

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

| | | |
|----------|---------------------------------|-----------|
| 0 | Getting Things Ready | 2 |
| 1 | Profiling with VTune | 3 |
| 1.1 | bfs.cpp | 3 |
| 1.2 | matrix_multi.cpp | 6 |
| 1.3 | matrix_multi_2.cpp | 9 |
| 1.4 | quicksort.cpp | 12 |
| 2 | Simulating with ChampSim | 15 |
| 2.1 | Prepare traces | 15 |
| 2.2 | Setup Configurations | 15 |

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

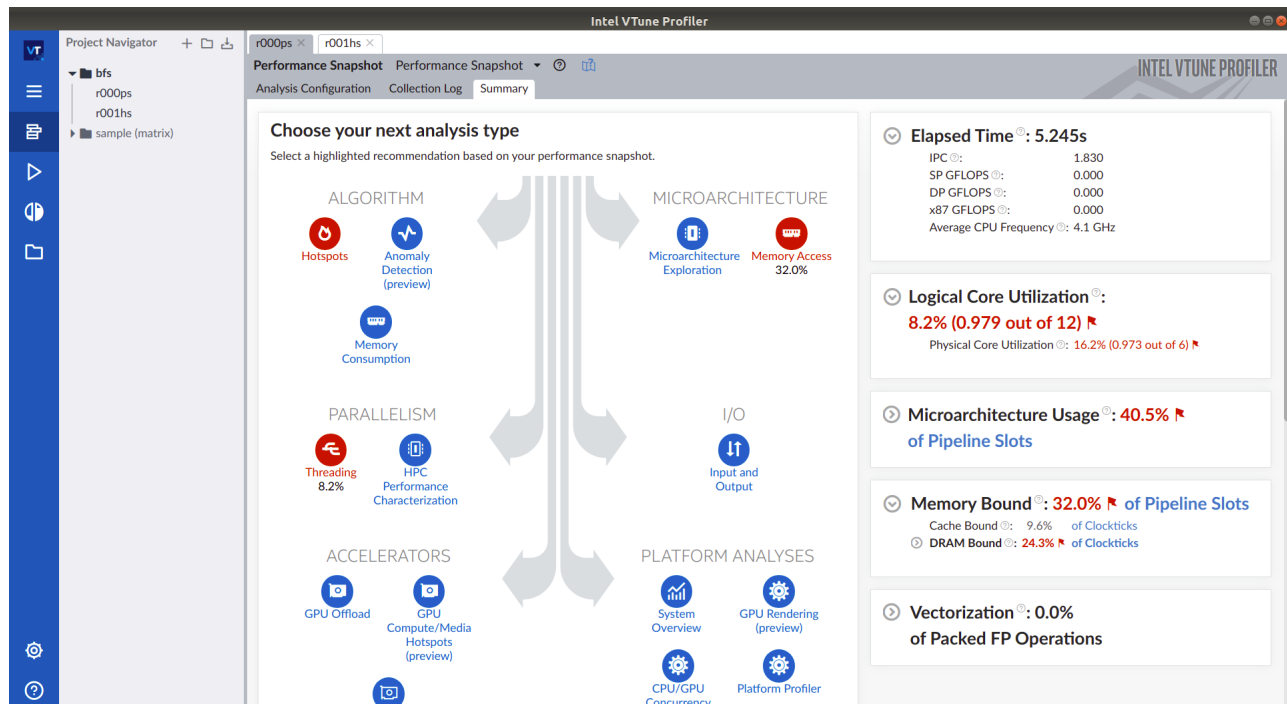


Figure 1.1: Performance Snapshot for bfs.cpp

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 |

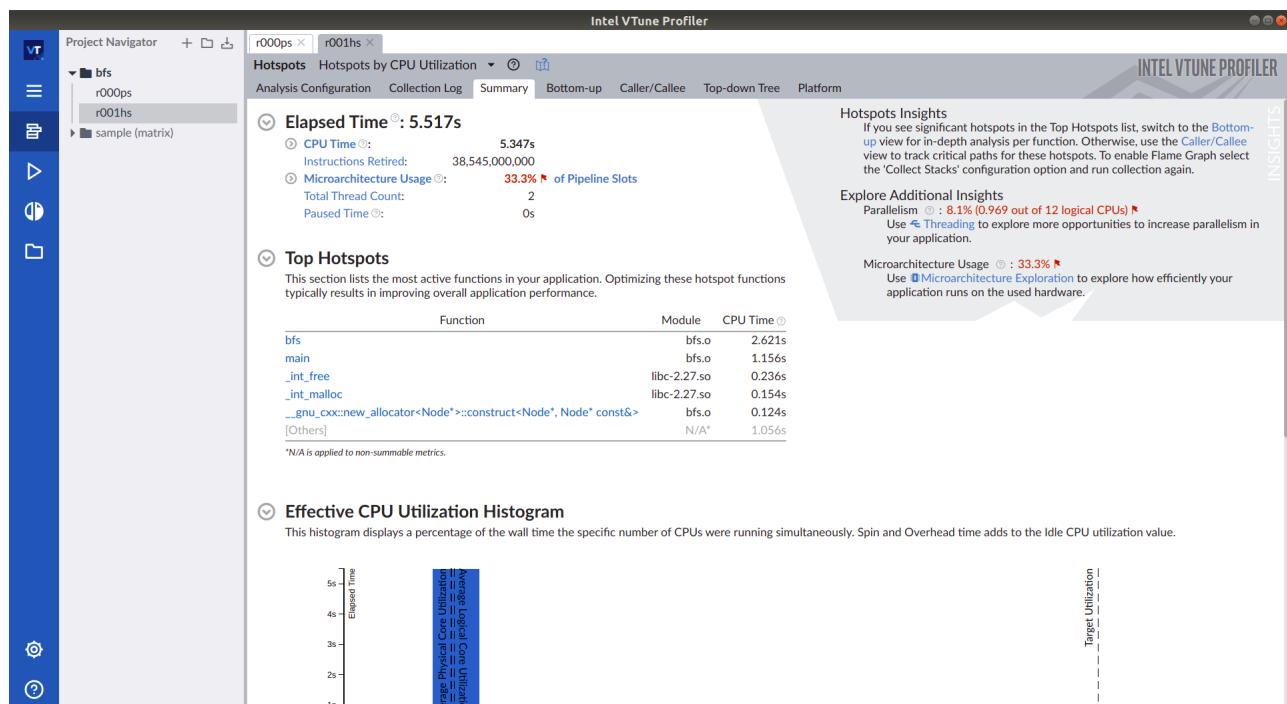


Figure 1.2: Top Functions by CPU Time for bfs.cpp

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% |

