

Assignment No. 9

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 input

such that $X = \{ A \}$

A = Binary executable

Y = set of output

such 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

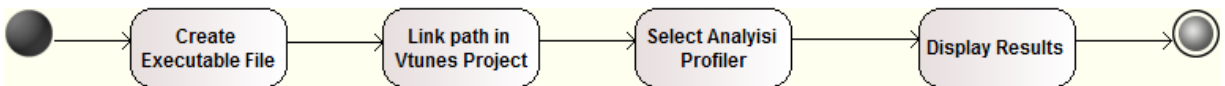
DD = Deterministic data
DD = $\{ A, f1 \}$

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 Amplifier 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 Amplifier XE.

Program Code

```
//quicksort.cpp
#include<iostream>
#include<omp.h>
using namespace std;
int k=0;
class sort
{
    int a[20];
    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)
    {
        if(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++)
        {
            if(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;

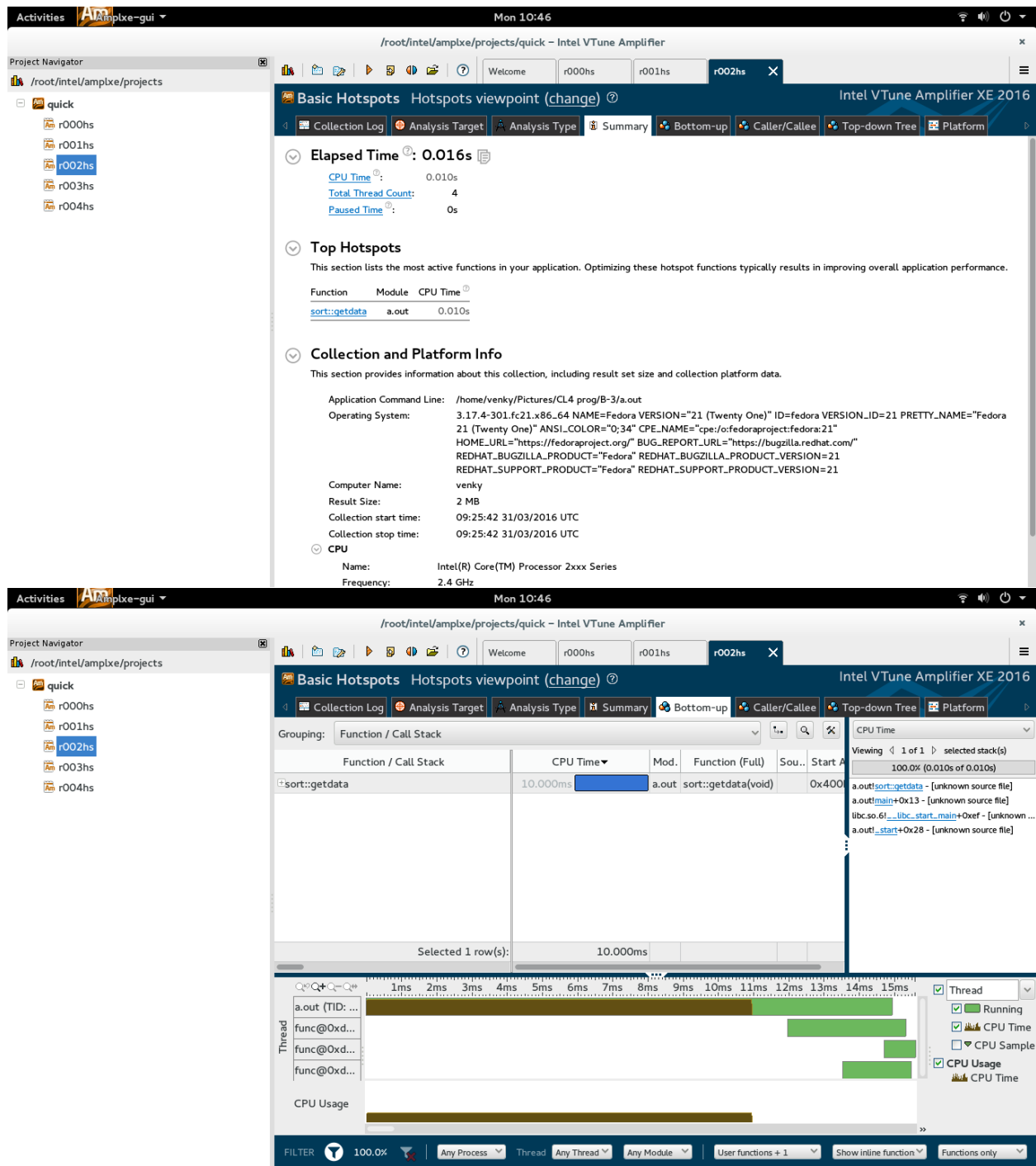
do
    {
        s1.getdata();
        s1.putdata();

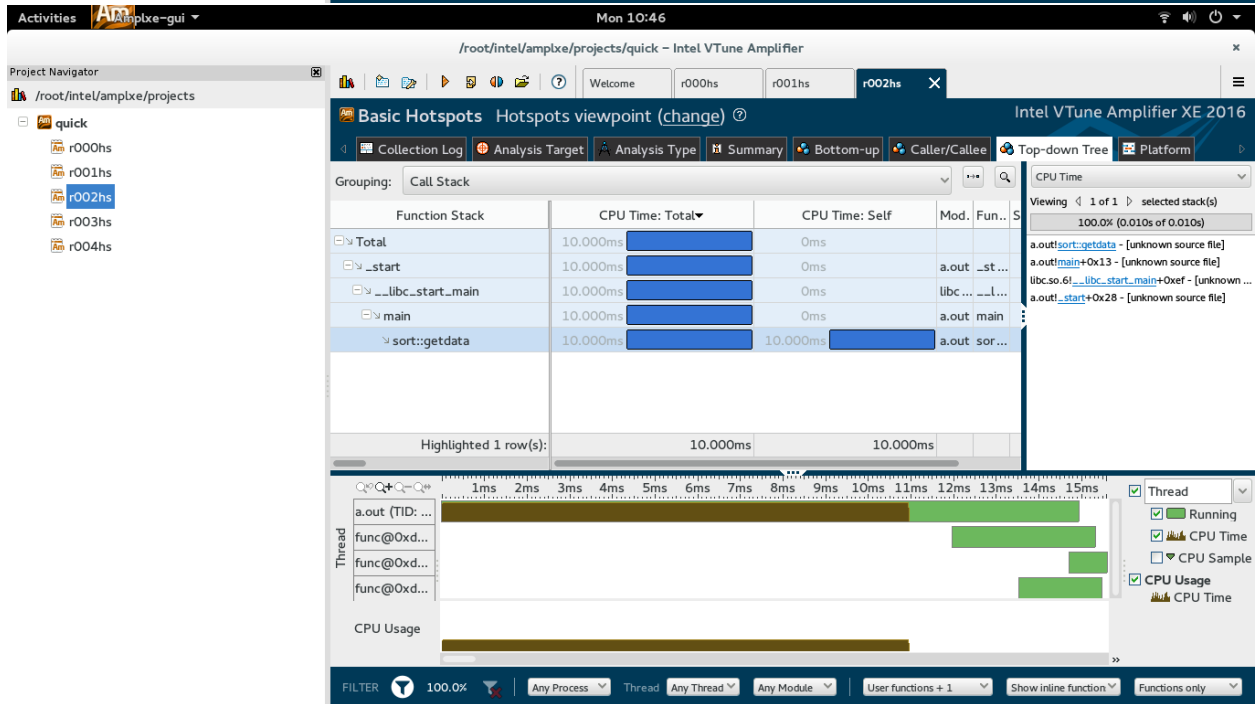
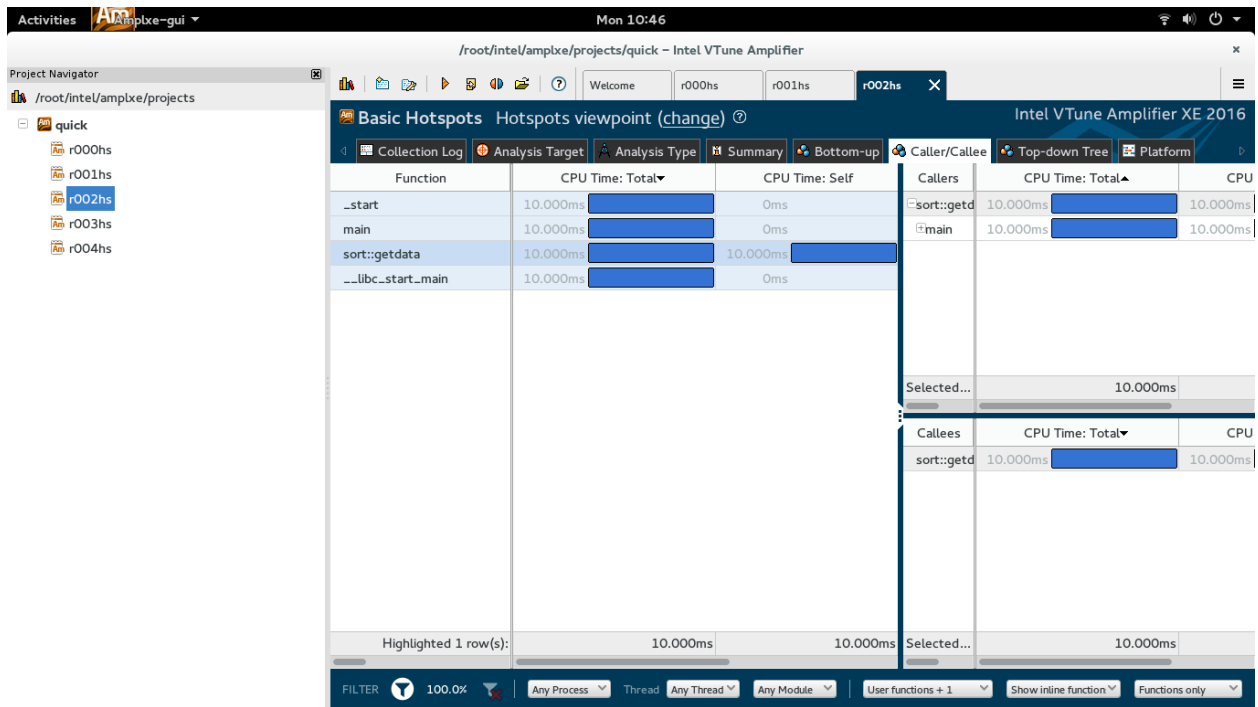
