Assignment No. B3

Aim

VTune Amplifier

Problem Definition

Develop a stack sampling using threads using VTune Amplifier.

Learning Objectives

- Learn how to use VTune Amplifier
- Understanding stack sampling

Learning Outcome

- Implemented stack sampling using VTune Amplifier

Software And Hardware Requirements

- Latest 64-BIT Version of Linux Operating System
- VTune Amplifier

Mathematical Model

Let S be the system of solution set for given problem statement such that,

S = { s, e, X, Y, F, DD, NDD, Su, Fu }

where,

s = start state
such that, binary executable A is present

e = end state such that, stack sampling S is done.

X = set of inputsuch that X = { A } A = Binary executable

Y = set of outputsuch that Y = { S } where,
S→ stack sampling output

F = set of function

such that F = { f1, f2 }
where, f1 =
generate a.out
f2 = use VTune Amplifier and view results

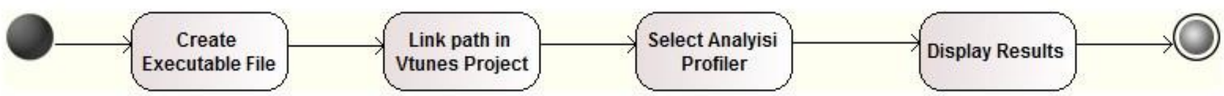
DD = Deterministic data
DD = { A, f1 }

NDD = Nondeterministic data
NDD = { S, f2 }

Su = Success cases
Stack sampling using threads is implemented

Fu = Failure case
VTune Amplifier is unable to run a.out
is not found

State Diagram



Theory

Intel VTune Amplifier is a commercial application for software performance analysis for 32 and 64-bit x86 based machines, and has both GUI and command line interfaces. It is available for both Linux and Microsoft Windows operating systems. Although basic features work on both Intel and AMD hardware, advanced hardware-based sampling requires an Intel-manufactured CPU.

VTune Amplifier

Intel VTune Amplifier provides a rich set of performance insight into CPU and GPU performance, threading performance and scalability, bandwidth, caching and much more. Analysis is faster and easier because VTune Amplifier understands common threading models and presents information at a higher level that is easier to interpret. Use its powerful analysis to sort, filter and visualize results on the timeline and on your source.

Stack Sampling

VTune Amplifier assists in various kinds of code profiling including stack sampling, thread profiling and hardware event sampling. The profiler result consists of details such as time spent in each sub routine which can be drilled down to the instruction level. The time taken by the instructions are indicative of any stalls in the pipeline during instruction execution. The tool can be also used to analyze thread performance. The new GUI can filter data based on a selection in the timeline.

Features

- Software sampling

Works on x86 compatible processors and gives both the locations where time is spent and the call stack used.

- JIT profiling support
  Profiles dynamically generated code.

- Locks and waits analysis
  Finds long synchronization waits that occur when cores are underutilized.

- Threading timeline
  Shows thread relationships to identify load balancing and synchronization issues. It can also be used to select a region of time and filter the results. This can remove the clutter of data gathered during uninteresting times like application start-up.

- Source view
  Sampling results are displayed line by line on the source / assembly code.

- Hardware event sampling
  This uses the on chip performance monitoring unit and requires an Intel processor. It can find specific tuning opportunities like cache misses and branch mispredictions.

- Performance Tuning Utility (PTU)
  PTU Was a separate download that gave VTune Amplifer XE users access to experimental tuning technology. This includes things like Data Access Analysis that identifies memory hotspots and relates them to code hotspots. PTU now is fully integrated into VTune Amplifer XE.

