



Software

# Profiling your PYTHON application with Intel® Vtune™ Amplifier

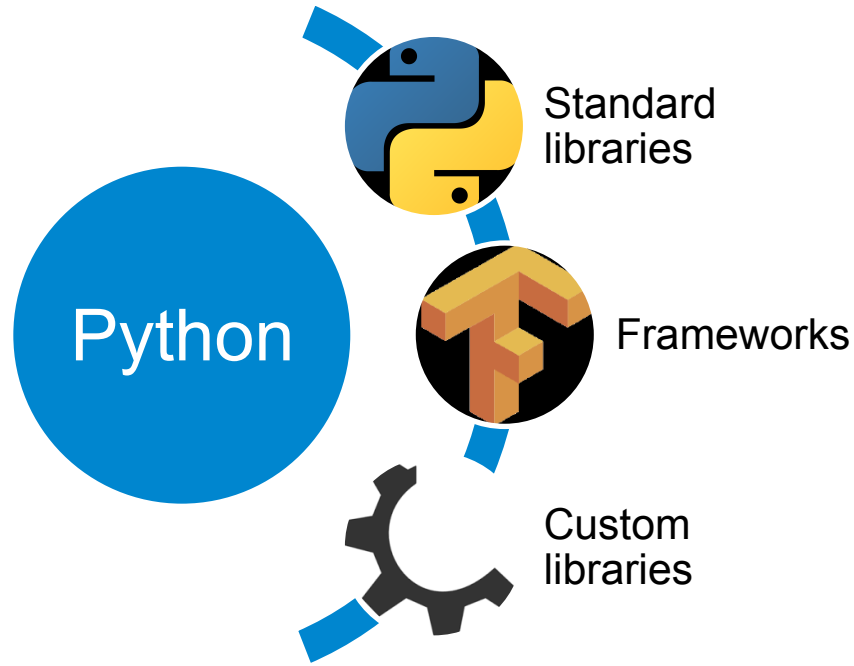
Paulius Velesko

Application Engineer, Intel Corporation

# High Performance Python

- Rapid prototyping
- Strong user base
- Lots of libraries

# 3 Types of Python Workloads



# Intel® VTune™ Amplifier

VTune Amplifier is a full system profiler

- Accurate
- Low overhead
- Comprehensive ( microarchitecture, memory, IO, treading, ... )
- Highly customizable interface
- Direct access to source code and assembly

Analyzing code access to shared resources is critical to achieve good performance on multicore and manycore systems

VTune Amplifier takes over where Intel® Advisor left

# Predefined Collections

Many available analysis types:

- advanced-hotspots Advanced Hotspots
  - concurrency Concurrency
  - disk-io Disk Input and Output
  - general-exploration General microarchitecture exploration
  - gpu-hotspots GPU Hotspots
  - gpu-profiling GPU In-kernel Profiling
  - hotspots Basic Hotspots →
  - hpc-performance HPC Performance Characterization
  - locksandwaits Locks and Waits →
  - memory-access Memory Access
  - memory-consumption Memory Consumption →
  - system-overview System Overview
  - ...
- Python Support
- 
- A diagram showing a list of predefined analysis collections on the left. On the right, a vertical line is labeled 'Python Support'. Three horizontal arrows point from the 'hotspots', 'locksandwaits', and 'memory-consumption' items to this vertical line.

# Vtune™ Syntax

`source /soft/compilers/intel/vtune_amplifier/amplxe-vars.sh` <- sets up the PATH

`amplxe-cl <action> <action options> -- <application>`

`amplxe-cl -c hotspots -- python foo.py`

`amplxe-cl -h collect`

`amplxe-cl -h collect hotspots`

<https://software.intel.com/en-us/vtune-amplifier-help-amplxe-cl-command-syntax>

# Running an analysis

- The “application” should be the full path to the python interpreter used
- The python code should be passed as “arguments” to the “application”

```
$: amplxe-cl -c hotspots --  
python mycode.py 10000
```

```
$: amplxe-gui &
```

## mycode.py

```
import numpy as np  
n = sys.argv[1]  
arr1 = np.random.rand(n, n)  
arr2 = np.random.rand(n, n)  
  
arr3 = arr1 * arr2  
arr3 = np.log(arr3)  
arr3 = np.exp(arr3)
```

# Summary

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

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

## Elapsed Time: 24.331s

CPU Time: 24.039s  
Total Thread Count: 1  
Paused Time: 0s

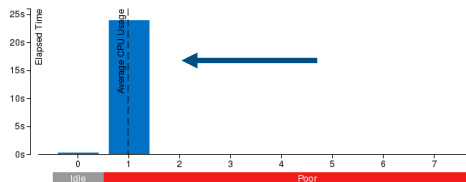
## Top Hotspots

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

| Function                    | Module    | CPU Time |
|-----------------------------|-----------|----------|
| exp                         | libm.so.6 | 8.749s   |
| __ieee754_log_avx           | libm.so.6 | 7.780s   |
| PyUFunc_d_d                 | umath.so  | 3.671s   |
| rk_random                   | mtrand.so | 1.150s   |
| __pyx_f_6mtrand_cont0_array | mtrand.so | 0.930s   |
| [Others]                    |           | 1.759s   |

## CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneou



## Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: python ./workload.py  
Operating System: 3.10.0-693.11.6.el7.x86\_64 NAME="Red Hat Enterprise Linux Server" VERSION="7.4 (Maipo)" ID="rhel" ID\_LIKE="fedora" VARIANT="Server" VARIANT\_ID="server" VERSION\_ID="7.4" PRETTY\_NAME="Red Hat Enterprise Linux Server 7.4 (Maipo)" ANSI\_COLOR="0;31" REDHAT\_SUPPORT\_PRODUCT="Red Hat Enterprise Linux" REDHAT\_SUPPORT\_PRODUCT\_VERSION="7.4"  
jlselogn2  
Computer Name: jlselogn2  
Result Size: 4 MB  
Collection start time: 14:10:40 12/06/2018 UTC  
Collection stop time: 14:11:04 12/06/2018 UTC  
Collector Type: User-mode sampling and tracing

