM bandwidth, while the total memory bandwidth is round 100 GB/s. Therefore, we would like to know why the memory bound is high ? Open the bottom-up view tab to find more details



Bottom-up view of HPC Characterization

At the left side, we find the hotspot function for memory usage, it is still the parallel for loop in computing the matrix-vector multiplication. At the right side, it gives a breakdown of the memory bound.

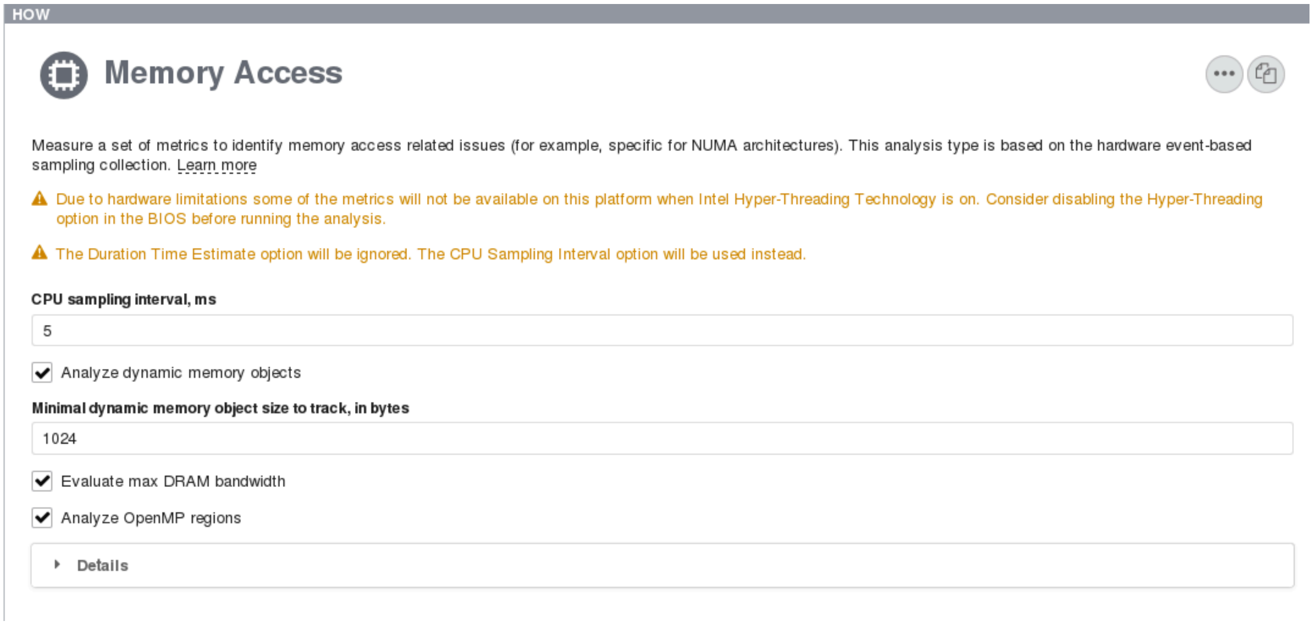
- Cache bound is about 7%, which means the cache usage is relatively good.
- NUMA remote access bound is as high as 54%.

Obviously, the memory bound is mainly due to the remote access of NUMA memory. In Haswell node, each node has two CPU sockets, and each socket has two memory packages. If all of the memory are allocated on one socket package, the threads running on the other socket will need a remote access, which has much higher latency than that on local socket.

Memory Access Evaluation

Note that running memory access evaluation may cause the node down.

To dig out more details about the memory access latency, we launch another profiling type named memory access.

