CS341: Computer Architecture Lab

Lab Assignment 4

Report

Devansh Jain (190100044)

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
2021-2022

Contents

Abstract

Summarize the objective of the lab, what experiments you have conducted, what were the results that you have obtained in a clear and concise manner. Numbers matter, not just words only, for ex. *very high*, *slow* etc.

Part 0: Getting Things Ready

Install `Intel VTune Profiler`

Installed successfully using the stand-alone app using offline installer script present in this link.

It was pretty easy to install `VTune` using the script.
While installing it showed that I didn't have `XCB` and `DRM` packages installed. Upon checking, I confirmed that they were already present.
Even though it failed prerequisites, there was a next option. I didn't face any issues for the rest of installation process.
From start to end, it took around 10-12 minutes to have the application installed. Followed by 3-5 minutes for tutorial.

Install `Docker`

Had docker setup from other projects.
Version: 20.10.9

Pull `ChampSim` Image

Pulled `0xd3ba/champsim-lab:latest`

Part 1: Profiling with VTune

1.1 `bfs.cpp`

Performance Snapshot

- IPC: 1.830
- Logical Core Utilization: 8.2% (0.979 out of 12)
- Physical Core Utilization: 16.2% (0.973 of 6)
- Memory bound: 32.0% of Pipeline slots

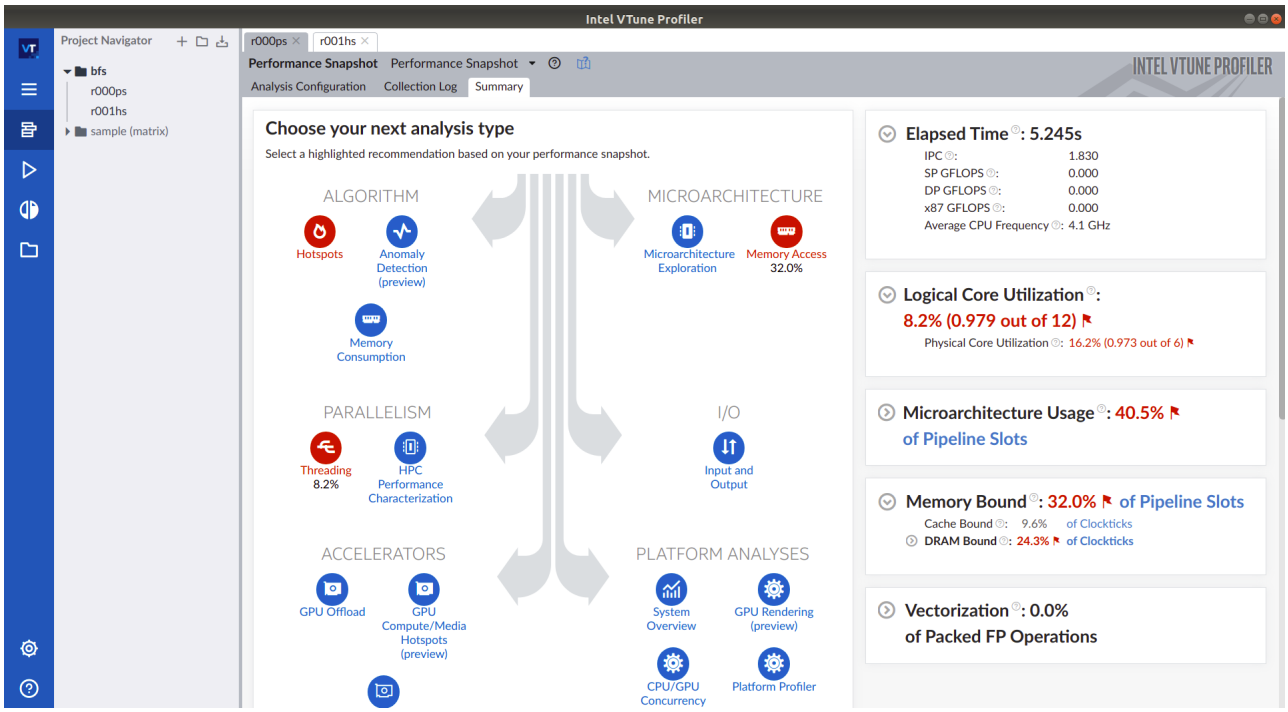

Figure 1.1: Performance Snapshot for `bfs.cpp`

Top 5 Functions by CPU Time

Function	Module	CPU Time
`bfs`	`bfs.o`	2.621s
`main`	`bfs.o`	1.156s
`_int_free`	`libc-2.27.so`	0.236s
`_int_malloc`	`libc-2.27.so`	0.154s
`__gnu_cxx::new_allocator<Node*>::construct<Node*, Node* const&>`	`bfs.o`	0.124s

