

# 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<sup>®</sup> VTune<sup>™</sup> Amplifier 2018, a performance analyzer in Intel<sup>®</sup> 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

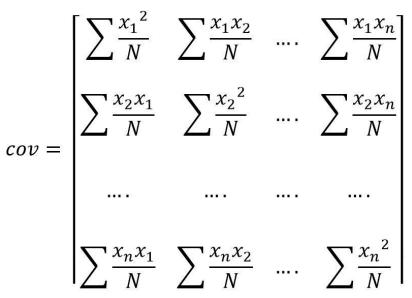
Collection Log 🌒 Analysis Target	Analysis Type 📓 Summary 🗟 Bottom-u	n 🙆 Cal	lor/Ca	Ilee 🔒 Top-down Tree	E Platform
rouping: Function / Call Stack	Analysis type a Summary of Bottom-t	P Car	ier/ca	- L. Q X	CPU Time
Function / Call Stack	CPU Time♥ Effective Time by Utilization Idle ■ Poor ■ Ok ■ Ideal ■ Over	Spin Time	≪ Ove. Tim.	Module	Viewing
process_slow	10.403s	0.406s	0s	demo.py	run.pyl <u><module></module></u> +0x57 - run.py:13 python.exe! tmainCRTStartup+0x119 - cr
process_fast	4.097s	0.190s	0s	demo.py	KERNEL32.DLLIBaseThreadInitThunk+0
fast_encode	0.031s	Os	0s	run.py	ntdll.dll! <u>RtlUserThreadStart</u> +0x33 - [unkn
get_data	0.016s	0.010s	0s	<frozen importlibbo<="" td=""><td></td></frozen>	
_tmainCRTStartup	0.016s	Os	0s	python.exe	
<module></module>	0.009s	Os	0s	run.py	
_call_with_frames_removed	0.006s	Os	0s	<frozen importlibbo<="" td=""><td></td></frozen>	
Selected 1 row(s)	10.403	3s 0.406s	0s		
CPU Usage         15         25         31	✓ "" 45 55 65 75 85 41 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	9 <u>s 1(</u>	)s	<u>115 125 135</u> ⊢ N M	145 155 V Thread V Threa

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

# Performance analysis of the Covariance Matrix calculation



#### **Optimization Notice**



# 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<sup>®</sup> VTune<sup>™</sup> 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

**Optimization Notice** 

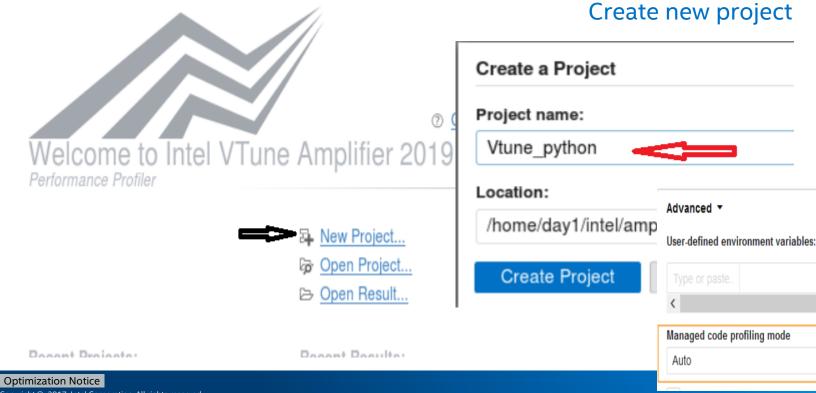


## Intel<sup>®</sup> VTune<sup>™</sup> Amplifier XE Analysis Types (based on technology)

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



# 3.1. Configure Amplifier activity



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

Image: Market State     Image: Market State     New A X       Image: Market State     Image: Market State     New A X	Find your analysis direction Hotspots Want to find out where your Want to see how efficiently
WHERE	app spends time and your code is using the optimize your algorithms? underlying hardware?
Local Host	choose analysis type Basic Hotspots General Exploration
WHAT	Advanced Hotspots Memory Access
Launch Application	Memory Consumption
Specify and configure your analysis target: an application or a script to execute. Press F1 for more details.	Parallelism Want to assess the compute efficiency of your multi-threaded app?
Application:         /home/intel/miniconda3/envs/idp/bin/python3	Concurrency Locks and Waits
Application parameters: /home/intel/sources/vtune/vtune_lab.py naive 100 1000	HPC Performance Characterization