CPU  
Name: Intel(R) Xeon(R) E5 Processor code named Jaketown  
Frequency: 2.2 GHz  
Logical CPU Count: 16

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

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

| Function | CPU Time: Total | CPU Time: Self | Module    | Callers | CPU Time: Total | CPU Time: Self |
|----------|-----------------|----------------|-----------|---------|-----------------|----------------|
| __start  | 100.0%          | 0s             | python2.7 | __start | 100.0%          | 7.780s         |

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

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

Grouping: Call Stack

| Function Stack                          | CPU Time: Total | CPU Time: Self | Module              | Source File | Function (Full)                  | Start Address |
|---|-----------------|----------------|---------------------|-------------|----------------------------------|---------------|
| ▼ Total                                 | 100.0%          | 0s             |                     |             |                                  |               |
| ▼ __start                               | 100.0%          | 0s             | python2.7           |             | __start                          | 0x856         |
| ▼ __libc_start_main                     | 100.0%          | 0.010s         | libc.so.6           |             | __libc_start_main                | 0x21b10       |
| ▼ <module>                              | 99.7%           | 0s             | workload.py         | workload.py | <module>                         | 0x71859eb...  |
| ▼ PyObject_Call                         | 84.9%           | 0s             | libpython2.7.so.1.0 |             | PyObject_Call                    | 0x48730       |
| ▼ ufunc_generic_call                    | 84.9%           | 0s             | umath.so            |             | ufunc_generic_call               | 0xe691a0      |
| ▼ PyUFunc_GenericFunction               | 84.9%           | 0s             | umath.so            |             | PyUFunc_GenericFunction          | 0xe52b0       |
| ▼ trivial_two_operand_loop              | 84.8%           | 0s             | umath.so            |             | trivial_two_operand_loop         | 0xd9e960      |
| ▼ PyUFunc_d_d                           | 84.4%           | 3.671s         | umath.so            |             | PyUFunc_d_d                      | 0x1b1a0       |
| ▼ exp                                   | 38.4%           | 8.749s         | libm.so.6           |             | exp                              | 0x25b10       |
| ▼ __ieee754_log_avx                     | 32.4%           | 7.780s         | libm.so.6           |             | __ieee754_log_avx                | 0x55a00       |
| ▼ log                                   | 0.4%            | 0.100s         | libm.so.6           |             | log                              | 0x26330       |
| ▶ [Unknown stack frame(s)]              | 0.3%            | 0s             |                     |             | [Unknown stack frame(s)]         | 0             |
| ▶ [Unknown stack frame(s)]              | 0.2%            | 0s             |                     |             | [Unknown stack frame(s)]         | 0             |
| ▶ __pyx_pw_6mtrand_11RandomState_29rand | 12.3%           | 0s             | mtrand.so           |             | __pyx_pw_6mtrand_11RandomStat... | 0xd070        |
| ▶ PyNumber_Multiply                     | 1.7%            | 0s             | libpython2.7.so.1.0 |             | PyNumber_Multiply                | 0x46b10       |
| ▶ PyEval_CallObjectWithKeywords         | 0.8%            | 0s             | libpython2.7.so.1.0 |             | PyEval_CallObjectWithKeywords    | 0xda690       |
| ▶ [Unknown stack frame(s)]              | 0.2%            | 0s             |                     |             | [Unknown stack frame(s)]         | 0             |

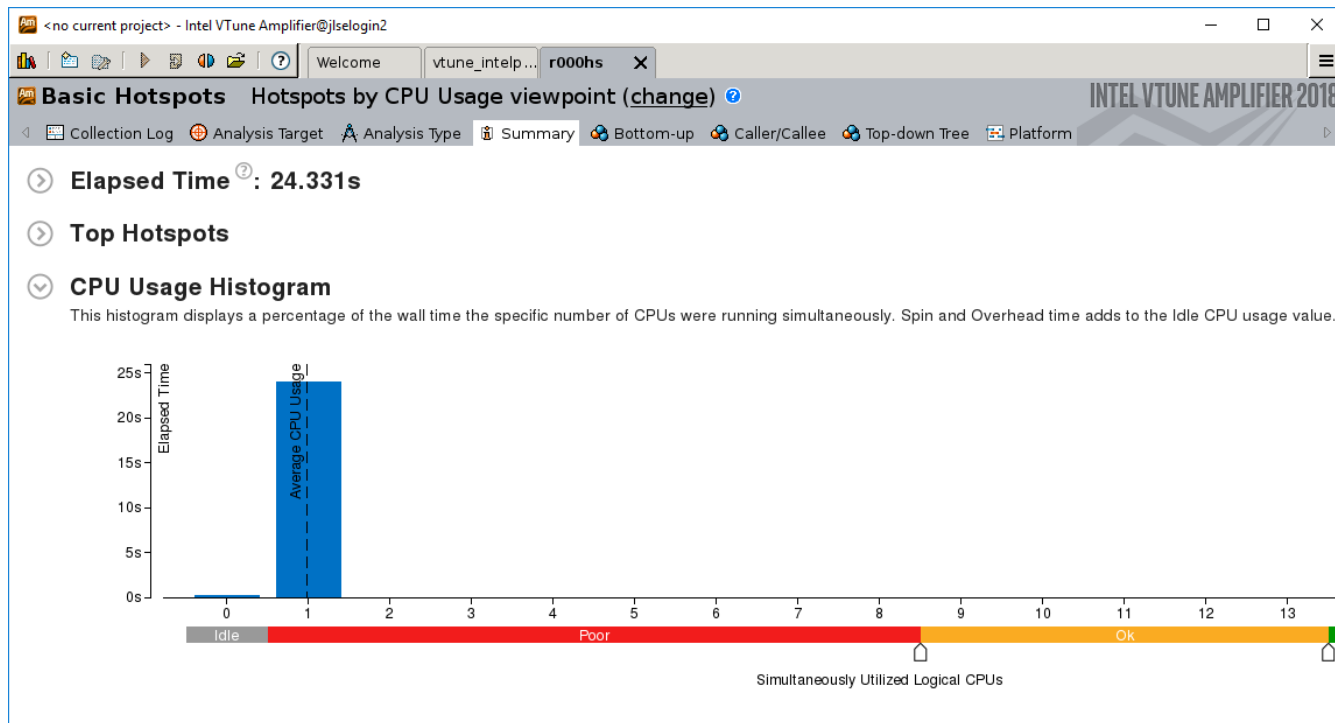
| Function                   | CPU Time: Total | CPU Time: Self | Module          |
|----------------------------|-----------------|----------------|-----------------|
| numpy_log                  | 0.5%            | 0.120s         | umath.so        |
| log                        | 0.4%            | 0.100s         | libm.so.6       |
| [Outside any known module] | 0.2%            | 0.059s         |                 |
| <module>                   | 0.2%            | 0s             | <module>        |
| memmove                    | 0.1%            | 0.030s         | libc-dynamic.so |
| memcpy                     | 0.1%            | 0s             | libc-dynamic.so |