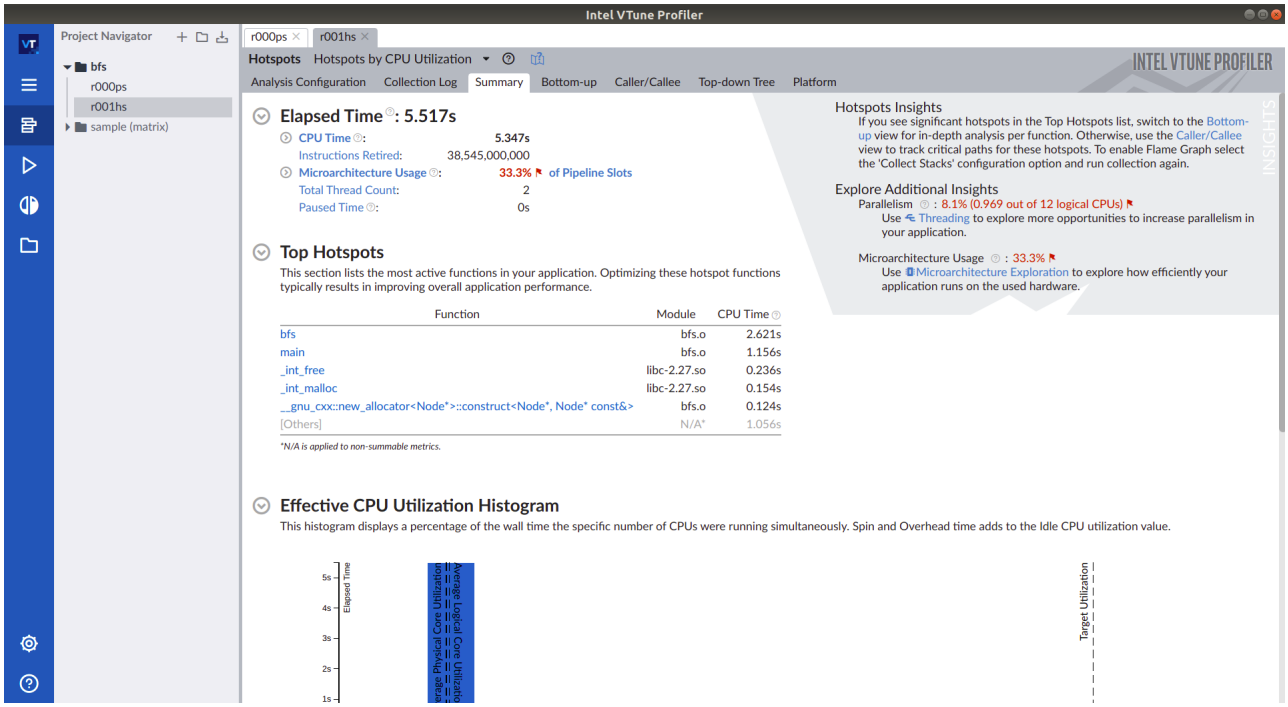

Figure 1.2: Top Functions by CPU Time for `bfs.cpp`

Top 5 Source lines by CPU Utilization

Source	Function	CPU Utilization
`if (left_child) node_Q.push(left_child);`	`inline void bfs(Node *root)`	22.7%
`bfs(root);`	`int main()`	21.6%
`right_child = curr_node->right;`	`inline void bfs(Node *root)`	17.2%
`for (int i = 0; i < q_size; i++) {`	`inline void bfs(Node *root)`	4.8%
`left_child = curr_node->left;`	`inline void bfs(Node *root)`	3.2%