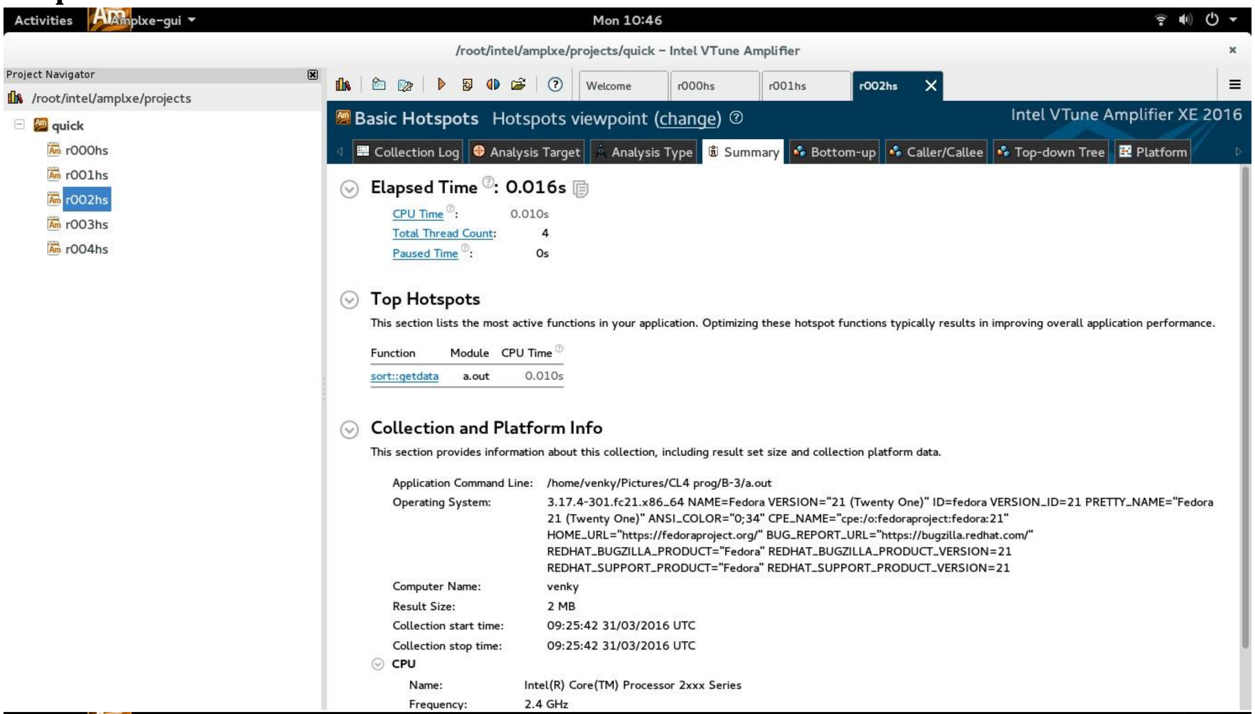
Program Code

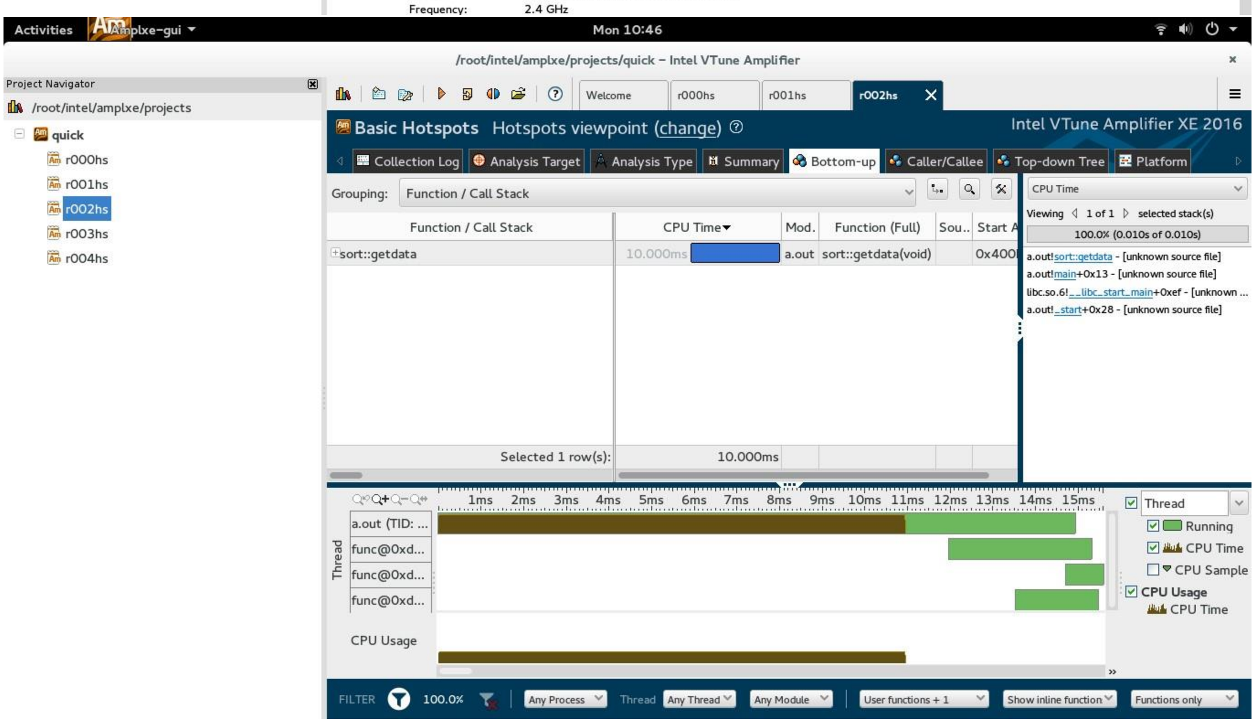
```
// quicksort . cpp
#include<iostream>
#include<omp.h> using
namespace std ; int k=0;
class     sort
{ int a [2 0] ; int n; public
:
        void   getdata   ();   void
        Quicksort ();
        void Quicksort ( int low ,       int       high ); int
            partition ( int low ,       int       high ); void
        putdata ();
};
void     sort : : getdata ()
{
        cout<<"Enter the no .  of        elements          in
            array\t "; cin>>n; cout<<"Enter the elements        of
            array:"<<endl ;
         for ( int      i =0;i<n; i++)
        { cin>>a [ i ] ;
        }
} void   sort : : Quicksort ()
```