Caller/Callee

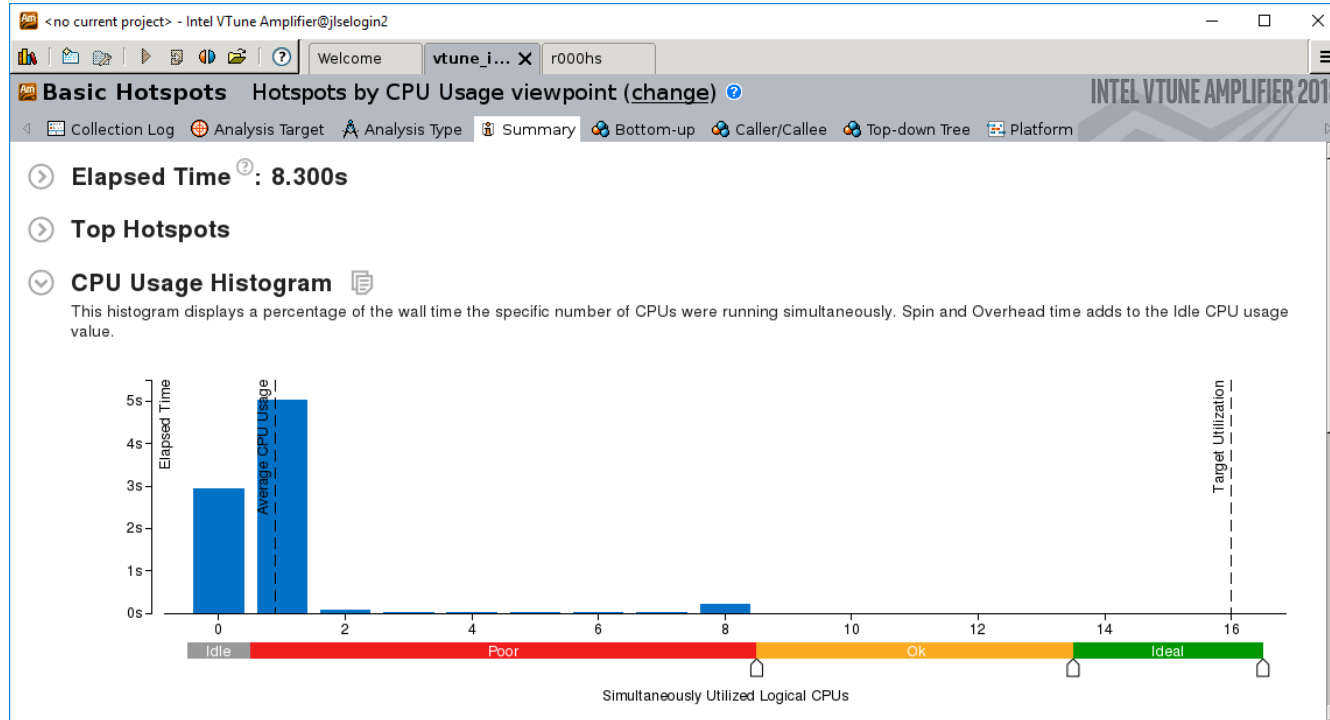
| Caller            | CPU Time: Total | CPU Time: Self |
|-------------------|-----------------|----------------|
| __ieee754_log_avx | 100.0%          | 7.780s         |



# Summary of Anaconda Python



# Summary of Intel® Python



# Bottom-up View

The screenshot displays the Intel VTune Amplifier 2018 interface in the Bottom-up View. The main window shows a table of hotspots by CPU usage, with the following data:

| Function / Call Stack       | Effective Time by Utilization | Spin Time | Overhead Time | Module    |
|-----------------------------|-------------------------------|-----------|---------------|-----------|
| exp                         | 8.749s                        | 0s        | 0s            | libm.so.6 |
| __ieee754_log_avx           | 7.780s                        | 0s        | 0s            | libm.so.6 |
| PyUFunc_d_d                 | 3.671s                        | 0s        | 0s            | umath.so  |
| rk_random                   | 1.150s                        | 0s        | 0s            | mtrand.so |
| __pyx_f_6mtrand_cont0_array | 0.930s                        | 0s        | 0s            | mtrand.so |
| rk_double                   | 0.870s                        | 0s        | 0s            | mtrand.so |
| sse2_binary_multiply_DOUBLE | 0.400s                        | 0s        | 0s            | umath.so  |
| npv_log                     | 0.120s                        | 0s        | 0s            | umath.so  |

Below the table is a thread view for 'python (TID: 8426)' showing CPU usage over time. The filter bar at the bottom includes the following options:

- FILTER: 100.0%
- Any Process
- Any Thread
- Any Module
- Any Utilizé
- User functions +
- Show inline fui
- Functions only

