



PERFORMANCE ANALYSIS OF PYTHON* APPLICATIONS USING INTEL® VTUNE™ AMPLIFIER

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Tune Python + native code for better Performance

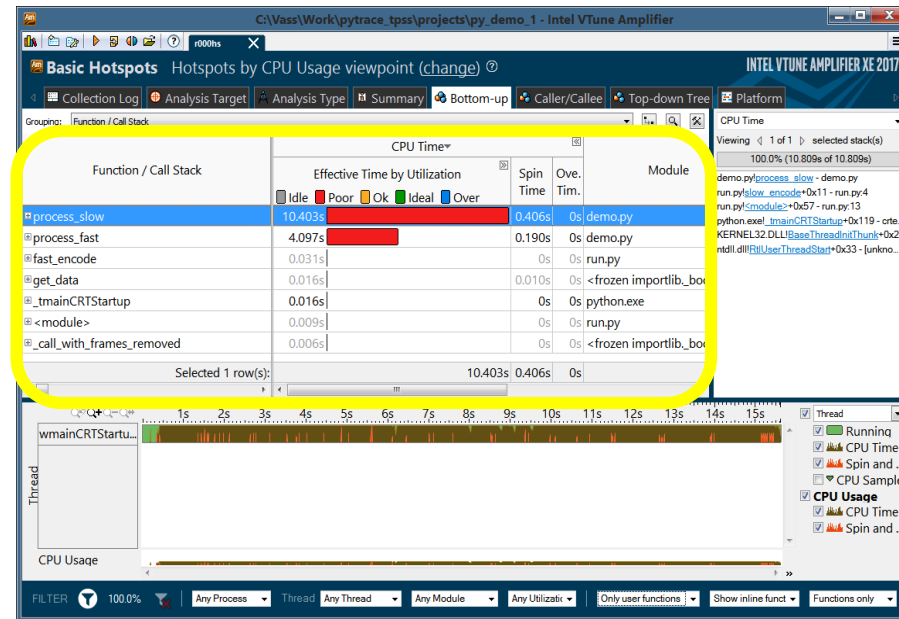
Intel® VTune™ Amplifier 2018, a performance analyzer in Intel® Parallel Studio XE suite

Challenge

- Full profile of Python + native applications
- Detect inefficient runtime execution

Solution

- Accurately identify performance hotspots at line-level
- Auto-detect mixed Python/C/C++ code and extensions
- Focus your tuning efforts for most impact on performance



Auto detection and performance analysis of Python and native functions

Available in Intel® VTune™ Amplifier 2017 Beta & Intel® Parallel Studio XE 2017

Download beta at <https://software.intel.com/en-us/python-profiling>

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Performance analysis of the Covariance Matrix calculation

$$cov = \begin{bmatrix} \sum \frac{x_1^2}{N} & \sum \frac{x_1 x_2}{N} & \cdots & \sum \frac{x_1 x_n}{N} \\ \sum \frac{x_2 x_1}{N} & \sum \frac{x_2^2}{N} & \cdots & \sum \frac{x_2 x_n}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \sum \frac{x_n x_1}{N} & \sum \frac{x_n x_2}{N} & \cdots & \sum \frac{x_n^2}{N} \end{bmatrix}$$

Activity #1: Run Vtune collection for Python: set environment

1. Setup parallel studio environment:

```
cd ~/lab3
```

```
source /opt/intel/parallel_studio_xe_2019/psxevars.sh intel64
```

Activity #1: Run Vtune collection for Python: start amplifier gui

1. Check the script correctness: do not forget parameters!

```
python3 ./lab3.py someVec 100 1000
```

2. Observe the output (of covariance matrix calculation)

3. Start Intel Vtune Amplifier:

```
amplxe-gui
```

Intel® VTune™ Amplifier XE

Feature Highlights

Basic Hot Spot Analysis (Statistical Call Graph)

- Locates the time consuming regions of your application
- Provides associated call-stacks that let you know how you got to these time consuming regions
- Call-tree built using these call stacks

Advanced Hotspot and architecture analysis

- Based on Hardware Event-based Sampling (EBS)
- Pre-defined tuning experiments

Thread Profiling

- Visualize thread activity and lock transitions in the timeline
- Provides lock profiling capability
- Shows CPU/Core utilization and concurrency information