#### Optimization Notice

# 3.3 Summary page: observer top hotspot list. Call function

Platform

INTEL VTUNE AMPLIFIER 2018

Basic Hotspots	Hotspots by CPU U	Jsage viewpo	int ( <u>change</u> )	0		
Collection Log O Anal	lysis Target 👌 Analysis T	ype 🔋 Summary	😪 Bottom-up	Caller/Callee	😪 Top-down Tree	E

✓ Elapsed Time <sup>②</sup>: 15.432s

```
    CPU Time<sup>(1)</sup>: 15.350s
    Total Thread Count: 1
    Paused Time<sup>(1)</sup>: 0s
```

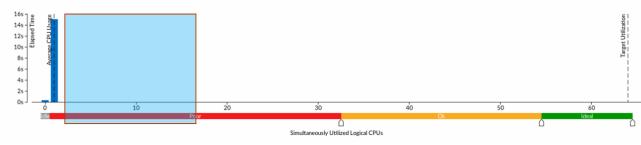
#### Sop Hotspots

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	multiarray.cpython-36m-x86_64-linux-gnu.so	0.784s
<genexpr></genexpr>	test_cov.py	0.450s
[Others]		10.003s

#### OPU 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**

## 3.4 Basic hotspots bottom up view

ouping: Module / Function / Call Stack							: 🛠 Q 👊	CPU Time
	CPU T	ïme ▼	6				0	Viewing ( 1 of 2 + selected stack(s)
Module / Function / Call Stack	Effective Time by Utilization	> Spin Time	Overhead Time	Module	Function (Full)	Source File		95.0% (1.228s of 1.292s) libpython3.6m.so.1.0!call_function
bpython3.6m.so.1.0 2	4.734s	0.264s	Os				/localdisk/anaconda3/envs/idp/bin/	test_cov.py!naive+0xcb - test_cov.p
r call_function	1.437s	0.192s	Os	libpytho	call_function	ceval.c	/localdisk/anaconda3/envs/idp/bin/	libpython3.6m.so.1.0!call_function
$\land$ naive $\leftarrow$ call_function $\leftarrow$ main $\leftarrow$ call_funct	1.108s	0.184s	Os	test_co	naive(fullArray)	test_cov.py	/localdisk/anaconda3/envs/idp/bin/	test_cov.py!main+0x33 - test_cov.p
▶ $\land$ _mean ← PyObject_Call ← mean ← call_fur	0.140s	0.008s	Os	_metho	_mean(a, axis,	_methods.py	/localdisk/anaconda3/envs/idp/bin/	libpython3.6m.so.1.0!call function
▶ $\land$ _count_reduce_items $\leftarrow$ call_function $\leftarrow$ _m	0.084s	Os	Os	_metho	_count_reduc	_methods.py	/localdisk/anaconda3/envs/idp/bin/	test_cov.py! <module>+0x1c1 - tes</module>
▶ $\land$ _compile_bytecode $\leftarrow$ call_function $\leftarrow$ get_	0.047s	Os	Os	<frozen< td=""><td>_compile_byt</td><td><frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>libpython3.6m.so.1.0!Py Main+0xf</td></frozen></td></frozen<>	_compile_byt	<frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>libpython3.6m.so.1.0!Py Main+0xf</td></frozen>	/localdisk/anaconda3/envs/idp/bin/	libpython3.6m.so.1.0!Py Main+0xf
▶ $\land$ asanyarray $\leftarrow$ call_function $\leftarrow$ _mean $\leftarrow$ Py	0.016s	Os	Os	numeric	asanyarray(a,	numeric.py	/localdisk/anaconda3/envs/idp/bin/	python3.6!main+0x16d - python.c
▶ < <module> ← _PyEval_EvalCodeWithName ·</module>	0.014s	Os	Os	_import	<module></module>	_import_to	/localdisk/anaconda3/envs/idp/bin/	libc.so.6! libc start main+0xef - li
▶ $\land$ _path_stat $\leftarrow$ call_function $\leftarrow$ _path_is_mod	0.012s	Os	Os	<frozen< td=""><td>_path_stat</td><td><frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>python3.6! start+0x28 - [unknown</td></frozen></td></frozen<>	_path_stat	<frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>python3.6! start+0x28 - [unknown</td></frozen>	/localdisk/anaconda3/envs/idp/bin/	python3.6! start+0x28 - [unknown
▶ < _load_unlocked ← call_function ← _find_an	0.008s	Os	Os	<frozen< td=""><td>_load_unlocked</td><td><frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>pythonoici_state control</td></frozen></td></frozen<>	_load_unlocked	<frozen im<="" td=""><td>/localdisk/anaconda3/envs/idp/bin/</td><td>pythonoici_state control</td></frozen>	/localdisk/anaconda3/envs/idp/bin/	pythonoici_state control
▶ namedtuple ← call_function ← <module></module>	0.008s	Os	Os	initpy	namedtuple(t	initpy	/localdisk/anaconda3/envs/idp/bin/	
sre_parse.py	0.012s	Os	Os				/localdisk/anaconda3/envs/idp/lib/pyt	
libc.so.6	0.012s	Os	Os				/lib/x86_64-linux-gnu/libc.so.6	
Q: + - = = = 05	25 45	65			10s		125 145	
			1					Thread :

