Report for Ques 2

Question 2: Matrix Multiplication

What the Program Does

This program multiplies two large matrices of size 1000×1000 each. Matrix multiplication involves taking each row of the first matrix and multiplying it with each column of the second matrix. For 1000×1000 matrices, this means doing 1 billion multiply-add operations (1000 × 1000 × 1000).

I tested two different parallel approaches. The first approach (1D threading) parallelizes the outer loop so each thread handles complete rows of the result matrix. The second approach (2D nested threading with collapse) distributes work more finely by treating both the row and column loops as parallel, spreading individual elements across threads.

Performance Results: 1D Threading

The sequential version took 3.886 seconds. When I used 2 threads, it took 3.838 seconds - barely any improvement at 1.01× speedup with 50.62% efficiency. With 4 threads it took 3.740 seconds for a tiny 1.04× speedup. Beyond 4 threads, performance actually got worse. With 8 threads it was 4.221 seconds (0.92× speedup - slower than sequential), and with 16 threads it remained slow at 4.109 seconds.

Performance Results: 2D Nested Threading

The 2D version started with a sequential time of 4.035 seconds (slightly slower baseline than 1D). With 2 threads it took 6.308 seconds for a terrible 0.64× speedup - 36% slower than sequential. Performance gradually improved with more threads but never beat sequential. With 4 threads it was 4.960 seconds (0.81× speedup), with 8 threads it was 4.343 seconds (0.93× speedup), and with 16 threads it reached 4.274 seconds (0.94× speedup).

Hardware Analysis Comparison

For the 1D version, perf showed task-clock of 69,181 milliseconds with CPU utilization at 0.872 cores. There were 63,691 context switches - that's a huge number indicating threads are being swapped constantly. Page faults were 7,951. Total elapsed time was 79.31 seconds but user time was 68.99 seconds with only 0.14 seconds system time.

For the 2D version, task-clock was 63,400 milliseconds with better CPU utilization at 0.927 cores. Context switches were 58,518 (still very high but less than 1D). Page faults were similar at 7,952. Elapsed time was 68.37 seconds with user time of 63.32 seconds and minimal system time of 0.05 seconds.

Why This Happened

Matrix multiplication has a fundamental problem in this implementation - the memory access pattern. When we access matrix B, we're reading it column by column which means jumping around in memory. Computer memory and caches work best when you read data sequentially. Reading column-wise causes what's called "cache misses" where the CPU constantly has to fetch new data from slow RAM instead of fast cache.