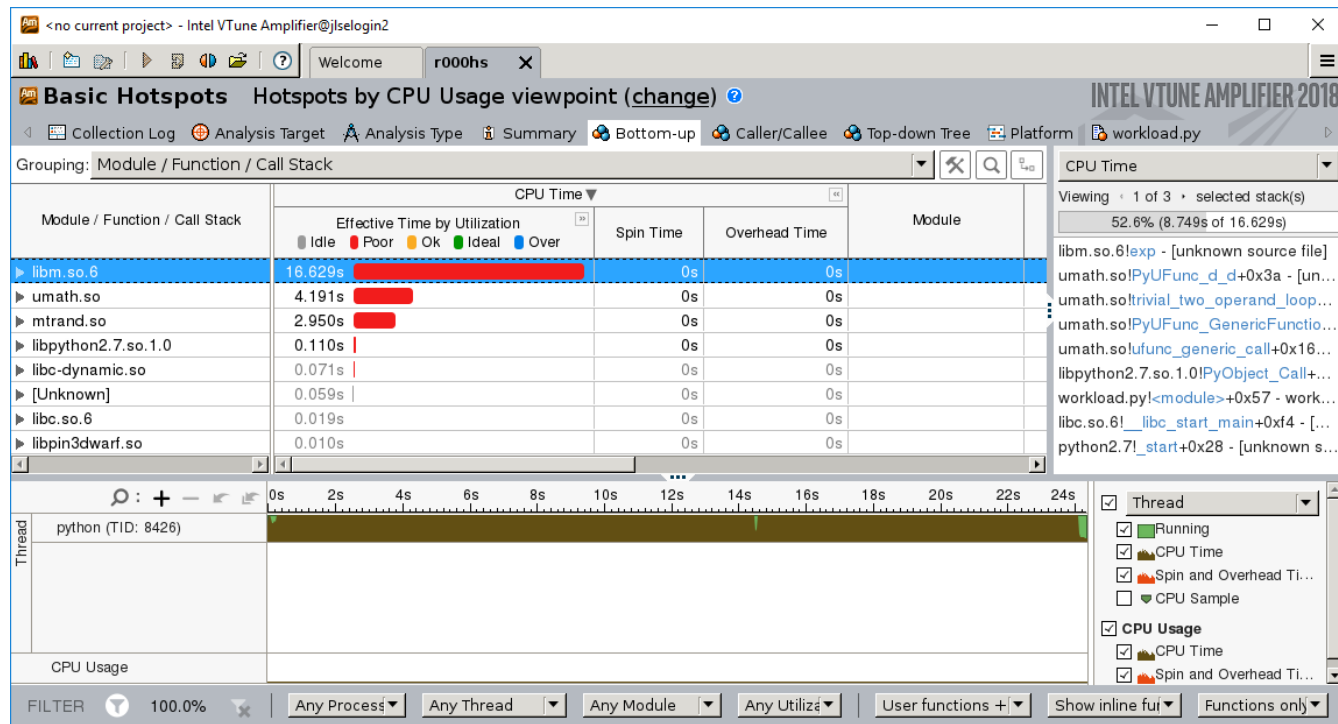
Blue arrows point to the following UI elements:

- Grouping: Function / Call Stack
- Function / Call Stack column header
- Effective Time by Utilization column header
- Filter bar options: Any Process, Any Thread, Any Module, Any Utilizé, User functions +, Show inline fui, Functions only

# Bottom-up View

The screenshot displays the Intel VTune Amplifier 2018 interface in the 'Bottom-up' view. The main window title is '<no current project> - Intel VTune Amplifier@jlselgin2'. The toolbar includes 'Welcome', 'r000hs', and a menu icon. The main title bar reads 'Basic Hotspots Hotspots by CPU Usage viewpoint (change)'. Below this, navigation tabs include 'Collection Log', 'Analysis Target', 'Analysis Type', 'Summary', 'Bottom-up', 'Caller/Callee', 'Top-down Tree', 'Platform', and 'workload.py'. The 'Grouping' dropdown is set to 'Function / Call Stack'. The central pane shows a tree view of hotspots, with 'Function / Call Stack' selected. The right pane shows 'CPU Time' for the selected stack, indicating 100.0% (8.749s of 8.749s). Below this, a list of stack frames is visible, including 'libm.so.6!exp', 'umath.so!PyUFunc\_d+0x3a', 'umath.so!trivial\_two\_operand\_loop...', 'umath.so!PyUFunc\_GenericFuncio...', 'umath.so!ufunc\_generic\_call+0x16...', 'libpython2.7.so.1.0!PyObject\_Call+', 'workload.py!<module>+0x57', 'libc.so.6!\_\_libc\_start\_main+0xf4', and 'python2.7!\_start+0x28'. The bottom of the interface features a 'FILTER' section with '100.0%' and various dropdown menus for 'Any Process', 'Any Thread', 'Any Module', 'Any Utilizã', 'User functions +', 'Show inline fu', and 'Functions only'.

# Bottom-up View



# Top-down View

The screenshot displays the Intel VTune Amplifier 2018 interface. The main window shows a 'Hotspots by CPU Usage' view for a file named 'workload.py'. The table below summarizes the data shown in the main view.

| Source Line | Source                              | CPU Time: Total | CPU Time: Self | Source File |
|-------------|-------------------------------------|-----------------|----------------|-------------|
| 1           | #!/usr/bin/env python               |                 |                |             |
| 3           | import numpy as np                  | 0.8%            | 0s             | workload.py |
| 5           | arr1 = np.random.rand(10000, 10000) | 12.3%           | 0s             | workload.py |
| 6           | arr2 = np.random.rand(10000, 10000) |                 |                |             |
| 8           | arr3 = arr1 * arr2                  | 1.7%            | 0s             | workload.py |
| 9           | arr3 = np.log(arr3)                 | 45.2%           | 0s             | workload.py |
| 10          | arr3 = np.exp(arr3)                 | 39.7%           | 0s             | workload.py |
| 12          | print("Complete.")                  |                 |                |             |

The right-hand pane shows a stack trace for the selected hotspot, listing various system and library frames such as 'libpython2.7.so.1.0|PyObject\_Call+', 'umath.soltrivial\_two\_operand\_loop...', and 'python2.7!\_start+0x28'.