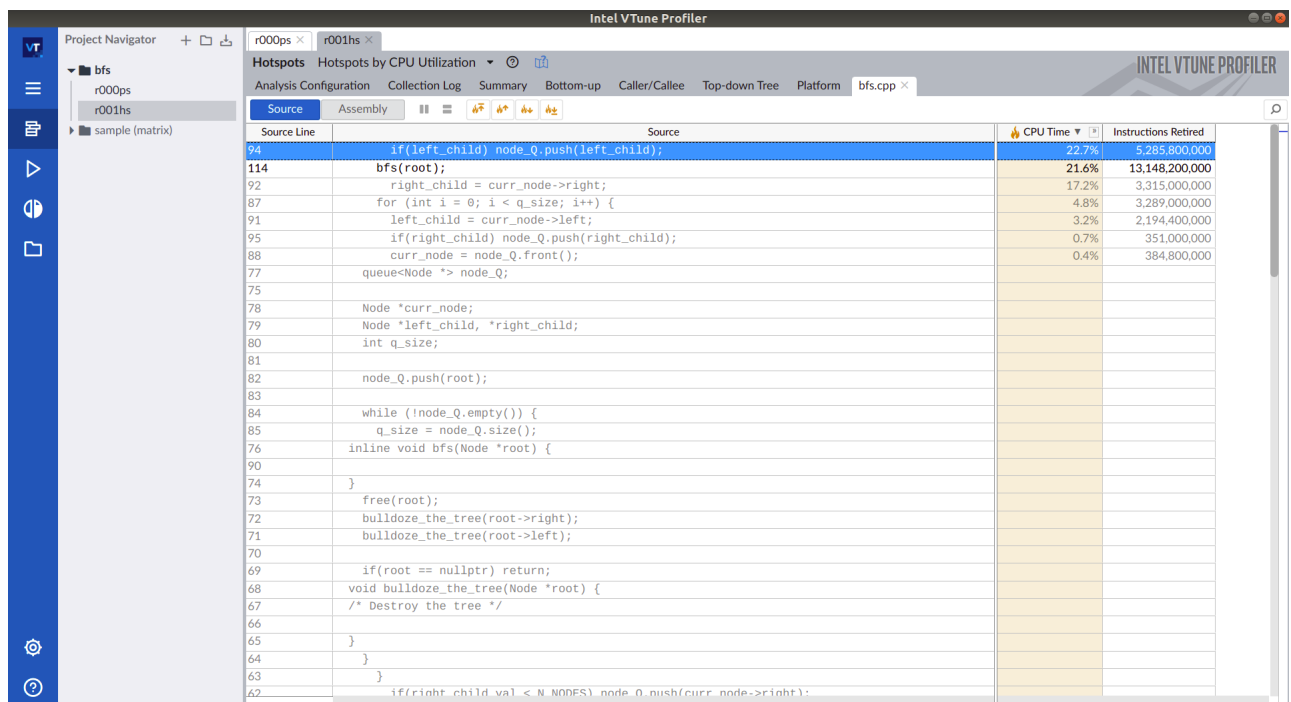


Figure 1.3: Top Source lines by CPU Utilization for bfs.cpp

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

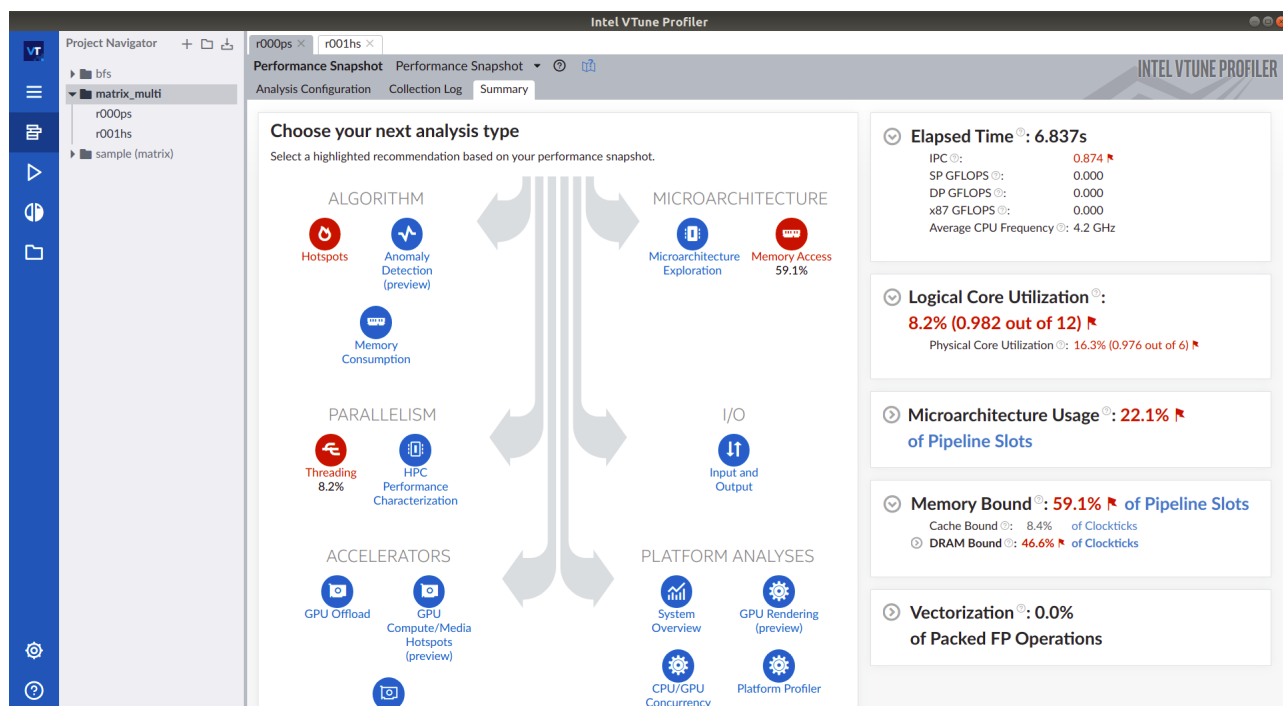


Figure 1.4: Performance Snapshot for `matrix_multi.cpp`

Top Functions by CPU Time

| Function | Module | CPU Time |
|----------------|----------------|----------|
| matrix_product | matrix_multi.o | 6.597s |

