Intel® VTune™ Amplifier XE – Python lab

Create a Project

To analyze your target the VTune Amplifier, you need to create a project, which is a container for an analysis target configuration and data collection results.

1. If working in a bash shell, set the `EDITOR` or `VISUAL` environment variable to associate your source files with the code editor (like emacs, vi, vim, gedit, and so on). For example:

   ```
   $ export EDITOR=gedit
   ```

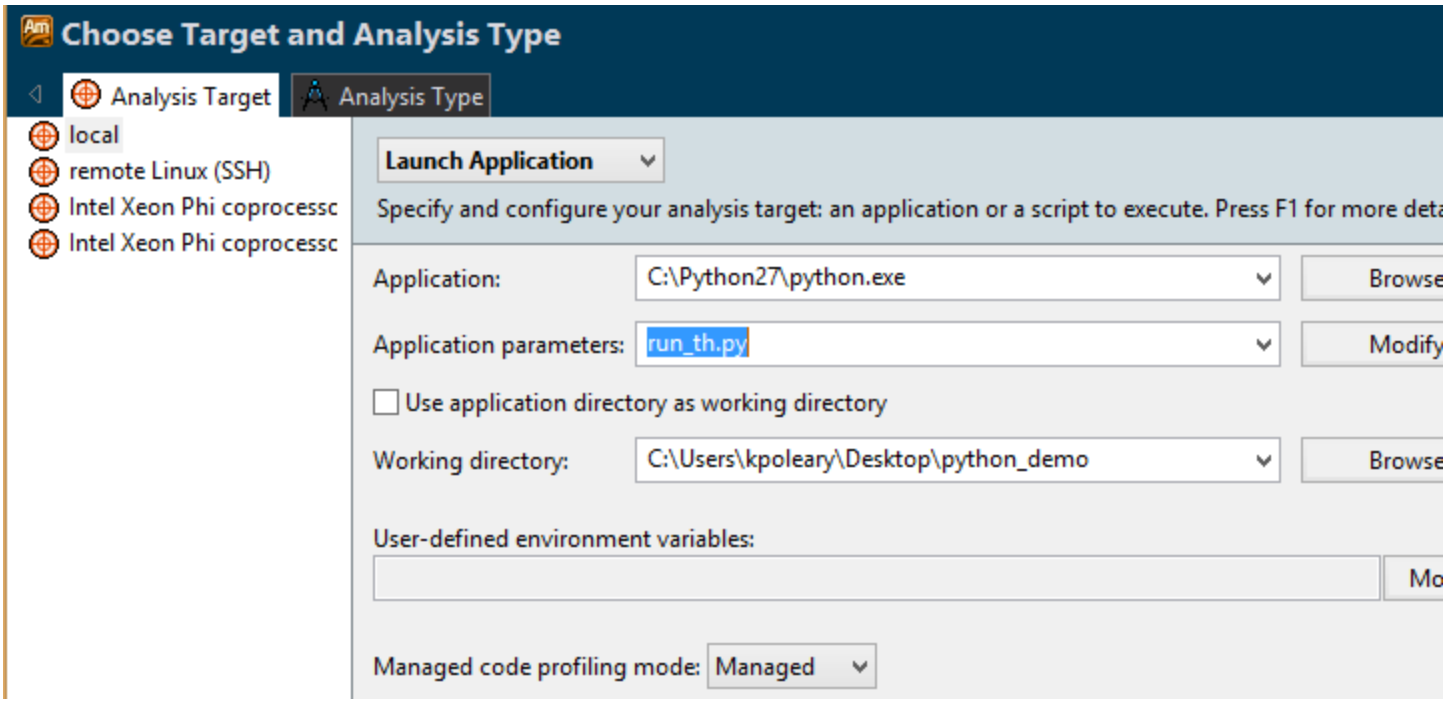
2. Run the `amplxe-gui` script launching the VTune Amplifier GUI.
3. Click the  menu button and select **New > Project...** to create a new project.

