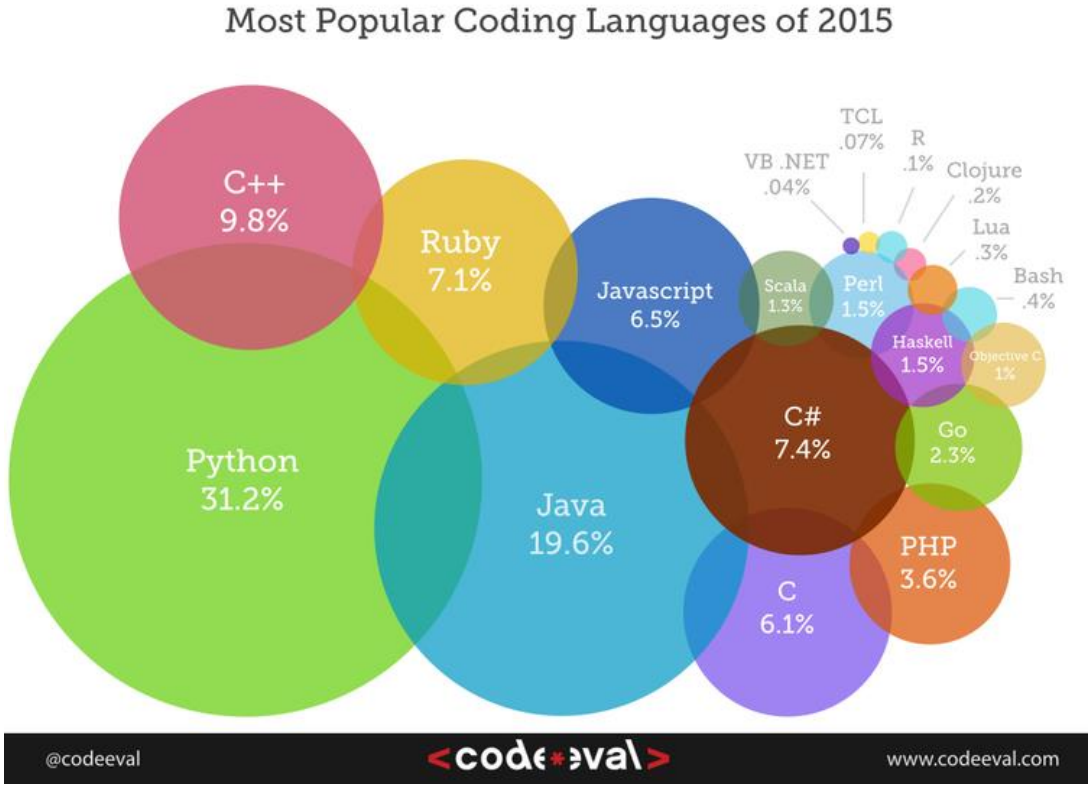
INTEL® VTUNE™ AMPLIFIER XE PYTHON TUTORIAL

Kevin O'Leary – Intel Developer Products Division Technical Consulting Engineer

Programming Languages by Popularity

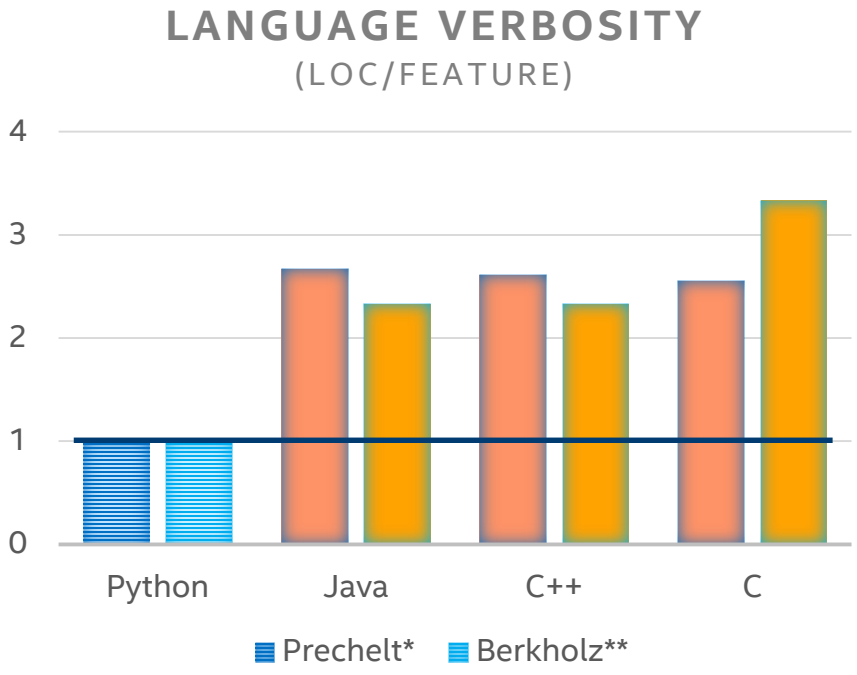
Python remains **#1** programming language in **hiring demand** followed by **Java** and **C++**

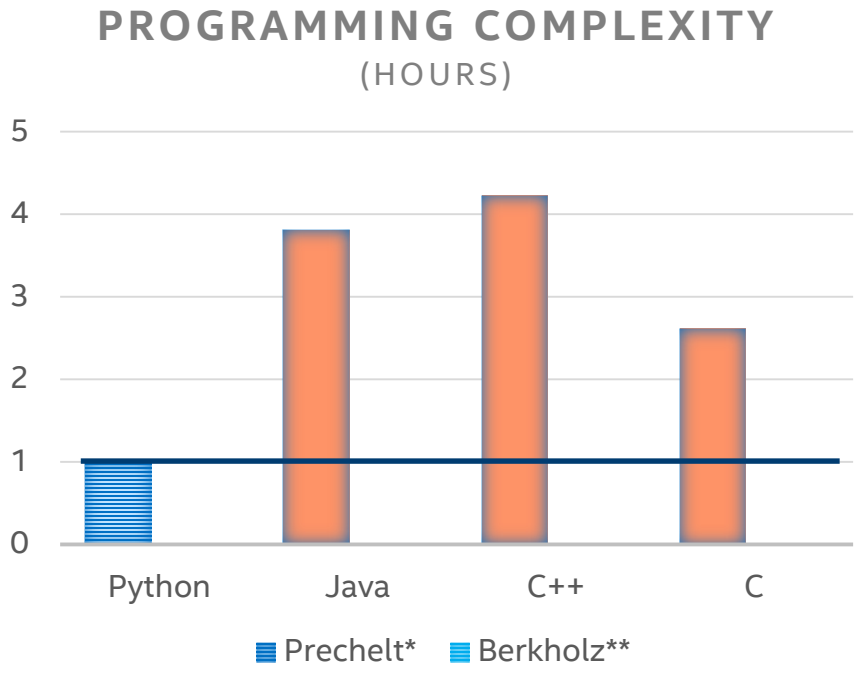
Go and **Scala** demonstrate **strong growth** for last 2 years



* Source: CodeEval, Feb 2015

Programming Languages Productivity





* L.Prechelt, An empirical comparison of seven programming languages, IEEE Computer 2000, Vol. 33, Issue 10, pp. 23-29
** RedMonk - D.Berkholz, Programming languages ranked by expressiveness

Profile Python & Go!

