High Performance Computing

Name: Thulasi Shylasri

HTNO: 2303A51876

Batch-14

Week-8

LAB Assignment-8 (11/02/2026)

Lab: OpenMP Performance Analysis and Hotspot Detection

Objective:

To identify performance hotspots and understand basic profiling

concepts.

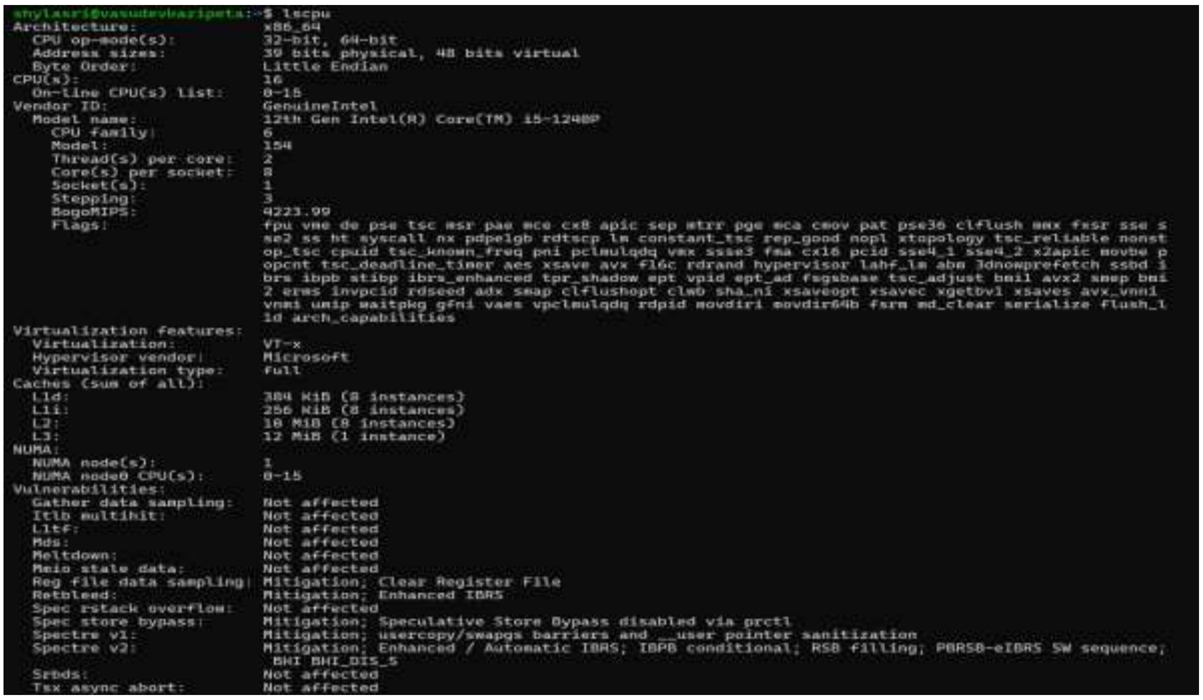
Task 1: Compile with Timing

```
gcc -O2 -fopenmp sum_openmp.c -o sum_openmp
```