   The **Create a Project** dialog box opens.
4. Specify the project name `test_python` that will be used as the project directory name.

   VTune Amplifier creates the `test_python` project directory under the `$HOME/intel/ampl/projects` directory and opens the **Choose Target and Analysis Type** window with the **Analysis Target** tab active.
5. From the left pane, select the **local** target system and from the right pane select the **Application to Launch** target type.

   The configuration pane on the right is updated with the settings applicable to the selected target type.
6. Specify and configure your target as follows:

- For the **Application** field, browse to your python executable
- For the **Application parameters** field, enter your python script
- Specify the working directory for your program to run



7. Click the **Choose Analysis** button on the right to switch to the **Analysis Type** tab.

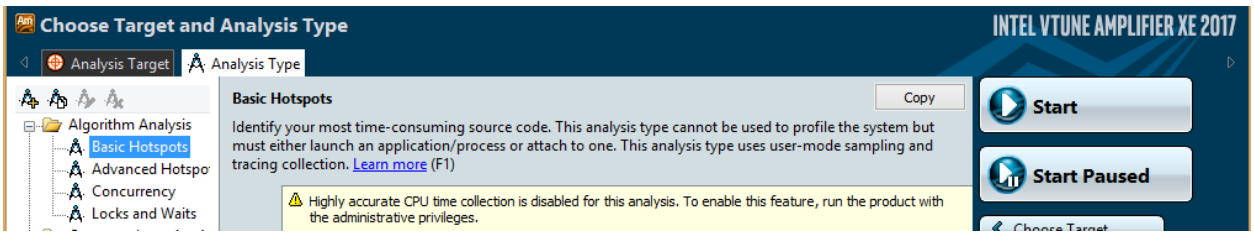
Run Basic Hotspots Analysis

Before running an analysis, choose a configuration level to influence Intel® VTune™ Amplifier analysis scope and running time. In this tutorial, you run the Python Hotspots analysis to identify the hotspots that took much time to execute.

To run an analysis:

1.
   a. In the **Choose Target and Analysis Type** window, switch to the **Analysis Type** tab.
2. From the analysis tree on the left, select **Algorithm Analysis > Basic Hotspots**.

   The right pane is updated with the default options for the Python Hotspots analysis.
3. Click the **Start** button on the right command bar to run the analysis.



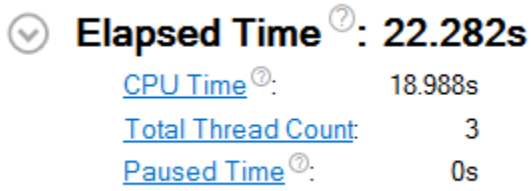
Interpret Result Data

When the sample application exits, the Intel® VTune™ Amplifier finalizes the results and opens the **Python Hotspots by** viewpoint where each window or pane is configured to display code regions that consumed a lot of CPU time. To interpret the data on the sample code performance, do the following:

- Understand the basic performance metrics provided by the Python Hotspots analysis.
- Analyze the most time-consuming functions and CPU usage.
- Analyze performance per thread.

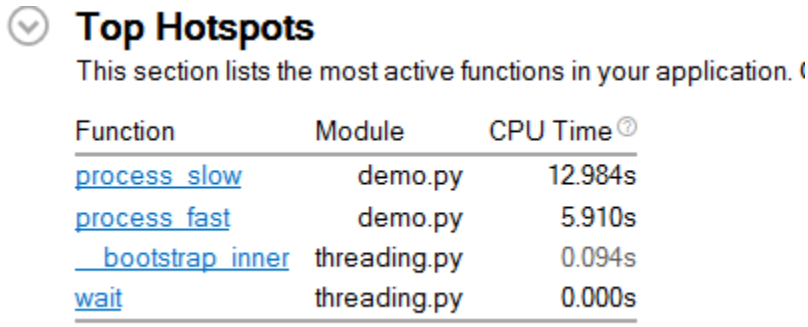
Understand the Python Hotspots Metrics

Start analysis with the **Summary** window. To interpret the data, hover over the question mark icons to read the pop-up help and better understand what each performance metric means.