GPU Compute Performance Analysis

- Collect GPU data for tuning OpenCL applications. Correlate GPU and CPU activities

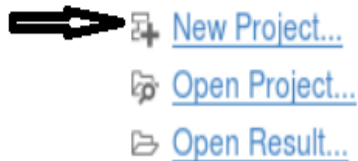
Intel® VTune™ Amplifier XE

Analysis Types (based on technology)

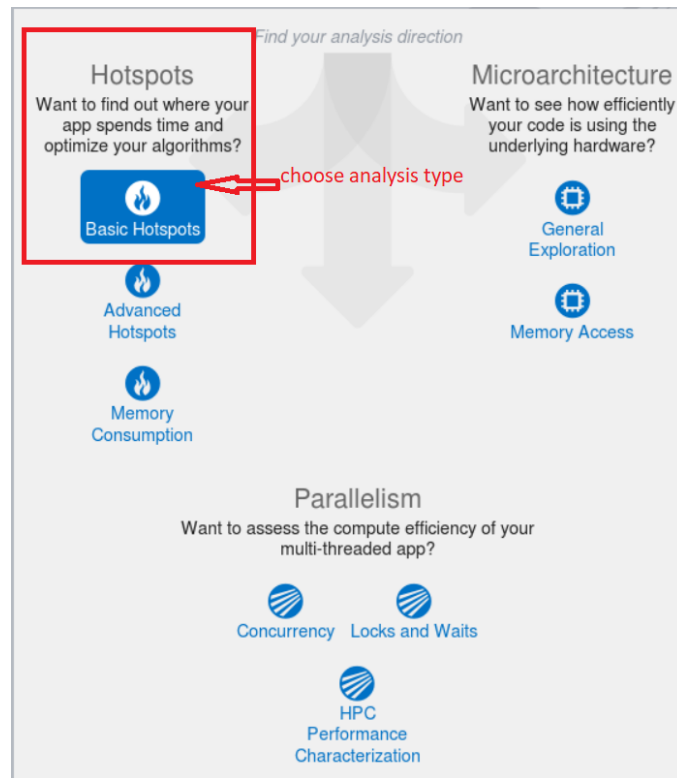
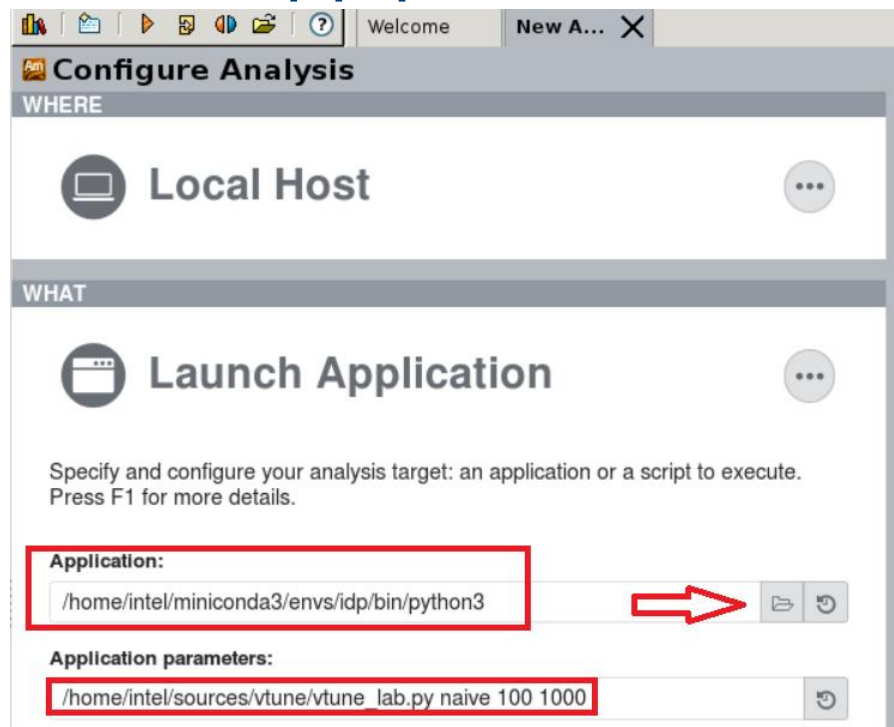
Software Collector Any x86 processor, any virtual, no driver	Hardware Collector Higher res., lower overhead, system wide
Basic Hotspots Which functions use the most time?	Advanced Hotspots Which functions use the most time? Where to inline? – Statistical call counts
Concurrency Tune parallelism. Colors show number of cores used.	General Exploration Where is the biggest opportunity? Cache misses? Branch mispredictions?
Locks and Waits Tune the #1 cause of slow threaded performance – waiting with idle cores.	Advanced Analysis Dig deep to tune bandwidth, cache misses, access contention, etc.

