High Performance Computing

Name: Thulasi Shylasri

HTNO: 2303A51876

Batch-14

Week-7

LAB Assignment-7 (11/02/2026)

Lab: OpenMP SIMD Vectorization

Objective:

To understand SIMD vectorization and observe how OpenMP

SIMD improves loop execution.

Task 1: Normal Loop Execution

File: simd_serial.c

```
#include &lt;stdio.h&gt;
#define N 100000000
int main() {
float a[N], b[N], c[N];
for (int i = 0; i &lt; N; i++)
c[i] = a[i] + b[i];
printf(&quot;Done\n&quot;);
return 0;
}
```

Objective

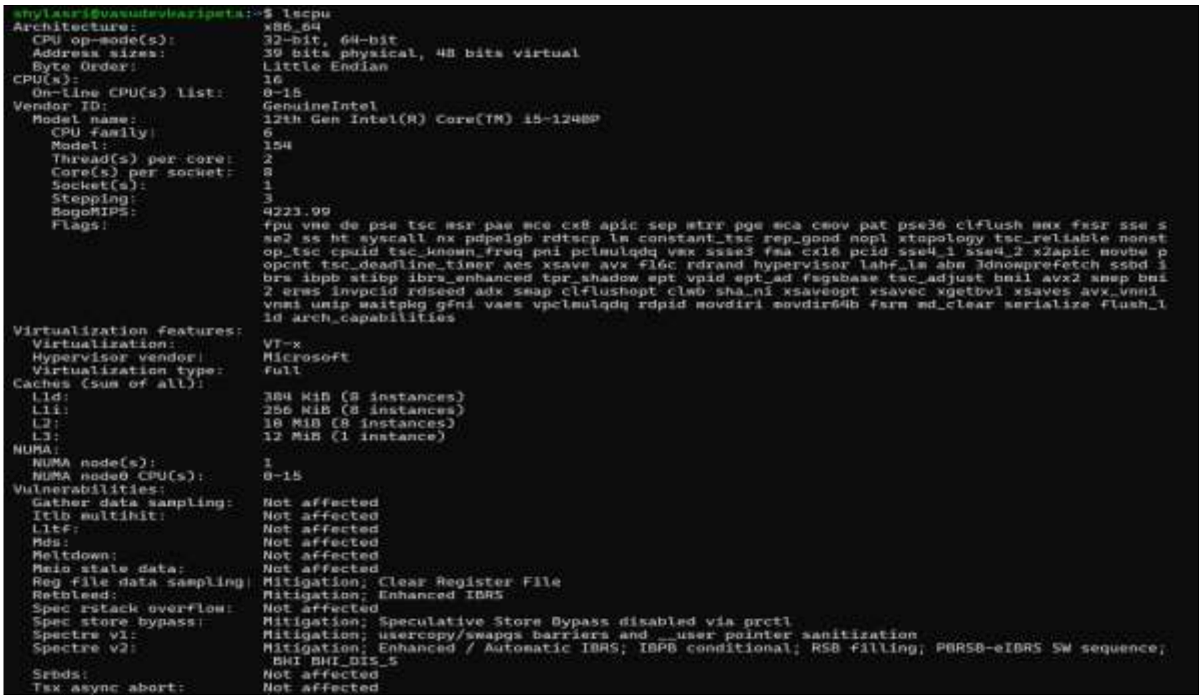
The objective of this experiment is to understand SIMD (Single Instruction Multiple Data) vectorization and to observe how OpenMP SIMD directives improve loop execution performance. This experiment helps in understanding how vector instructions process multiple data elements simultaneously.