Note that **CPU Time** for the sample application is equal to 18.988 seconds. It is the sum of CPU time for all application threads. **Total Thread Count** is 3, so the sample application is multi-threaded.

The **Top Hotspots** section provides data on the most time-consuming functions (*hotspot functions*) sorted by CPU time spent on their execution.



For the sample application, the `process_slow` function, which took 12.84 seconds to execute, shows up at the top of the list as the hottest function.

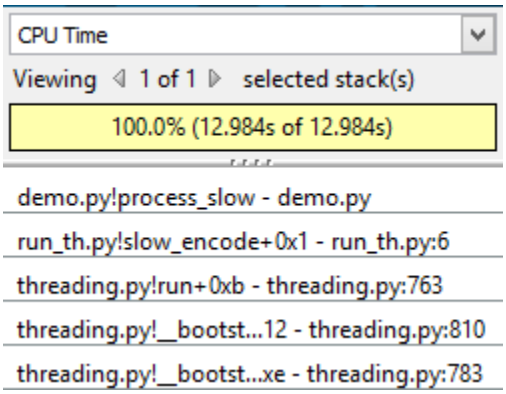
Analyze the Most Time-consuming Functions

Click the **Bottom-up** tab to explore the **Bottom-up** pane. By default, the data in the grid is sorted by Function. You may change the grouping level using the **Grouping** drop-down menu at the top of the grid.

Analyze the **CPU Time** column values. This column is marked with a yellow star as the Data of Interest column. It means that the VTune Amplifier uses this type of data for some calculations (for example, filtering, stack contribution, and others). Functions that took most CPU time to execute are listed on top.

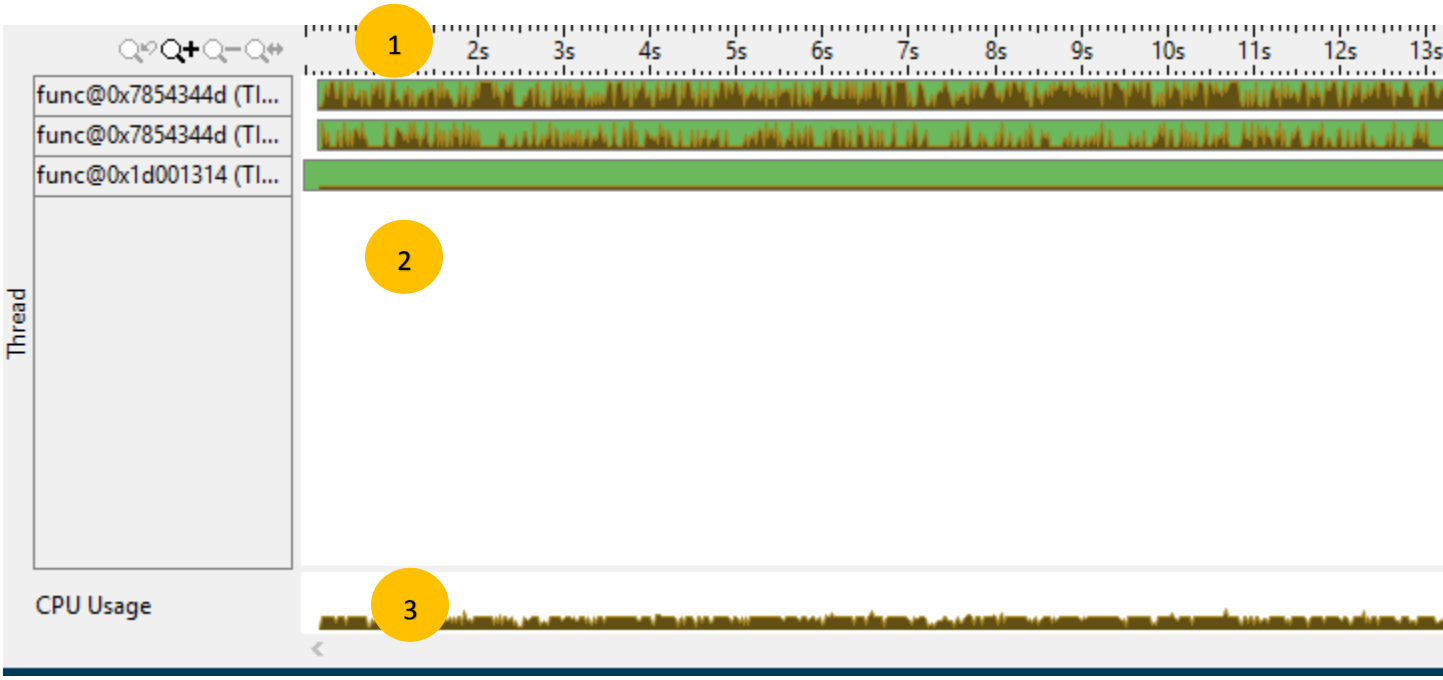
Select the `process_slow` function in the grid and explore the data provided in the **Call Stack** pane on the right. The **Call Stack** pane displays full stack data for each hotspot function, enables you to navigate between function call stacks and understand the impact of each stack to the function CPU time. The stack functions in the **Call Stack** pane are represented in the following format:

