www.locuz.com

Processor Cores Underutilization: OpenMP Serial Time

Presented By: Mandeep Kumar

Converge to the Cloud

Scenario

- This recipe shows how to identify a fraction of serial execution in an application parallelized with OpenMP, discover additional opportunities for parallelization, and improve scalability of the application.

- A fraction of Serial time in a parallel application is one of the factors that limits application scalability, which is an ability of the application to utilize available hardware resources, such as cores, for executing the application code.

- When your application is parallelized with OpenMP, the sequential code execution may be a result of the code executed out of OpenMP regions or executed inside `#pragma omp master` or `#pragma omp single` constructs.

Ingredients

This section lists hardware and software tools used for the performance analysis scenario.

- **Application**: a miniFE Finite Element Mini-Application that is available for download from https://github.com/Mantevo/miniFE (OpenMP version)

- **Compiler:** Intel® Compiler 13 Update 5 and later. This recipe relies on this compiler version to have necessary instrumentation inside Intel OpenMP runtime library used by the VTune Amplifier for analysis.

- **Performance analysis tools:**
  - **Intel VTune Amplifier 2019:** HPC Performance Characterization analysis
  - **Intel® Inspector 2019:** Threading Error analysis

- **Operating system:** Linux

- **CPU:** Intel Xeon® CPU E5-2699 v4 @ 2.20GHz

Create a Baseline

- Use the openmp/src/Makefile.intel.openmp make file to build the application.

- Add -g and -parallel-source-info=2 compiler options to enable debug information and provide source file information in OpenMP region names, which makes their identification easier.

- Running the compiled application with nx=200, ny=200, and nz=200 parameters, the number of OpenMP threads corresponding to the number of physical cores and with one thread running per core (OMP_NUM_THREADS=44, OMP_PLACES=cores) takes about 12 seconds.

- This is a performance baseline that could be used for further optimizations.

Run HPC Performance Characterization Analysis

To have a high-level understanding of potential performance bottlenecks for the sample, start with the HPC Performance Characterization analysis provided by the VTune Amplifier:

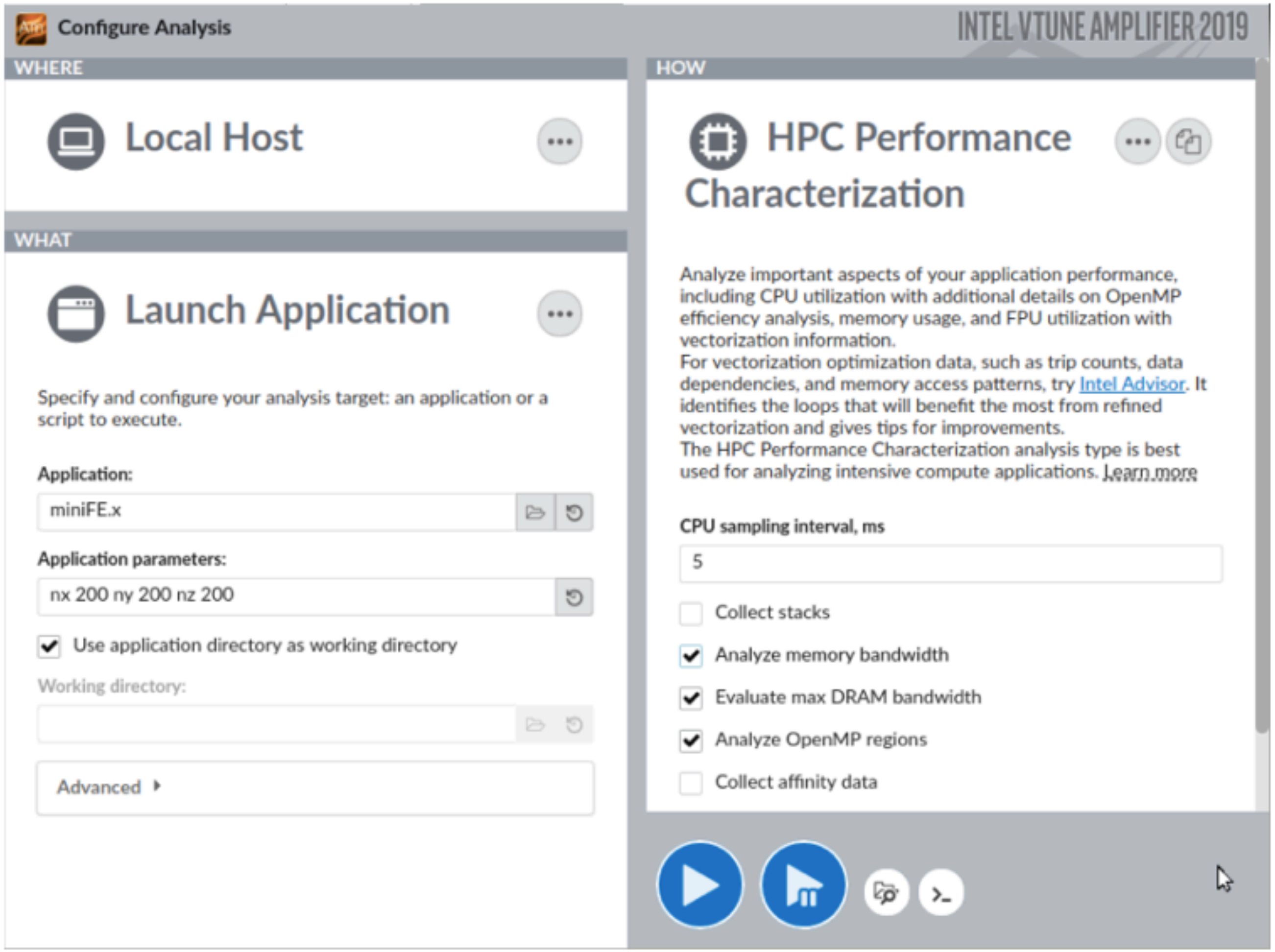
1. Click the **New Project** button on the toolbar and specify a name for the new project, for example: miniFE.
   The **Configure Analysis** window opens.

2. On the **WHERE** pane, select the **Local Host** target system type.
   On the **WHAT** pane, select the **Launch Application** target type and specify an application for analysis and its parameters: nx 200 ny 200 nz 200.

3. On the **HOW** pane, click the browse button and select **HPC Performance Characterization** from the **Parallelism** group.

4. Click the  Start button to run the analysis.

GUI



Configure Analysis
INTEL VTUNE AMPLIFIER 2019
WHERE
Local Host
WHAT
Launch Application
Specify and configure your analysis target: an application or a script to execute.
Application:
miniFE.x
Application parameters:
nx 200 ny 200 nz 200
Use application directory as working directory
Working directory:
Advanced
HOW
HPC Performance Characterization
Analyze important aspects of your application performance, including CPU utilization with additional details on OpenMP efficiency analysis, memory usage, and FPU utilization with vectorization information.
For vectorization optimization data, such as trip counts, data dependencies, and memory access patterns, try Intel Advisor. It identifies the loops that will benefit the most from refined vectorization and gives tips for improvements.
The HPC Performance Characterization analysis type is best used for analyzing intensive compute applications. Learn more
CPU sampling interval, ms
5
Collect stacks
Analyze memory bandwidth
Evaluate max DRAM bandwidth
Analyze OpenMP regions
Collect affinity data

Run the Analysis with Command Line

- You can also run the analysis from the command line:

```
amplxe-cl -collect hpc-performance -data-limit=0 ./miniFE.x nx 200 ny 200 nz 200
```

VTune Amplifier launches the application, collects data, and processes the data collection result resolving symbol information, which is required for successful source analysis.

Identify OpenMP Serial Time

- HPC Performance Characterization analysis collects and shows important HPC metrics that help understand such performance bottlenecks as CPU utilization (parallelism), memory access efficiency, and vectorization.

- For applications using the Intel OpenMP runtime, as in this recipe, you can benefit from special OpenMP efficiency metrics that help identify issues with threading parallelism.