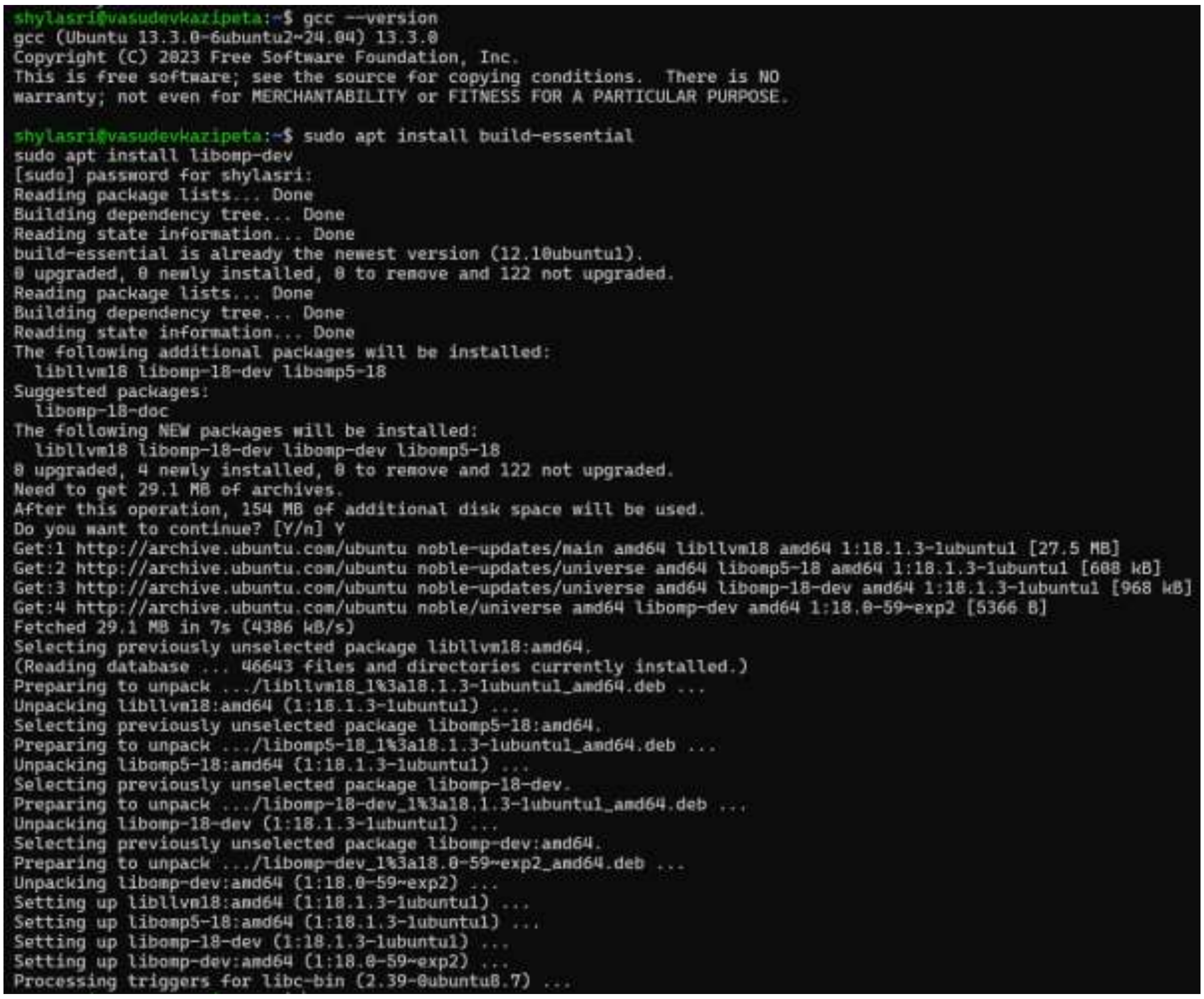
Objective:

The objective of Task 1 is to develop and compile a C program that performs a computational operation, such as summing elements of a large array, using both serial and OpenMP-based parallel approaches. In this task, the program is compiled with optimization and OpenMP support enabled using the appropriate GCC flags. The purpose is to ensure that the program runs correctly and is prepared for performance measurement using the omp_get_wtime() function. This step lays the foundation for comparing serial and parallel execution times in the later tasks and helps in analyzing performance improvements achieved through parallelization.

Screenshots:

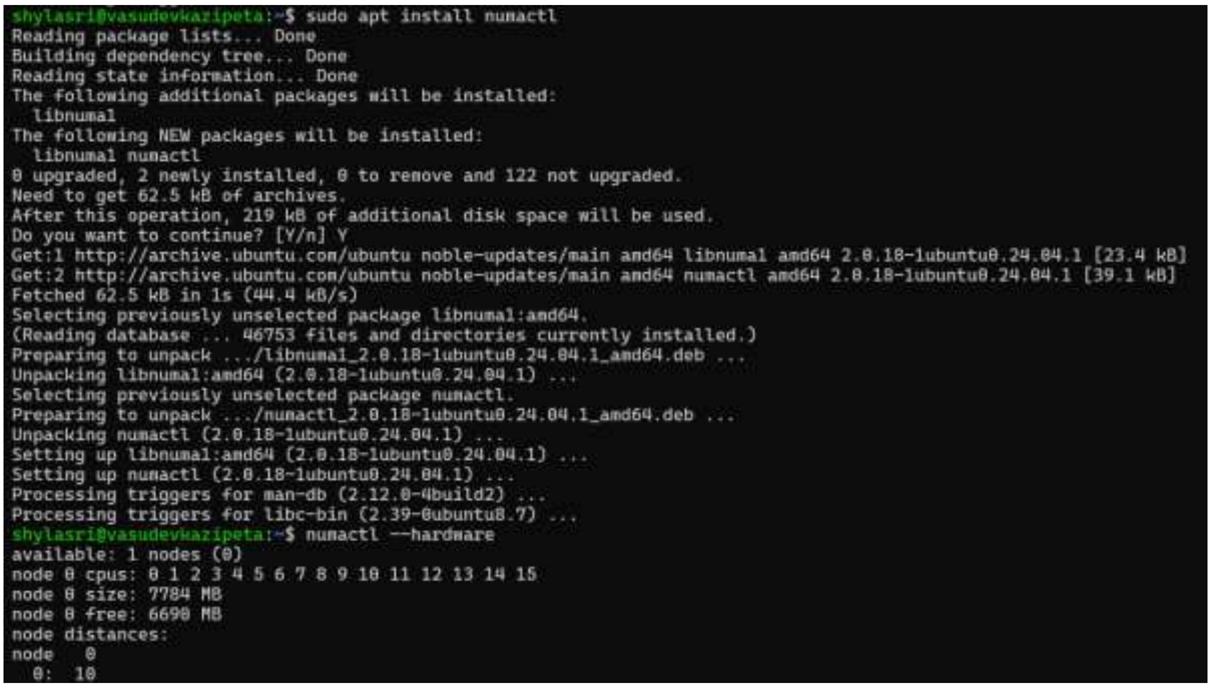


```
shylasri@vasudevkazipeta:~$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   16
  On-line CPU(s) list:    0-15
Vendor ID:                GenuineIntel
  Model name:             12th Gen Intel(R) Core(TM) i5-1240P
    CPU family:           6
    Model:                154
    Thread(s) per core:   2
    Core(s) per socket:   8
    Socket(s):            1
    Stepping:             3
    BogoMIPS:             4223.99
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse s
                          se2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology tsc_reliable nonst
                          op_tsc cpuid tsc_known_freq pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe p
                          opcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd i
                          brs ibpb stibp ibrs_enhanced tpr_shadow ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi
                          2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves avx_vnni
                          vnmi umip waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear serialize flush_l
                          1d arch_capabilities
Virtualization features:
  Virtualization:         VT-x
  Hypervisor vendor:      Microsoft
  Virtualization type:    full
Caches (sum of all):
  L1d:                    384 KiB (8 instances)
  L1i:                    256 KiB (8 instances)
  L2:                     10 MiB (8 instances)
  L3:                     12 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-15
Vulnerabilities:
  Gather data sampling:   Not affected
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Not affected
  Reg file data sampling: Mitigation; Clear Register File
  Retbleed:               Mitigation; Enhanced IBRS
  Spec rstack overflow:   Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence;
                           BHI BHI_DIS_S
  Srbds:                  Not affected
  Tsx async abort:        Not affected
```