<module>!<function> - <file>:<line number>, where the line number corresponds to the line calling the next function in the stack.



Analyze Performance per Thread

To get detailed information on the thread performance, explore the **Timeline** pane.



1. **Timeline** area. When you hover over the graph element, the timeline tooltip displays the time passed since the application has been launched.

2. **Threads** area that shows the distribution of CPU time utilization per thread. Hover over a bar to see the CPU time utilization in percent for this thread at each moment of time. Green zones show the time threads are active.

3. **CPU Usage** area that shows the distribution of CPU time utilization for the whole application. Hover over a bar to see the application-level CPU time utilization in percent at each moment of time.

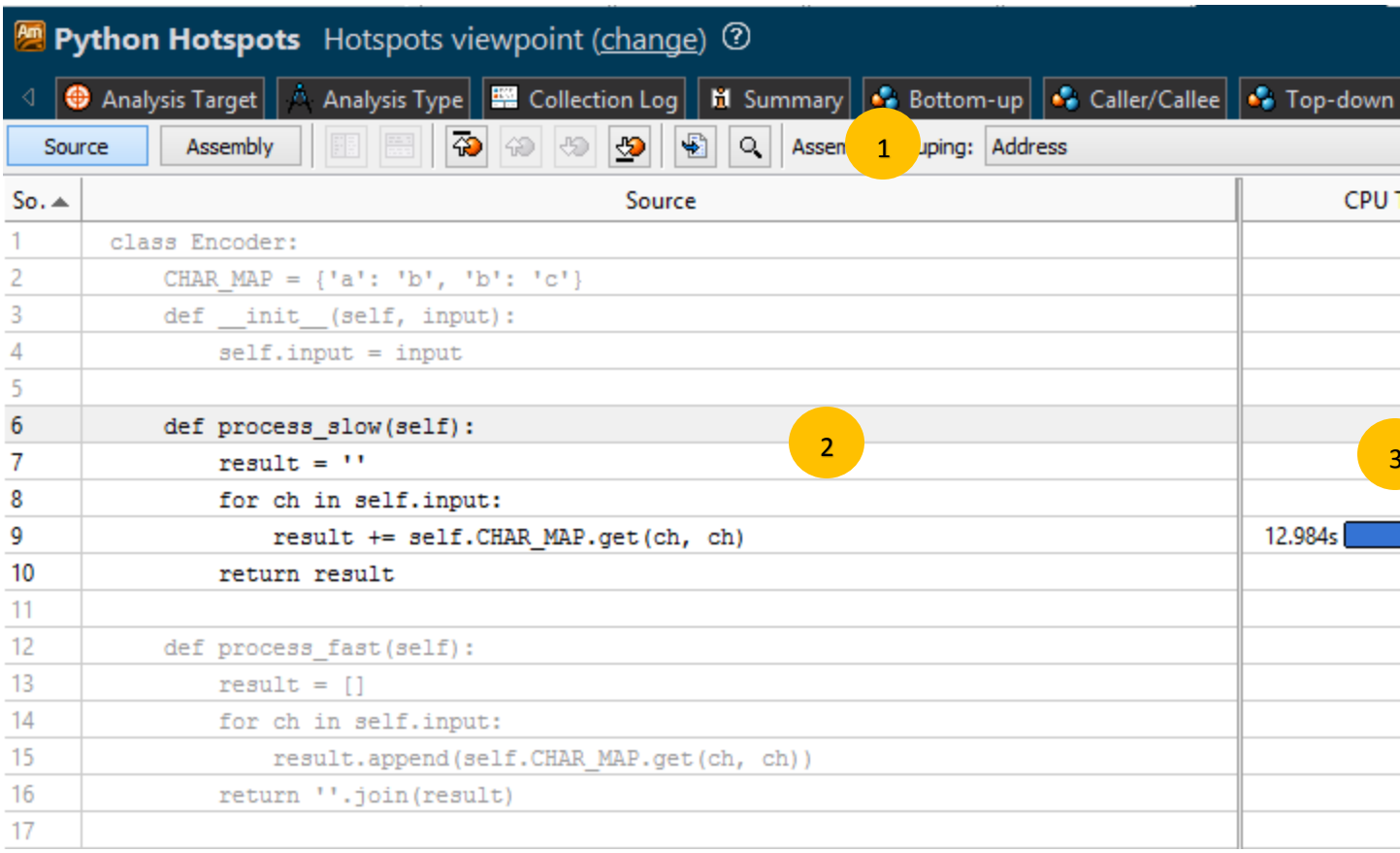
   VTune Amplifier calculates the overall **CPU Usage** metric as the sum of CPU time per each thread of the **Threads** area. Maximum **CPU Usage** value is equal to `[number of processor cores] x 100%`.

Analyze Code

You identified `process_slow` as the hottest function. In the **Bottom-up** pane, double-click this function to open the **Source** window and analyze the source code:

1. Understand basic options provided in the **Source** window.
2. Identify the hottest code lines.

Understand Basic Source Window Options



The table below explains some of the features available in the **Source** window when viewing the Python Hotspots analysis data.

1	Source window toolbar. Use the hotspot navigation buttons to switch between most performance-critical code lines. Hotspot navigation is based on the metric column selected as a Data of Interest. For the **Hotspots by CPU Usage** viewpoint, this is **CPU Time: Self**. Note: The Assembly button is currently unimplemented for Python.
2	**Source** pane displaying the source code of the application if the function symbol information is available. The hottest code line in the function is highlighted. The source code in the **Source** pane is not editable.
3	Processor time attributed to a particular code line. If the hotspot is a system function, its time, by default, is attributed to the user function that called this system function.