And Mixed Python / C++ / Fortran

New!

Low Overhead Sampling

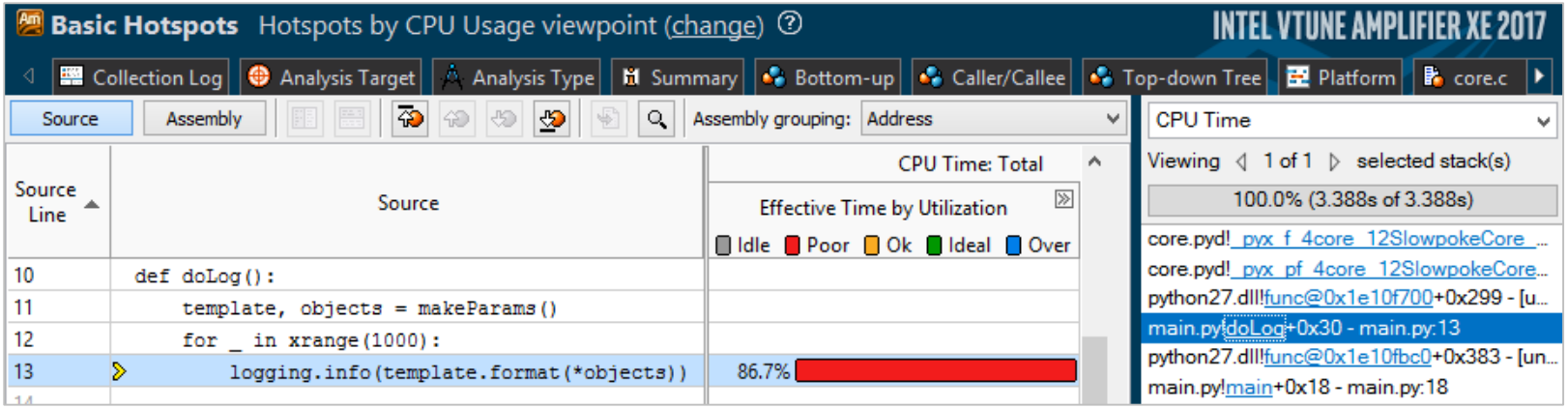
- Accurate performance data without high overhead instrumentation
- Launch application or attach to a running process

Precise Line Level Details

- No guessing, see source line level detail

Mixed Python / native C, C++, Fortran…

- Optimize native code driven by Python



GETTING YOUR PYTHON CODE TO RUN FASTER USING INTEL® VTUNE™ AMPLIFIER XE

Faster, Scalable Code, Faster

Intel® VTune™ Amplifier Performance Profiler

Accurate Data - Low Overhead

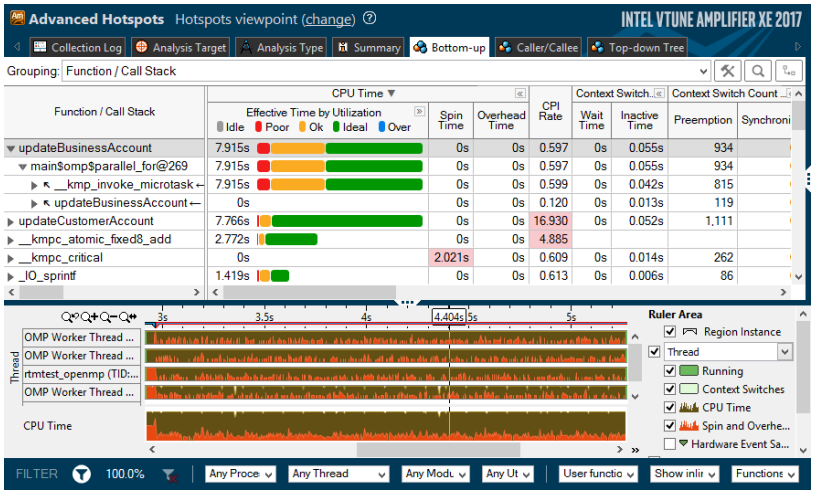
- CPU, GPU, FPU, threading, bandwidth…

Meaningful Analysis

- Threading, OpenMP region efficiency
- Memory access, storage device

Easy

- Data displayed on the source code
- Easy set-up, no special compiles



For Windows* and Linux* From $899
(UI only now available on OS X*)

"Last week, Intel® VTune™ Amplifier helped us find almost 3X performance improvement. This week it helped us improve the performance another 3X."

Claire Cates
Principal Developer
SAS Institute Inc.