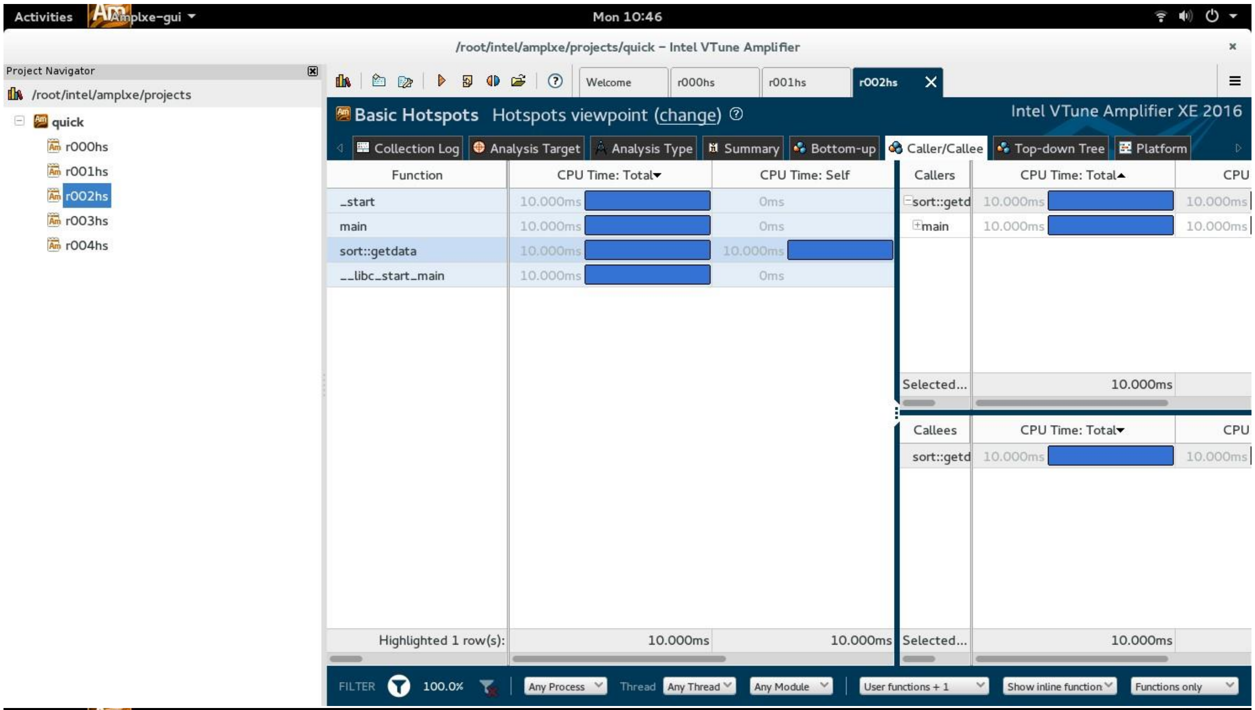
```
{
 Quicksort (0 ,n−1);
}

void          sort : : Quicksort ( int low ,          int     high )
{ i f (low<high )
{
                int     partn  ;  partn=partition  (low  ,
                high );

cout<<”\n\nThread Number: ”<<k<<”                    pivot element         selected         : ”<<a [ partn ] ;
                   #pragma omp parallel          sections
               {
                 #pragma omp section
                             { k=k+1;
                               Quicksort (low ,        partn −1);
                             }
                 #pragma omp section
                             { k=k+1;
                             Quicksort ( partn+1, high );
                             }
               }//pragma _omp Parallel _end
}

}
int           sort : : partition ( int low          , int     high )
{
                int pvt ;
                pvt=a [ high ] ;
                int     i ;
                i=low−1; int   j
                ;
                for ( j=low ; j<high ; j++)
                { i f (a [ j]<=pvt)
                             {
                                           int tem=0;
                                           tem=a  [ j ] ; a [ j
                                           ]=a  [  i  +1];  a  [
                                           i+1]=tem ;
                                           i=i +1;
                             }
                }
                int     te ; te=a [
                high ] ;
                a [ high]=a [ i +1]; a [
                i+1]=te ;
                 return      i +1;
}
```