# Caller/Callee

The screenshot displays the Intel VTune Amplifier 2018 interface. The main window shows the 'Basic Hotspots' view for a workload named 'r000hs'. The 'Caller/Callee' view is selected, showing a call graph for the function 'PyUFunc\_d\_d'. The main table lists the following functions and their CPU usage:

| Function                   | CPU Time: Total | CPU Time: Self | Module              |
|----------------------------|-----------------|----------------|---------------------|
| _start                     | 100.0%          | 0s             | python2.7           |
| __libc_start_main          | 100.0%          | 0.010s         | libc.so.6           |
| <module>                   | 99.7%           | 0s             | workload.py         |
| PyUFunc_GenericFunction    | 86.6%           | 0s             | umath.so            |
| ufunc_generic_call         | 86.6%           | 0s             | umath.so            |
| PyObject_Call              | 84.9%           | 0s             | libpython2.7.so.1.0 |
| trivial_two_operand_loop   | 84.8%           | 0s             | umath.so            |
| PyUFunc_d_d                | 84.4%           | 3.671s         | umath.so            |
| exp                        | 36.4%           | 8.749s         | libm.so.6           |
| __ieee754_log_avx          | 32.4%           | 7.780s         | libm.so.6           |
| __pyx_pw_6mtrand_11Rar     | 12.3%           | 0s             | mtrand.so           |
| __pyx_f_6mtrand_cont0_at   | 12.3%           | 0.930s         | mtrand.so           |
| __pyx_pw_6mtrand_11Rar     | 12.3%           | 0s             | mtrand.so           |
| rk_double                  | 8.4%            | 0.870s         | mtrand.so           |
| rk_random                  | 4.8%            | 1.150s         | mtrand.so           |
| PyObject_CallFunctionObjA  | 1.7%            | 0s             | libpython2.7.so.1.0 |
| trivial_three_operand_loop | 1.7%            | 0s             | umath.so            |
| sse2_binary_multiply_DOU   | 1.7%            | 0.400s         | umath.so            |
| PyNumber_Multiply          | 1.7%            | 0s             | libpython2.7.so.1.0 |
| array_multiply             | 1.7%            | 0s             | multiarray.so       |

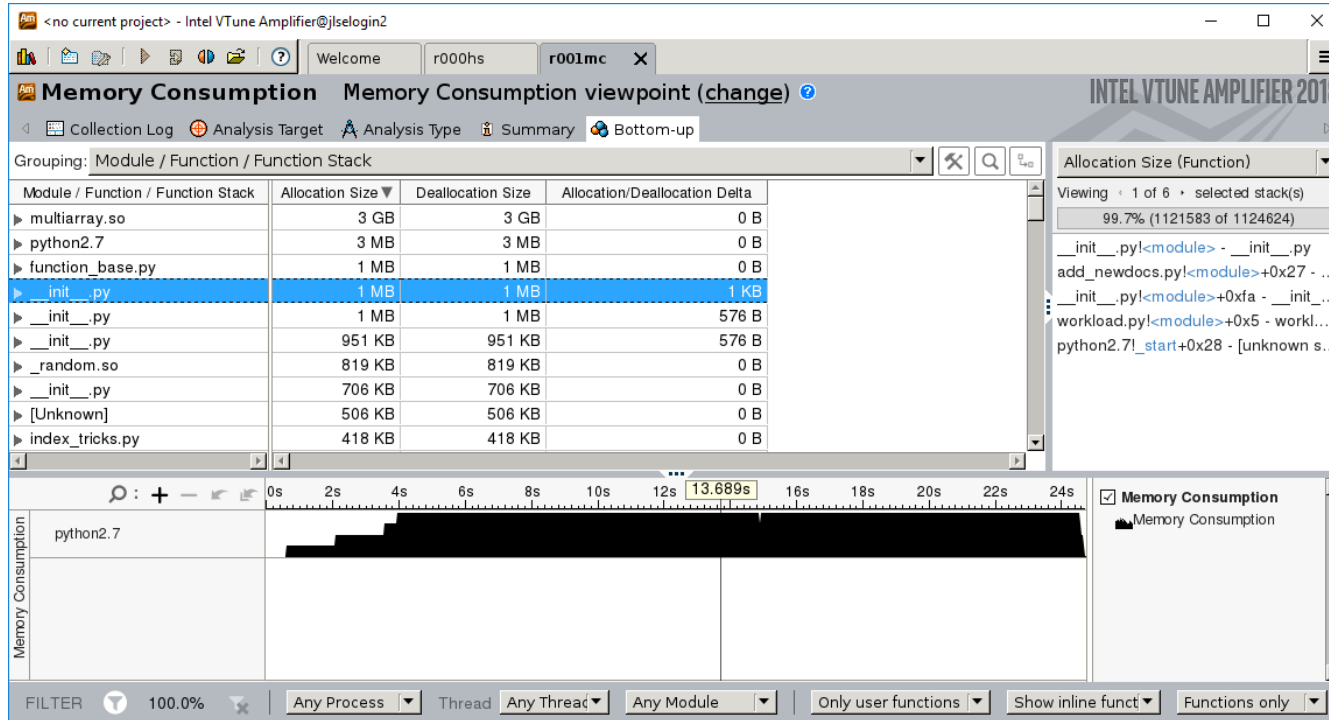
The right-hand pane shows the call graph for 'PyUFunc\_d\_d', which is highlighted in blue. The call graph shows the following callers and callees:

| Callers                | CPU Time: Total | CPU Time: Self |
|------------------------|-----------------|----------------|
| PyUFunc_d_d            | 100.0%          |                |
| trivial_two_operand_lo | 100.0%          |                |
| PyUFunc_GenericFu      | 100.0%          |                |
| ufunc_generic_cal      | 100.0%          |                |
| PyObject_Call          | 100.0%          |                |
| <module>               | 100.0%          |                |
| __libc_start           | 100.0%          |                |
| <b>_start</b>          | <b>100.0%</b>   |                |