Task 1: Normal Loop Execution

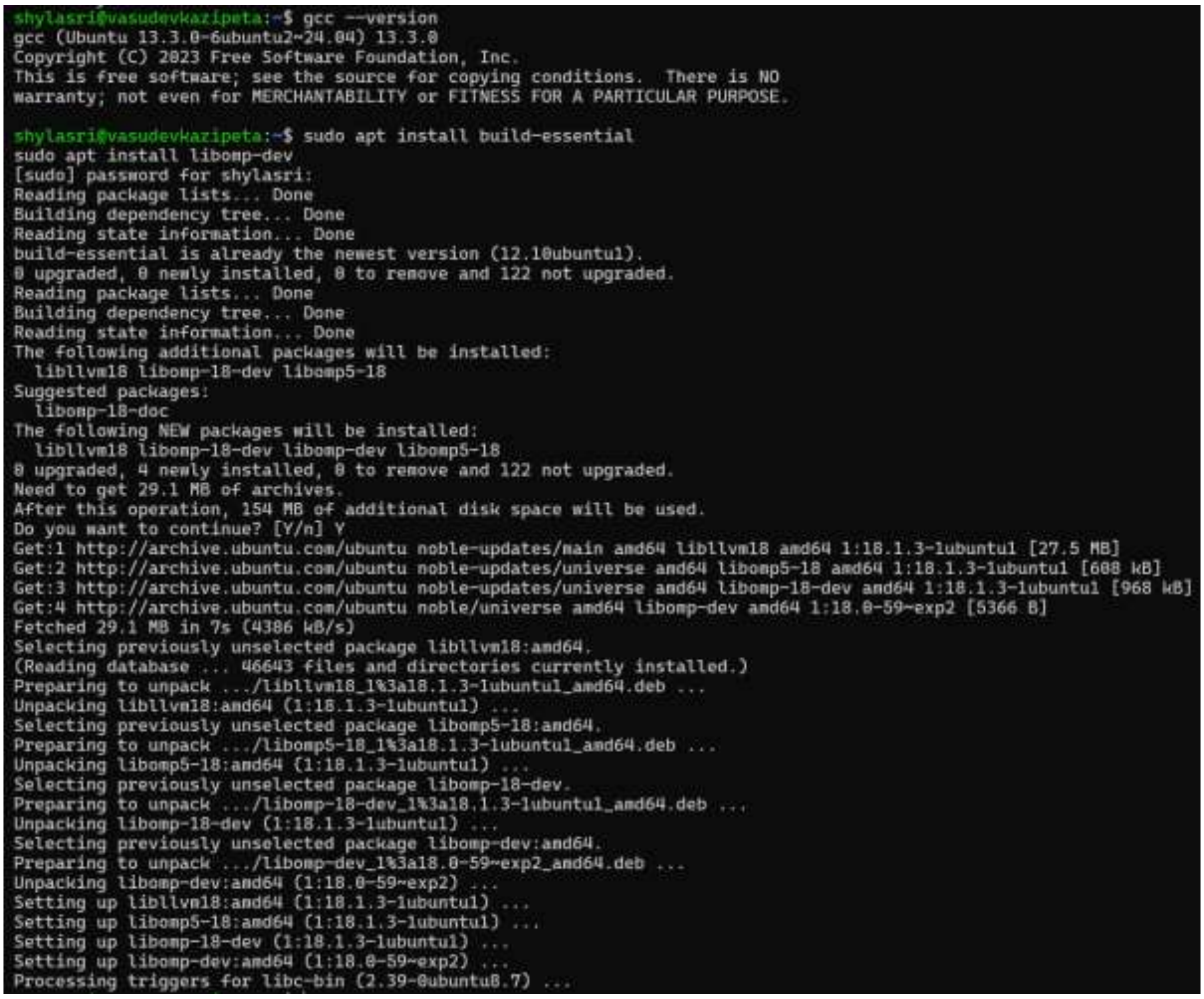
Program Description

In this task, a simple loop is written to add two arrays element by element.

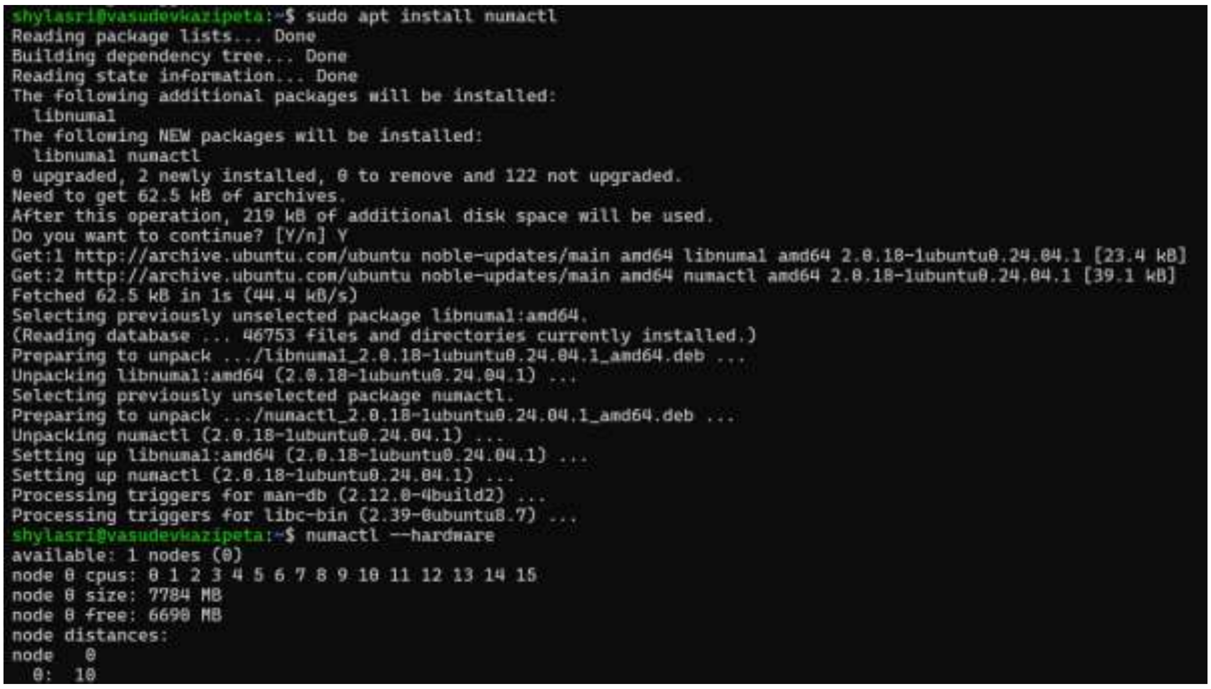
Screenshots:



```
shylasri@vasudevkaripeta:~$ 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
```