Identify OpenMP Serial Time

- Start your analysis with the **Summary** view that displays application-level statistics. Flagged **Effective Physical Core Utilization** metric (on some systems just **CPU Utilization**) signals a performance problem that should be explored:

Effective Physical Core Utilization: 62.2% (27.374 out of 44)

Effective Logical Core Utilization: 31.5% (27.690 out of 88)

Serial Time (outside parallel regions): 3.085s (24.9%)

Top Serial Hotspots (outside parallel regions)

This section lists the loops and functions executed serially in the master thread outside of any OpenMP region and consuming the most CPU time. Improve overall application performance by optimizing or parallelizing these hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

Function	Module	Serial CPU Time
[Loop at line 133 in MatrixInitOp<miniFE::CSRMatrix<double, int, int>>::operator()]	miniFE.x	0.636s
func@0xffffffff813f5ed2	vmlinux	0.286s
[Loop at line 768 in std::__fill_n_a<double*, unsigned long, double>]	miniFE.x	0.281s
std::local_Rb_tree_decrement	libstdc++.so.6.0.21	0.271s
miniFE::find_row_for_id<int>	miniFE.x	0.195s
[Others]		1.233s

**N/A is applied to non-summable metrics.*

Parallel Region Time: 9.319s (75.1%)

Estimated Ideal Time: 7.749s (62.5%)

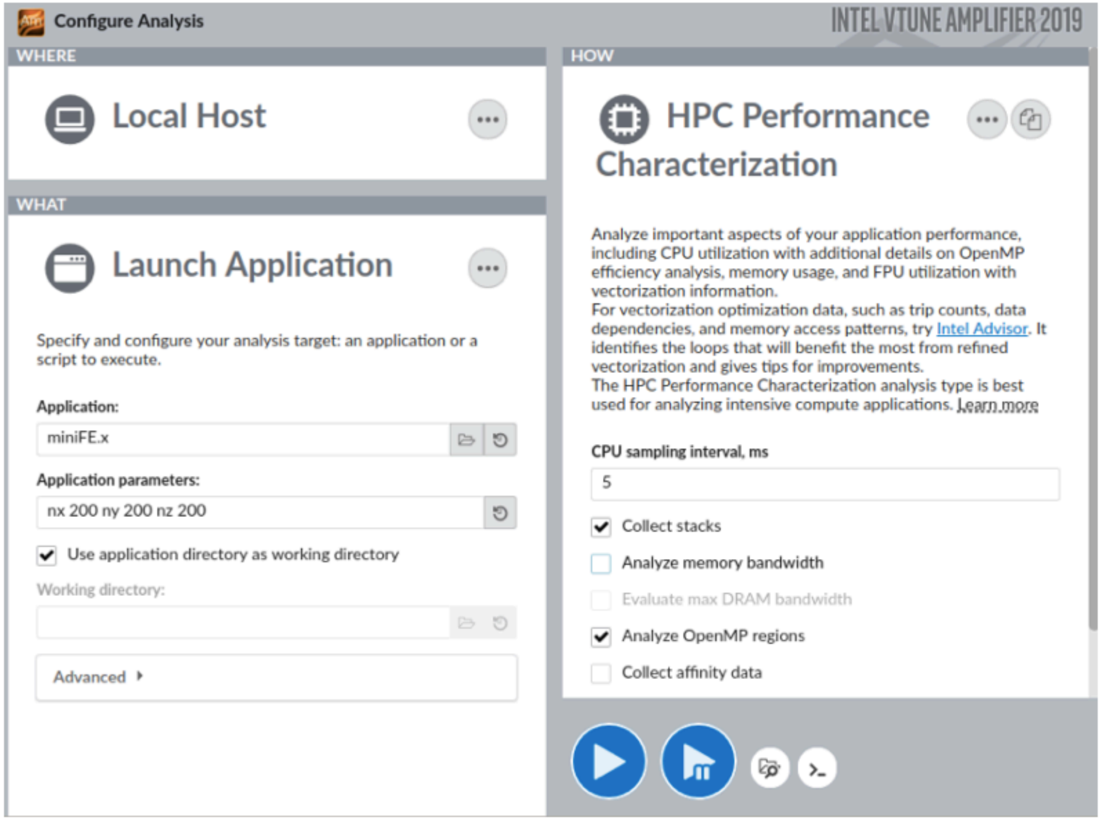
OpenMP Potential Gain: 1.571s (12.7%)