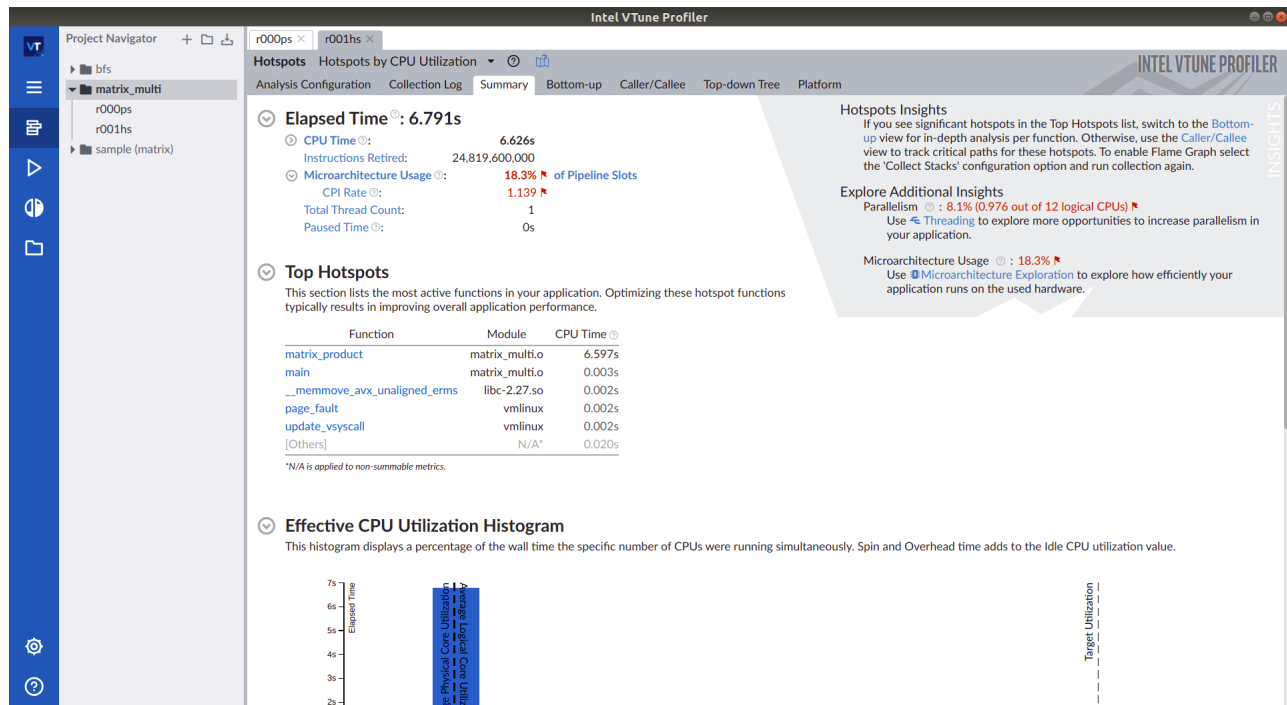


Figure 1.5: Top Functions by CPU Time for matrix_multi.cpp

Top 2 Source lines by CPU Utilization

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

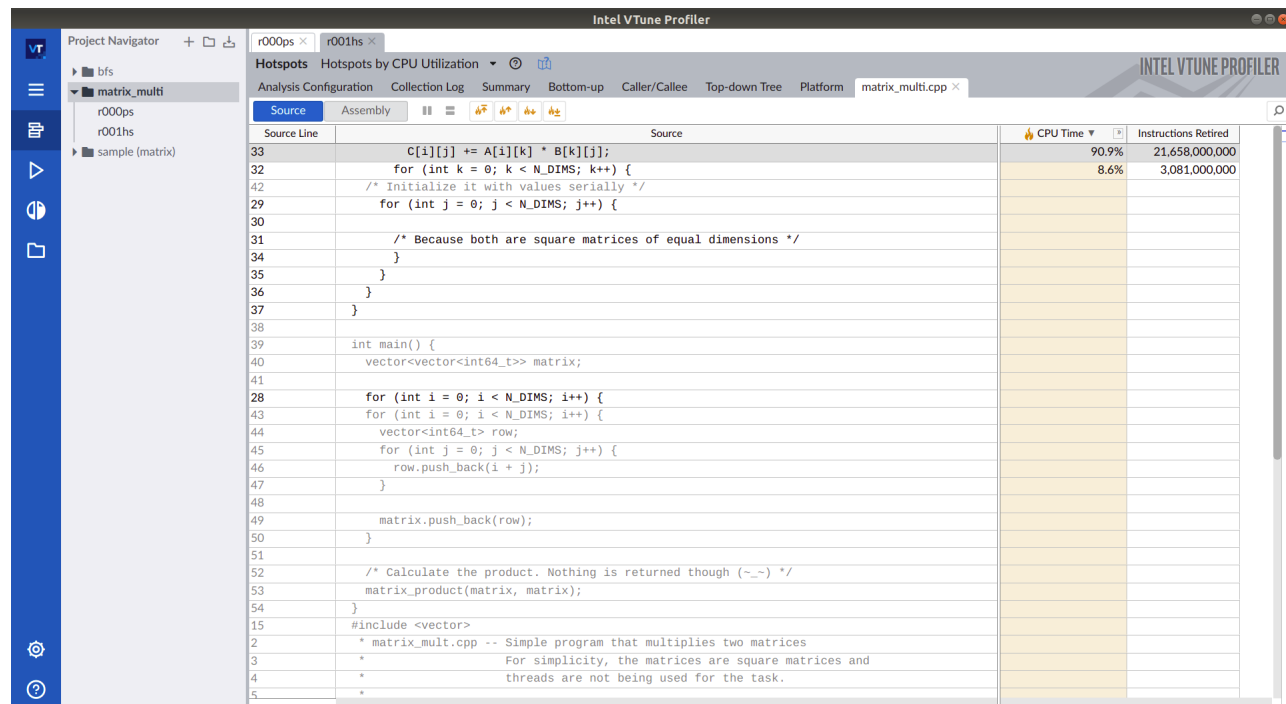


Figure 1.6: Top Source lines by CPU Utilization for `matrix_multi.cpp`