The callees for 'PyUFunc\_d\_d' are:

| Callees   | CPU Time: Total | CPU Time: Self |
|-----------|-----------------|----------------|
| PyUFunc_d | 100.0%          | 3.671s         |
| exp       | 43.1%           | 8.749s         |
| __ieee754 | 38.3%           | 7.780s         |
| log       | 0.5%            | 0.100s         |

# Memory Consumption Analysis







# Profiling MPI4py

<https://github.com/jbornschein/mmpi4py-examples>

# Profiling python MPI jobs

- 2 options
  - Collect on every rank
    - Might be unable to launch enough ampxe-cl instances
  - Collect on select ranks
    - Smaller result

# Profiling python MPI jobs - every rank

```
$: mpirun -n 2 \  
amplxe-cl -c hotspots -r vtune_res \  
-- ~/intel/intelpython2/bin/python 07-matrix-vector-product
```

# Profiling python MPI jobs - Select rank

```
$: mpirun -n 1 \  
amplxe-cl -c hotspots -r vtune_res \  
-- ~/intel/intelpython2/bin/python 07-matrix-vector-product \  
\  
: -n 1 ~/intel/intelpython2/bin/python 07-matrix-vector-product
```

# Profiling python MPI jobs - Select rank Cray

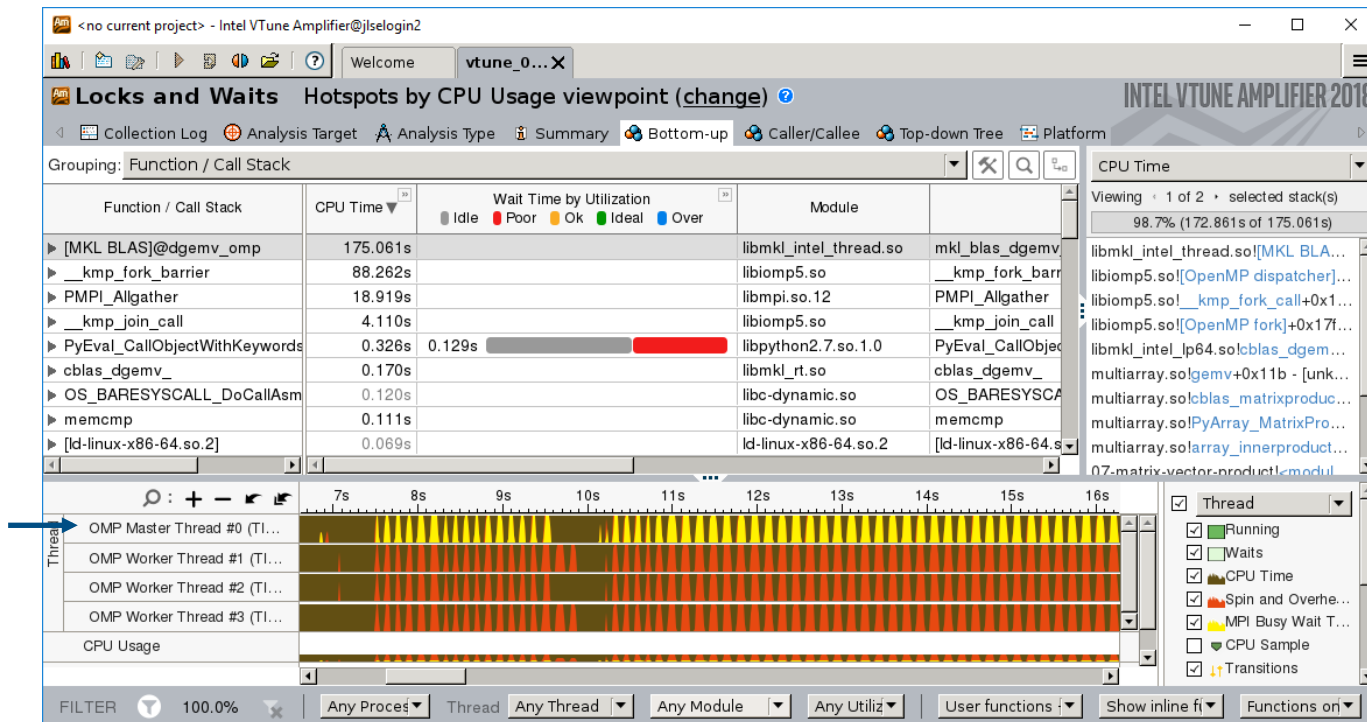
profile1.sh

```
1 #!/bin/bash
2 # source /opt/intel/parallel_studio_xe_2018/psxevars.sh intel64
3 # export LD_LIBRARY_PATH=/opt/intel/advisor/lib64:$LD_LIBRARY_PATH
4 # export LD_LIBRARY_PATH=/opt/intel/vtune_amplifier/lib64:$LD_LIBRARY_PATH
5
6 export PE_RANK=$ALPS_APP_PE
7 export PMI_NO_FORK=1
8 if [ "$PE_RANK" == 0 ];then
9 $1 -- $2
10 else
11 $2
12 fi
```

# Profiling python MPI jobs - Select rank Cray

```
aprun -n 2 ./profile1.sh \  
"amplxe-cl -c hotspots -r vtune_res" \  
"~/intel/intelpython2/bin/python 07-matrix-vector-product"
```

# Bottom-up View – MPI





# Profiling Libraries

Python Module for Quantum Chemistry

<https://github.com/sunqm/pyscf>



# Prerequisites

Build your libraries with `-g` to include debug symbols

Might have to add `--search-dir src:=/path/to/library/source` to your collection line

# Two-Step Process

Many available analysis types:

- advanced-hotspots Advanced Hotspots
- concurrency Concurrency
- disk-io Disk Input and Output
- general-exploration General microarchitecture exploration
- gpu-hotspots GPU Hotspots
- gpu-profiling GPU In-kernel Profiling
- hotspots Basic Hotspots
- hpc-performance HPC Performance Characterization
- locksandwaits Locks and Waits
- memory-access Memory Access
- memory-consumption Memory Consumption
- system-overview System Overview
- ...