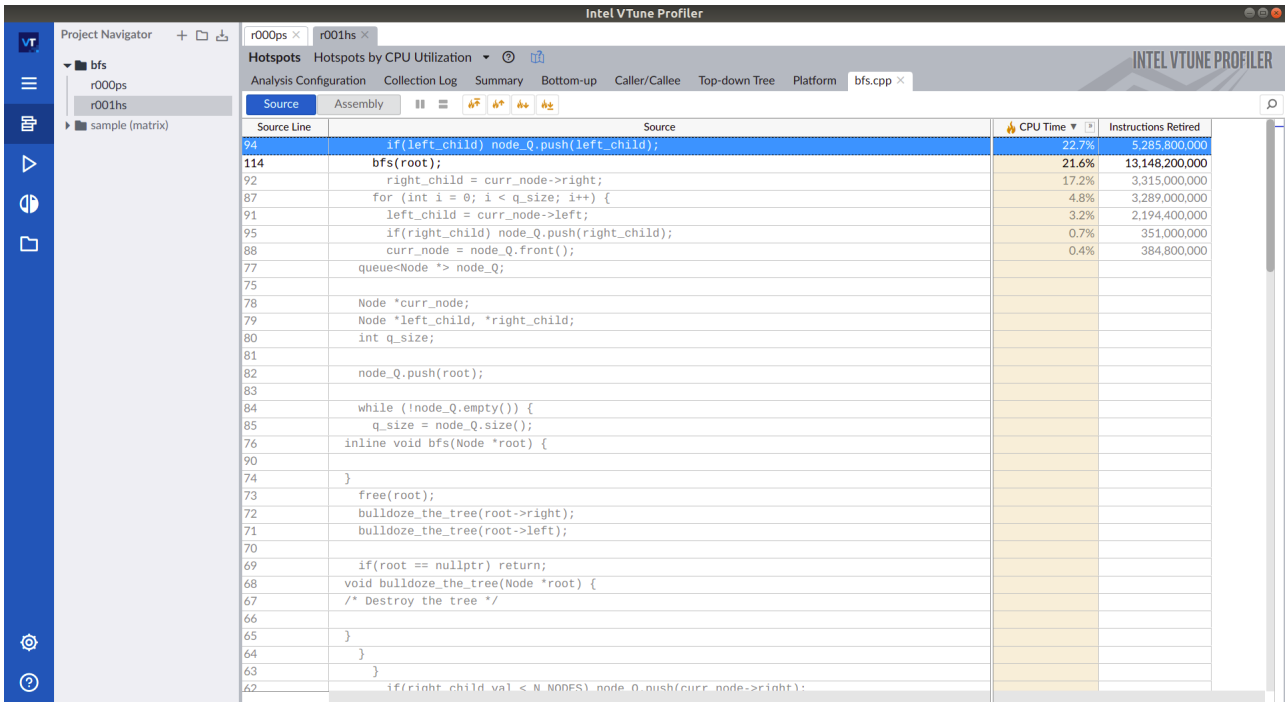

Figure 1.3: Top Source lines by CPU Utilization for `bfs.cpp`

1.2 `matrix_multi.cpp`

Performance Snapshot

- IPC: 0.874
- Logical Core Utilization: 8.2% (0.982 out of 12)
- Physical Core Utilization: 16.3% (0.976 of 6)
- Memory bound: 59.1% of Pipeline slots