#### **Optimization Notice**

## 3.5 Source code view

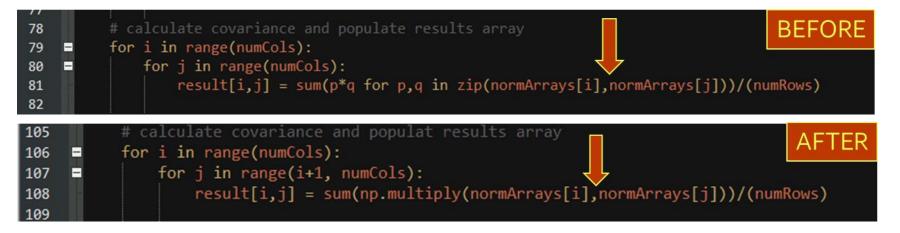
Sour	ce Assembly 🔲 🔄 🔄 🧐 🧐 🗐 🔍 Assembly grouping: Address			\$	CPU Time
		CPU Time: Total			Viewing 1 of 2 + selected stack(s)
iou Line	Source	Effective Time by Utilization	≫ sp		95.0% (1.228s of 1.292s)
		🔲 Idle 📕 Poor 🚺 Ok 📕 Ideal 📕 Over	Ti		libpython3.6m.so.1.0!call_function
3					test_cov.py!naive+0xcb - test_cov.p
4	except:				libpython3.6m.so.1.0!call_function+
5	print('PyDAAL not found, skipping daal analysis')				test_cov.py!main+0x33 - test_cov.p
6					libpython3.6m.so.1.0!call function
7					test_cov.py! <module>+0x1c1 - test</module>
8	111 Davis Dafining Functionalli				libpython3.6m.so.1.0!Py_Main+0xf2
i9 i0	''' Begin Defining Functions'''			-	python3.6!main+0x16d - python.c:
	def naive(fullArray):				<ul> <li>libc.so.6!_libc_start_main+0xef - lib</li> </ul>
j1 j2	print('Calculate by Hand Naive For Loops')			10	python3.6!_start+0x28 - [unknown
3	start = time.time()				python3.6:_start+0x26 - [unknown
4	start = time.time()			· •	
5	#initialize results array				
6	result = np.zeros((numCols, numCols), dtype=float)				
7	·····				
8	# initialize norm arrays list				
9	normArrays = []				
0					
1	# calculate norm arrays and populate norm arrays dict				
2	for i in range(numCols):				
3	<pre>normArrays.append(np.zeros((numRows, 1), dtype=float))</pre>				
4	for j in range(numRows):				
5	normArrays[i][j]=fullArray[:, i][j]-np.mean(fullArray[:, i])	0.3%	0.		
6					
7					
8	# calculate covariance and populate results array				
9	for i in range(numCols):				
0	for j in range(numCols):	6.00/	1		
2	result[i,j] = sum(p*q for p,q in zip(normArrays[i],normArrays[j]))/(numRows)	6.9%	1.	ŝ	
3	<pre>end = time.time()</pre>				
4	print('overall runtime = ' + str(end - start))				
14	<pre>print( overall runtime = " + str(end - start)) print(result[:5, :5])</pre>				