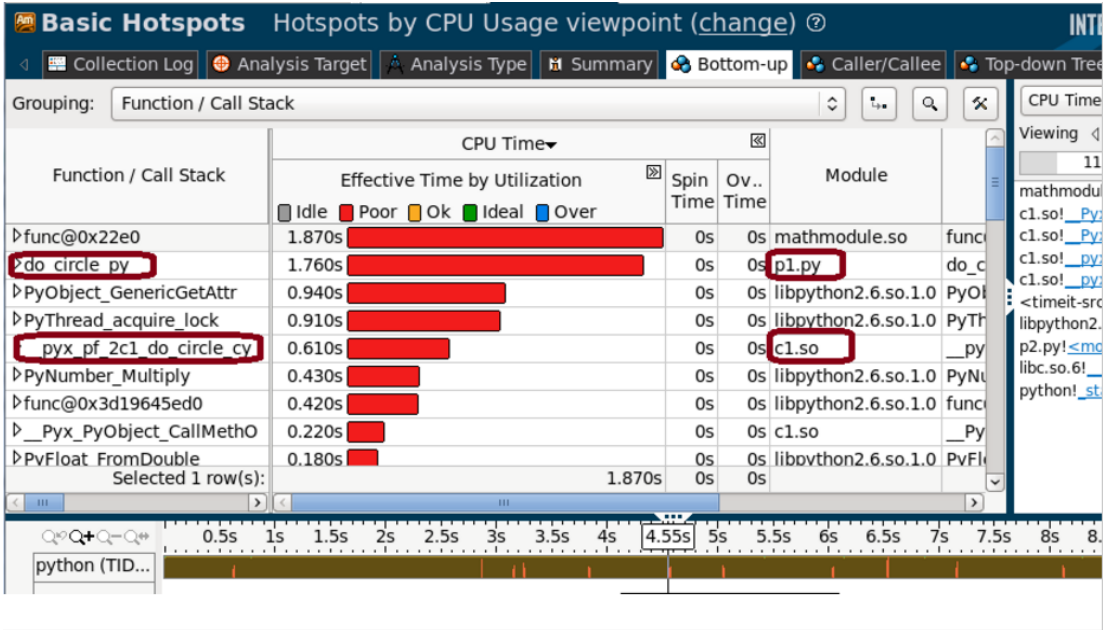
http://intel.ly/vtune-amplifier-xe

Intel® VTune™ Amplifier

Tune Applications for Scalable Multicore Performance

Agenda

- Why python optimization is important
- How do you find places that need optimization
- Overview of profilers
- Profiling Python using VTune Amplifier
- Mixed mode profiling
- Summary



Why do you need Python optimization?

Python is used to power a wide range of software, including those where application performance matters.

- web server code
- complex automation scripts (even build systems)
- scientific calculations, etc.

Python allows you to quickly write code that may not scale well, but you won't know it unless you give it enough workload to process.

How do you find the places that need optimization?

Code examination

- Easiest in terms of you don't need any tools except code editor
- Difficult in practice, also assumes you know how certain code constructs perform
- This might not work for even moderately large code base because there is just too much information to grasp.

How do you find the places that need optimization? (continued)

Logging

- Done by making special version of source code augmented with timestamp logging
- Involves looking at the logs trying to find the part of your code that is slow.
- This analysis can be tedious and it also involves changing your source.

How do you find the places that need optimization? (continued)

Profiling

- Profiling is gathering some metrics on how your application works under certain workloads
- In this paper we will be focused on CPU hotspot profiling. Finding places in your code that consume a lot of CPU cycles.
- In theory you could also profile other interesting cases such as waiting on a lock, memory consumption, etc. (not currently implemented in VTune™ Amplifier)

Overview of existing profilers

There are three basic types of profilers:

- Event

Event based profilers collect data when certain events occur. For example on function entry/exit or when classes are loaded/unloaded, etc. The built-in Python profiler cProfile is an example of an event based profiler.

- Instrumentation

In an instrumentation based profiler the target application is modified and basically the application profiles itself. This can be done by manually modifying the application or by support built inside the compiler.

Overview of existing profilers (contd)

- Statistical

A statistically based profiler samples data at regular intervals. The hottest functions should be at the top of the sample distribution. This type of profiling provides approximate results but are much less intrusive on the target application. Profiling overhead is also much less workload dependent. Intel® VTune™ Amplifier is an example of a statistically based profiler.

Short overview of Python profilers

Tool	Description	Platforms	Profile level	Avg. overhead *
Intel® VTune™ Amplifier	• Rich GUI viewer • Mixed C/C++/Python code	Windows Linux	Line	~1.1-1.6x
cProfile (built-in)	• Text interactive mode: “pstats” (built-in) • GUI viewer: RunSnakeRun (Open Source) • PyCharm	Any	Function	1.3x-5x
Python Tools	• Visual Studio (2010+) • Open Source	Windows	Function	~2x
line_profiler	• Pure Python • Open Source • Text-only viewer	Any	Line	Up to 10x or more

* Measured against Grand Unified Python Benchmark

Machine specs: HP EliteBook 850 G1; Intel® Core™ i5-4300U @1.90 Ghz (4 cores with HT on) CPU; 16 GB RAM; Windows 8.1 x86_64

Profiling Python code using Intel® VTune™ Amplifier XE

Intel® VTune™ Amplifier XE now has the ability to profile Python code. It can give you line level profiling information with very low overhead. Some key features are:

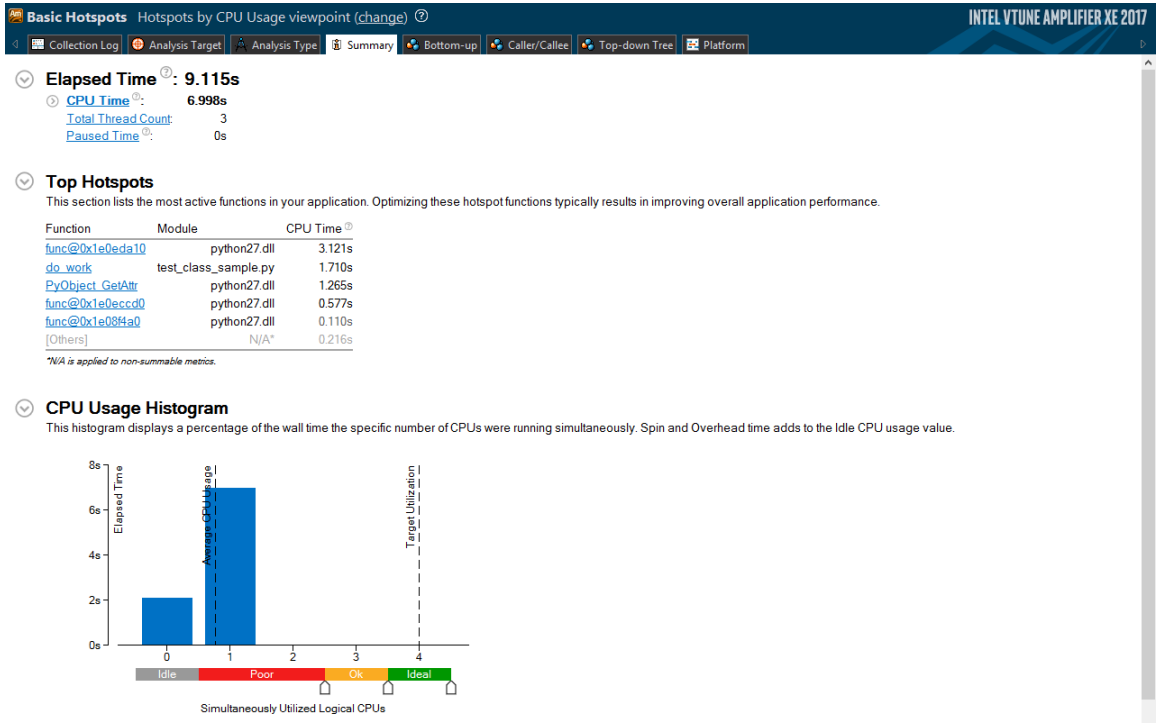
Both Python 32- and 64-bit are supported, 2.7.x and 3.4.x-3.5.x versions

Remote collection via SSH supported

Rich user interface with multithreaded support; zoom & filter; source drill-down

- Supported workflows
  - Start application, wait for it to finish
  - Attach to application, profile, detach

Profiling Python code using Intel® VTune™ Amplifier XE



Steps to analyze

Using Intel VTune™ Amplifier

Create a VTune™ Amplifier project.