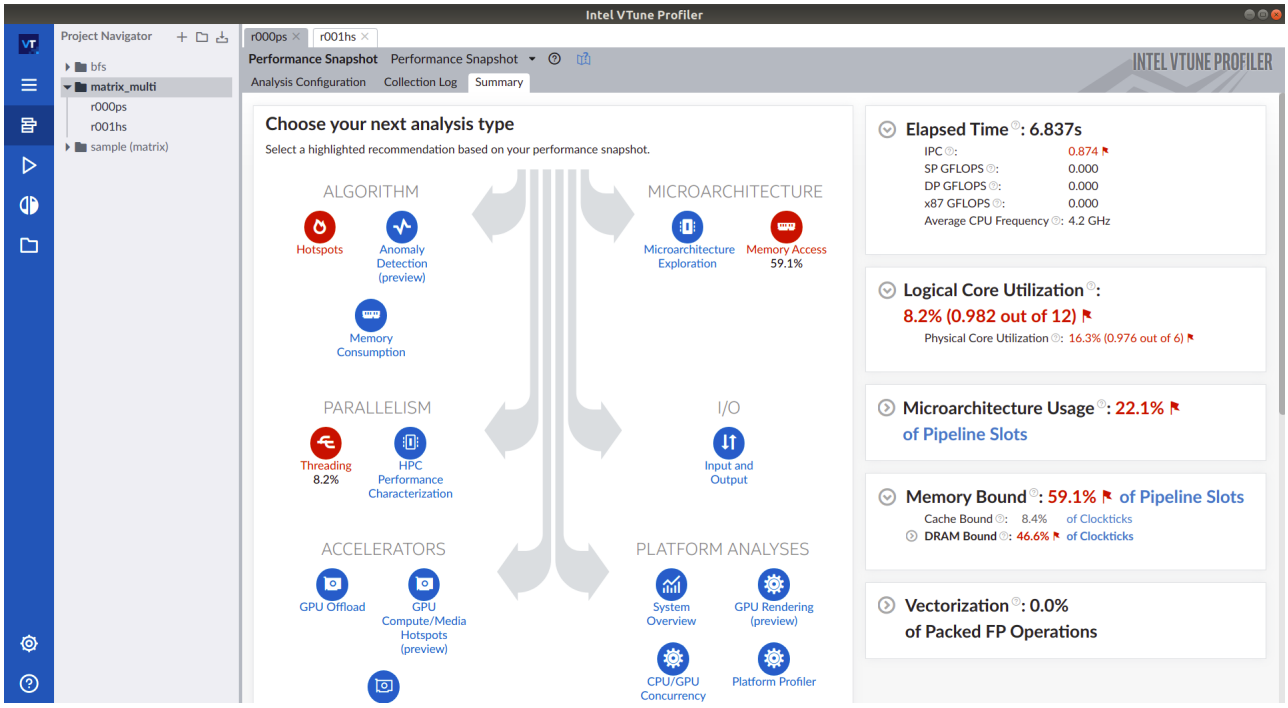

Figure 1.4: Performance Snapshot for `matrix_multi.cpp`

Top Functions by CPU Time

Function	Module	CPU Time
`matrix_product`	`matrix_multi.o`	6.597s

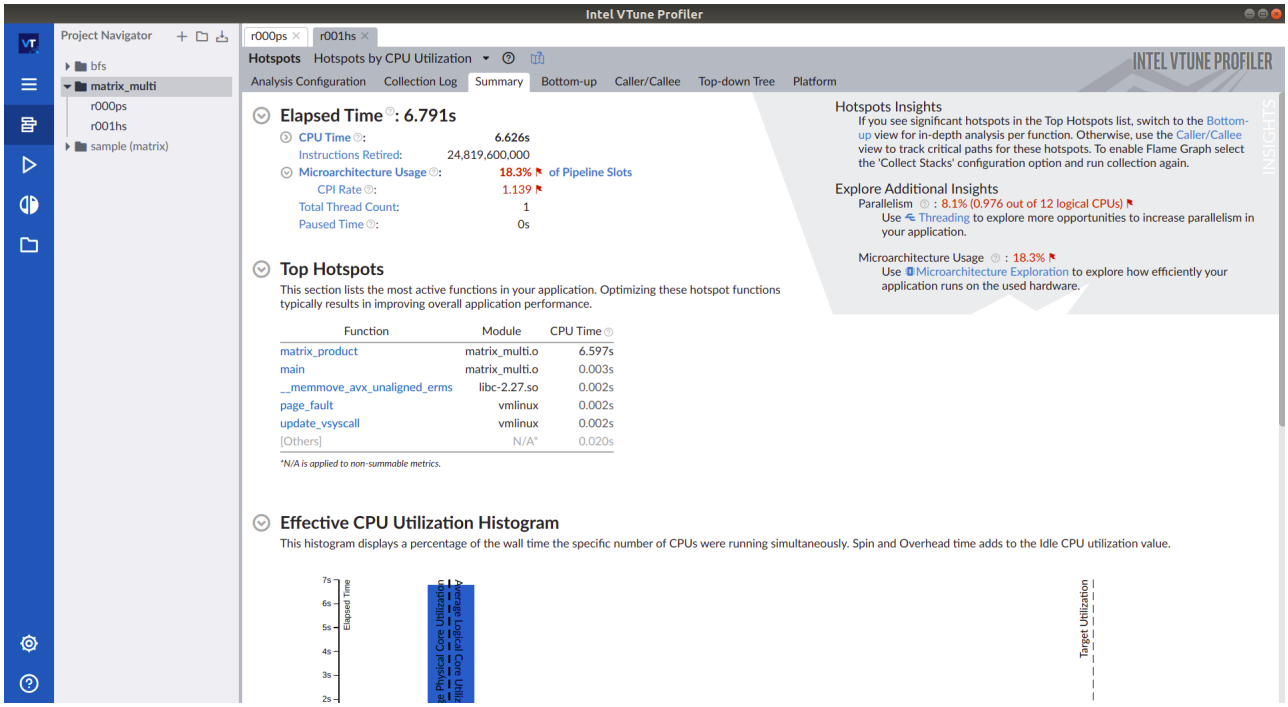

Figure 1.5: Top Functions by CPU Time for `matrix_multi.cpp`

Top 2 Source lines by CPU Utilization

Source	Function	CPU Utilization
`C[i][j] += A[i][k] * B[k][j];`	`void matrix_product()`	90.9%
`for (int k = 0; k < N_DIMS; k++) {`	`void matrix_product()`	8.6%

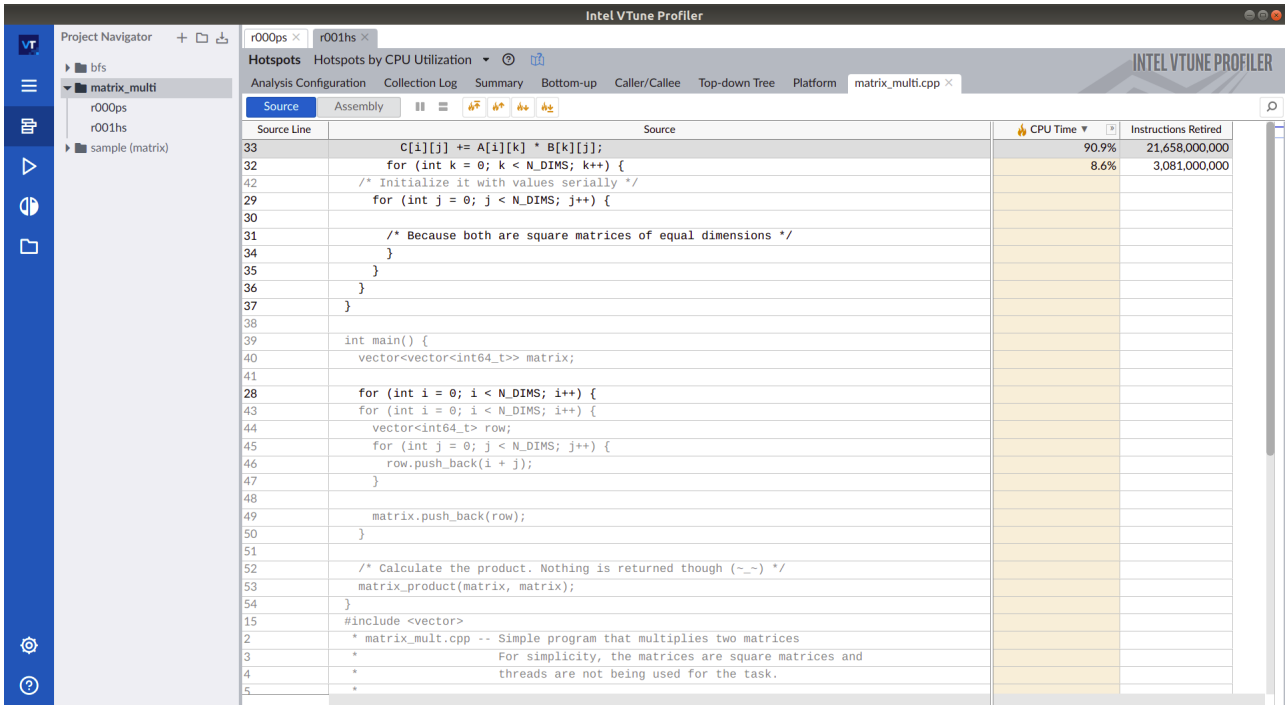

Figure 1.6: Top Source lines by CPU Utilization for `matrix_multi.cpp`

1.3 `matrix_multi_2.cpp`

Performance Snapshot

- IPC: 1.339
- Logical Core Utilization: 8.2% (0.981 out of 12)
- Physical Core Utilization: 16.2% (0.973 of 6)
- Memory bound: 38.0% of Pipeline slots