3.1. Configure Amplifier activity

Create new project

The image shows the 'Create a Project' dialog box. It has a title bar 'Create a Project'. Below the title bar, there are three main sections: 'Project name:', 'Location:', and 'Managed code profiling mode'. The 'Project name:' section has a text input field containing 'Vtune_python' and a red arrow pointing to it. The 'Location:' section has a text input field containing '/home/day1/intel/amp' and a blue 'Create Project' button below it. The 'Managed code profiling mode' section has a dropdown menu with 'Auto' selected. There is also a section for 'User-defined environment variables:' with a text input field and a scroll bar.

3.2 Configuring application to launch: do not forget to add app parameters!



3.3 Summary page: observer top hotspot list. Call function

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform

INTEL VTUNE AMPLIFIER 2018

Elapsed Time: 15.432s

CPU Time: 15.350s
Total Thread Count: 1
Paused Time: 0s

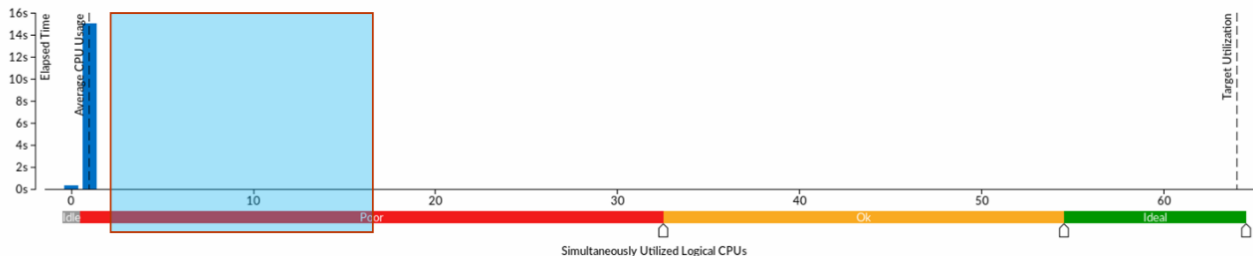
Top Hotspots

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

Function	Module	CPU Time
call_function	libpython3.6m.so.1.0	1.629s
PyObject_CallFunctionObjArgs	libpython3.6m.so.1.0	1.586s
PyUFunc_GenericFunction	umath.cpython-36m-x86_64-linux-gnu.so	0.898s
PyArray_NewFromDescr	multitarray.cpython-36m-x86_64-linux-gnu.so	0.784s
<genexpr>	test_cov.py	0.450s
[Others]		10.003s

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.



