Trirmadura J Ariyawansa
Viraj Malia

VTune Amplifier performance profiler is a commercial application for software performance analysis of 32 and 64-bit x86 based machines. It has both a GUI (graphical user interface) and command line and comes in versions for Linux or Microsoft Windows operating systems. An optional download lets you analyze the Windows or Linux data with a GUI on OS X. Many features work on both Intel and AMD hardware, but advanced hardware-based sampling requires an Intel-manufactured CPU. (Wikipedia)

Some features we are gonna focus are:

- Basic Hotspots
- Concurrency
- Locks and Waits
- Memory Consumption
- Memory Access

But first, we need to install Intel Parallel Studio which includes Intel's compilers, math library and Vtune

1. Go on https://registrationcenter.intel.com/en/forms/?licensetype=2&productid=2867 and create an account and click the Custom option
2. Then unzip the tar file and run the install_GUI.sh DON'T CHANGE the installation directory!
3. After the installation is done, run the *initialize_vtune.sh* This set the environment variables and starts up Vtune. OR

 # Set compiler environment variables
source /opt/intel/compilers_and_libraries_2018/linux/bin/compilervars.sh -arch intel64 -platform linux

 # Set Vtune environment variables
source /opt/intel/vtune_amplifier_2018/amplxe-vars.sh

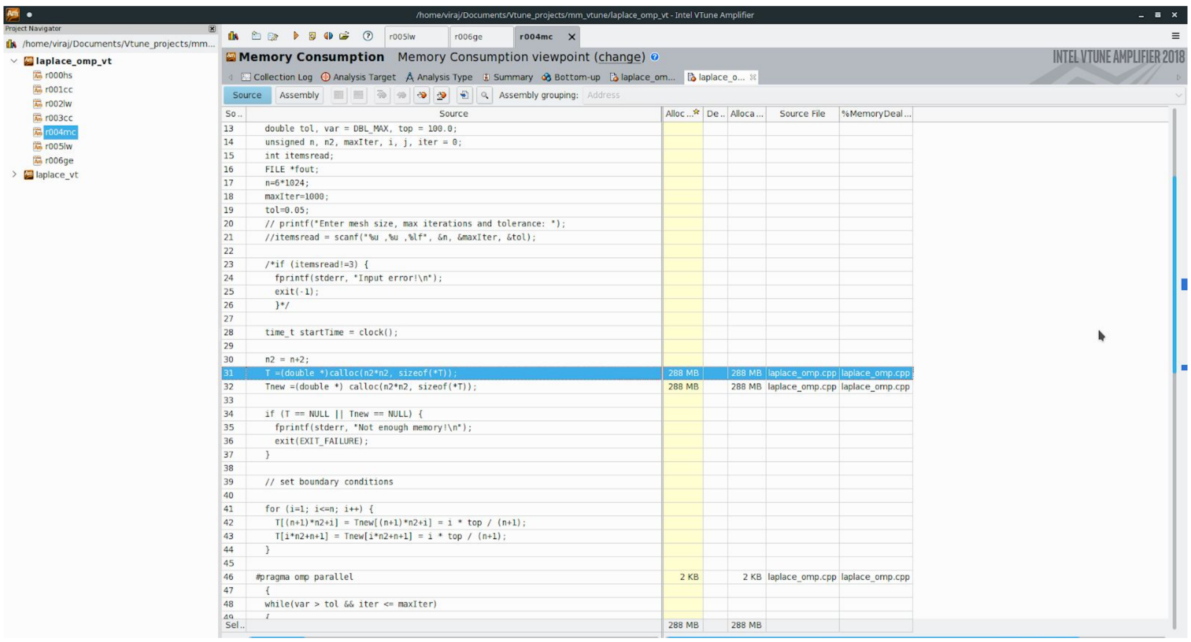
4. Once Vtunes opens up, create a new project
5. Make in the git directory containing laplace.cpp
6. In Vtunes, click on the file button under application and select laplace binary. Then click on the *Choose Analysis*

Hotspot analysis:

This analysis shows the CPU usage by each function call, CPU usage by threads and total CPU usage. It shows how many CPUs were utilized and what was the utilization per CPU.