Run basic hotspot analysis

Interpret result data

Steps to analyze – Create a Project

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

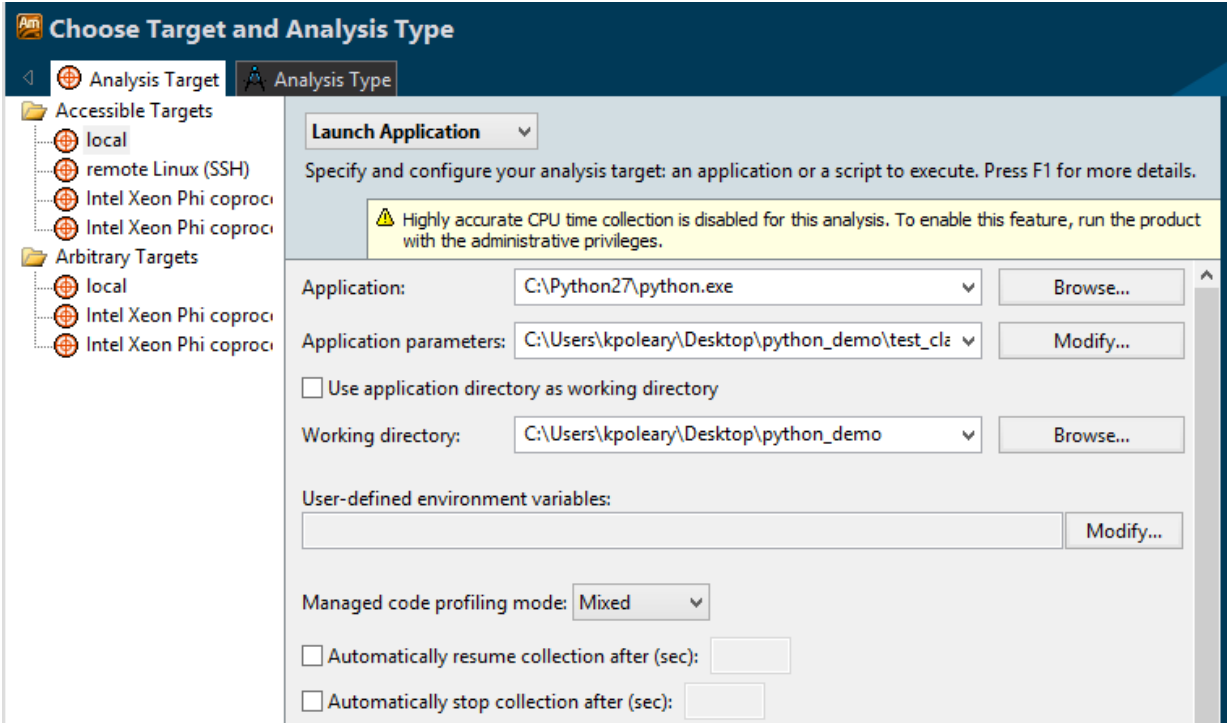
- Run the amplxe-gui that launches the VTune Amplifier GUI
- Click on the menu button select **New->Project**
- Specify the project name test_python

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.

Steps to analyze – Create a Project

- 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.
- 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
  - Use the Managed code profiling mode pull down to specify Mixed

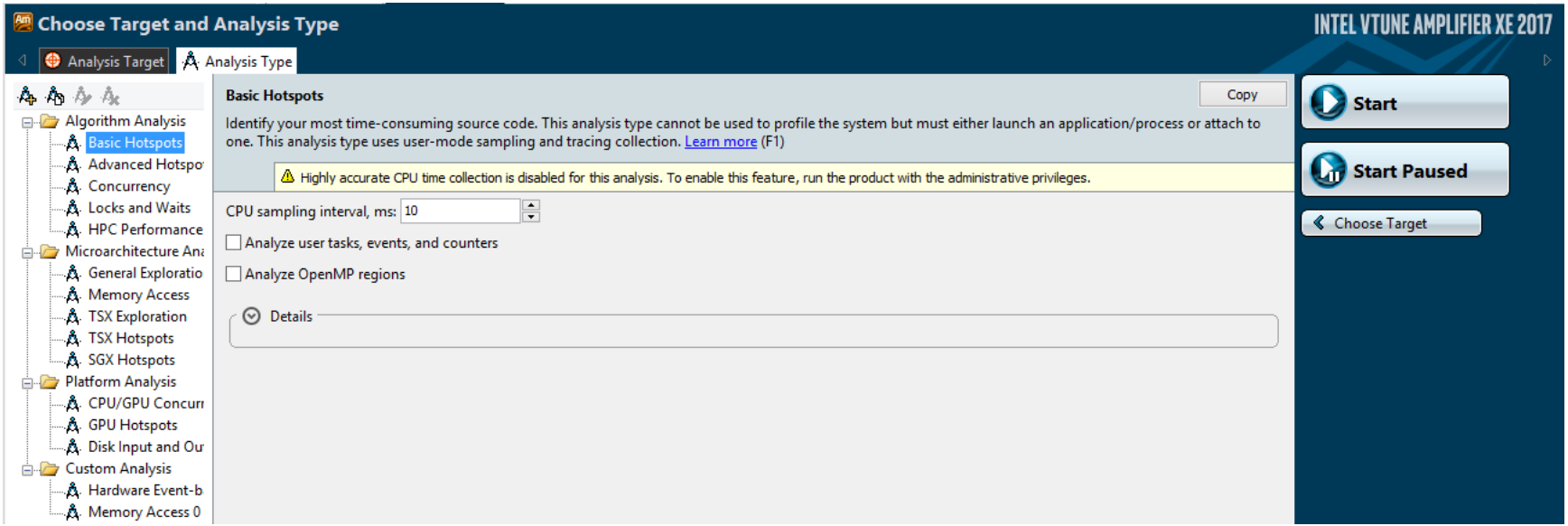
Steps to analyze – Create a Project



Steps to analyze – Run Basic Hotspot analysis

- Click the **Choose Analysis** button on the right to switch to the **Analysis Type** tab
- In the **Choose Target and Analysis Type** window, switch to the **Analysis Type** tab.
- From the analysis tree on the left, select **Algorithm Analysis > Basic Hotspots**
- The right pane is updated with the default options for the Hotspots analysis
- Click the **Start** button on the right command bar to run the analysis.