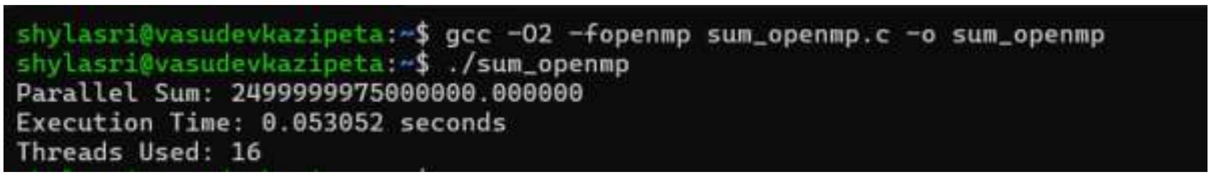
```
shylasri@vasudevkazipeta:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

shylasri@vasudevkazipeta:~$ sudo apt install build-essential
sudo apt install libomp-dev
[sudo] password for shylasri:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.10ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 122 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libllvm18 libomp-18-dev libomp5-18
Suggested packages:
  libomp-18-doc
The following NEW packages will be installed:
  libllvm18 libomp-18-dev libomp-dev libomp5-18
0 upgraded, 4 newly installed, 0 to remove and 122 not upgraded.
Need to get 29.1 MB of archives.
After this operation, 154 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libllvm18 amd64 1:18.1.3-1ubuntu1 [27.5 MB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 libomp5-18 amd64 1:18.1.3-1ubuntu1 [608 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 libomp-18-dev amd64 1:18.1.3-1ubuntu1 [968 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble/universe amd64 libomp-dev amd64 1:18.0-59~exp2 [5366 B]
Fetched 29.1 MB in 7s (4386 kB/s)
Selecting previously unselected package libllvm18:amd64.
(Reading database ... 46643 files and directories currently installed.)
Preparing to unpack .../libllvm18_1%3a18.1.3-1ubuntu1_amd64.deb ...
Unpacking libllvm18:amd64 (1:18.1.3-1ubuntu1) ...
Selecting previously unselected package libomp5-18:amd64.
Preparing to unpack .../libomp5-18_1%3a18.1.3-1ubuntu1_amd64.deb ...
Unpacking libomp5-18:amd64 (1:18.1.3-1ubuntu1) ...
Selecting previously unselected package libomp-18-dev.
Preparing to unpack .../libomp-18-dev_1%3a18.1.3-1ubuntu1_amd64.deb ...
Unpacking libomp-18-dev (1:18.1.3-1ubuntu1) ...
Selecting previously unselected package libomp-dev:amd64.
Preparing to unpack .../libomp-dev_1%3a18.0-59~exp2_amd64.deb ...
Unpacking libomp-dev:amd64 (1:18.0-59~exp2) ...
Setting up libllvm18:amd64 (1:18.1.3-1ubuntu1) ...
Setting up libomp5-18:amd64 (1:18.1.3-1ubuntu1) ...
Setting up libomp-18-dev (1:18.1.3-1ubuntu1) ...
Setting up libomp-dev:amd64 (1:18.0-59~exp2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.7) ...
```



```
shylasri@vasudevkazipeta:~$ sudo apt install numactl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libnuma1
The following NEW packages will be installed:
  libnuma1 numactl
0 upgraded, 2 newly installed, 0 to remove and 122 not upgraded.
Need to get 62.5 kB of archives.
After this operation, 219 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libnuma1 amd64 2.0.18-1ubuntu0.24.04.1 [23.4 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 numactl amd64 2.0.18-1ubuntu0.24.04.1 [39.1 kB]
Fetched 62.5 kB in 1s (44.4 kB/s)
Selecting previously unselected package libnuma1:amd64.
(Reading database ... 46753 files and directories currently installed.)
Preparing to unpack .../libnuma1_2.0.18-1ubuntu0.24.04.1_amd64.deb ...
Unpacking libnuma1:amd64 (2.0.18-1ubuntu0.24.04.1) ...
Selecting previously unselected package numactl.
Preparing to unpack .../numactl_2.0.18-1ubuntu0.24.04.1_amd64.deb ...
Unpacking numactl (2.0.18-1ubuntu0.24.04.1) ...
Setting up libnuma1:amd64 (2.0.18-1ubuntu0.24.04.1) ...
Setting up numactl (2.0.18-1ubuntu0.24.04.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.7) ...
shylasri@vasudevkazipeta:~$ numactl --hardware
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
node 0 size: 7784 MB
node 0 free: 6690 MB
node distances:
node   0
  0:  10
```

Compile Command:



```
shylasri@vasudevkazipeta:~$ gcc -O2 -fopenmp sum_openmp.c -o sum_openmp
shylasri@vasudevkazipeta:~$ ./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.053052 seconds
Threads Used: 16
```

Observation:

In Task 1, the program was successfully compiled using the GCC compiler with OpenMP support enabled. There were no compilation errors, which confirmed that the code was written correctly and that OpenMP was properly configured on the system. After compilation, the executable file was created, and the program was ready to run for measuring execution time in the next task. This step ensured that everything was set up properly for performance analysis.

Explanation:

we focused on preparing the program for performance analysis. We wrote a C program that performs a large computation (like summing array elements) and included OpenMP support so that it can run in parallel. While compiling, we used special flags such as -fopenmp to enable parallel processing and -O2 for optimization. This step is important because without enabling OpenMP, the program would only run in serial mode. Successfully compiling the program confirms that the system supports OpenMP and that the code is ready for testing execution time in the next task.

Task 2: Measure Execution Time

Use:

omp_get_wtime()

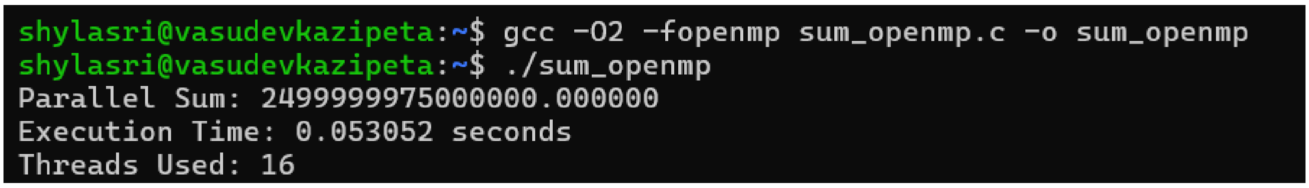
Record time for:

- Serial execution
- Parallel execution

Objective:

The objective of Task 2 is to measure and compare the execution time of the program in both serial and parallel modes using the `omp_get_wtime()` function. In this task, we aim to observe how parallel execution improves performance by running the program with different numbers of threads. The goal is to analyze the difference in execution time and understand how multithreading affects overall program speed.