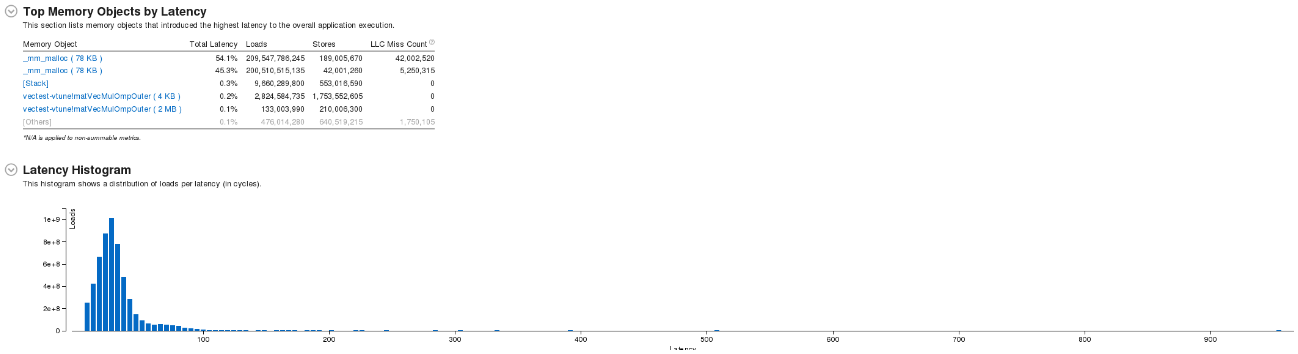


Launch the memory access profiling

The summary of memory access

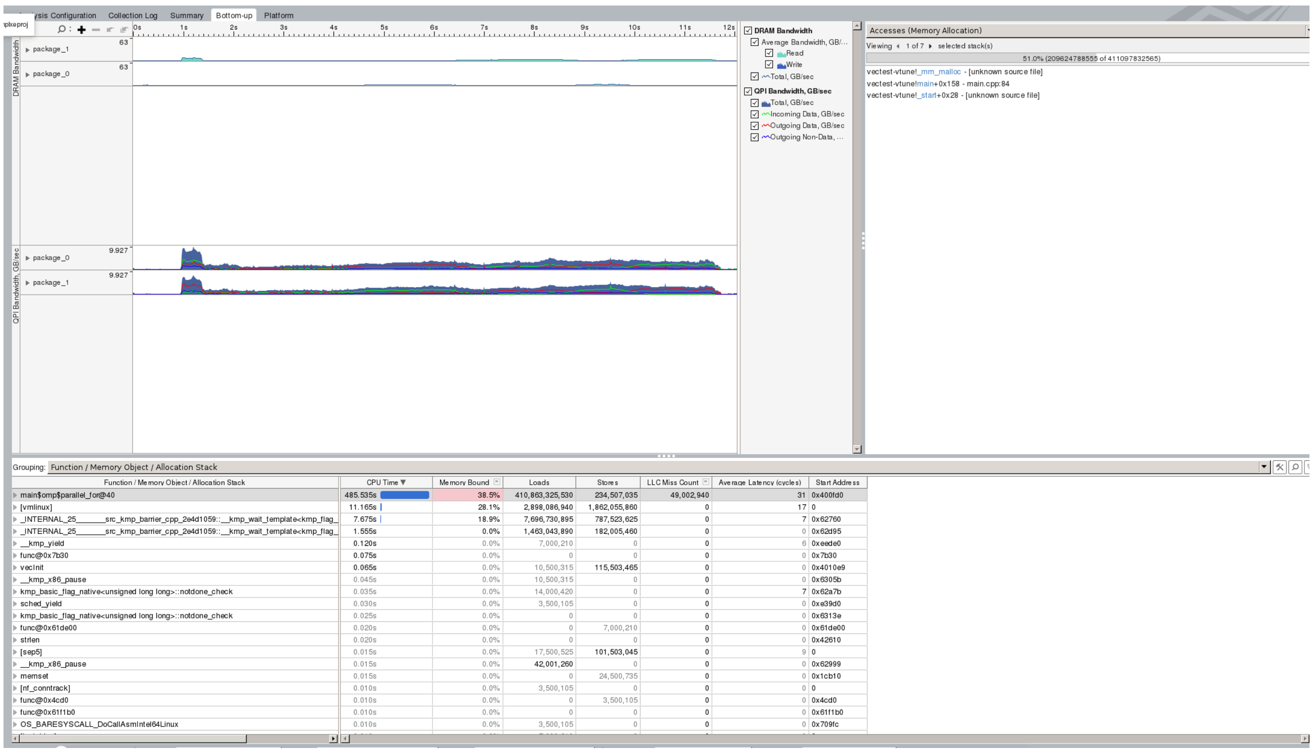
The summary of Memory Access Profiling

Beside the memory bound, we have the memory load/store instruction numbers and the LLC (last-level cache) miss count. We find that the LLC counts are trivial compared to the total memory load/store instructions. It means that the cache performs well in our program, and the latency per operation is not high.



The Memory access latency histogram

The timeline breakdown of Memory Access



The timeline breakdown of memory access

We see that the DRAM memory bandwidth usage is low while there is a substantial QPI bandwidth usage.

> The Intel QuickPath Interconnect (QPI) is a point-to-point processor interconnect developed by Intel

Therefore, we know that all of the memory are initially allocated on the memory from one of the socket, which causes QPI data transfer between the two sockets.