Steps to analyze – Run Basic Hotspot analysis



Steps to analyze – Interpret Result Data

When the sample application exits, Intel® VTune™ Amplifier finalizes the results and opens the **Hotspots** 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:

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

Steps to analyze – Interpret Result Data

Elapsed Time: 9.538s

- CPU Time: 7.004s
- Total Thread Count: 3
- Paused Time: 0s

Note that **CPU Time** for the sample application is equal to 7.004 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.

Top Hotspots

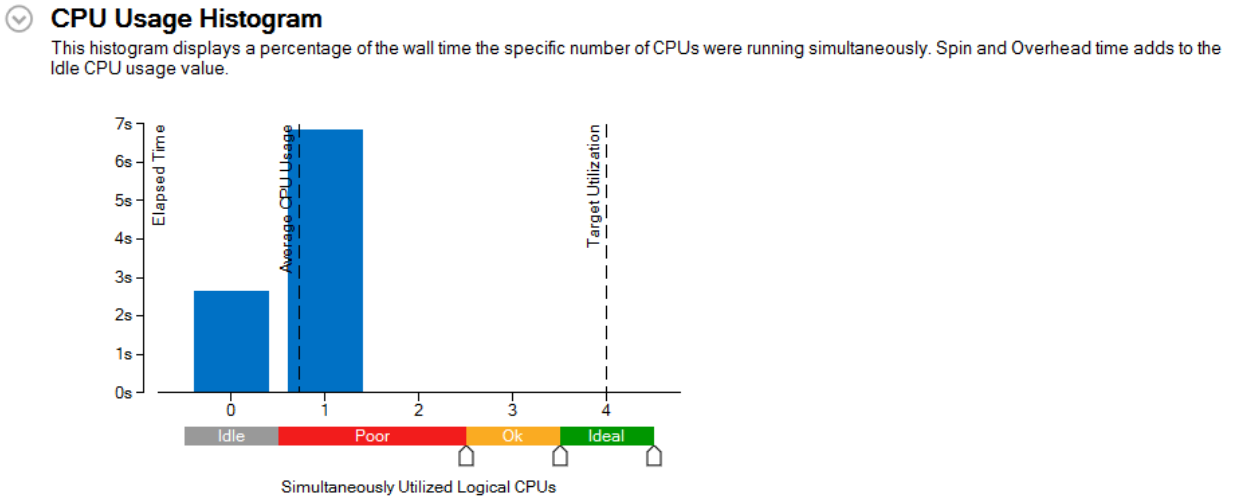
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
func@0x1e0eda10	python27.dll	2.894s
do_work	test_class_sample.py	1.580s
PyObject_GetAttr	python27.dll	1.229s
func@0x1e0eccd0	python27.dll	0.610s
func@0x1e08f4a0	python27.dll	0.280s
[Others]	N/A*	0.411s

*N/A is applied to non-summable metrics.

Steps to analyze – Interpret Result Data

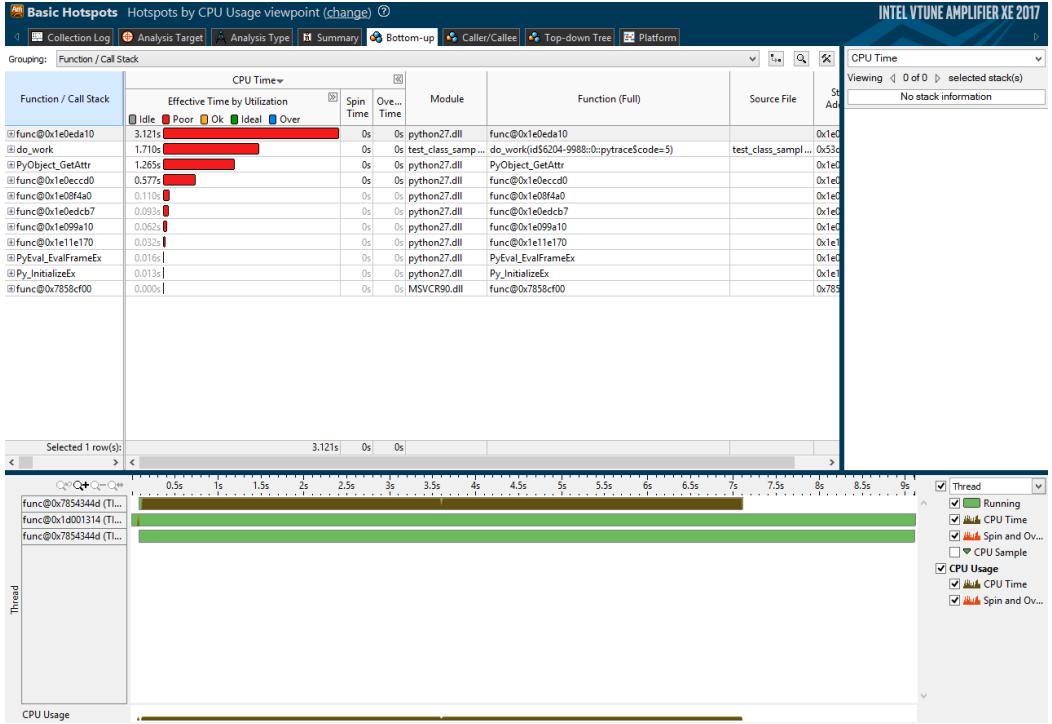
The **CPU Usage Histogram** shows you how well you are utilizing the different cores of your system. It indicates the **Target Utilization** which is the maximum cares available and also the **Average Utilization**. You can use the sliders at the bottom of the graph to set which values you would like to be Ideal, Ok, Poor and Idle.