## Step # 2

## Step # 1

Python Support

# Step # 1

**\$: ampxe-cl -c hotspots -- python ./workload.py**

**Basic Hotspots** Hotspots by CPU Usage viewpoint (change) INTEL VTUNE AMPLIFIER 2018

Grouping: Call Stack

| Function Stack                | CPU Time: Total | CPU Time: Self | Module              | F              |
|-------------------------------|-----------------|----------------|---------------------|----------------|
| Total                         | 100.0%          | 0s             |                     |                |
| _start                        | 99.8%           | 0s             | python2.7           | _start         |
| __libc_start_main             | 99.8%           | 0.049s         | libc.so.6           | __libc_start_m |
| <module>                      | 99.8%           | 0s             | 20-k_points_sct.py  | <module>       |
| PyEval_EvalCodeEx             | 99.7%           | 0s             | libpython2.7.so.1.0 | PyEval_EvalCo  |
| M                             | 73.9%           | 0s             | cell.py             | M(**kwargs)    |
| kernel                        | 23.1%           | 0s             | newton_ah.py        | _CIAH_SOSCF    |
| kernel                        | 2.6%            | 0s             | hf.py               | SCF.kernel(se  |
| ufunc_generic_call            | 0.0%            | 0s             | umath.so            | ufunc_generic  |
| array_divide                  | 0.0%            | 0s             | multiarray.so       | array_divide   |
| PyEval_CallObjectWithKeywords | 0.1%            | 0s             | libpython2.7.so.1.0 | PyEval_CallOb  |
| [Unknown stack frame(s)]      | 0.0%            | 0.010s         |                     | [Unknown stac  |
| PyCFuncPtr_call               | 0.0%            | 0s             | _ctypes.so          | PyCFuncPtr_c   |
| ufunc_generic_call            | 0.0%            | 0s             | umath.so            | ufunc_generic  |
| array_assign_subscript        | 0.0%            | 0s             | multiarray.so       | array_assign_  |
| __clone                       | 0.2%            | 0s             | libc.so.6           | __clone        |
| [Outside any known module]    | 0.0%            | 0.040s         |                     | [Outside any k |
| [OpenMP dispatcher]           | 0.0%            | 0.020s         | libiomp5.so         | _kmp_invoke    |

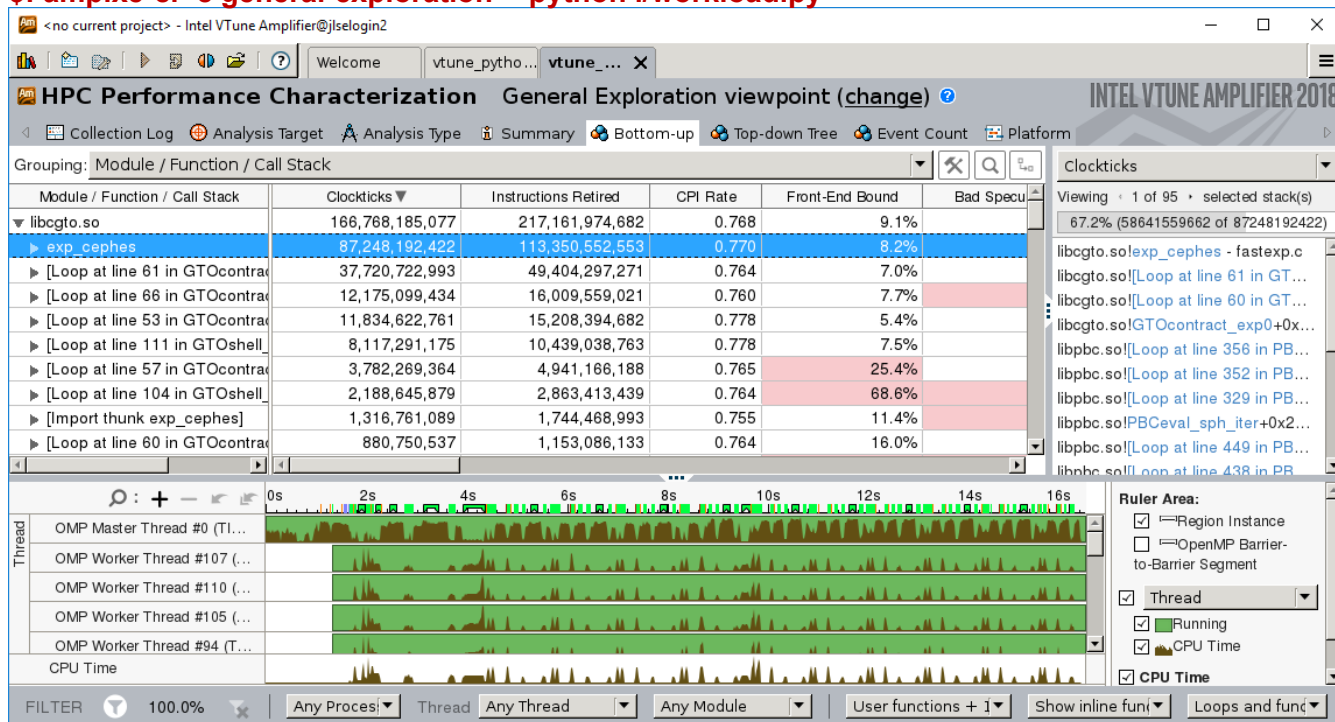
Viewing 1 of 2173 selected stack(s)  
15.9% (248.801s of 1564.138s)