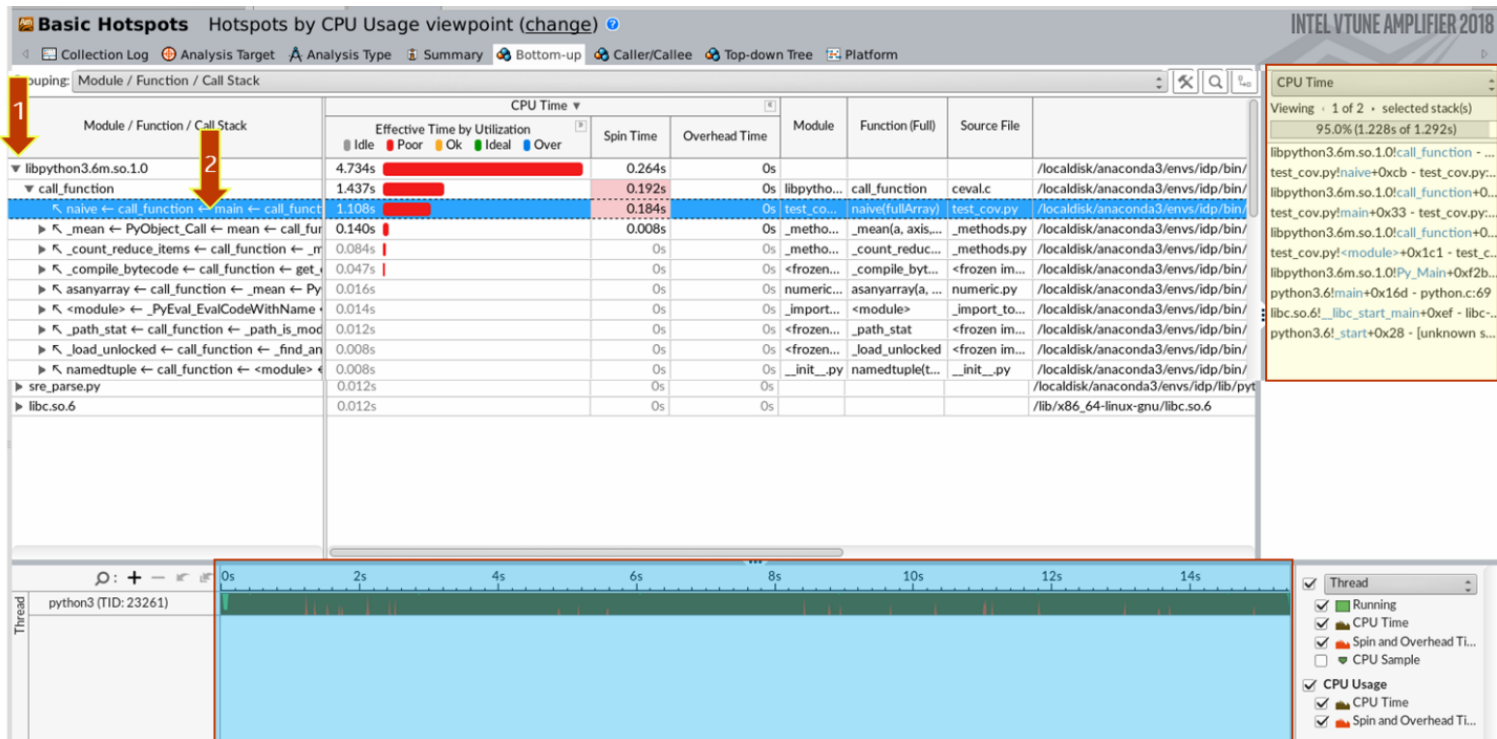
Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.