        cout<<"\nUsing Quick Sort";
        double start = omp_get_wtime();
        s1.Quicksort();
        double end = omp_get_wtime();
        cout<<"\nThe Sorted ";
        s1.putdata();
        cout<<"\nExcecution time : "<<end - start<<" seconds ";

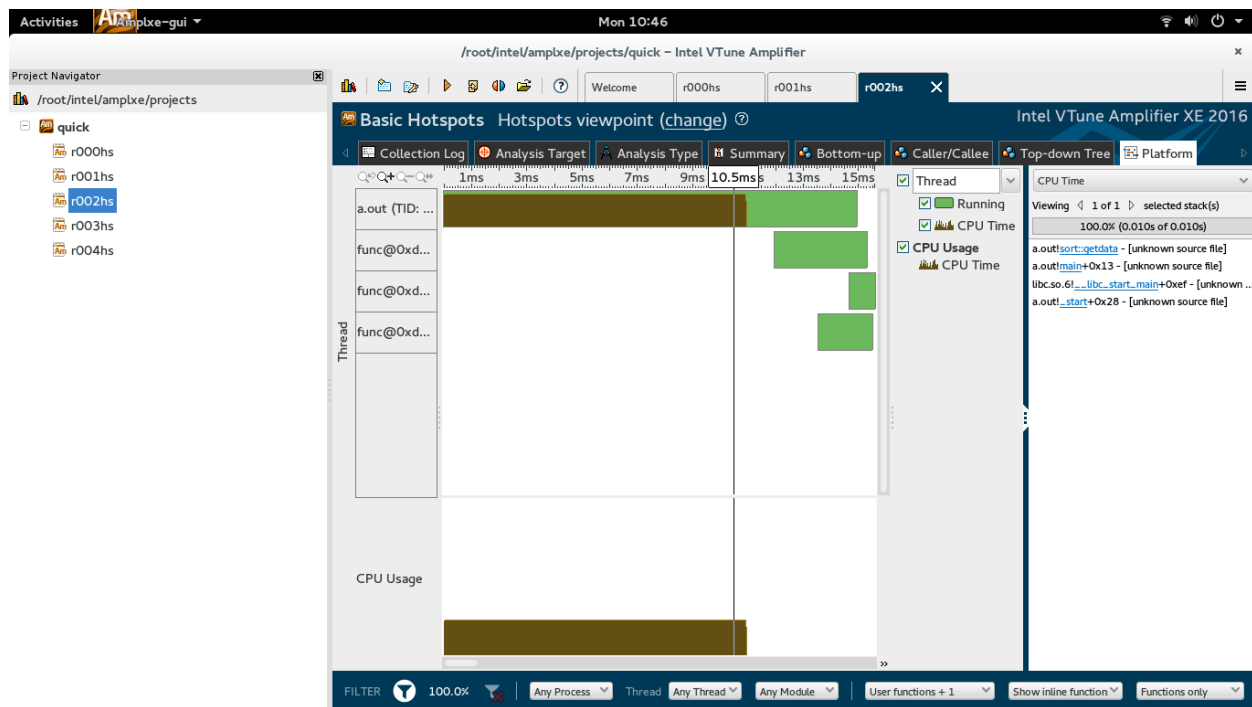
        cout<<"Would you like to continue? (1/0 y/n)"<<endl;
        cin>>ch;
    }while(ch==1);
}

```

Output







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.