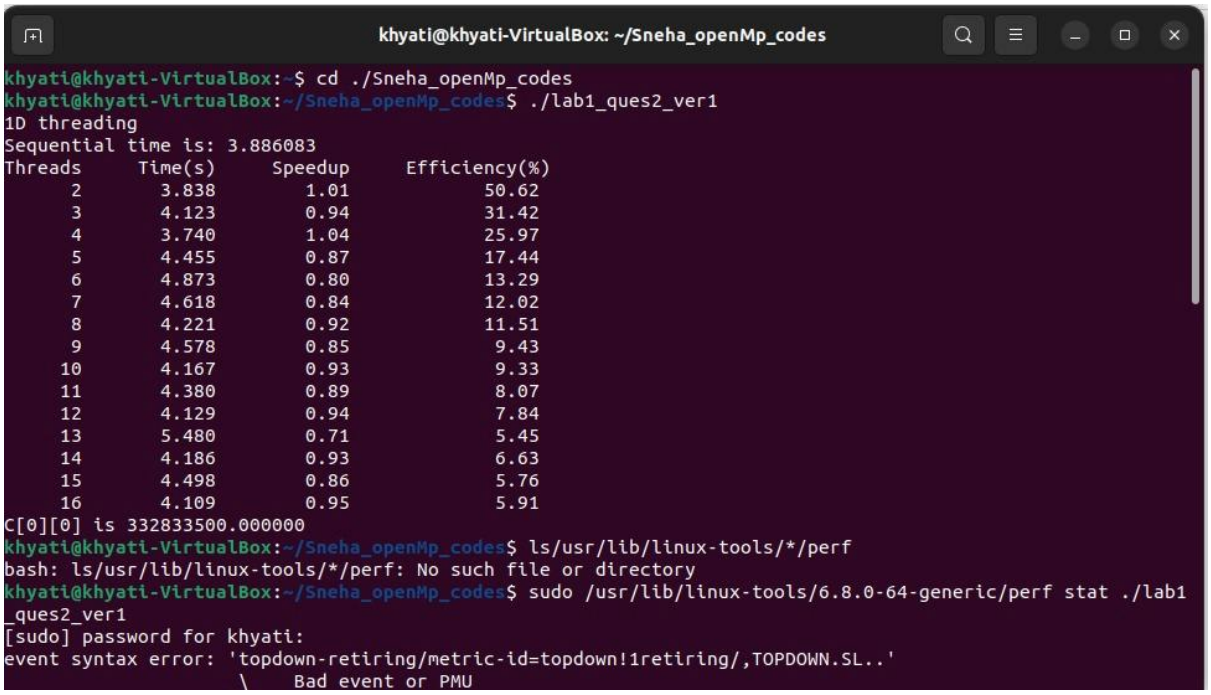
With 8 megabytes per matrix, the data doesn't fit in the CPU's cache. Every thread is fighting to access memory, creating a bottleneck. The massive number of context switches (63,000+) shows the threads are being paused and resumed constantly, which wastes time and destroys cache efficiency further.

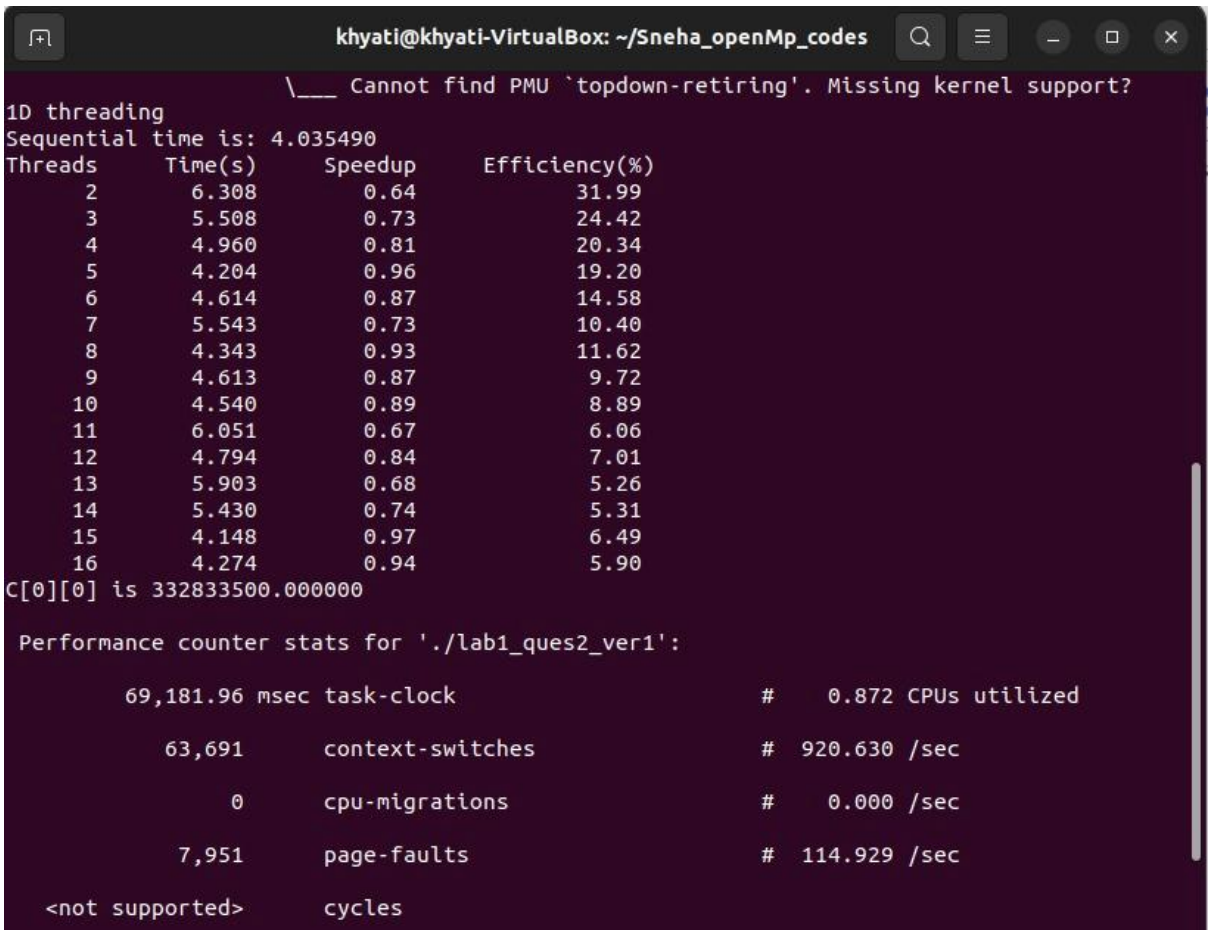
The 2D version had slightly better CPU utilization (92.7% vs 87.2%) because work was distributed more evenly across threads, but this didn't help performance because the fundamental memory bottleneck remained. The additional coordination overhead of collapsing two loops actually made things slightly worse initially, though it eventually caught up with more threads.