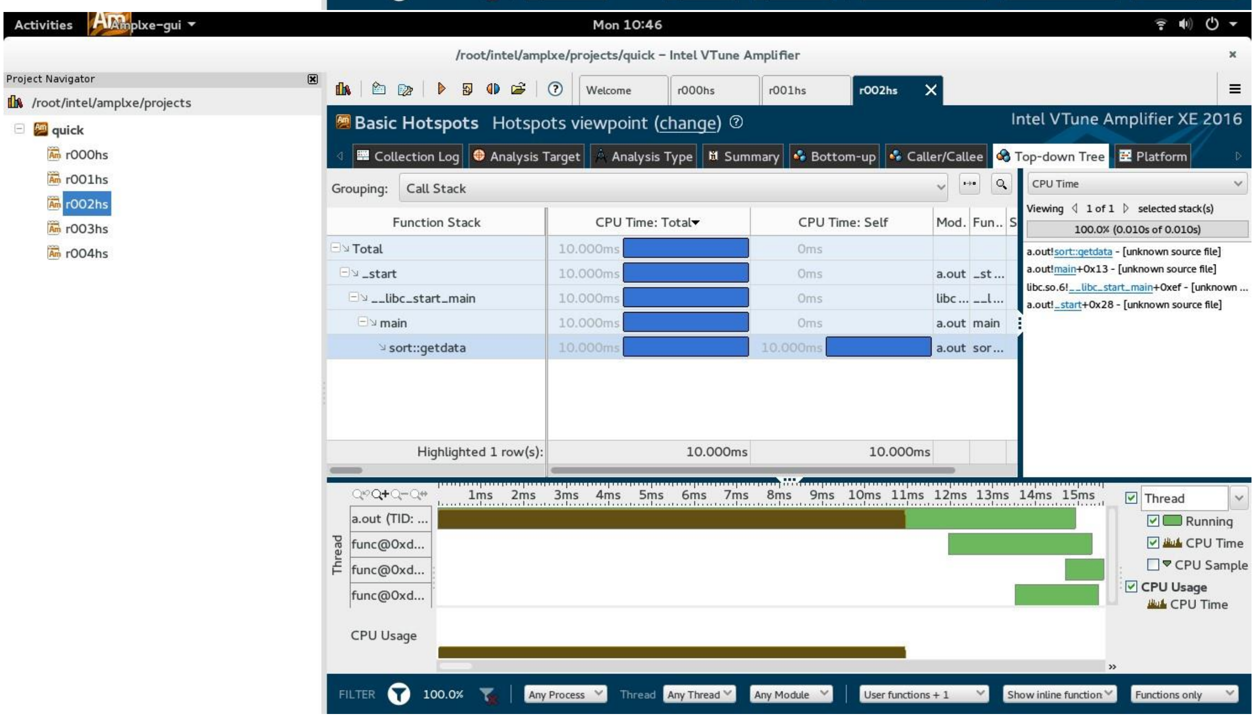
```
void        sort : : putdata ()
{ cout<<endl<<"\nThe Array is:"<<endl ;
                 for ( int       i =0;i<n; i++)
                cout<<" "<<a [ i ] ;
}
 int main()
{
                int  n;  sort
                s1 ; int ch ;
do
{
                               s1  .  getdata  ();  s1  .
                               putdata ();

                                     cout<<"\nUsing Quick Sort ";
        double   start   =   omp   get   wtime   ();   s1   .
                               Quicksort ();
        double end = omp get wtime (); cout<<"\nThe
                               Sorted       "; s1 . putdata ();
             cout<<"\nExcecution time                 : "<<end − start <<" seconds           ";

cout<<"Would you like    to continue? (1/0 y/n)"<<endl ; cin>>ch ;
}while (ch==1);
}
```