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

(intel)

# Activity #4: code modification. Numpy function

//			
78	# calculate covariance and populate results array		
79	<pre>for i in range(numCols):</pre>		
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			





## 4.1 Re-run the performance.

- Run in the terminal

python3 lab3.py somevec 100 1000

Calculate by Hand Some Vectorization

overall runtime = 5.254168748855591

what could be improved further?

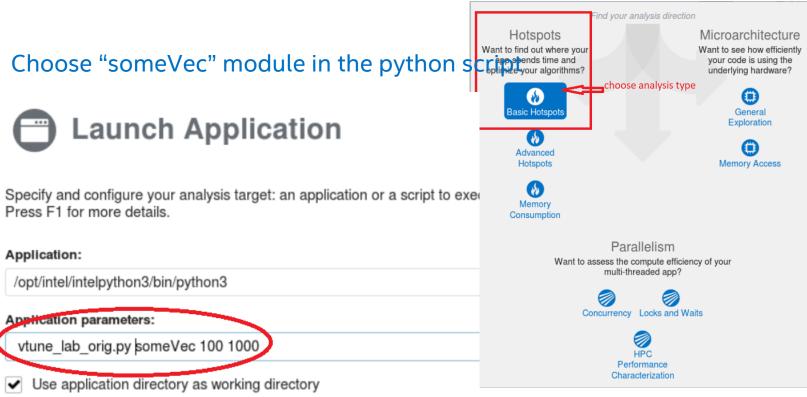




Further vectorization and performance improvement

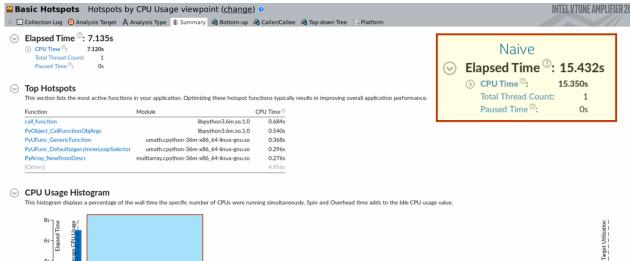


# 5.1 Configuring application to launch:



#### **Optimization Notice**

## 5.2 Updated code: twice as fast.



#### **Optimization Notice**



## 5.3 Top down view to find the issue

•	tspots by CPU U		•					INTEL VTUNE AMP
A Analysis Configuration E C Grouping: Call Stack	ollection Log 👔 Sumr	nary 🐝 Bottom-u	p 🔥 Calle	er/Callee 🐼 Top	-down Tree 🗈	Platform		CPU Time
arouping. Can stack						×		CPUTIme
Function Stack	CPU Time: Total 🔻 🔌	CPU Time: Self	Module	Function (Full)	Source File	Start Address	<b>^</b>	Viewing < 1 of 65 + select
libc_start_main	89.3%	0s	libc.so.6	libc_start		0x20740		10.4% (0.112s of 1
main	89.3%	0s	python	main	python.c	0x400a60		libpython3.6m.so.1.0!Py
	89.3%	0.010s	libpyth	Py_Main	main.c	0x1bb7b0	1	umath.cpython-36m-x86
▼ <module></module>	89.1%	0s	vtune_I	<module></module>	vtune_lab	0x7efd6f48b	1	umath.cpython-36m-x86
	89.1%	0s	libpyth	call_function	ceval.c	0x166370	1	umath.cpython-36m-x86
▼ main	89.1%	0s	vtune_I	main()	vtune_lab	0x7efd6dcf1	1	umath.cpython-36m-x86
	89.1%	0s	libpyth	call function	ceval.c	0x166370	1	<ul> <li>libpython3.6m.so.1.0!cal</li> </ul>
⊤ someVec	89.1%	0.028s	vtune_I	someVec(full	vtune_lab	0x7efd6f433	1	methods.py! mean+0x
	86.6%	0.236s	libpyth	call function	ceval.c	0x166370	1	libpython3.6m.so.1.0!Py
▶ array ad	49.8%	0.020s	multiar	array add		0x3ce0a0	1 🔲	fromnumeric.py!mean+0
nean 🚺	17.6%	0.060s	fromnu	mean(a, axis,	fromnume	0x7efd63a9	í	libpython3.6m.so.1.0!cal
▶ anay_de	8.2%	0.084s	multiar	array dealloc		0x21a40	1	vitune lab orig.pylome
▶ array ite	5.2%	0.020s	multiar	array item		0x3572d0	i 📕	libpython3.6m.so.1.0!cal
⊨ ufunc ge	1.6%	0s	umath	ufunc generi		0x20640	1	vtune lab orig.py!main+
► [Unknow	0.4%	0s		[Unknown sta		0	1	017
▶ PyDict Lo	0.4%	0.024s	libpyth	PyDict Loa	dictobject.c	0xbde70	•	libpython3.6m.so.1.0!cal
							Þ	vtune_lab_orig.py! <mod< td=""></mod<>
0: +				s 3.5s	4s 4.5s	5s 5.5		6s 🔽 Thread
python3 (TID: 26055)			1	1 1		1 11	1	Running