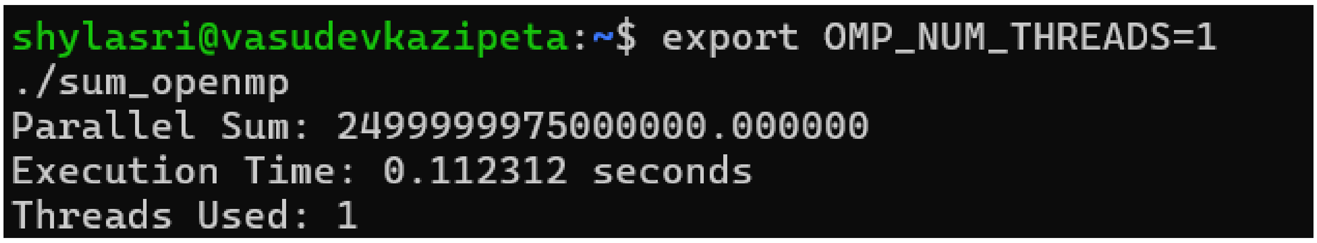
Screenshots:



```
shylasri@vasudevkazipeta:~$ gcc -O2 -fopenmp sum_openmp.c -o sum_openmp
shylasri@vasudevkazipeta:~$ ./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.053052 seconds
Threads Used: 16
```

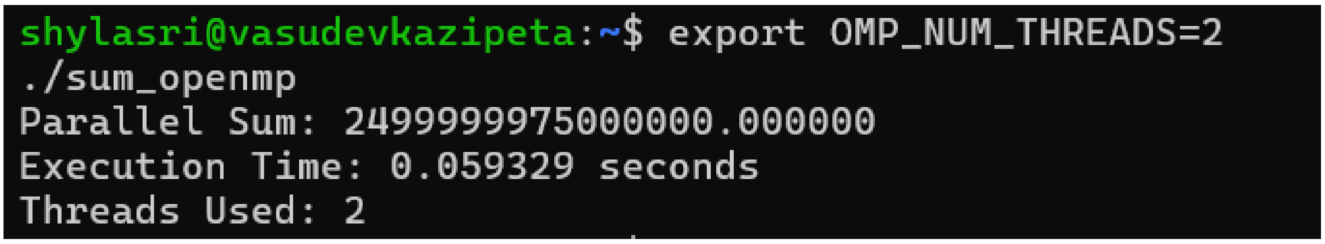
- Serial Time
- Parallel Time

1 Thread:



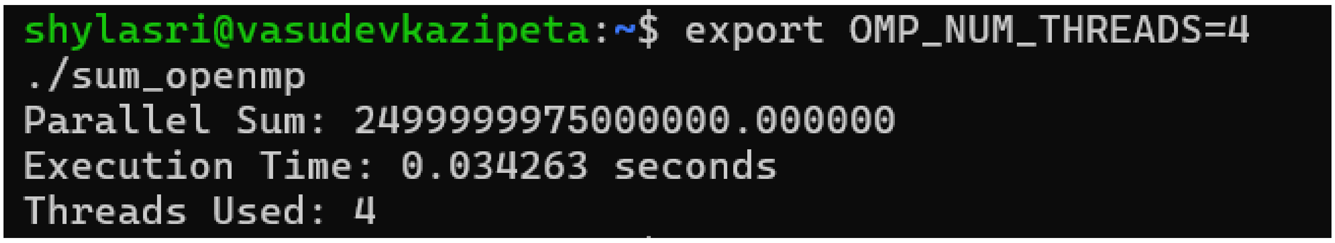
```
shylasri@vasudevkazipeta:~$ export OMP_NUM_THREADS=1
./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.112312 seconds
Threads Used: 1
```