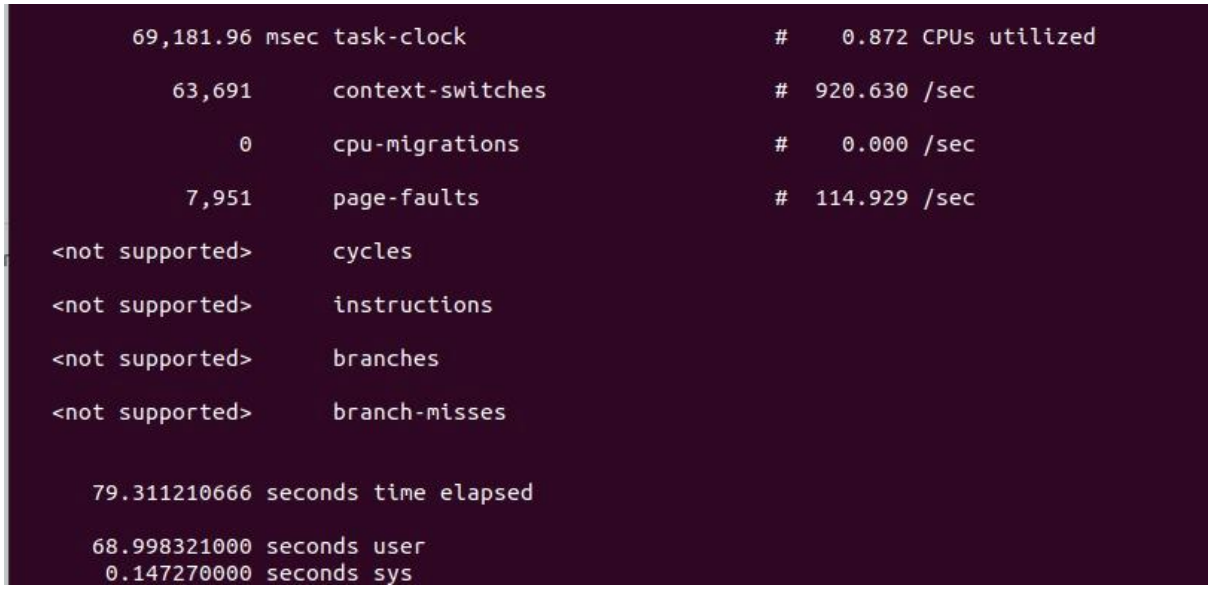
Both versions show that simply adding threads doesn't solve algorithmic inefficiencies. The naive matrix multiplication algorithm is inherently memory-bound and needs algorithmic improvements like blocking or transposing matrices, not just more threads.