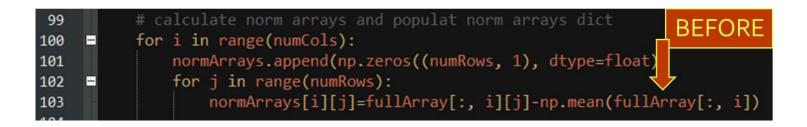
#### **Optimization Notice**

## 5.4 Drill down to the source view

78	#"some vectorization" covariance function		
79	<pre>def someVec(fullArray):</pre>		
80	print('Calculate by Hand Some Vectorization')		
81	<pre>start = time.time()</pre>		
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 corm arrays and populat norm arrays dict		
91	for 1 in range(numCols):		
92	<pre>normArrays.append(np.zeros((numRows, 1), dtype=float))</pre>		-
93	for j in range(numRows):	0.1%	
94	<pre>normArrays[i][j]=fullArray[:, i][j]-np.mean(fullArray[:, i])</pre>	21.0%	
95			
96	# calculate covariance and populat results array		
97	for i in range/numCole).		

#### **Optimization Notice**

## 5.5 Code changes we need to do...



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



## 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\*



### 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!



#### Basic Hotspots Hotspots by CPU Usage viewpoint (<u>change</u>)

🗉 🖭 Collection Log \ominus Analysis Target 🗍 Analysis Type 🔋 Summary 🐼 Bottom-up 🗞 Caller/Callee 🗞 Top-down Tree 🖭 Platform

### ⊘ Elapsed Time<sup>⑦</sup>: 1.394s

```
    Section 2 CPU Time<sup>(1)</sup>: 1.380s
    Total Thread Count: 1
    Paused Time<sup>(1)</sup>: 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
Gl_fetestexcept	libm.so.6	0.164s
PyCFunction_Call	libpython3.6m.so.1.0	0.137s
npy_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

## Naive Elapsed Time<sup>(2)</sup>: 15.432s (3) CPU Time<sup>(0)</sup>: 15.350s Total Thread Count: 1 Paused Time<sup>(0)</sup>: 0s

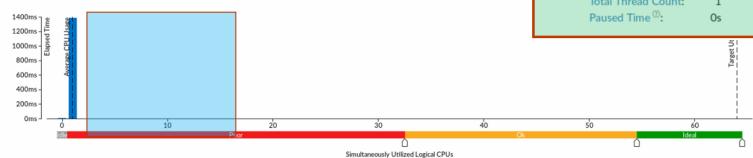
INTEL VTUNE AMPLIFIER 201

### Some Vectorization Elapsed Time<sup>®</sup>: 7.135s CPU Time<sup>®</sup>: 7.120s Total Thread Count: 1

 $\sim$ 

### ⊘ 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 va





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







## Resources

## Get Intel's Distribution for Python:

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

•



