

**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:

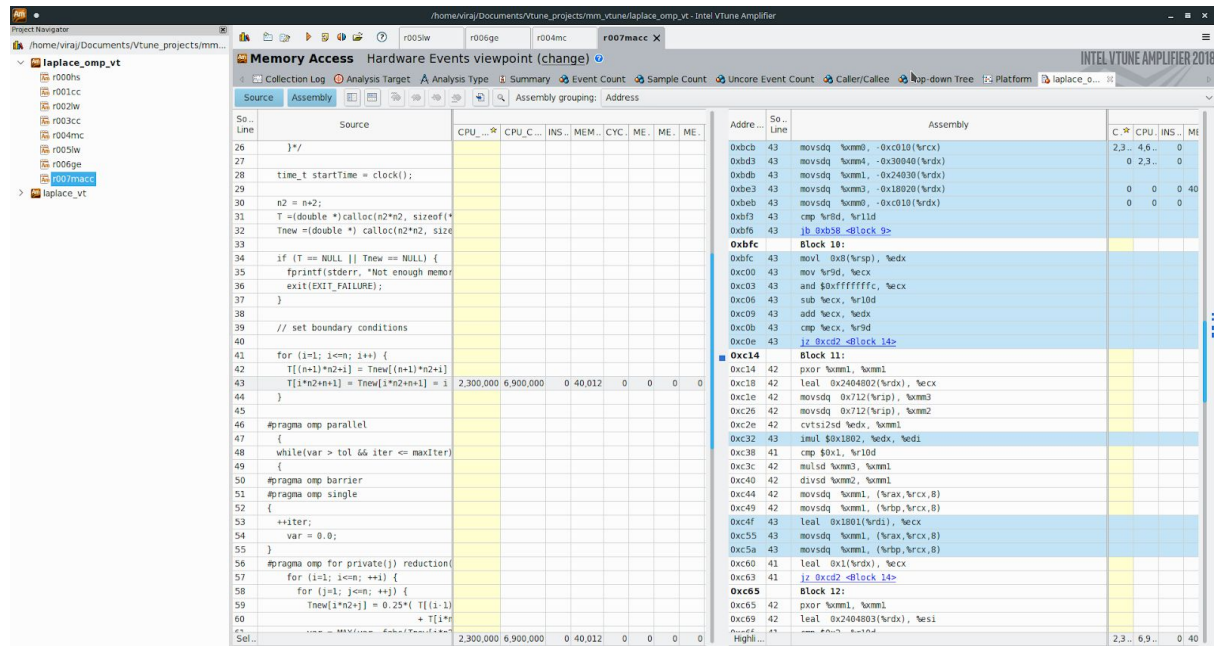
Shows memory allocation and usage by functions. Clicking on them opens up detailed instruction level report. Assembly code can also be viewed to find memory allocations and deallocations.

The screenshot shows the Intel VTune Amplifier 2018 interface. The left pane displays the project structure with 'laplace\_omp\_vt' selected. The main pane shows the 'Memory Consumption' analysis. The 'Memory Consumption viewpoint (change)' tab is active, displaying a table of memory allocation and deallocation events. The table has columns: Source, Address, Alloc, De, Alloc, Source File, and %MemoryDeal. The source code for 'laplace\_omp.cpp' is visible on the left, showing memory allocation for 'T' and 'Tnew'.

Source	Address	Alloc	De	Alloc	Source File	%MemoryDeal
13 double tol, var = DBL_MAX, top = 100.0;						
14 unsigned n, n2, maxiter, i, j, iter = 0;						
15 int itemsread;						
16 FILE *fout;						
17 n=6*1024;						
18 maxiter=1000;						
19 tol=0.05;						
20 // printf("Enter mesh size, max iterations and tolerance: ");						
21 //itemsread = scanf("%u %u %lf", &n, &maxiter, &tol);						
22						
23 /*if (itemsread!=3) {						
24 fprintf(stderr, "Input error!\n");						
25 exit(1);						
26 }						
27						
28 time_t startTime = clock();						
29						
30 n2 = n*2;						
31 T = (double *) calloc(n2*n2, sizeof(*T));		288 MB	288 MB		laplace_omp.cpp	laplace_omp.cpp
32 Tnew = (double *) calloc(n2*n2, sizeof(*T));		288 MB	288 MB		laplace_omp.cpp	laplace_omp.cpp
33						
34 if (T == NULL    Tnew == NULL) {						
35 fprintf(stderr, "Not enough memory!\n");						
36 exit(EXIT_FAILURE);						
37 }						
38						
39 // set boundary conditions						
40						
41 for (i=1; i<=n; i++) {						
42 T[(n+1)*n2+i] = Tnew[(n+1)*n2+i] = 1 * top / (n+1);						
43 T[1*n2+n+1] = Tnew[1*n2+n+1] = 1 * top / (n+1);						
44 }						
45						
46 #pragma omp parallel		2 KB	2 KB		laplace_omp.cpp	laplace_omp.cpp
47 {						
48 while(var > tol && iter <= maxiter)						
49 {						
50 Sel...		288 MB	288 MB			

