**Profiling Software Using Intel VTune**

**The content of this guide is written by adopting from the following sources:**

1. <https://software.intel.com/content/www/us/en/develop/documentation/advisor-user-guide/top.html>
2. <https://hpc.llnl.gov/software/development-environment-software/intel-advisor>

**Chapter1: Introduction**

Intel's VTune Profiler is a performance profiling tool for C, C++, and Fortran code that can identify where in the code time is being spent in both serial and threaded applications. For threaded applications, it can also determine the amount of concurrency and identify bottlenecks created by synchronization primitives.

VTune Profiler uses dynamic instrumentation and thus does not require use of Intel compilers or the use of any special compiler flags.

VTune Amplifier includes both a graphical user interface (GUI) and a command line (CL) interface that can be accessed with the *vtune-gui* and *vtune* commands, respectively. Once the VTune Amplifier project is created, set up a new analysis. The analysis types include: **Hotspots**; **Threading**; and **Memory Consumption**.

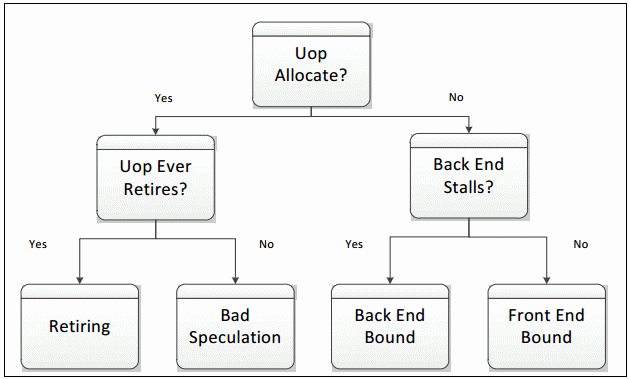
* Hotspots will profile your code's execution to determine which functions are consuming the most time and thus are targets for optimization. The hotspots analysis includes timing information from all threads and from sub-processes.
* The threading analysis analyzes how well a threaded application takes advantage of multi-core hardware and identifies functions and times during execution where available CPUs aren't fully utilized.
* The memory consumption analysis identifies RAM usage over time and identifies memory objects allocated and released during the analysis run.

VTune Amplifier uses sampling to gather profile information and should only incur a 5% execution-time overhead. There are other, more advanced analyses, such as HPC Performance Characterization, Microarchitecture Exploration, and Memory Access.

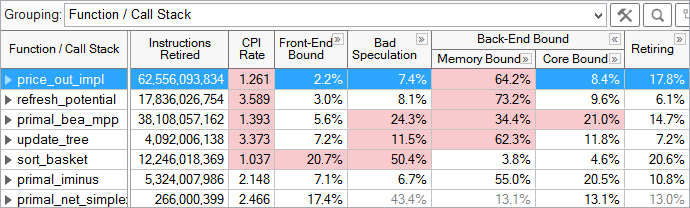
**Chapter 2: Metrics**

Intel® VTune™ Profiler provides a Microarchitecture Exploration analysis type that is pre-configured to collect the events defined in the Top-Down Characterization. The Top-Down Characterization is a hierarchical organization of event-based metrics that identifies the dominant performance bottlenecks in an application. Its aim is to show, on average, how well the CPU’s pipeline(s) were being utilized while running an application.

The pipeline of a modern high-performance CPU is quite complex. In the simplified view blow, the pipeline is divided conceptually into two halves, the Front-end and the Back-end. The Front-end is responsible for fetching the program code represented in architectural instructions and decoding them into one or more low-level hardware operations called micro-ops (uOps). The uOps are then fed to the Back-end in a process called allocation. Once allocated, the Back-end is responsible for monitoring when uOp’s data operands are available and executing the uOp in an available execution unit. The completion of a uOp’s execution is called retirement, and is where results of the uOp are committed to the architectural state (CPU registers or written back to memory). Usually, most uOps pass completely through the pipeline and retire, but sometimes speculatively fetched uOps may get cancelled before retirement – like in the case of mis-predicted branches.



Microarchitecture Exploration results are displayed in hierarchical columns to reinforce the top-down nature of the characterization. The Summary window gives the percentage of pipeline slots in each category for the whole application. You can explore results in multiple ways. The most common way to explore results is to view metrics at the function level:



For each function, the fraction of pipeline slots in each category is shown. For example, the price\_out\_impl function, selected above, had 2.2% of its pipeline slots in the Front-End Bound category, 7.4% in Bad Speculation, 64.2% in Memory Bound, 8.4% in Core Bound, and 17.8% in the Retiring category.

**Chapter 3: Microarchitectural Tuning Methodology**

When doing any performance tuning, it is important to focus on the top hotspots of the application. Hotspots are the functions taking the most CPU time. Focus on these spots will ensure that optimizations impact the overall application performance. VTune Profiler has a Hotspots analysis with two specific collection modes: user-mode sampling and hardware event-based sampling. Within the Microarchitecture Exploration viewpoint, hotspots can be identified by determining the functions or modules with the highest Clockticks event counts, which measures the number of CPU clockticks.

* Select a hotspot function (one with a large percentage of the application's total clockticks).
* Evaluate the efficiency of that hotspot using the Top-Down Method and the guidelines given below.
* If inefficient, drill down the category representing the primary bottleneck, and use the next levels of sub-bottlenecks to identify causes.
* Optimize the issue(s). VTune Profiler tuning guides contain specific tuning suggestions for many of the underlying performance issues in each category.
* Repeat until all significant hotspots have been evaluated.