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

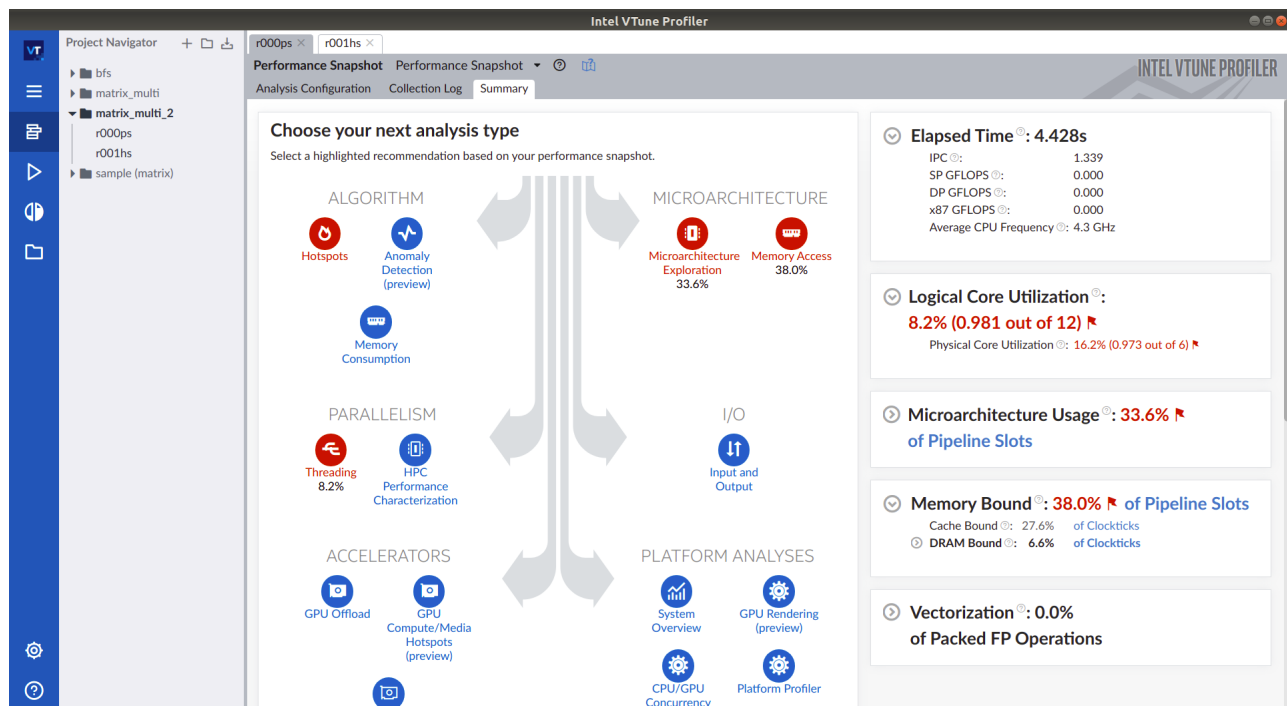


Figure 1.7: Performance Snapshot for `matrix_multi_2.cpp`

Top Functions by CPU Time

| Function | Module | CPU Time |
|----------------|------------------|----------|
| matrix_product | matrix_multi_2.o | 4.492s |