Identify the Hottest Code Lines

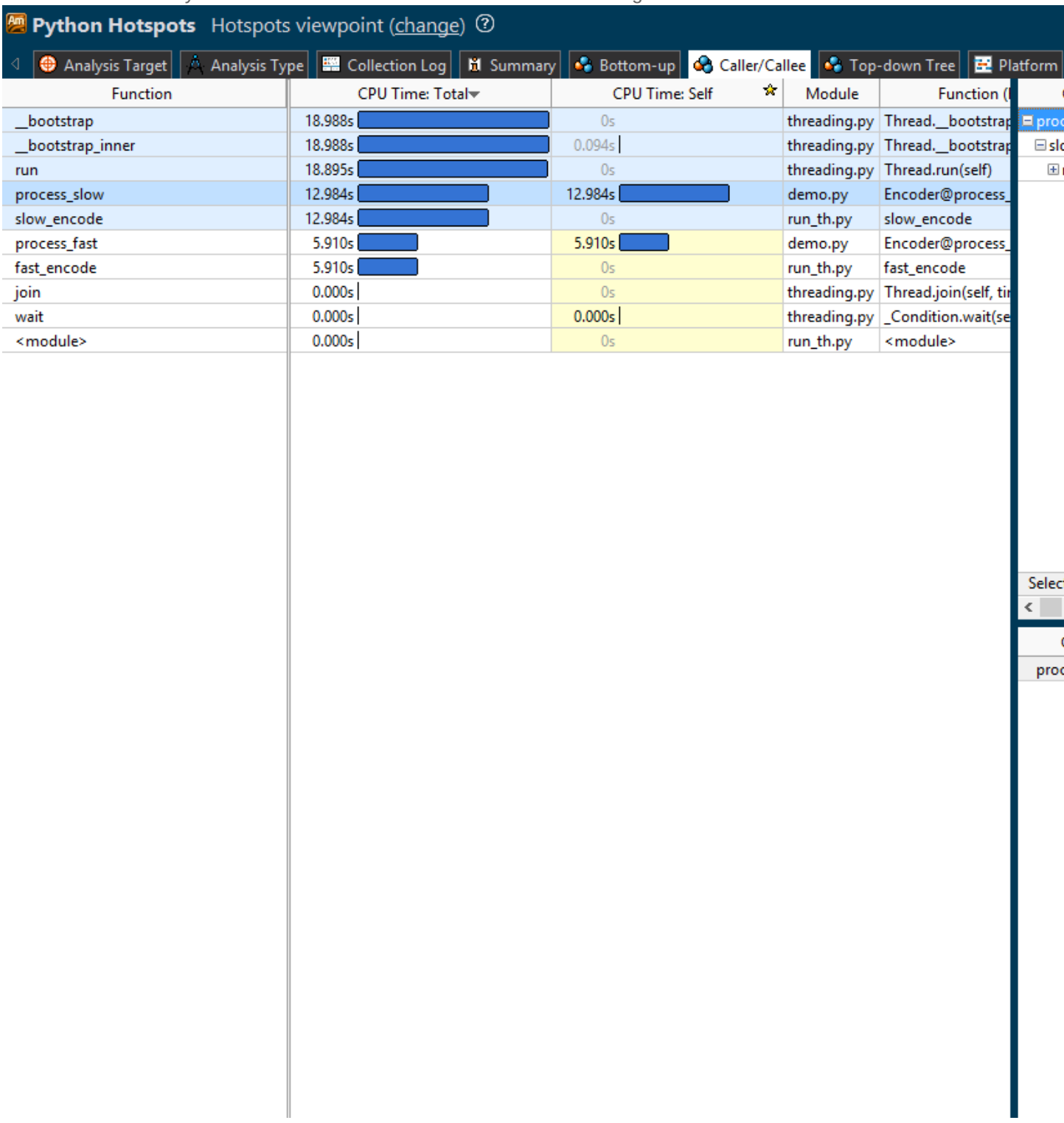
When you identify a hotspot in the serial code, you can make some changes in the code to tune the algorithms and speed up that hotspot. Another option is to parallelize the sample code by adding threads

to the application so that it performs well on multi-core processors. This tutorial focuses on algorithm tuning.

By default, when you double-click the hotspot in the **Bottom-up** pane, VTune Amplifier opens the source file positioning at the most time-consuming code line of this function.

Identify Caller/Callee for a function

You can identify the functions called and who call the function using the Caller/Callee tab.