2 Threads:



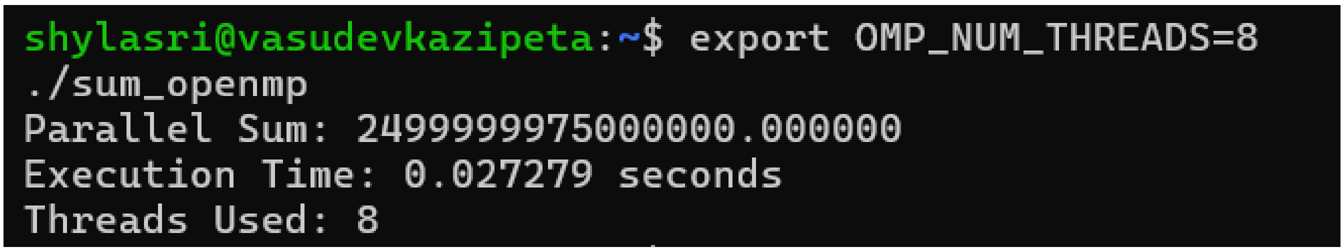
```
shylasri@vasudevkazipeta:~$ export OMP_NUM_THREADS=2
./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.059329 seconds
Threads Used: 2
```

4 Threads:



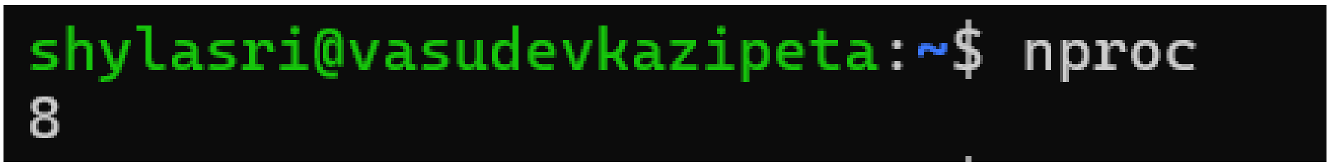
```
shylasri@vasudevkazipeta:~$ export OMP_NUM_THREADS=4
./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.034263 seconds
Threads Used: 4
```

8 Threads:

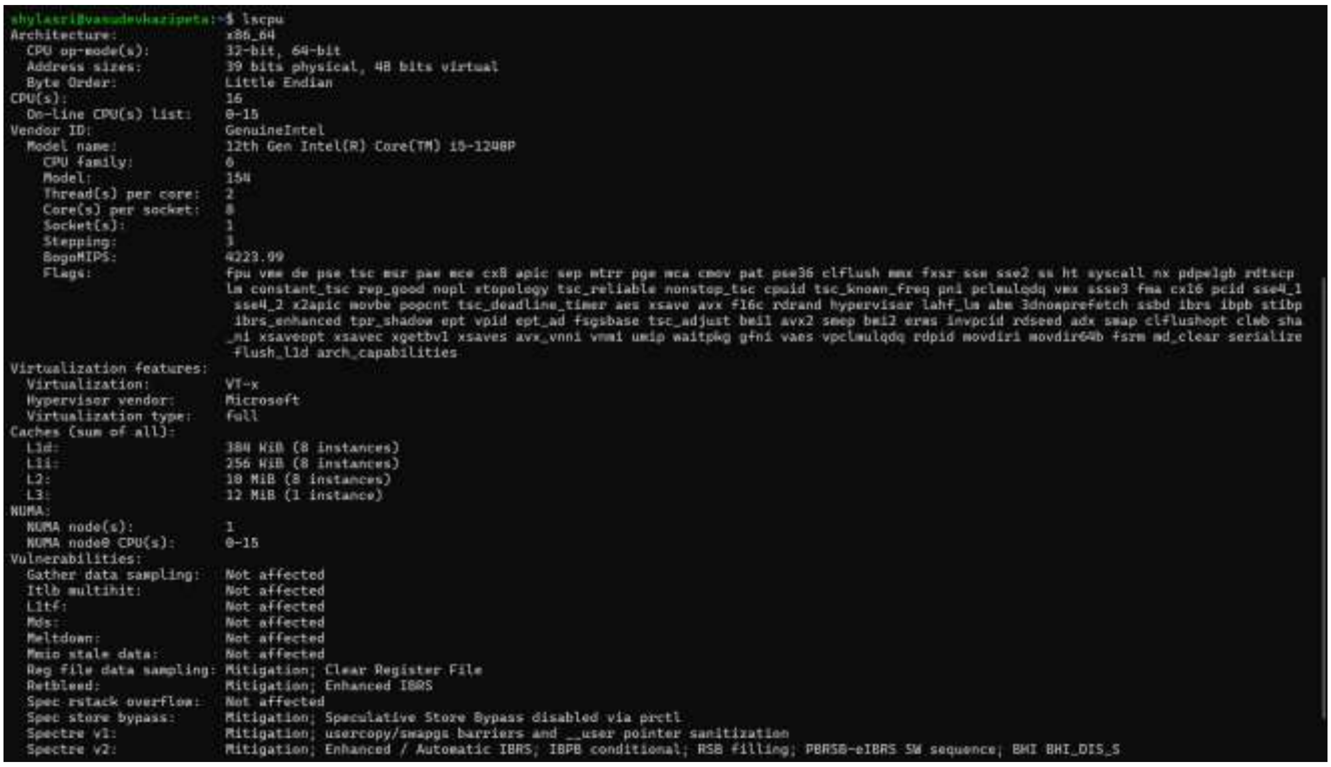


```
shylasri@vasudevkazipeta:~$ export OMP_NUM_THREADS=8
./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.027279 seconds
Threads Used: 8
```