Steps to analyze – Interpret Result Data

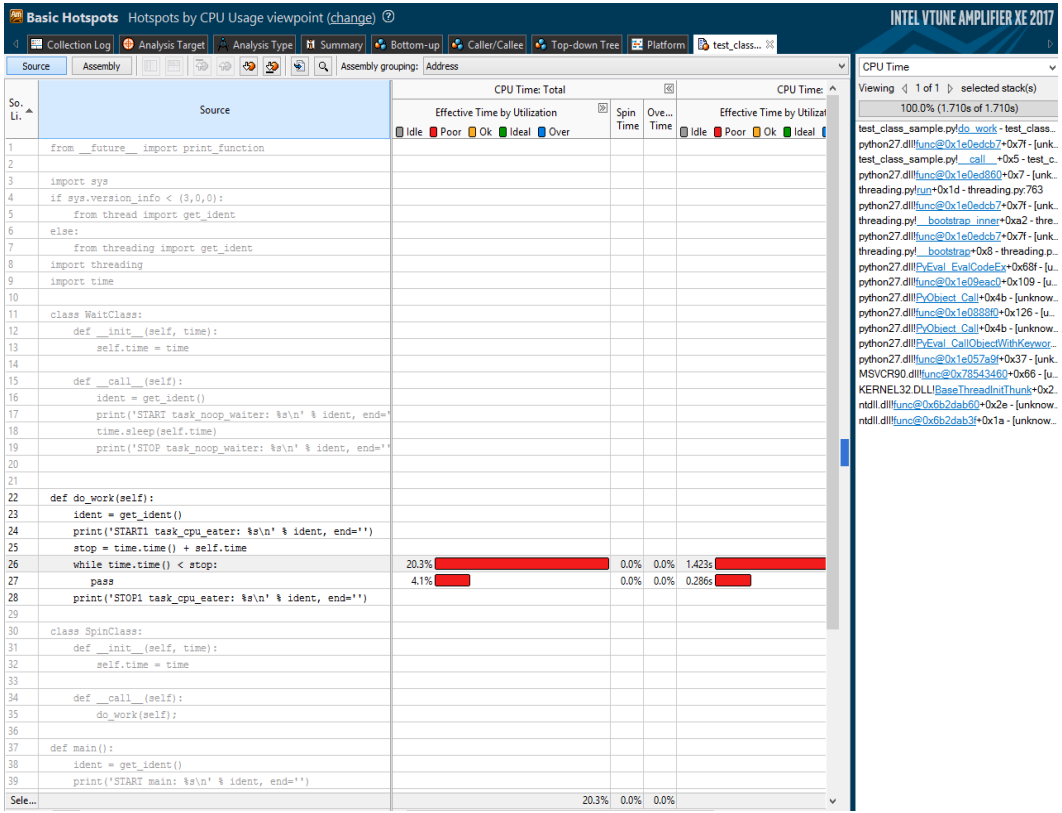
Click on the **Bottom-up** tab. You can see the hottest functions in your application. You can also see the threads in your application and how much CPU time was spent in each thread, you can easily see if your workload is balanced between your threads. In addition, VTune™ Amplifier color codes your effective time by utilization to indicate whether you are utilizing your CPU efficiently.

Steps to analyze – Interpret Result Data



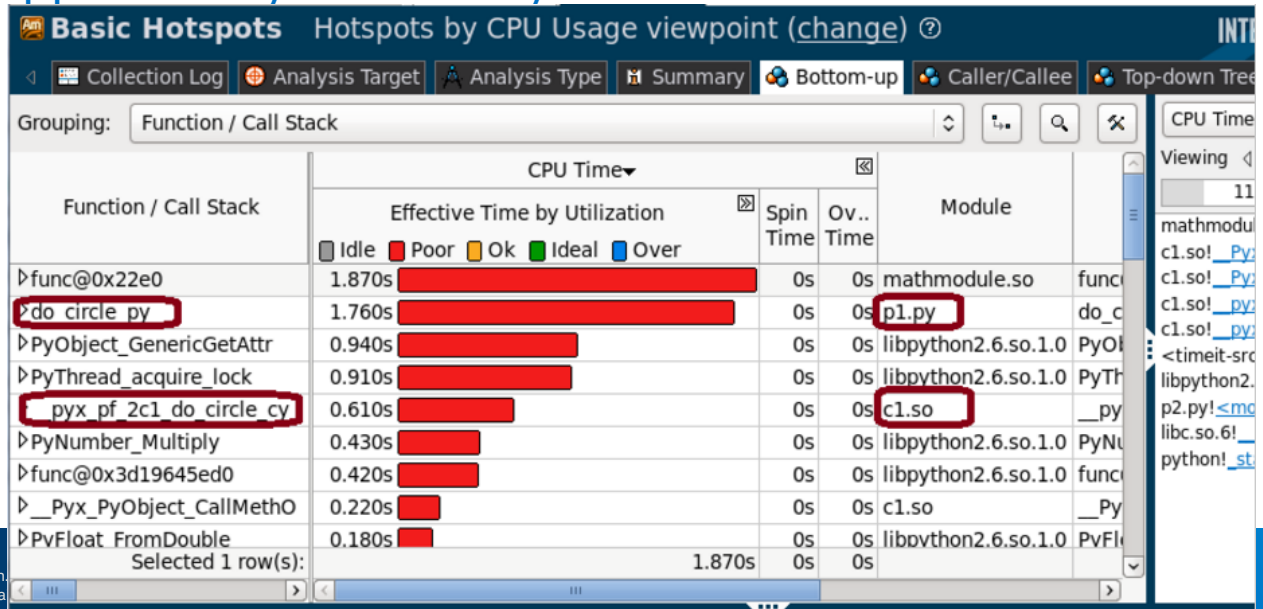
Steps to analyze – Interpret Result Data

Double click on one of your functions, this brings up the source code for your application, You can view how much time you are spending on each line of your application



Running mixed mode analysis

Python is an interpreted language and it does not use a compiler to generate binary execution code. Cython is also an interpreted language but a C-extension, it can be used to generate native code. The VTune™ Amplifier XE 2017 fully supports of Python and Cython code.