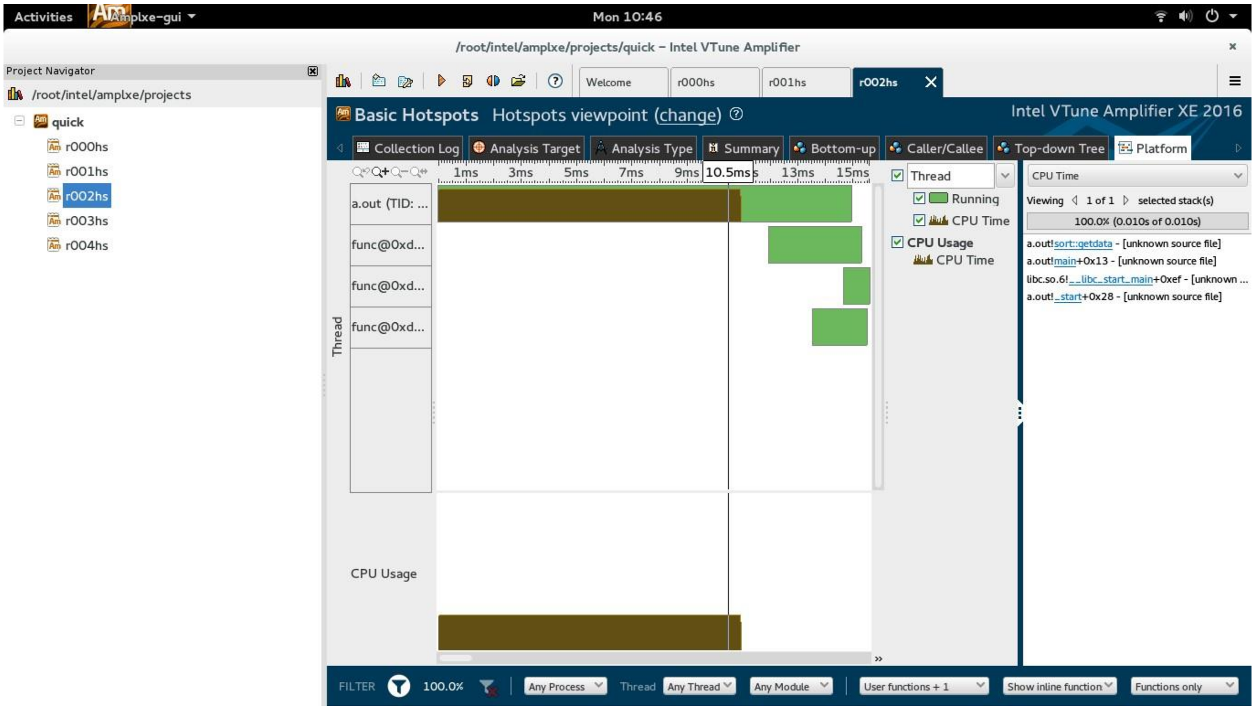
Output


Activities
Amplxe-gui
Mon 10:46
/root/intel/amplxe/projects/quick – Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
quick
r000hs
r001hs
r002hs
r003hs
r004hs
Welcome
r000hs
r001hs
r002hs
Basic Hotspots Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Elapsed Time: 0.016s
CPU Time: 0.010s
Total Thread Count: 4
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
sort::getdata a.out 0.010s
Collection and Platform Info
This section provides information about this collection, including result set size and collection platform data.
Application Command Line: /home/venky/Pictures/CL4 prog/B-3/a.out
Operating System: 3.17.4-301.fc21.x86_64 NAME=Fedora VERSION="21 (Twenty One)" ID=fedora VERSION_ID=21 PRETTY_NAME="Fedora 21 (Twenty One)" ANSI_COLOR="0;34" CPE_NAME="cpe:/o:fedoraproject:fedora:21" HOME_URL="https://fedoraproject.org/" BUG_REPORT_URL="https://bugzilla.redhat.com/" REDHAT_BUGZILLA_PRODUCT="Fedora" REDHAT_BUGZILLA_PRODUCT_VERSION=21 REDHAT_SUPPORT_PRODUCT="Fedora" REDHAT_SUPPORT_PRODUCT_VERSION=21
Computer Name: venky
Result Size: 2 MB
Collection start time: 09:25:42 31/03/2016 UTC
Collection stop time: 09:25:42 31/03/2016 UTC
CPU
Name: Intel(R) Core(TM) Processor 2xxx Series
Frequency: 2.4 GHz


Activities
Amplxe-gui
Mon 10:46
/root/intel/amplxe/projects/quick – Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
quick
r000hs
r001hs
r002hs
r003hs
r004hs
Welcome
r000hs
r001hs
r002hs
Basic Hotspots Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Grouping: Function / Call Stack
CPU Time
Viewing 1 of 1 selected stack(s)
100.0% (0.010s of 0.010s)
Function / Call Stack
CPU Time
Mod.
Function (Full)
Sou..
Start A
sort::getdata
10.000ms
a.out
sort::getdata(void)
0x400
a.out!sort::getdata - [unknown source file]
a.out!main+0x13 - [unknown source file]
libc.so.6!__libc_start_main+0xef - [unknown ...