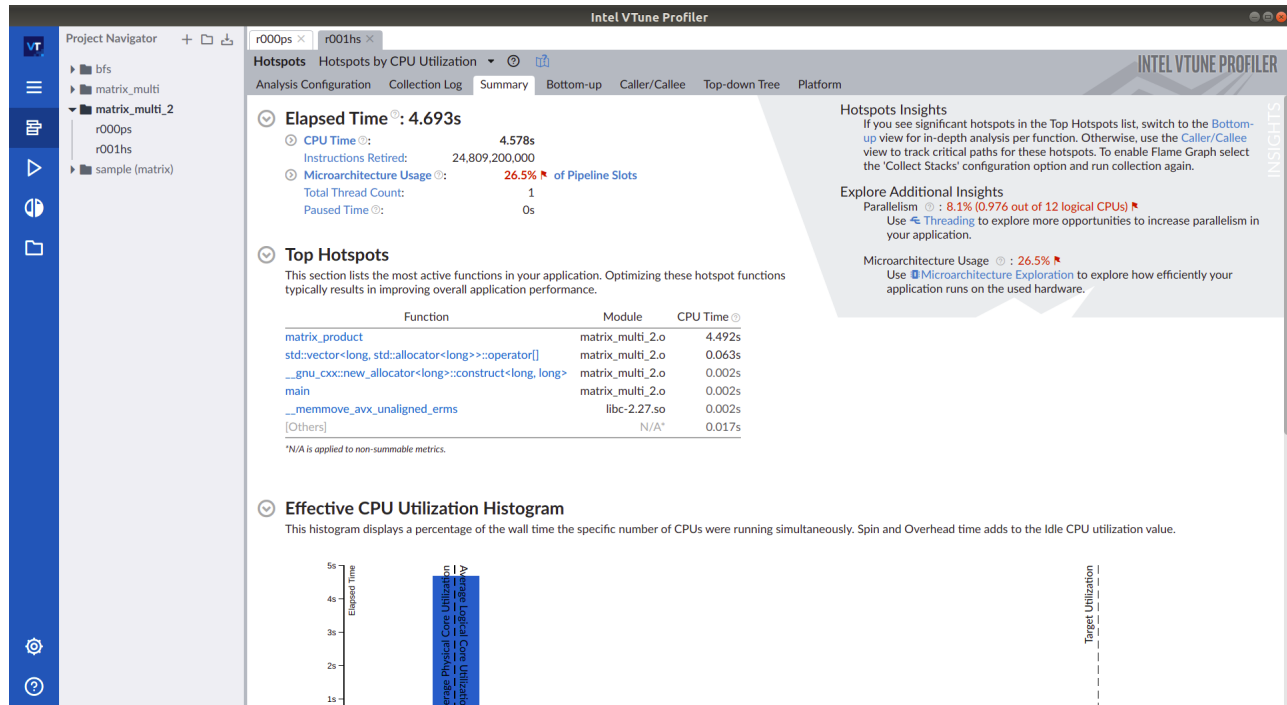


Figure 1.8: Top Functions by CPU Time for matrix_multi_2.cpp

Top 2 Source lines by CPU Utilization

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

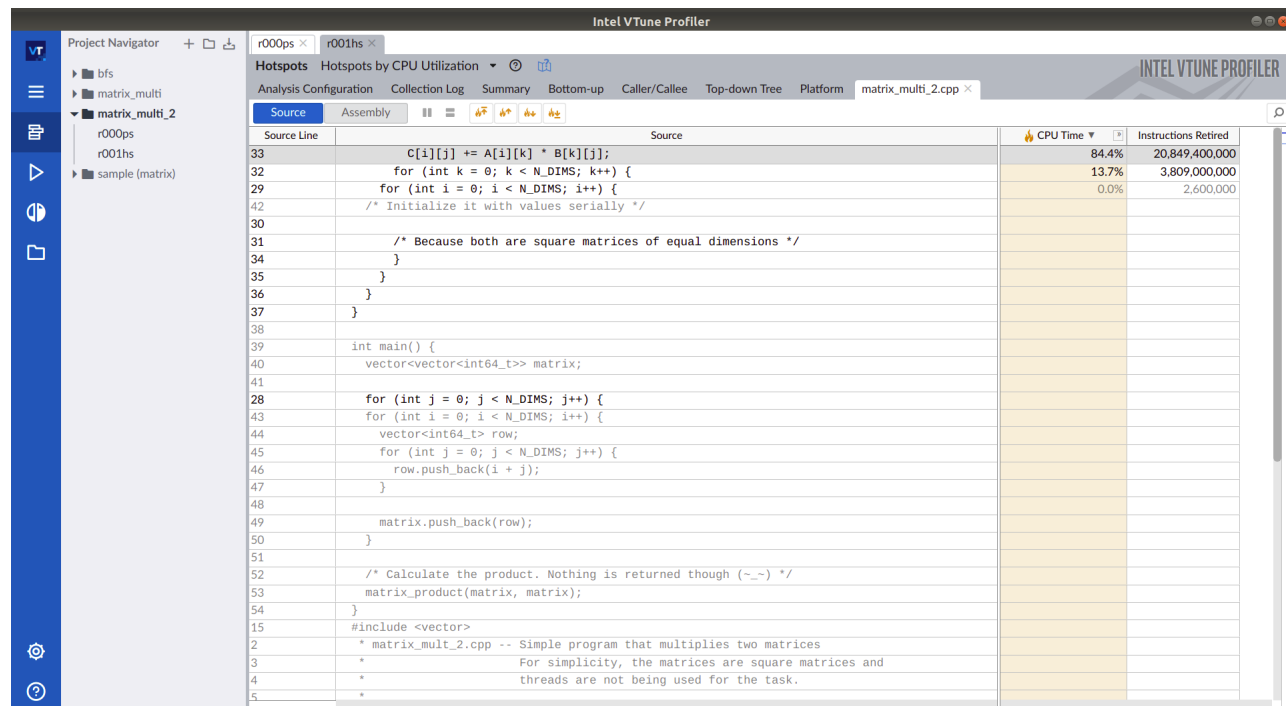


Figure 1.9: Top Source lines by CPU Utilization for matrix_multi_2.cpp

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

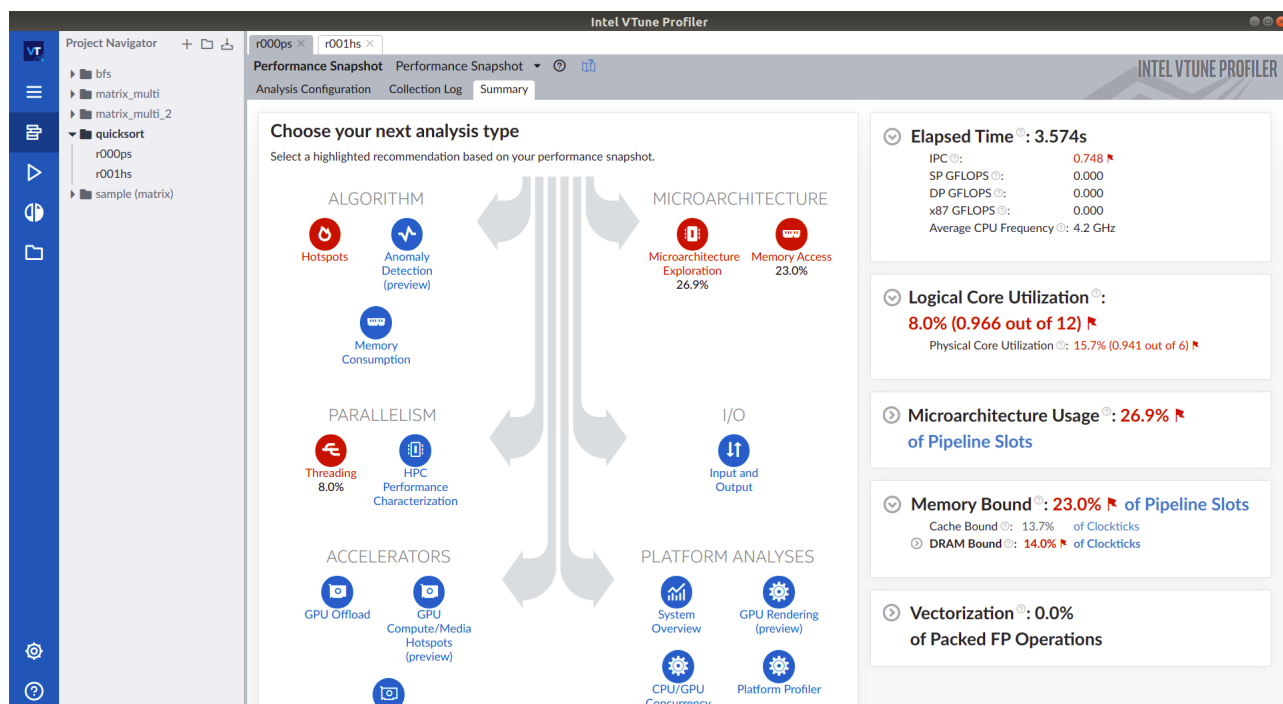


Figure 1.10: Performance Snapshot for quicksort.cpp

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 |

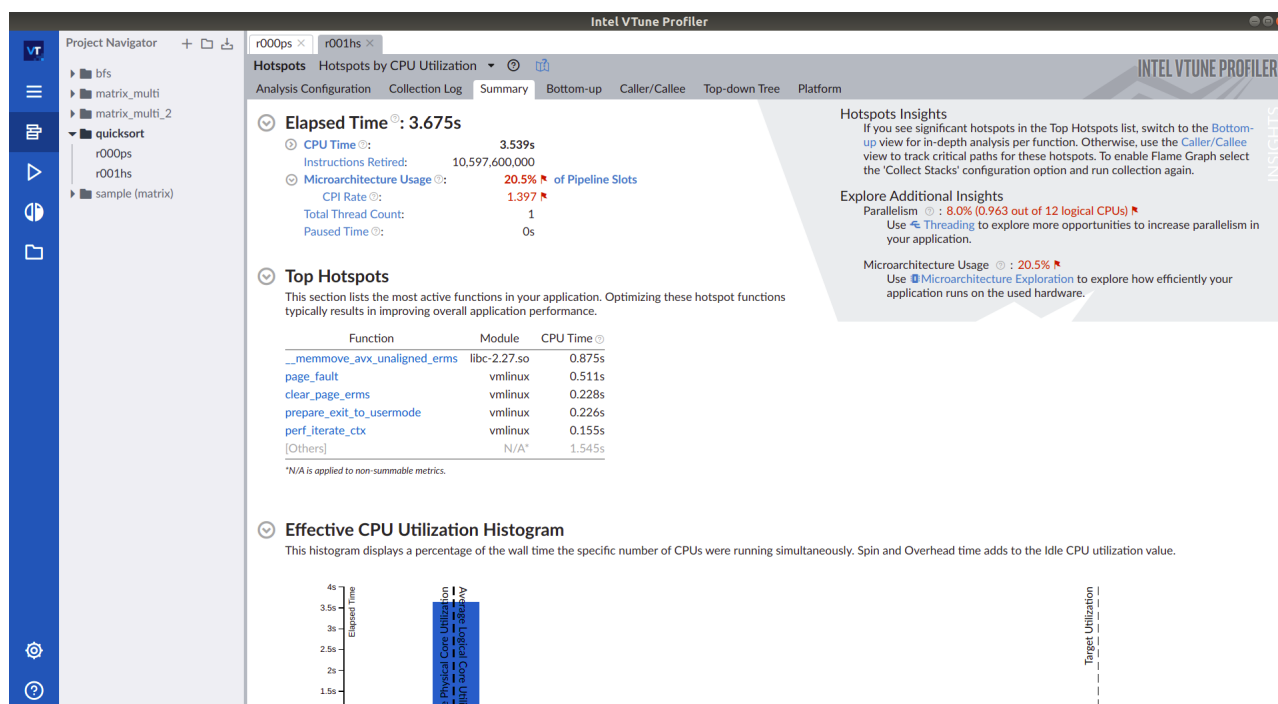


Figure 1.11: Top Functions by CPU Time for quicksort.cpp

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% |

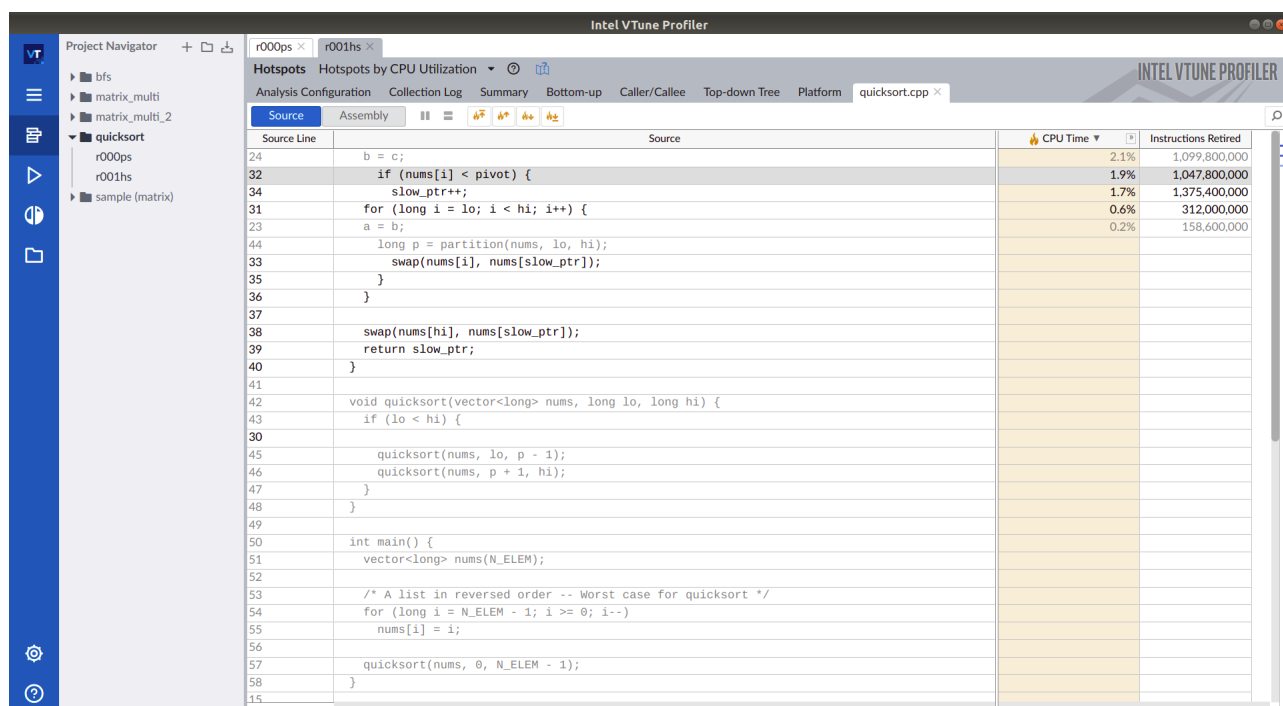


Figure 1.12: Top Source lines by CPU Utilization for quicksort.cpp