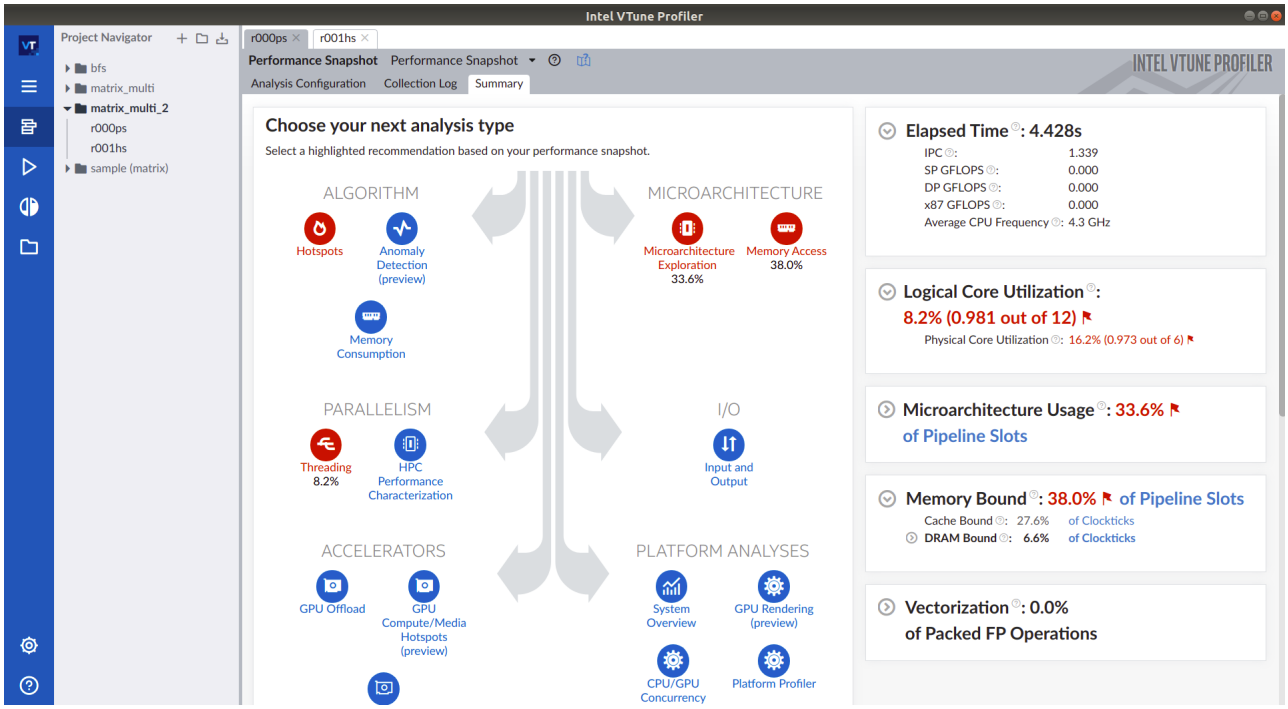

Figure 1.7: Performance Snapshot for `matrix_multi_2.cpp`

Top Functions by CPU Time

Function	Module	CPU Time
`matrix_product`	`matrix_multi_2.o`	4.492s

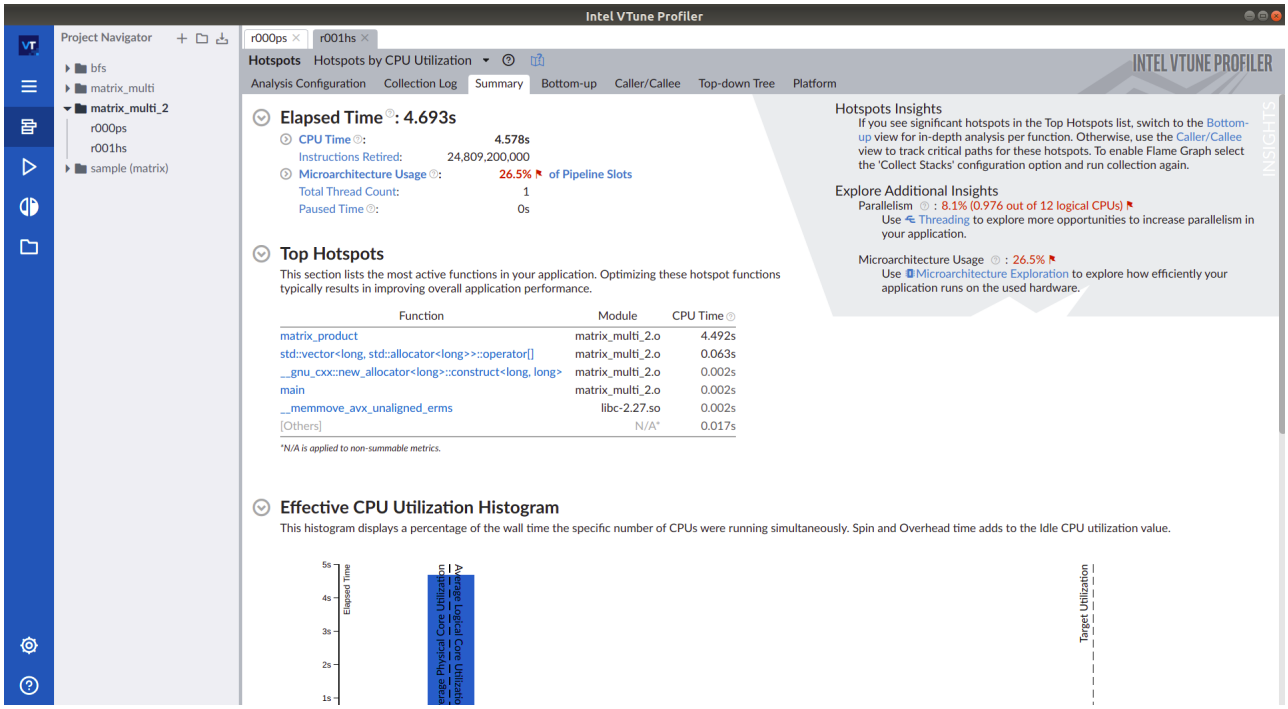

Figure 1.8: Top Functions by CPU Time for `matrix_multi_2.cpp`

Top 2 Source lines by CPU Utilization

Source	Function	CPU Utilization
`C[i][j] += A[i][k] * B[k][j];`	`void matrix_product()`	84.4%