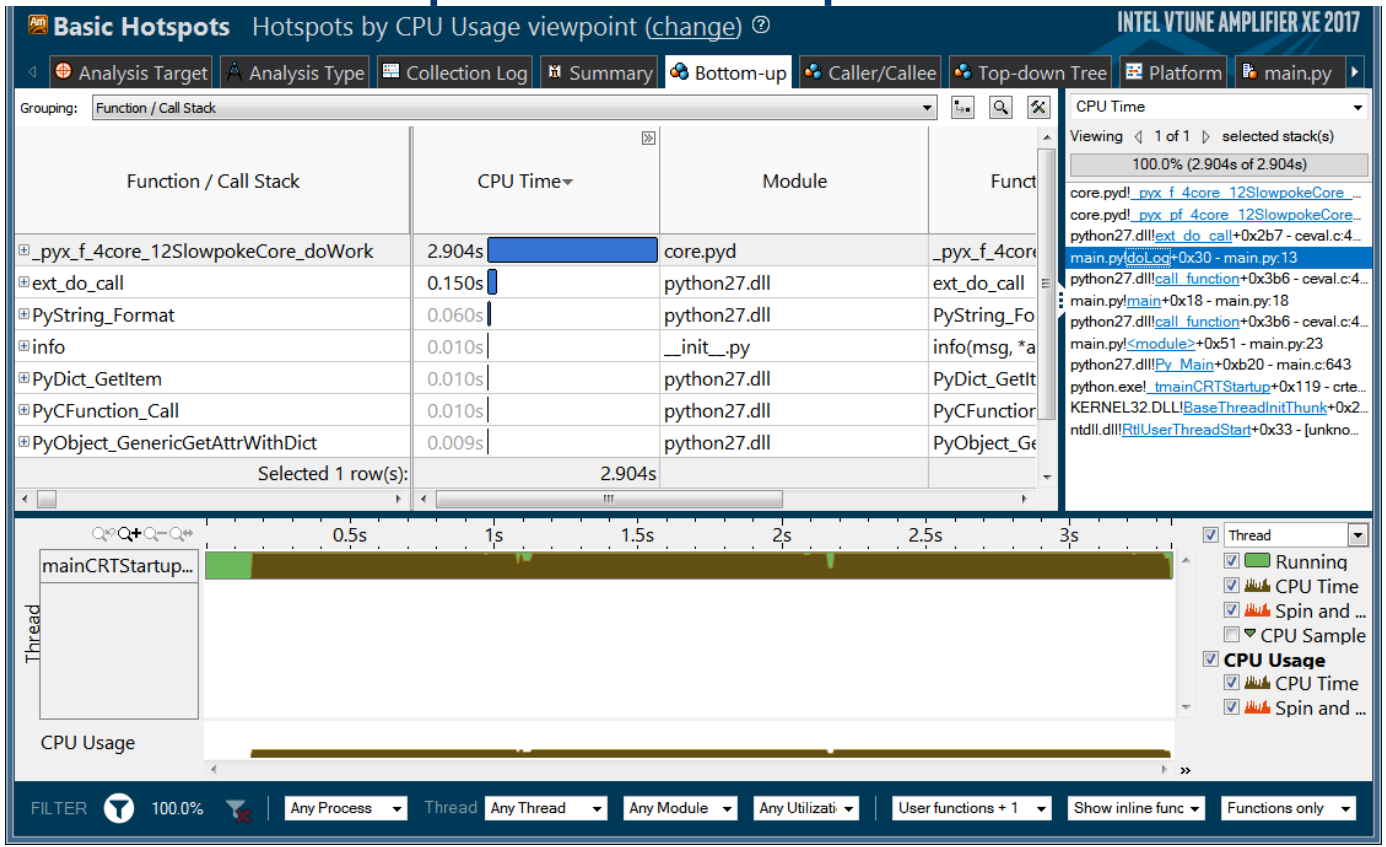
Mixed C/Python example to profile: **core.pyx** (Cython-based)

```
import math
cdef class SlowpokeCore:
    cdef public object N
    def __init__(self, N):
        self.N = N

    cdef double doWork(self, int N) except *:
        cdef int i, j, k
        cdef double res
        res = 0
        for j in range(N):
            k = 0
            for i in range(N):
                k += 1
            res += k
        return math.log(res)

    def __str__(self):
        return 'SlowpokeCore: %f' % self.doWork(self.N)
```

Mixed C/Python example to profile: **main.py**

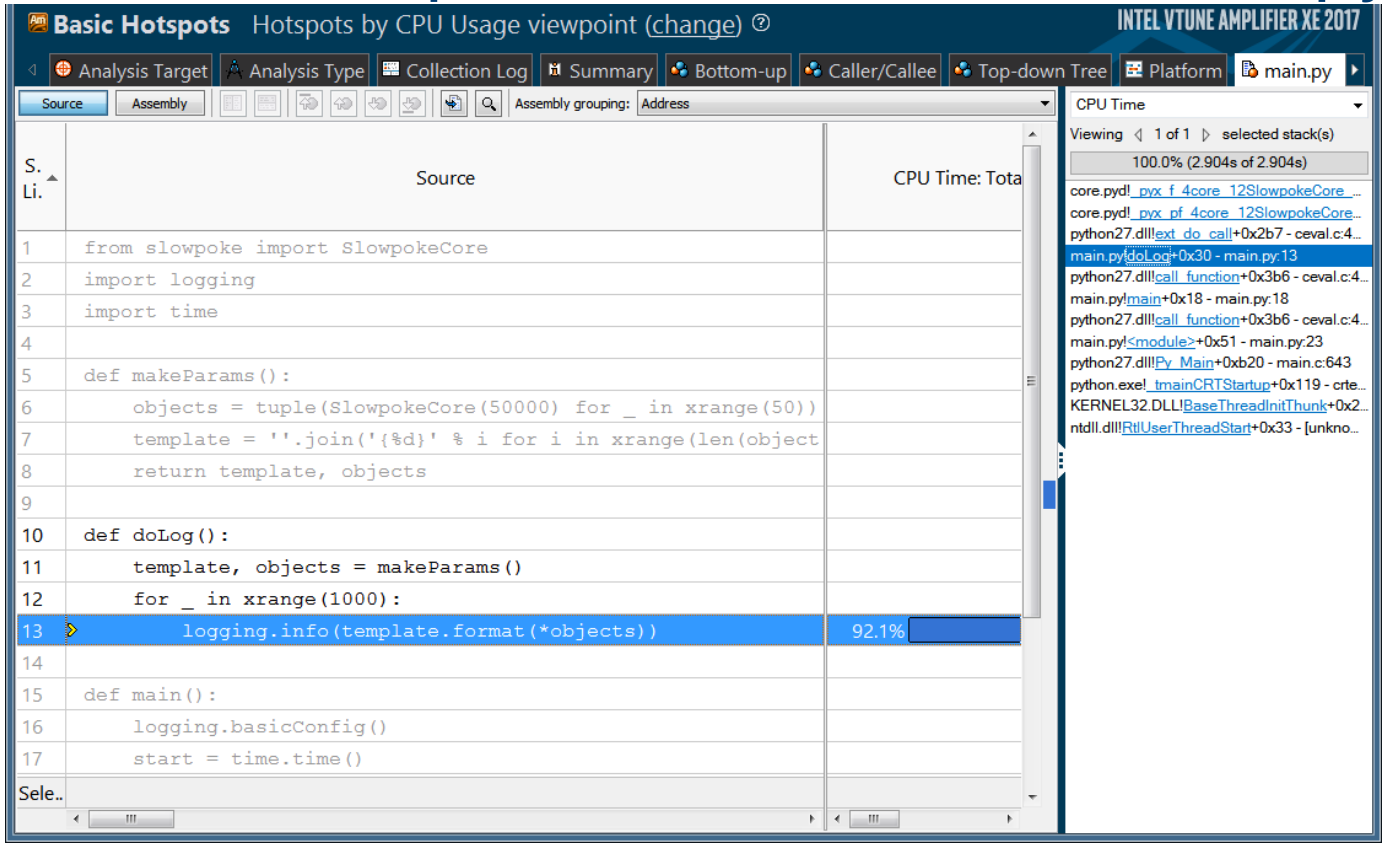
```
from slowpoke import SlowpokeCore
import logging
import time

def makeParams():
    objects = tuple(SlowpokeCore(50000) for _ in xrange(50))
    template = ''.join('{%d}' % i for i in xrange(len(objects)))
    return template, objects

def calc_pi():
   # removed for readability; pure-Python function was here

def doLog():
    template, objects = makeParams()
    for _ in xrange(1000):
        calc_pi()
        logging.info(template.format(*objects))

def main():
    logging.basicConfig()
    start = time.time()
    doLog()
    stop = time.time()
    print('run took: %.3f' % (stop - start))

if __name__ == '__main__':
    main()
```