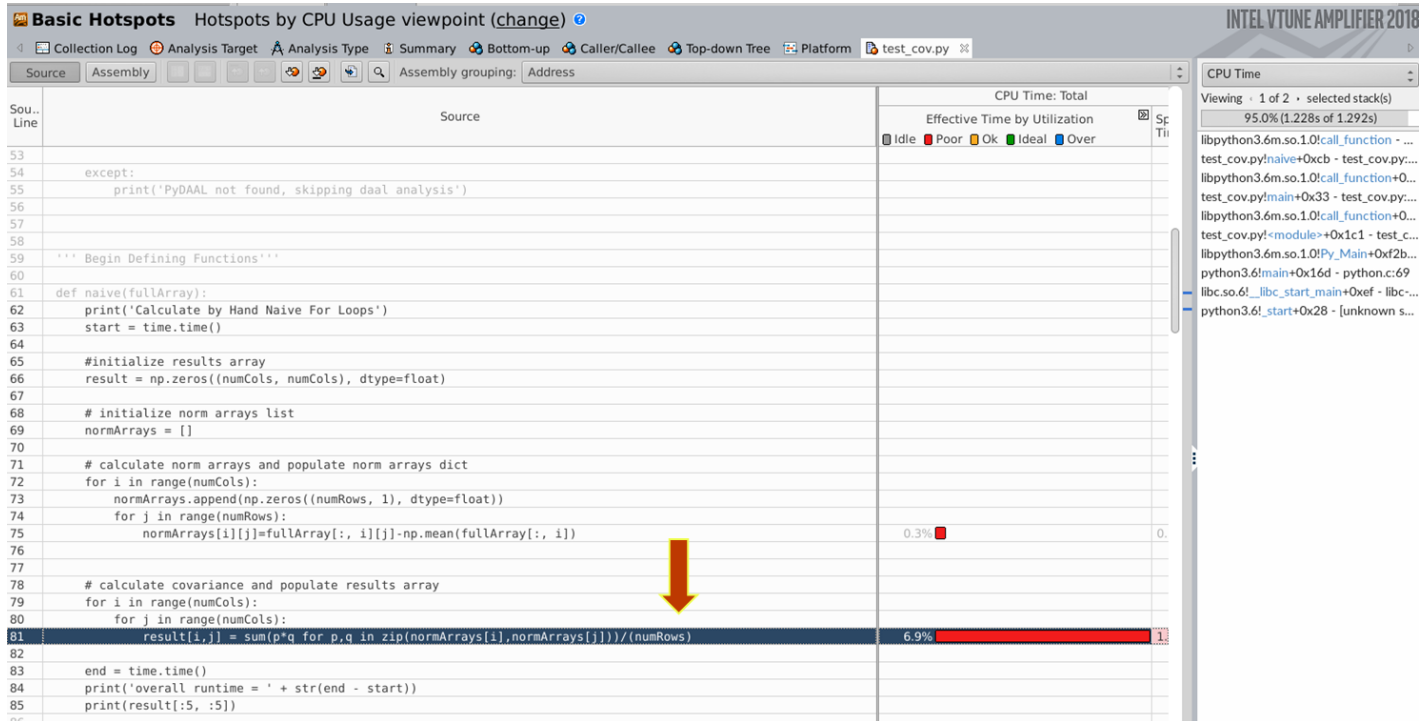
*Other names and brands may be claimed as the property of others.



3.4 Basic hotspots bottom up view



3.5 Source code view



Optimization Notice


Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity #4: code modification. Numpy function substitute


77		
78	# calculate covariance and populate results array	
79	for i in range(numCols):	
80	for j in range(numCols):	
81	result[i,j] = sum(p*q for p,q in zip(normArrays[i],normArrays[j]))/(numRows)	6.9% 1
82		

```
77
78 # calculate covariance and populate results array
79 = for i in range(numCols):
80 =     for j in range(numCols):
81         result[i,j] = sum(p*q for p,q in zip(normArrays[i],normArrays[j]))/(numRows)
82
```



BEFORE

```
105 # calculate covariance and populate results array
106 = for i in range(numCols):
107 =     for j in range(i+1, numCols):
108         result[i,j] = sum(np.multiply(normArrays[i],normArrays[j]))/(numRows)
109
```



AFTER

4.1 Re-run the performance.

- Run in the terminal

```
python3 Tab3.py someVec 100 1000
```

Calculate by Hand Some Vectorization

overall runtime = 5.254168748855591

what could be improved further?

Activity #5

Further vectorization and performance improvement

5.1 Configuring application to launch:

Choose “someVec” module in the python script



Launch Application

Specify and configure your analysis target: an application or a script to execute. Press F1 for more details.