`for (int k = 0; k < N_DIMS; k++) {`	`void matrix_product()`	13.7%

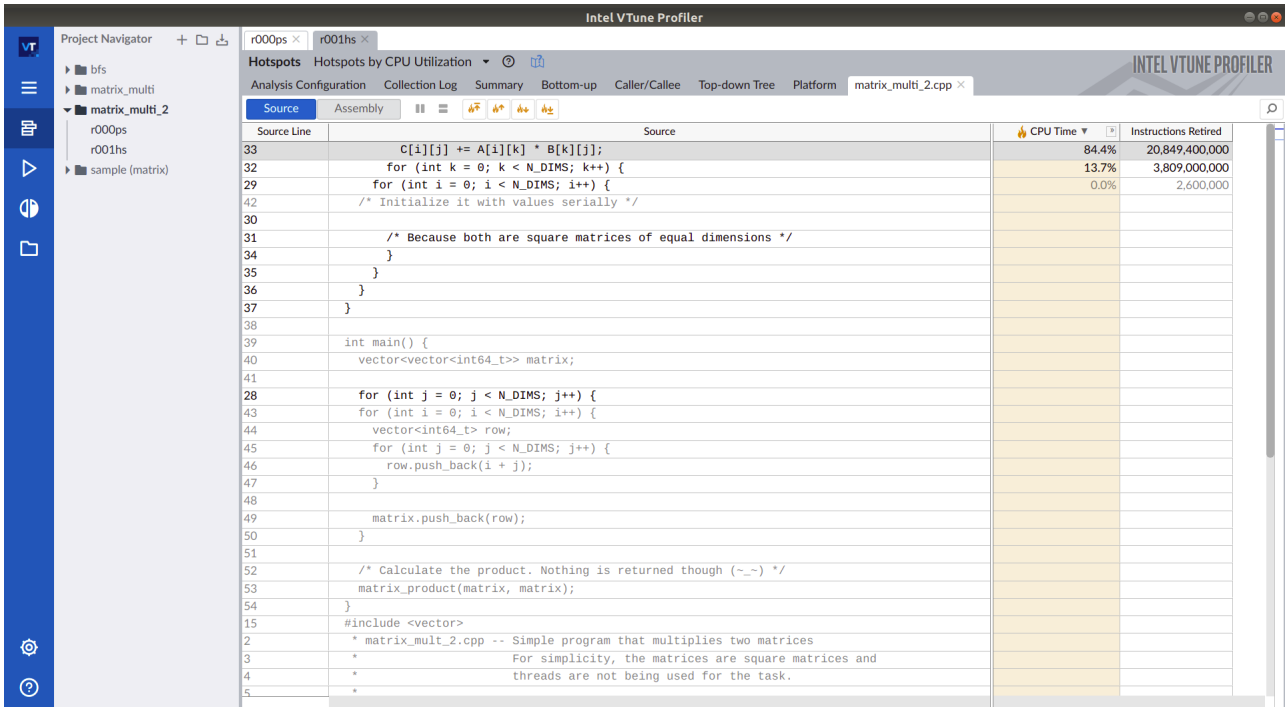

Figure 1.9: Top Source lines by CPU Utilization for `matrix_multi_2.cpp`

1.4 `quicksort.cpp`

Performance Snapshot

- IPC: 0.748
- Logical Core Utilization: 8.0% (0.966 out of 12)
- Physical Core Utilization: 15.7% (0.941 of 6)
- Memory bound: 23.0% of Pipeline slots