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 |

2.2 Setup Configurations

To speed up the task and to avoid having to reset to default values again and again, I prepared 7 copies of ChampSim in the docker container.

This helped me run the 28 simulations in parallel and the entire experiment took 5-6 minutes to finish.

To setup each of the Configurations, I had to modify `./inc/cache.h`.

| Line | Parameter | Baseline | Direct Mapped | Fully Associative | Reduced Size | Doubled Size | Reduced MSHR | Doubled MSHR |
|------|---------------|----------|------------------|----------------------|-----------------|-----------------|-----------------|-----------------|
| 46 | L1I_SET | 64 | 64*8 | 1 | 32 | 128 | 64 | 64 |
| 47 | L1I_WAY | 8 | 1 | 8*64 | 8 | 8 | 8 | 8 |
| 51 | L1I_MSHR_SIZE | 8 | 8 | 8 | 8 | 8 | 4 | 16 |
| 52 | L1I_LATENCY | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 55 | L1D_SET | 64 | 64*12 | 1 | 32 | 128 | 64 | 64 |
| 56 | L1D_WAY | 12 | 1 | 12*64 | 12 | 12 | 12 | 12 |
| 60 | L1D_MSHR_SIZE | 16 | 16 | 16 | 16 | 16 | 8 | 32 |
| 61 | L1D_LATENCY | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 64 | L2C_SET | 1024 | 1024*8 | 1 | 512 | 2048 | 1024 | 1024 |
| 65 | L2C_WAY | 8 | 1 | 8*1024 | 8 | 8 | 8 | 8 |
| 69 | L2C_MSHR_SIZE | 32 | 32 | 32 | 32 | 32 | 16 | 64 |
| 70 | L2C_LATENCY | 10 | 10 | 10 | 9 | 13 | 10 | 10 |
| 73 | LLC_SET | 2048 | 2048*16 | 1 | 1024 | 4096 | 2048 | 2048 |
| 74 | LLC_WAY | 16 | 1 | 16*2048 | 16 | 16 | 16 | 16 |
| 78 | LLC_MSHR_SIZE | 64 | 64 | 64 | 64 | 64 | 32 | 128 |
| 79 | LLC_LATENCY | 20 | 20 | 20 | 17 | 24 | 20 | 20 |