My output:

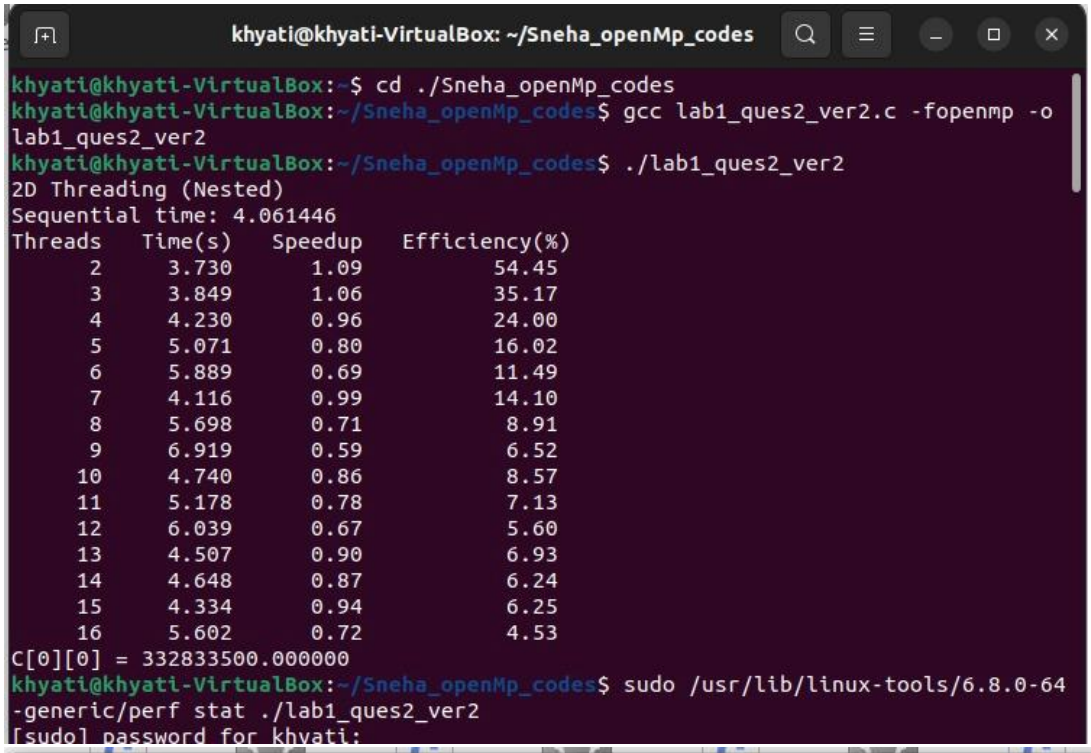
Version 1 terminal output as well as performance stats:

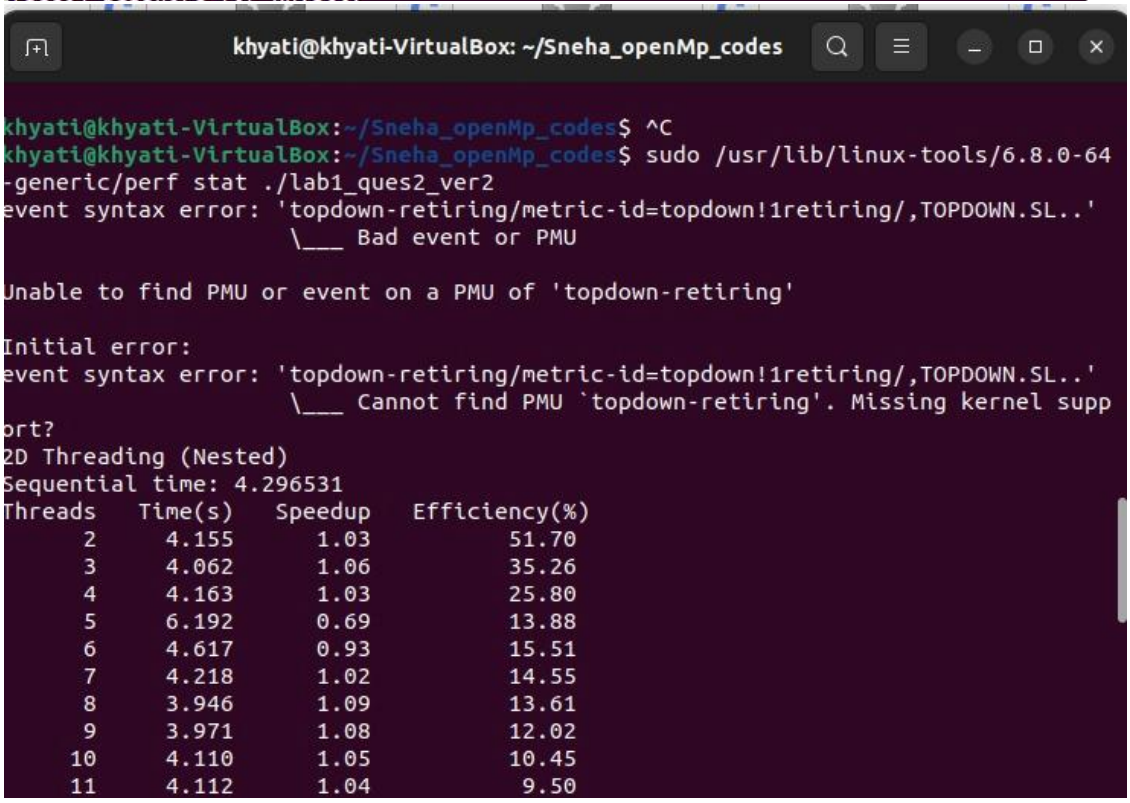

khyati@khyati-VirtualBox: ~/Sneha_openMp_codes
khyati@khyati-VirtualBox:~$ cd ./Sneha_openMp_codes
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ ./lab1_ques2_ver1
1D threading
Sequential time is: 3.886083
Threads Time(s) Speedup Efficiency(%)
2 3.838 1.01 50.62
3 4.123 0.94 31.42
4 3.740 1.04 25.97
5 4.455 0.87 17.44
6 4.873 0.80 13.29
7 4.618 0.84 12.02
8 4.221 0.92 11.51
9 4.578 0.85 9.43
10 4.167 0.93 9.33
11 4.380 0.89 8.07
12 4.129 0.94 7.84
13 5.480 0.71 5.45
14 4.186 0.93 6.63
15 4.498 0.86 5.76
16 4.109 0.95 5.91
C[0][0] is 332833500.000000
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ ls/usr/lib/linux-tools/*/perf
bash: ls/usr/lib/linux-tools/*/perf: No such file or directory
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ sudo /usr/lib/linux-tools/6.8.0-64-generic/perf stat ./lab1
_ques2_ver1
[sudo] password for khyati:
event syntax error: 'topdown-retiring/metric-id=topdown!1retiring/,TOPDOWN.SL..'
\ Bad event or PMU