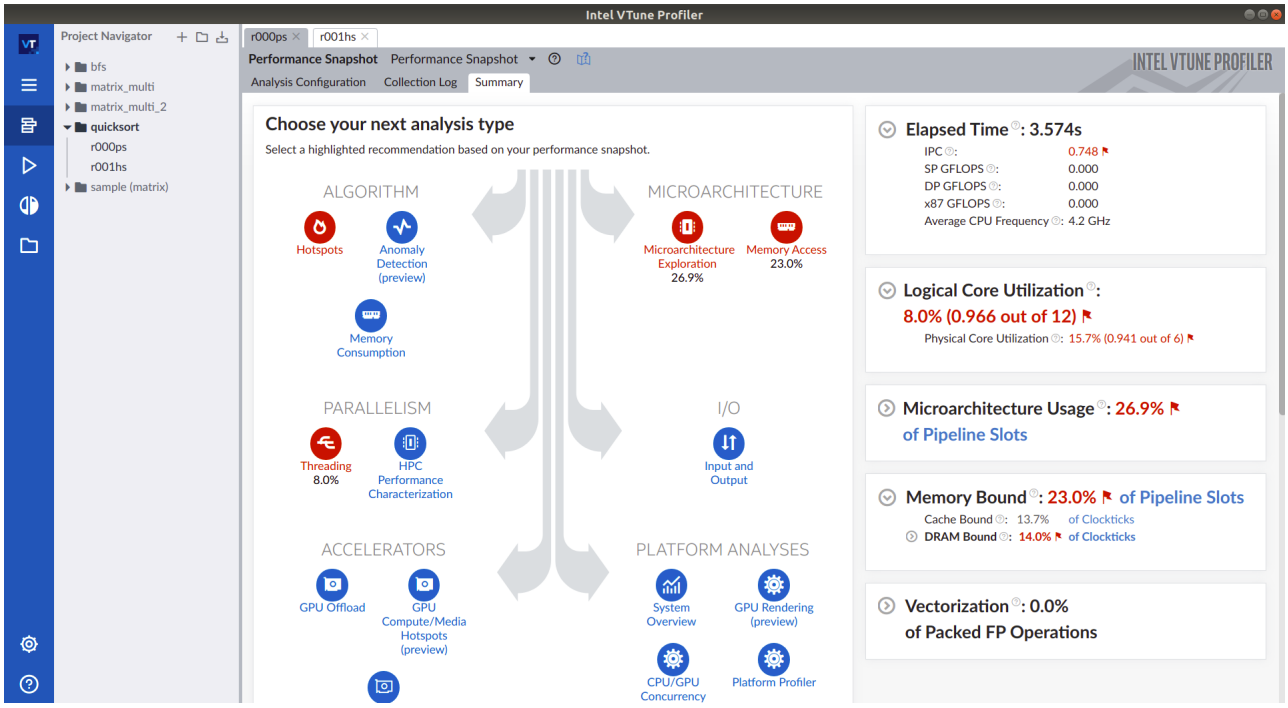

Figure 1.10: Performance Snapshot for `quicksort.cpp`

Top Functions by CPU Time

Function	Module	CPU Time
`__memmove_avx_unaligned_erms`	`libc-2.27.so`	0.875s
`page_fault`	`vmlinux`	0.511s
`clear_page_erms`	`vmlinux`	0.228s
`prepare_exit_to_usermode`	`vmlinux`	0.226s
`perf_iterate_ctx`	`vmlinux`	0.155s
Others	N/A	1.545s