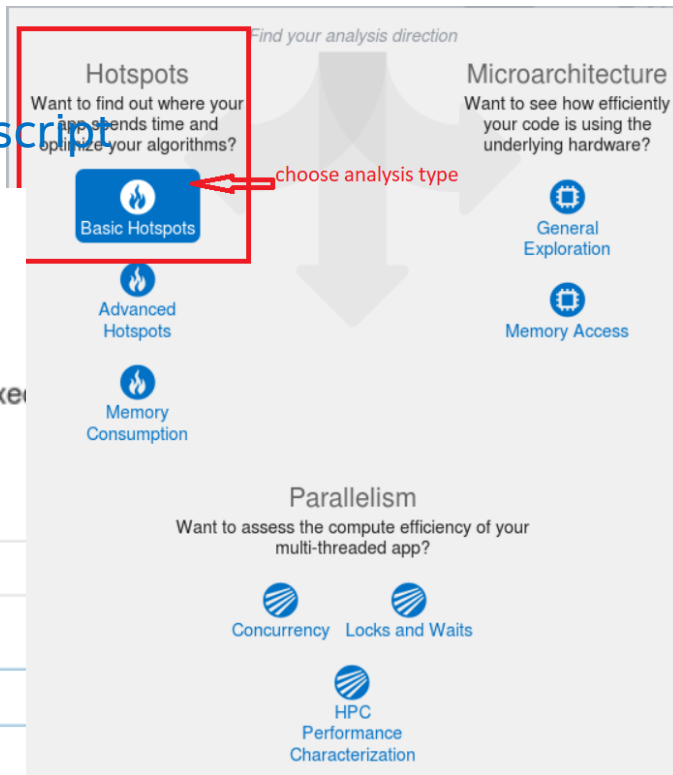
Application:

/opt/intel/intelpython3/bin/python3

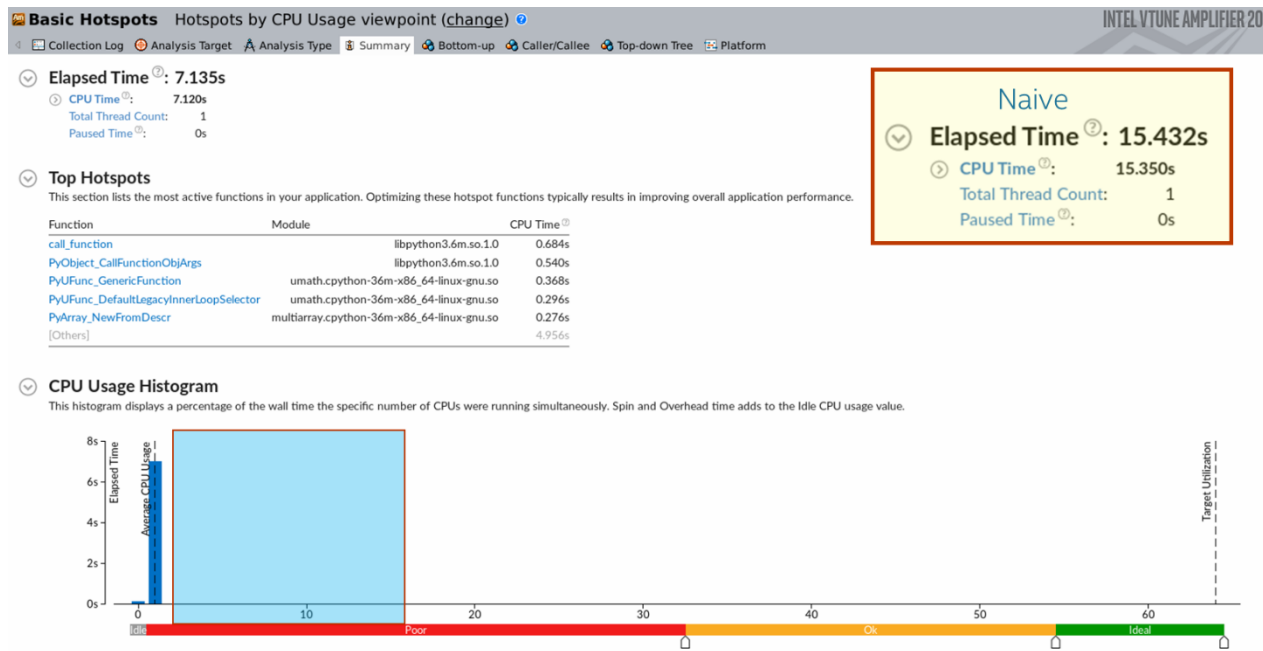
Application parameters:

vtune_lab_orig.py someVec 100 1000

☒ Use application directory as working directory



5.2 Updated code: twice as fast.

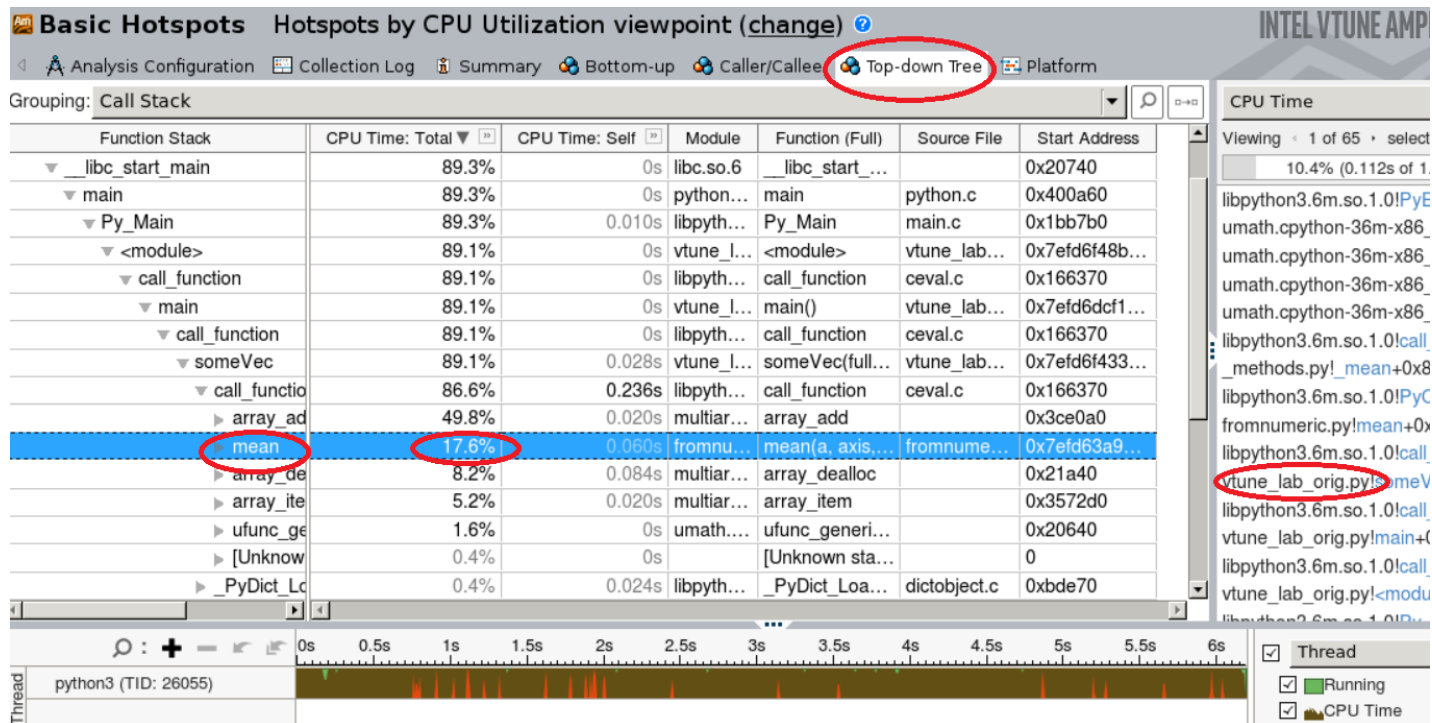


Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



5.3 Top down view to find the issue



5.4 Drill down to the source view

```
//  
78 # "some vectorization" covariance function  
79 def someVec(fullArray):  
80     print('Calculate by Hand Some Vectorization')  
81     start = time.time()  
82  
83     # initialize results array  
84     result = np.zeros((numCols, numCols), dtype=float)  
85  
86  
87     # initialize norm arrays list  
88     normArrays = []  
89  
90     # calculate norm arrays and populat norm arrays dict  
91     for i in range(numCols):  
92         normArrays.append(np.zeros((numRows, 1), dtype=float))  
93         for j in range(numRows):  
94             normArrays[i][j]=fullArray[:, i][j]-np.mean(fullArray[:, i])  
95  
96     # calculate covariance and populat results array  
97     for i in range(numCols):
```