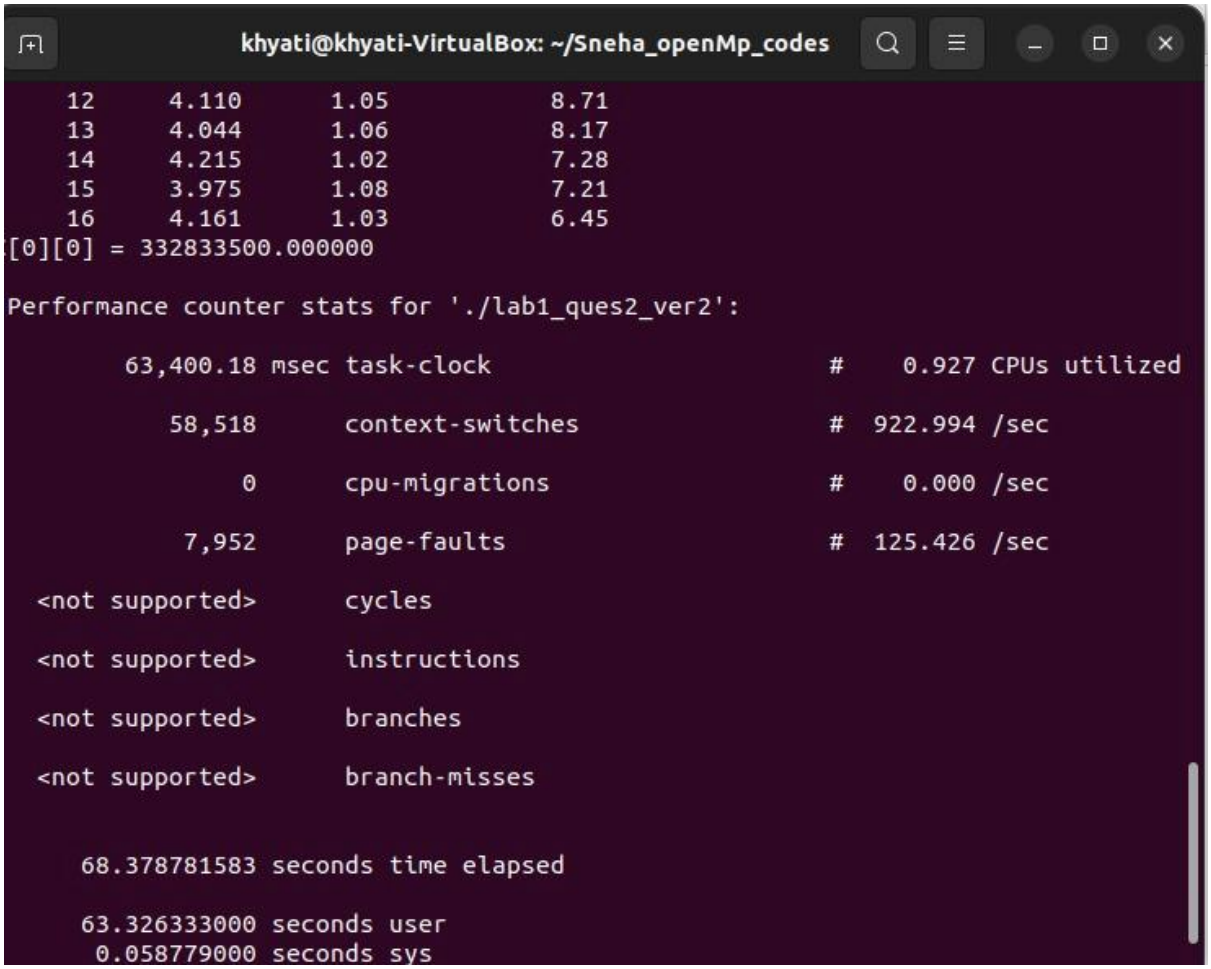

khyati@khyati-VirtualBox: ~/Sneha_openMp_codes
___ Cannot find PMU `topdown-retiring'. Missing kernel support?
1D threading
Sequential time is: 4.035490
Threads Time(s) Speedup Efficiency(%)
2 6.308 0.64 31.99
3 5.508 0.73 24.42
4 4.960 0.81 20.34
5 4.204 0.96 19.20
6 4.614 0.87 14.58
7 5.543 0.73 10.40
8 4.343 0.93 11.62
9 4.613 0.87 9.72
10 4.540 0.89 8.89
11 6.051 0.67 6.06
12 4.794 0.84 7.01
13 5.903 0.68 5.26
14 5.430 0.74 5.31
15 4.148 0.97 6.49
16 4.274 0.94 5.90
C[0][0] is 332833500.000000
Performance counter stats for './lab1_ques2_ver1':
69,181.96 msec task-clock # 0.872 CPUs utilized
63,691 context-switches # 920.630 /sec
0 cpu-migrations # 0.000 /sec
7,951 page-faults # 114.929 /sec
<not supported> cycles