Compare with Previous Result

After you have optimized your code your can use VTune Amplifier to compare the two different runs. To understand whether you got rid of the hotspot and what kind of optimization you got per function, re-run the Python Hotspots analysis on the optimized code and compare results:

1. Compare results before and after optimization.
2. Identify the performance gain.

Compare Results Before and After Optimization

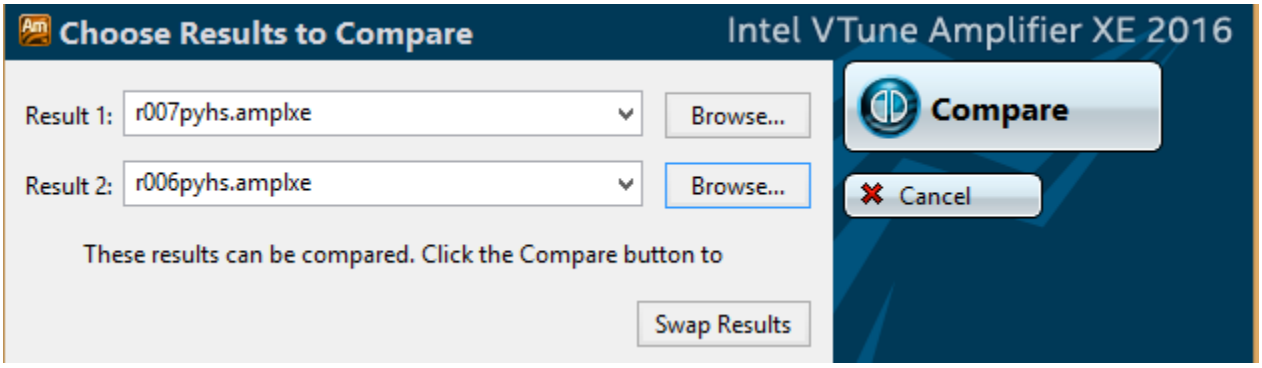
1. Run the Python Hotspots analysis on the modified code.

   VTune Amplifier collects data and opens the result in the Result tab. Make sure to close the results before comparison.

2. Select the result in the **Project Navigator**, right-click and choose **Compare Results** from the context menu.

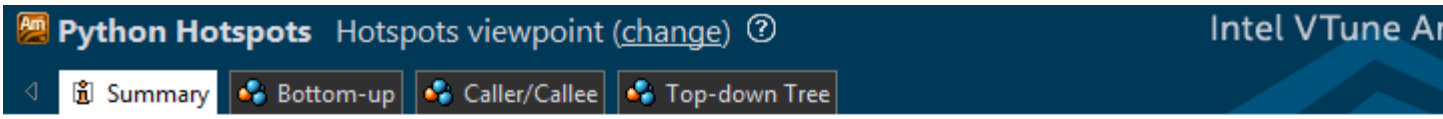
   The **Compare Results** window opens.

3. Specify the Python Hotspots analysis results you want to compare and click the **Compare Results** button:



The Hotspots **Summary** window opens, providing a high-level picture of performance improvements in the following format: *<result 1 value> - <result 2 value>*.

Identify the Performance Gain



Elapsed Time: 22.282s - 22.752s = -0.470s

CPU Time:	18.988s - 19.178s = -0.190s
Total Thread Count:	Not changed, 3
Paused Time:	Not changed, 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in application performance.

Function	Module	CPU Time
process_slow	demo.py	12.984s - 12.850s = 0.134s
process_fast	demo.py	5.910s - 6.234s = -0.324s
__bootstrap_inner	threading.py	Not changed, 0.094s
wait	threading.py	Not changed, 0.000s

Top Hotspots by Difference

This section displays the performance difference between two selected results for the most active functions in your

Function	Module	CPU Time, sorted by abs. difference
process_fast	demo.py	5.910s - 6.234s = -0.324s
process_slow	demo.py	12.984s - 12.850s = 0.134s
wait	threading.py	Not changed, 0.000s
__bootstrap_inner	threading.py	Not changed, 0.094s