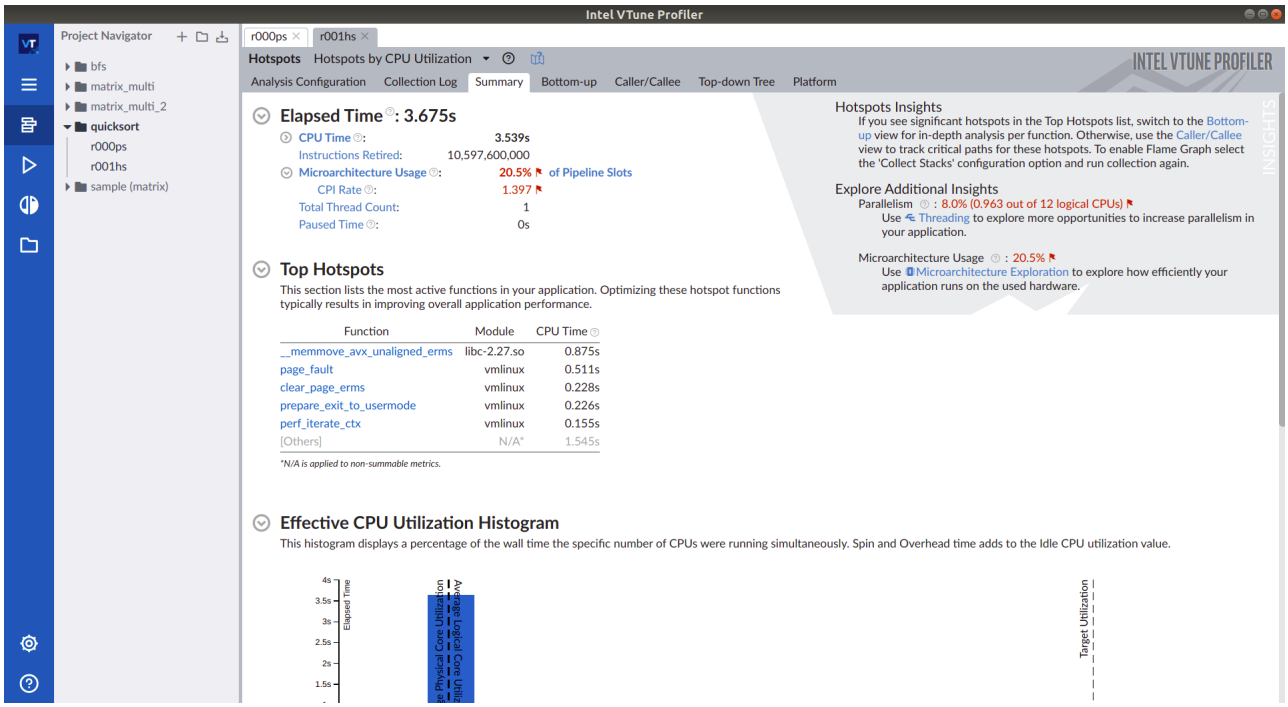

Figure 1.11: Top Functions by CPU Time for `quicksort.cpp`

Top 5 Source lines by CPU Utilization

Source	Function	CPU Utilization
`b = c;`	`void swap()`	2.1%
`if (nums[i] < pivot) {`	`long partition()`	1.9%
`slow_ptr++;`	`long partition()`	1.7%
`for (long i = lo; i < hi; i++) {`	`long partition()`	0.6%
`a = b;`	`void swap()`	0.2%

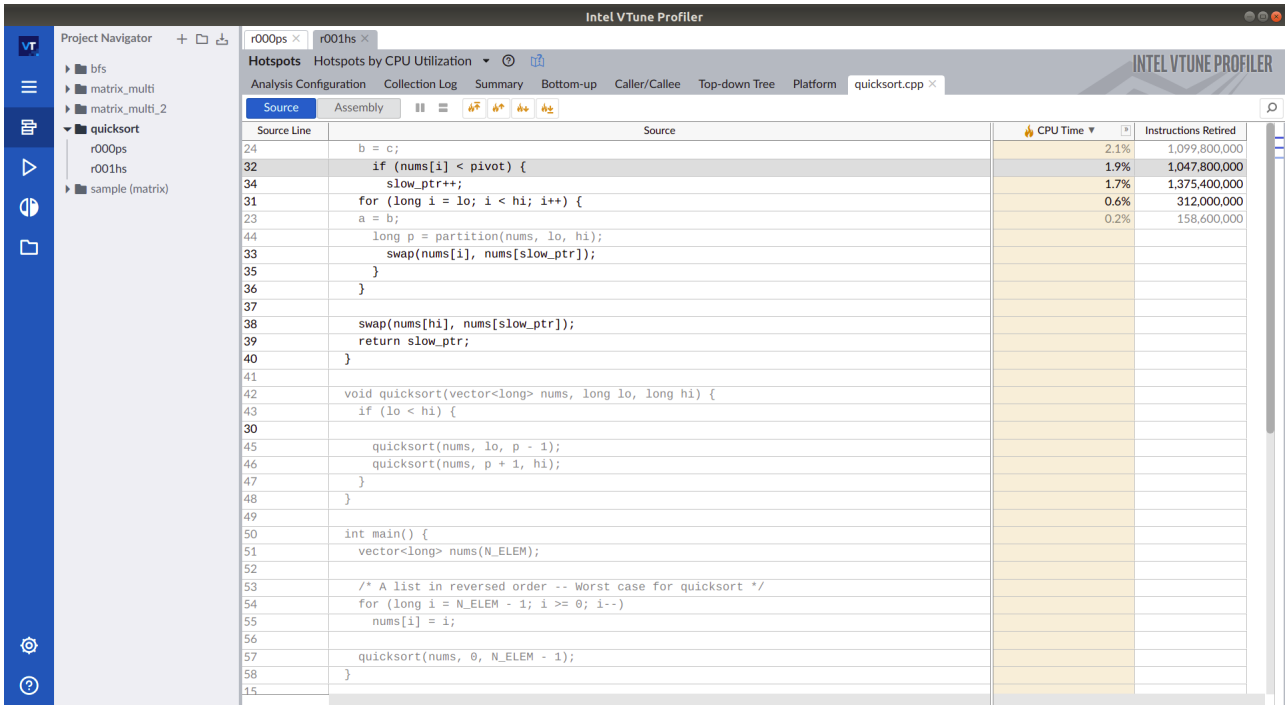

Figure 1.12: Top Source lines by CPU Utilization for `quicksort.cpp`

Part 2: Simulating with ChampSim

2.1 Prepare traces

Generate tracer for champsim:
`cd /champsim/ChampSim/tracer; ./make_tracer.sh;`
Used pin to generate traces:
`pin -t obj-intel64/champsim_tracer.so -o <program>.trace -t 21000000 - <executable>;`
Used xz to compress the traces:
`xz -vz <program>.trace -threads=0;`

Program	Parameters	Execution time	Trace size
`bfs.o`	`N_NODES (1«15)`; `N_LOOPS 1000`;	3.4 s	2092 KB
`matrix_multi.o`	`N_DIMS 700`;	4.5 s	2172 KB
`matrix_multi_2.o`	`N_DIMS 700`;	4.2 s	2160 KB
`quicksort.o`	`N_ELEM (1«14)`;	3.1 s	4172 KB