69,181.96 msec task-clock # 0.872 CPUs utilized
63,691 context-switches # 920.630 /sec
0 cpu-migrations # 0.000 /sec
7,951 page-faults # 114.929 /sec
<not supported> cycles
<not supported> instructions
<not supported> branches
<not supported> branch-misses
79.311210666 seconds time elapsed
68.998321000 seconds user
0.147270000 seconds sys

Version 2:


khyati@khyati-VirtualBox: ~/Sneha_openMp_codes
khyati@khyati-VirtualBox:~$ cd ./Sneha_openMp_codes
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ gcc lab1_ques2_ver2.c -fopenmp -o lab1_ques2_ver2
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ ./lab1_ques2_ver2
2D Threading (Nested)
Sequential time: 4.061446
Threads Time(s) Speedup Efficiency(%)
2 3.730 1.09 54.45
3 3.849 1.06 35.17
4 4.230 0.96 24.00
5 5.071 0.80 16.02
6 5.889 0.69 11.49
7 4.116 0.99 14.10
8 5.698 0.71 8.91
9 6.919 0.59 6.52
10 4.740 0.86 8.57
11 5.178 0.78 7.13
12 6.039 0.67 5.60
13 4.507 0.90 6.93
14 4.648 0.87 6.24
15 4.334 0.94 6.25
16 5.602 0.72 4.53
C[0][0] = 332833500.000000
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ sudo /usr/lib/linux-tools/6.8.0-64-generic/perf stat ./lab1_ques2_ver2
[sudo] password for khyati:


khyati@khyati-VirtualBox: ~/Sneha_openMp_codes
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ ^C
khyati@khyati-VirtualBox:~/Sneha_openMp_codes$ sudo /usr/lib/linux-tools/6.8.0-64-generic/perf stat ./lab1_ques2_ver2
event syntax error: 'topdown-retiring/metric-id=topdown!1retiring/,TOPDOWN.SL..'
___ Bad event or PMU
Unable to find PMU or event on a PMU of 'topdown-retiring'
Initial error:
event syntax error: 'topdown-retiring/metric-id=topdown!1retiring/,TOPDOWN.SL..'
___ Cannot find PMU `topdown-retiring'. Missing kernel support?
2D Threading (Nested)
Sequential time: 4.296531
Threads Time(s) Speedup Efficiency(%)
2 4.155 1.03 51.70
3 4.062 1.06 35.26
4 4.163 1.03 25.80
5 6.192 0.69 13.88
6 4.617 0.93 15.51
7 4.218 1.02 14.55
8 3.946 1.09 13.61
9 3.971 1.08 12.02
10 4.110 1.05 10.45
11 4.112 1.04 9.50


khyati@khyati-VirtualBox: ~/Sneha_openMp_codes
12 4.110 1.05 8.71
13 4.044 1.06 8.17
14 4.215 1.02 7.28
15 3.975 1.08 7.21
16 4.161 1.03 6.45
[0][0] = 332833500.000000
Performance counter stats for './lab1_ques2_ver2':
63,400.18 msec task-clock # 0.927 CPUs utilized
58,518 context-switches # 922.994 /sec
0 cpu-migrations # 0.000 /sec
7,952 page-faults # 125.426 /sec
<not supported> cycles
<not supported> instructions
<not supported> branches
<not supported> branch-misses
68.378781583 seconds time elapsed
63.326333000 seconds user
0.058779000 seconds sys