a.out!_start+0x28 - [unknown source file]
Selected 1 row(s): 10.000ms
1ms 2ms 3ms 4ms 5ms 6ms 7ms 8ms 9ms 10ms 11ms 12ms 13ms 14ms 15ms
Thread
a.out (TID: ...
func@0xd...
func@0xd...
func@0xd...
CPU Usage
Thread
Running
CPU Time
CPU Sample
CPU Usage
CPU Time
FILTER 100.0%
Any Process
Thread Any Thread
Any Module
User functions + 1
Show inline function
Functions only


Activities
amplxe-gui
Mon 10:46
/root/intel/amplxe/projects/quick - Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
quick
r000hs
r001hs
r002hs
r003hs
r004hs
Welcome
r000hs
r001hs
r002hs
Basic Hotspots Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Function
CPU Time: Total
CPU Time: Self
_start
main
sort::getdata
__libc_start_main
Callers
sort::getd
Selected...
Callees
Highlighted 1 row(s):
10.000ms
FILTER
100.0%
Any Process
Thread
Any Thread
Any Module
User functions + 1
Show inline function
Functions only


Activities
amplxe-gui
Mon 10:46
/root/intel/amplxe/projects/quick - Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
quick
r000hs
r001hs
r002hs
r003hs
r004hs
Basic Hotspots Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Grouping:
Call Stack
Function Stack
CPU Time: Total
CPU Time: Self
Total
_start
__libc_start_main
main
sort::getdata
Highlighted 1 row(s):
10.000ms
CPU Time
Viewing 1 of 1 selected stack(s)
100.0% (0.010s of 0.010s)
a.out!sort::getdata - [unknown source file]
a.out!main+0x13 - [unknown source file]
libc.so.6!__libc_start_main+0xef - [unknown ...
a.out!_start+0x28 - [unknown source file]
Thread
a.out (TID: ...
func@0xd...
CPU Usage
Running
CPU Time
CPU Sample
FILTER
100.0%
Any Process
Any Thread
Any Module
User functions + 1
Show inline function
Functions only



Conclusion

After analyzing the VTune main features, we can say that it has an easy-to-use interface and very rich functionality. Thus, we have studied VTune Amplifier and implemented stack sampling using threads using Vtune Amplifier.