Top OpenMP Regions by Potential Gain

When you dive deeper to the metric hierarchy, you see that the Serial Time (outside parallel regions) of the application occupies ~25% of its elapsed time. The main serial hotspot is in the matrix initialization code.

Run HPC Performance Characterization Analysis with Call Stacks

- Consider running the HPC Performance Characterization analysis with call stacks to explore available optimization opportunities. Call stacks can help to find a candidate for parallelism at a proper level of granularity. Since the call stack collection is not compatible with memory bandwidth analysis, make sure to disable the **Analyze memory bandwidth** configuration option:



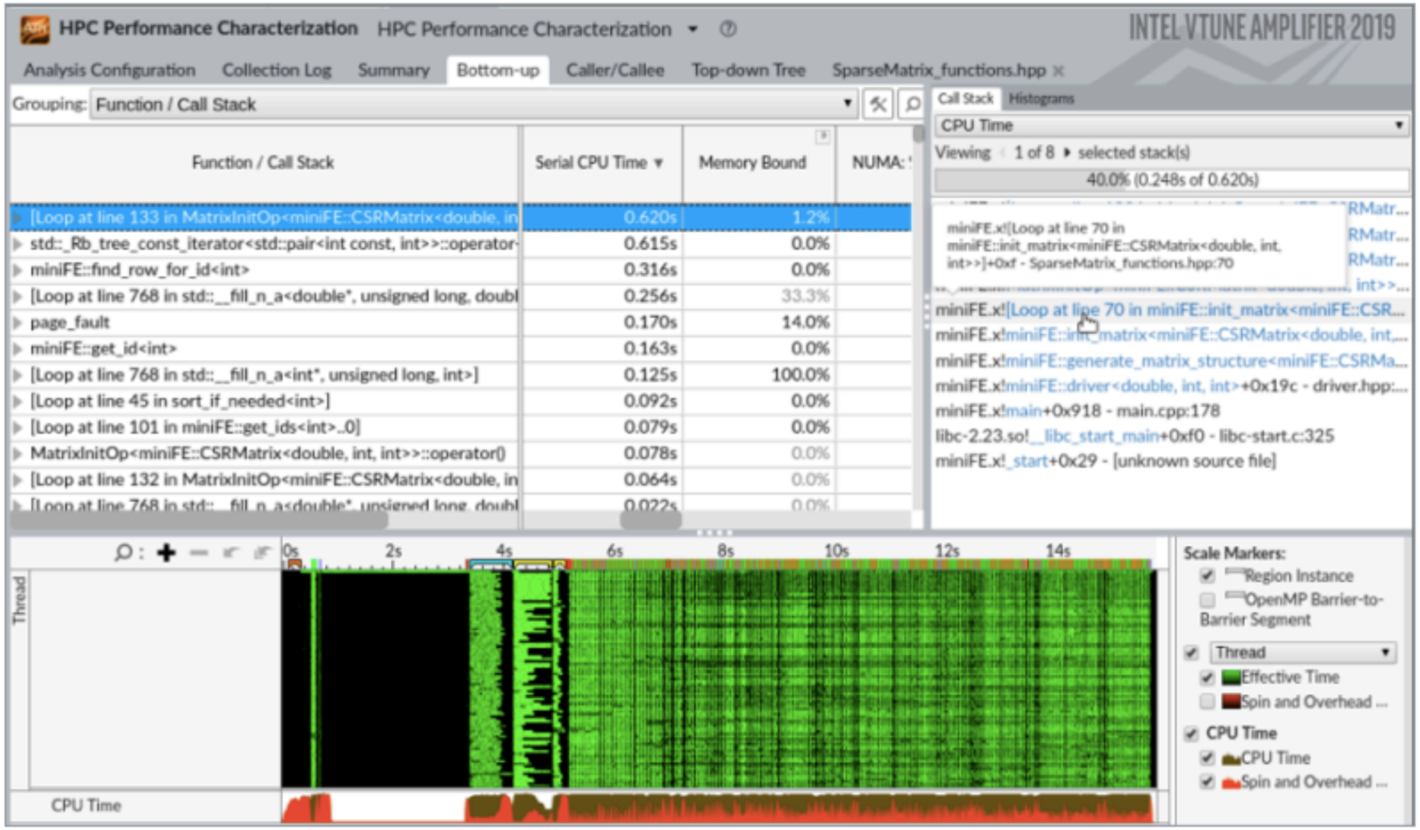
Run the Analysis with Command Line

- To run the same configuration from the command line, enter:

```
amplxe-cl -collect hpc-performance -data-limit=0 -knob enable-stack-collection=true -knob
collect-memory-bandwidth=false ./miniFE.x nx 200 ny 200 nz 200
```

Identify top hotspots and explore their call stacks

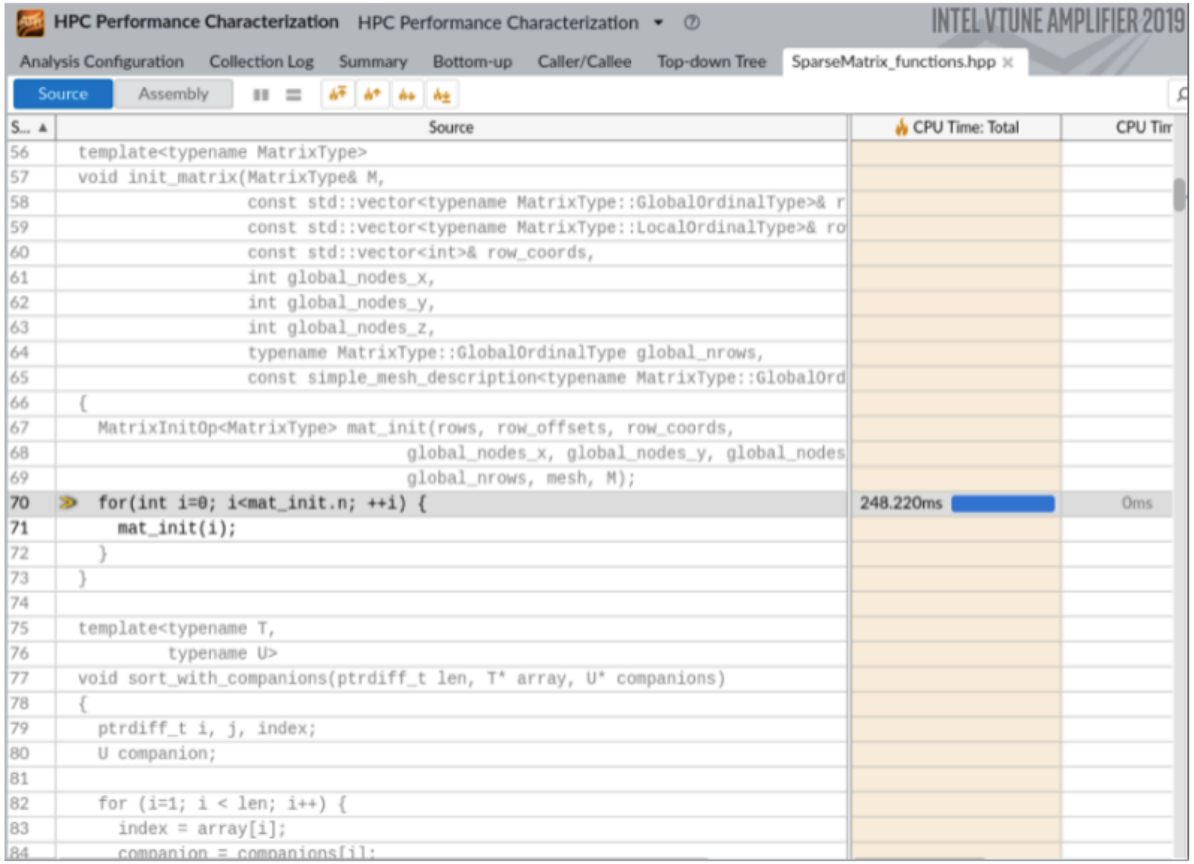
- To identify top hotspots and explore their call stacks, switch to the Bottom-up view and sort it by Serial CPU Time column:



You see that the right place to insert parallelism is line 70 in SparseMatrix_functions.hpp where there is a loop with iterations by matrix elements.

Top hotspot loop

- Double-click the row on the **Call Stack** pane to open the source file automatically positioned on the hottest line of the top hotspot loop:



Parallelize the Code

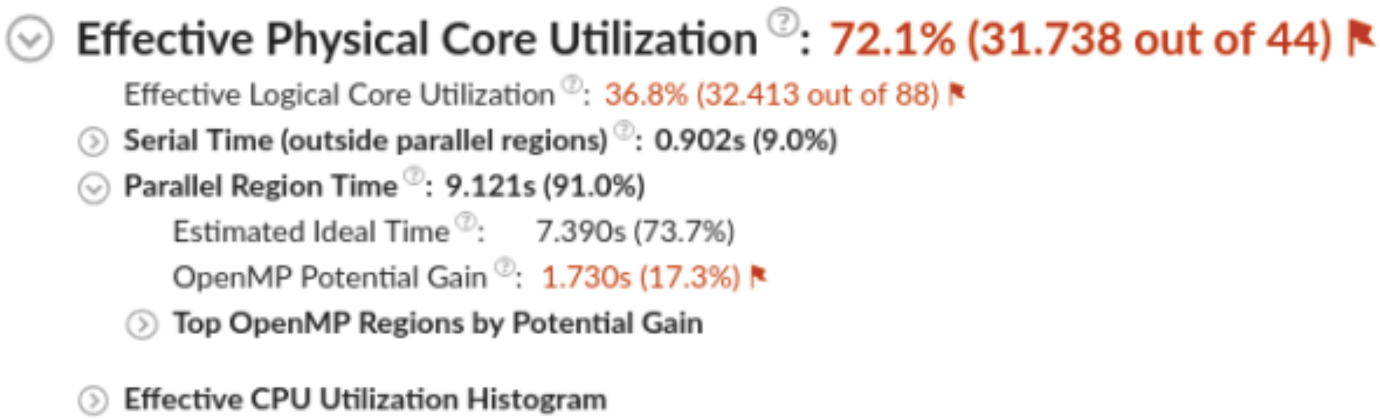
- Insert the omp parallel for pragma to make the matrix initialization parallelized by OpenMP:

```
#pragma omp parallel for
for(int i=0; i<mat_init.n; ++i) {
   mat_init(i);
```

- Re-compile the application and compare the execution time versus the original performance baseline to verify your optimization.
- In this recipe, the Elapsed time of the application after optimization is approximately 10 seconds, which is ~16% speed-up of the application execution.

Re-run the HPC Performance Characterization analysis

- Re-run the HPC Performance Characterization analysis for the optimized version of the application:



Overall **Effective Physical Core Utilization** has improved by 10%. The fraction of OpenMP Serial Time is reduced to 9% and it is not flagged by VTune Amplifier as an issue (the threshold for the metric is 15%).

Inspect Threading Errors

- To complete your analysis for parallelism, check your code for threading errors like data races or deadlocks.

- To do this, the recipe uses the Intel Inspector, which is able to find even potential data races and deadlocks that might not happen on application runs on particular hardware but can hurt in another environment or even on the same environment with different settings.

- If you use the command line interface and reduce the workload size, the check runs faster but still is representative:

```
inspxe-cl -collect ti3 ./miniFE.x nx 40 ny 40 nz 40
```

Intel Inspector Summary

- You see that the Intel Inspector does not report any issues for the parallelized code.

Thanks!