libcgto.so!GTOshell\_eval\_grid\_c...  
libpbc.so!PBCeval\_sph\_iter+0x8...  
libpbc.so!PBCeval\_loop\_omp\_fn...  
libiomp5.so!OpenMP\_dispatcher]...  
libiomp5.so!GOMP\_parallel\_start...  
libpbc.so!PBCeval\_loop+0x19c - ...  
libpbc.so!PBCeval\_sph\_drv+0xc2...  
libpbc.so!PBCGTOval\_sph\_deriv...  
libffi.so.6!ffi\_call\_unix64+0x4b - [...  
libffi.so.6!ffi\_call+0x22c - [unkno...  
\_ctypes.so!\_ctypes\_callproc+0x4...  
\_ctypes.so!PyCFuncPtr\_call+0xe...  
libpython2.7.so.1.0!PyObject\_Cal...  
eval\_gto.py!eval\_gto+0x3b9 - ev...  
libpython2.7.so.1.0!PyEval\_Eval...  
cell.py!pbc\_eval\_gto+0x20 - cell...  
libpython2.7.so.1.0!PyEval\_Eval...  
numint.py!eval\_ao\_kpts+0xba - n...

FILTER 100.0% Any Proces Thread Any Thread Any Module Any Utiliz User functions Show inline f Functions on

# Step # 2- General Exploration

**\$: ampxe-cl -c general-exploration -- python ./workload.py**



# Step # 2b- Advisor

\$: **advixe-cl -c roofline -- python ./workload.py**

Intel Advisor 2018 interface showing a roofline analysis. The main table displays the following data:

| ROOFLINE | Function Call Sites and Loops                   | Performance Issues | Self Time       | Total Time       | Type                   | Vectorized Loops      | Location            | Why No |
|----------|---|--------------------|-----------------|------------------|------------------------|-----------------------|---------------------|--------|
|          |   |                    |                 |                  |                        | Vec. Gain... VL (...) |                     |        |
|          | [loop in _aligned_strided_to_contig_size8]      |                    | 0.760s          | 0.760s           | Vectorized (Bo...      | SSE4 4                |                     |        |
|          | <b>[loop in GTO_Gv_general at ft_ao.c:1071]</b> | 3 Scalar m...      | <b>415.294s</b> | <b>5337.661s</b> | <b>Vectorized (...</b> | <b>SSE2 2</b>         | <b>ft_ao.c:1071</b> |        |
|          | [loop in vrr1d_withGv at ft_ao.c:246]           | 1 Serialized ...   | 219.230s        | 520.631s         | Vectorized (Bo...      | SSE2 2                | ft_ao.c:246         |        |
|          | [loop in vrr1d_withGv at ft_ao.c:268]           | 1 Serialized ...   | 96.173s         | 204.733s         | Vectorized (Bo...      | SSE2 2                | ft_ao.c:268         |        |
|          | [loop in vrr1d_withGv at ft_ao.c:263]           | 1 Serialized ...   | 65.306s         | 173.904s         | Vectorized (Bo...      | SSE2 2                | ft_ao.c:263         |        |
|          | [loop in GTO_Gv_orth at ft_ao.c:1132]           | 3 Scalar mat...    | 18.898s         | 51.716s          | Vectorized (Bo...      | SSE2 2                | ft_ao.c:1132        |        |
|          | [loop in GTOcontract_exp0 at deriv1.c:61]       | 1 Serialized ...   | 1.090s          | 3.690s           | Vectorized (Bo...      | SSE2 2                | deriv1.c:61         |        |

The selected loop (ft\_ao.c:1071) is detailed below:

| Line | Source   | Total Time | % | Loop/Function Time | % | Traits |
|------|--|------------|---|--------------------|---|--------|
| 1068 | const double cutoff = EXPCUTOFF * aij * 4;                                 |            |   |                    |   |        |
| 1069 | int n;   |            |   |                    |   |        |
| 1070 | double kR, kk;   |            |   |                    |   |        |
| 1071 | for (n = 0; n < NGV; n++) {  | 16.790s    |   | 5,337.660s         |   |        |
|      | [loop in GTO_Gv_general at ft_ao.c:1071]                                   |            |   |                    |   |        |
|      | Vectorized SSE2 loop processes Float64 data type(s) and includes Divisions |            |   |                    |   |        |
| 1072 | kk = kx[n] * kx[n] + ky[n] * ky[n] + kz[n] * kz[n];                        | 49.650s    |   |                    |   |        |
| 1073 | if (kk < cutoff) {   | 17.799s    |   |                    |   |        |
| 1074 | kR = kx[n] * rij[0] + ky[n] * rij[1] + kz[n] * rij[2];                     | 47.097s    |   |                    |   |        |
| 1075 | out[n] = exp(-.25*kk/aij) * fac * (cos(kR) - sin(kR))*_Complex_            | 280.868s   |   |                    |   |        |
| 1076 | } else {   |            |   |                    |   |        |
| 1077 | out[n] = 0;  |            |   |                    |   |        |

Selected (Total Time): 16.790s

# Note on Collection/Finalization

- Your job is too short to collect sufficient information
  - Increase sampling freq `--interval 0`
- Finalization takes way too long
  - `--finalization-mode=none`
- Unknowns in your results
  - Libraries compiled with `-g`?
- Sources not being found
  - `--search-dir src:=/path/to/source/dir`

# 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.

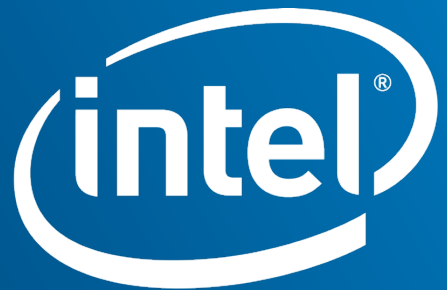
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may 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. For more complete information visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks).

Copyright © 2018, 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



Software