0.1%

21.0%

5.5 Code changes we need to do...


```
99      # calculate norm arrays and populat norm arrays dict
100  =    for i in range(numCols):
101      |        normArrays.append(np.zeros((numRows, 1), dtype=float))
102  =    |        for j in range(numRows):
103      |        |        normArrays[i][j]=fullArray[:, i][j]-np.mean(fullArray[:, i])
104
```

BEFORE



```
126      # calculate norm arrays and populat norm arrays dict
127  =    for i in range(numCols):
128      |        normArrays.append(np.zeros((numRows, 1), dtype=float))
129      |        normArrays[i]=np.subtract(fullArray[:, i], np.mean(fullArray[:, i]))
130  =    |        for j in range(i+1):
131      |        |        result[i,j] = sum(np.multiply(normArrays[i],normArrays[j]))/(numRows)
132
```

AFTER



5.6 Re-run the test in command line

If the source code is modified within the same module please to this:

Run

```
python3 lab3.py someVec 100 1000
```

Calculate by Hand Some Vectorization

overall runtime = ?

Summary:

we identified the slowest line in our “Some Vectorization” covariance matrix function, and replaced it with a better organized for-loop and vector subtract method from Numpy*

Activity 6

Run

```
python3 lab3.py moreVec 100 1000
```

Calculate by Hand More Vectorization

overall runtime = 1.182539701461792

Let's make sure it is the best possible performance!

Elapsed Time[?]: 1.394s

CPU Time[?]: 1.380s
Total Thread Count: 1
Paused Time[?]: 0s

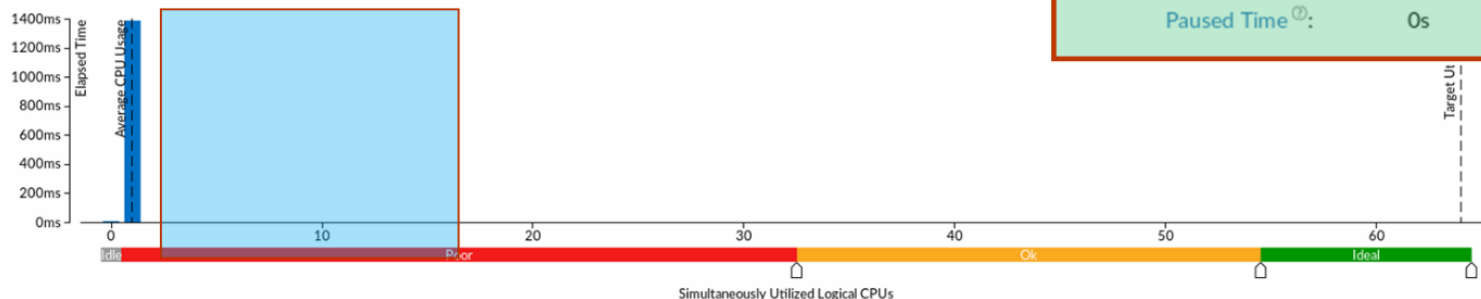
Top Hotspots

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

Function	Module	CPU Time [?]
call_function	libpython3.6m.so.1.0	0.182s
__GI_fetestexcept	libm.so.6	0.164s
PyCFunction_Call	libpython3.6m.so.1.0	0.137s
numpy_get_floatstatus	umath.cpython-36m-x86_64-linux-gnu.so	0.100s
double_add	umath.cpython-36m-x86_64-linux-gnu.so	0.084s
[Others]		0.713s

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.



Naive

Elapsed Time[?]: 15.432s

CPU Time[?]: 15.350s
Total Thread Count: 1
Paused Time[?]: 0s

Some Vectorization

Elapsed Time[?]: 7.135s

CPU Time[?]: 7.120s
Total Thread Count: 1
Paused Time[?]: 0s

summary

VTUNE is the best tool ever and it now helps with Python code analysis, too!!!

backup

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Resources

Get Intel's Distribution for Python:

- Visit <https://software.intel.com/en-us/python-distribution> for download, documentation and support
- Also available at Intel channel at Anaconda (you can use “conda install!”):
<https://anaconda.org/intel>
- <https://software.intel.com/en-us/vtune-amplifier-help-python-code-analysis>
-