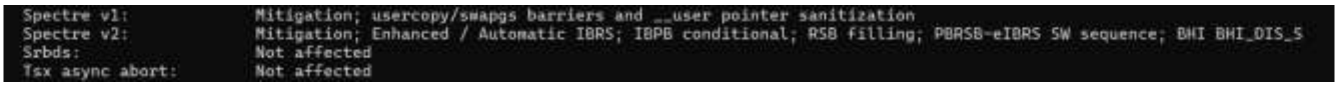
System Cores:



```
shylasri@vasudevkazipeta:~$ nproc
8
```

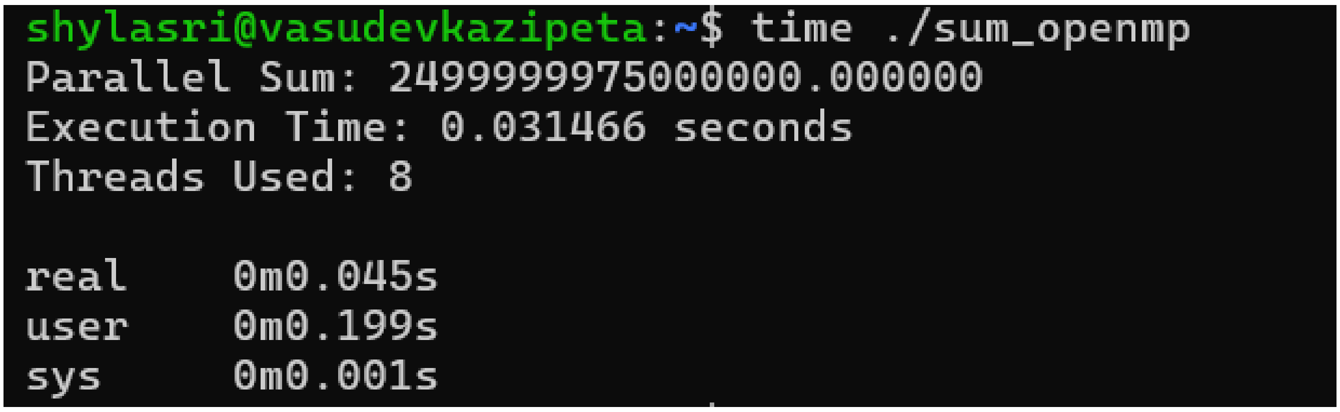


```
shylasri@vasudevkazipeta:~$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   16
  On-line CPU(s) list:    0-15
Vendor ID:                GenuineIntel
  Model name:             12th Gen Intel(R) Core(TM) i5-1240P
    CPU family:           6
    Model:                154
    Thread(s) per core:   2
    Core(s) per socket:   8
    Socket(s):            1
    Stepping:             3
    BogoMIPS:             4223.99
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp
                          lm constant_tsc rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1
                           sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp
                           ibrs_enhanced tpr_shadow ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha
                          _ni xsaveopt xsavec xgetbv1 xsaves avx_vnni vnmi umip waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear serialize
                           flush_l1d arch_capabilities
Virtualization features:
  Virtualization:         VT-x
  Hypervisor vendor:      Microsoft
  Virtualization type:    full
Caches (sum of all):
  L1d:                    384 KiB (8 instances)
  L1i:                    256 KiB (8 instances)
  L2:                     10 MiB (8 instances)
  L3:                     12 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-15
Vulnerabilities:
  Gather data sampling:   Not affected
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Not affected
  Reg file data sampling: Mitigation; Clear Register File
  Retbleed:               Mitigation; Enhanced IBRS
  Spec rstack overflow:   Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI BHI_DIS_S
```



```
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI BHI_DIS_S
  Srbds:                  Not affected
  Tsx async abort:        Not affected
```