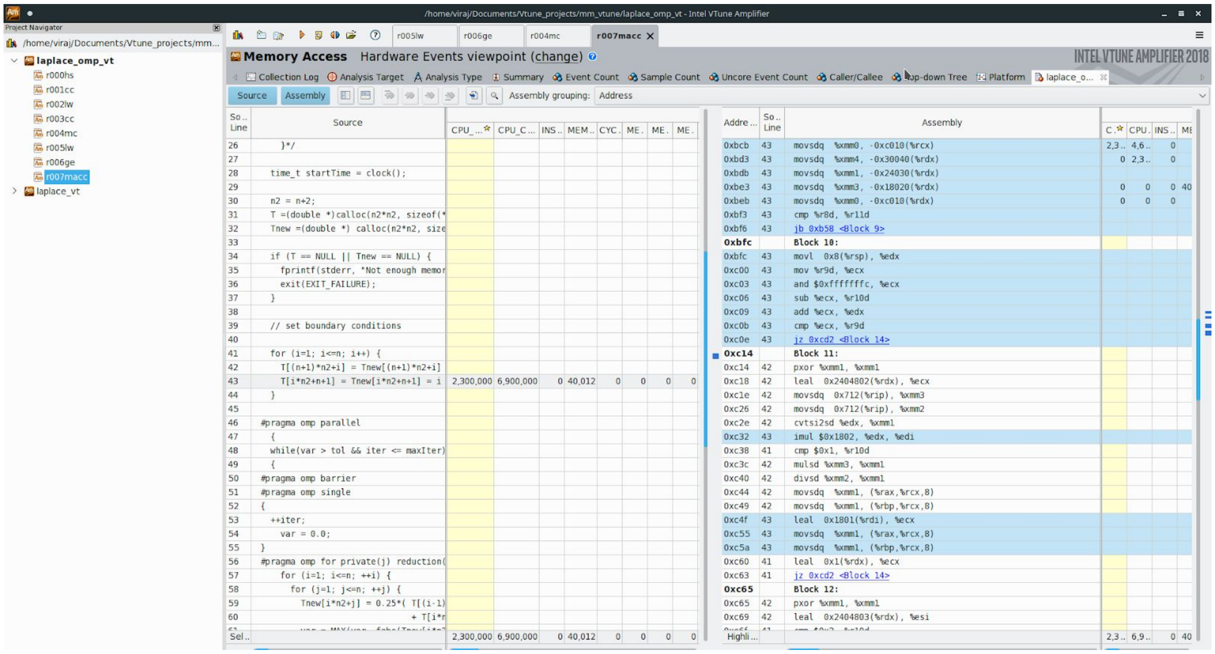
Memory Consumption:

Shows memory allocation and usage by functions. Clicking on them opens up detailed instruction level report. Assembly code can also be viewed to find memory allocations and deallocations.



Memory Access:

This is an extensive analysis and exposes issues and details from the hardware level to the software. This section shows the Cache hits, misses, number of loads, stores, stalled loads, stalled stores and more.



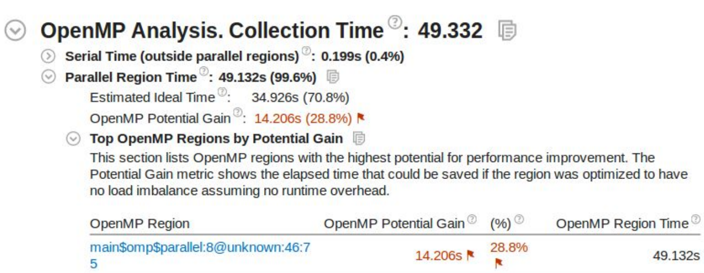
Concurrency Exercise:

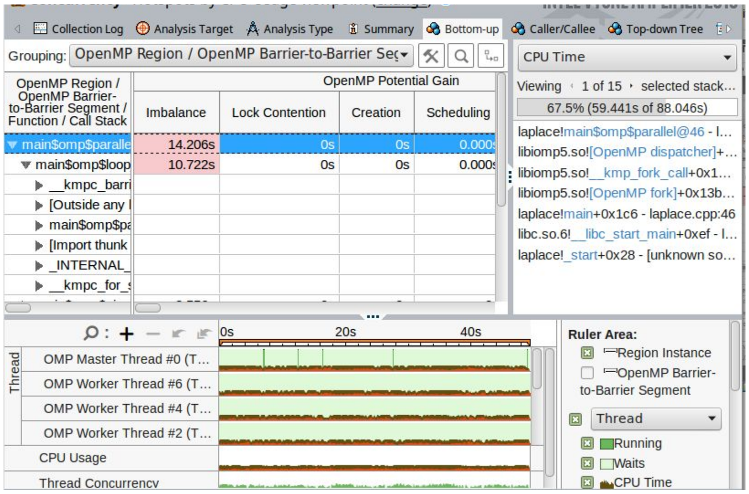
1. Click on the Concurrency Analysis and run it **Make sure the CPU sampling interval is 1ms.**

Hover over any of the questions to get any definitions of the terms used in the summary

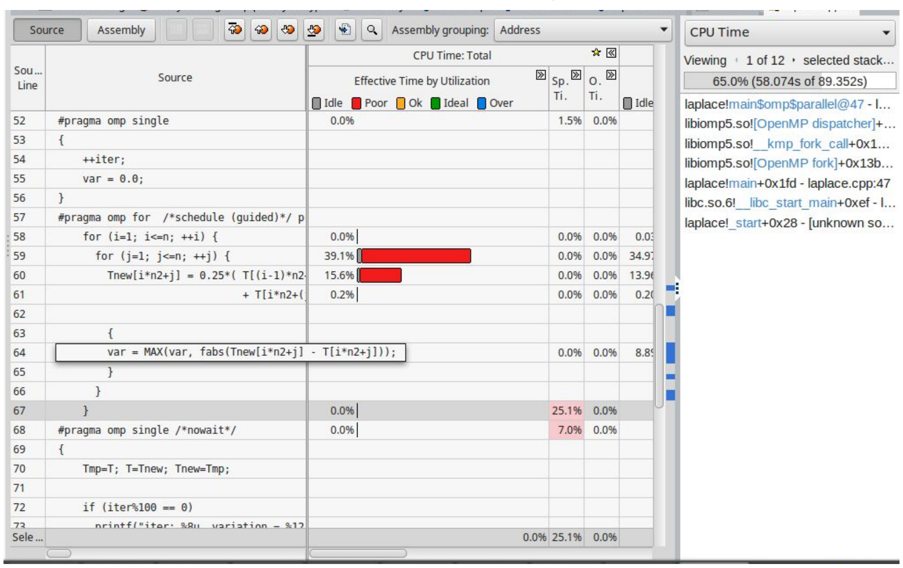
2. Look at the OpenMP Analysis Summary and it should say that the top Open MP region that could have a potential gain is *mainompparallel:8@unknown:46:75*

Concurrency could be used to find which OpenMP regions have the most CPU imbalances. Imbalances happens when threads are just spinning due to locks or waits.





3. Its shows that *main omploop_barrier_segment@unknown:66* which tells you that the omp parallel for loop is causing the imbalance. Left-click on the segment and click *View the Source* to see where in the code the imbalance and spin times happens. Add certain optimizations that could make it faster. *Remember scheduling!*



4. Once you edit it, make and run the concurrency hotspots again to see if its fixed

5. If it doesn't recognize icpc, reset the variables using the bold statements above.

6. Repeat steps 3-4 as many times as possible.