## 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.**

✓ Elapsed Time<sup>Ⓢ</sup>: 50.501s

✓ CPU Time <sup>Ⓢ</sup> :	89.455s
✓ Effective Time <sup>Ⓢ</sup> :	58.083s
✓ Spin Time <sup>Ⓢ</sup> :	31.356s
Imbalance or Serial Spinning <sup>Ⓢ</sup> :	30.212s
Lock Contention <sup>Ⓢ</sup> :	0s
Other <sup>Ⓢ</sup> :	1.144s
✓ Overhead Time <sup>Ⓢ</sup> :	0.016s
✓ Wait Time <sup>Ⓢ</sup> :	136.534s
Total Thread Count:	8
Paused Time <sup>Ⓢ</sup> :	0s

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 **main\$omp\$parallel: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.**

✓ OpenMP Analysis. Collection Time<sup>Ⓢ</sup>: 49.332

✓ Serial Time (outside parallel regions)<sup>Ⓢ</sup>: 0.199s (0.4%)

✓ Parallel Region Time<sup>Ⓢ</sup>: 49.132s (99.6%)

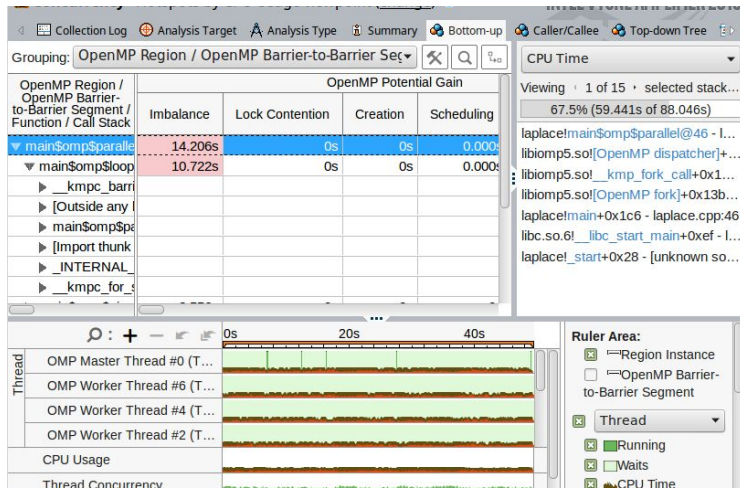
Estimated Ideal Time<sup>Ⓢ</sup>: 34.926s (70.8%)

OpenMP Potential Gain<sup>Ⓢ</sup>: 14.206s (28.8%)

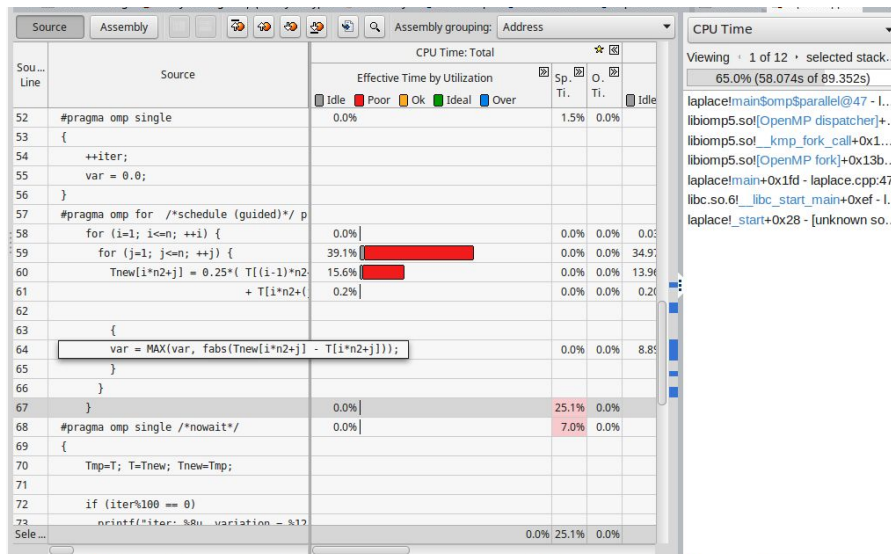
✓ Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain <sup>Ⓢ</sup> (%)	OpenMP Region Time <sup>Ⓢ</sup>
main\$omp\$parallel:8@unknown:46:75	14.206s 28.8%	49.132s
5		



3. It shows that `main $omp$loop_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!*



- Once you edit it, make and run the concurrency hotspots again to see if its fixed
- If it doesn't recognize icpc, reset the variables using the bold statements above.
- Repeat steps 3-4 as many times as possible.