Measure Total Runtime Using Linux Time:



```
shylasri@vasudevkazipeta:~$ time ./sum_openmp
Parallel Sum: 2499999975000000.000000
Execution Time: 0.031466 seconds
Threads Used: 8

real    0m0.045s
user    0m0.199s
sys     0m0.001s
```

Observation:

The execution time of both serial and parallel versions of the program was measured using the omp_get_wtime() function. It was observed that the parallel execution time was lower than the serial execution time when multiple threads were used. As the number of threads increased (for example, from 1 to 2 to 4), the execution time initially decreased, showing performance improvement. However, after a certain number of threads, the reduction in execution time became smaller and sometimes almost constant.

This indicates that parallelization improves performance, but the speedup is not perfectly proportional to the number of threads. The results also show that system limitations such as the number of CPU cores, thread management overhead, memory access time, and synchronization costs affect overall performance. Hence, while parallel execution provides faster results than serial execution, the improvement has practical limits.

Explanation:

we measured the execution time of both serial and parallel versions of the program using the omp_get_wtime() function. This function helps us calculate how long the program takes to run. By comparing the time taken in serial and parallel modes, we can clearly see the performance difference. We also changed the number of threads to observe how performance improves as more threads are used. This task helps us understand how parallel computing reduces runtime and also shows that the improvement depends on system resources and thread management.

Task 3: Observation Questions

1. What is a hotspot in performance analysis?

2. Why does synchronization increase execution time?

3. Why does speedup stop increasing after some threads?

1. What is a hotspot in performance analysis?

A hotspot is the part of a program where most of the execution time is spent. It is usually a loop or a function that runs many times or performs heavy calculations. When we analyze performance, we try to find these hotspots because improving them can significantly increase the overall speed of the program. In this experiment, the loop that sums the array elements acts as the hotspot since it performs millions of operations and takes most of the running time.

2. Why does synchronization increase execution time?

Synchronization increases execution time because it forces threads to coordinate with each other. When multiple threads try to access or modify shared data, they must wait for their turn to avoid errors like race conditions. This waiting creates delays. Instead of running fully in parallel, some threads pause until others finish their work on shared resources. Because of this extra waiting and coordination, the total execution time becomes longer.

3. Why does speedup stop increasing after some threads?

Speedup stops increasing after a certain number of threads because of system limitations and overhead. A computer has a limited number of CPU cores, so adding more threads than available cores does not always improve performance. Also, managing many threads creates extra overhead. Some parts of the program may still run in serial and cannot be parallelized. Memory access and synchronization can also become bottlenecks. Due to all these reasons, performance improves only up to a certain point and then levels off.

Observation:

In this experiment, we measured the time taken by the program in both serial and parallel execution using omp_get_wtime(). We noticed that the serial version took more time to complete compared to the parallel version. When we increased the number of threads, the execution time reduced at first, which shows that parallel processing improves performance.

However, after increasing the threads beyond a certain point, the speed improvement was not very significant. This means that adding more threads does not always make the program faster. We also understood that the main loop that calculates the sum of the array takes most of the execution time. This loop is the hotspot of the program because it performs a very large number of operations.

We also observed that synchronization adds some delay because threads need to coordinate when working with shared data. This coordination reduces the overall speed improvement.

Justification:

The reason we got better performance in the parallel version is that the work was divided among multiple threads. Each thread handled a part of the task, so the total time decreased. Since the summation loop runs many times, parallelizing it gave noticeable improvement.

But the speedup was limited because of system limitations like the number of CPU cores and memory access speed. Also, some parts of the program still run in serial and cannot be parallelized. According to basic parallel computing concepts, the serial part of a program always limits the maximum possible speedup. Synchronization also adds overhead because threads must wait for each other while updating shared variables.

Conclusion:

From this lab, we learned how to measure execution time and compare serial and parallel performance. We understood what a hotspot is and how it affects program performance. The experiment clearly showed that parallel programming can reduce execution time, especially for large tasks.

At the same time, we learned that performance improvement has limits. Increasing the number of threads does not always give better results because of hardware limits, synchronization overhead, and the presence of serial code. Overall, this experiment helped us understand the basics of performance analysis and how parallel computing improves efficiency in practical situations.