Intel® VTune™ Amplifier example



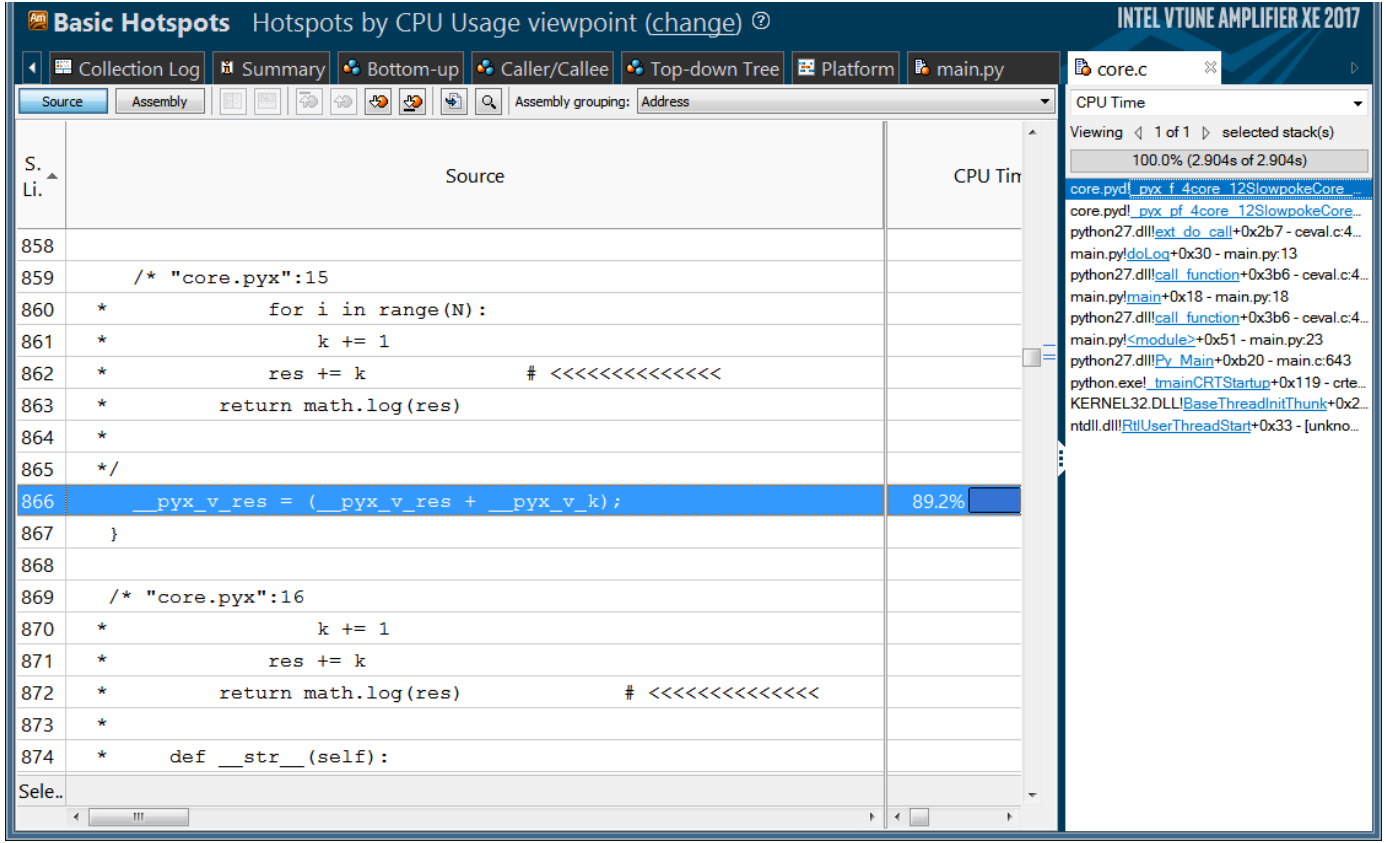
Machine specs: HP EliteBook 850 G1; Intel® Core™ i5-4300U @1.90 Ghz (4 cores with HT on) CPU; 16 GB RAM; Windows 8.1 x86_64

Intel® VTune™ Amplifier – source view (main.py)



Machine specs: HP EliteBook 850 G1; Intel® Core™ i5-4300U @1.90 Ghz (4 cores with HT on) CPU; 16 GB RAM; Windows 8.1 x86_64

Intel® VTune™ Amplifier – source view (core.c)



Machine specs: HP EliteBook 850 G1; Intel® Core™ i5-4300U @1.90 Ghz (4 cores with HT on) CPU; 16 GB RAM; Windows 8.1 x86_64

Summary

Tuning can dramatically increase the performance of your code. Intel® VTune™ Amplifier XE now has the ability to profile Python code. You can also analyze mixed Python/C code as well as pure Python. VTune™ Amplifier has a powerful set of features that will allow you to quickly identify your performance bottlenecks.

Call to action

Get Intel® Parallel Studio XE 2017 and start profiling your Python code today!

https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2017

Legal Disclaimer & Optimization Notice

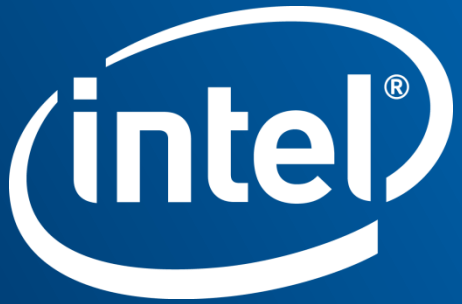
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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.

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


intel®