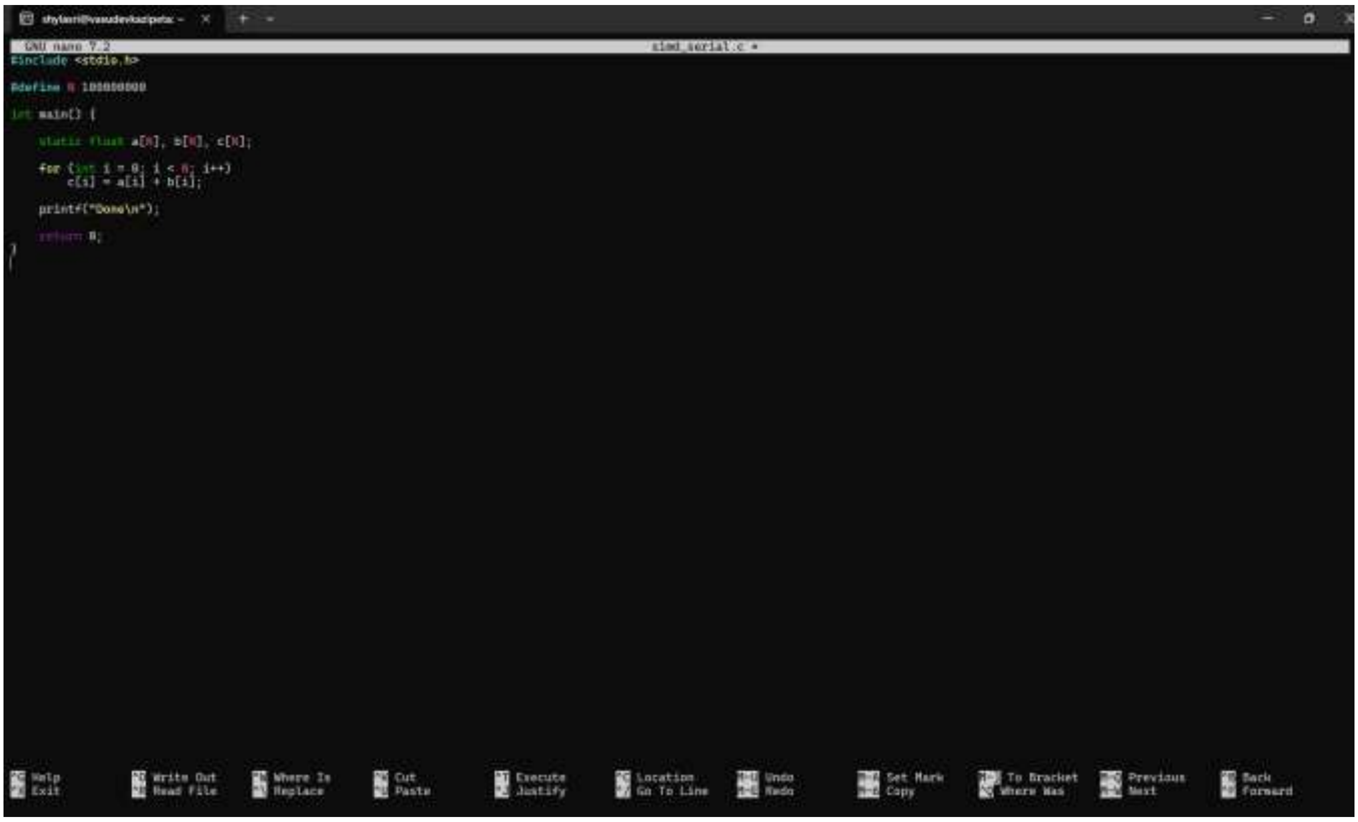
```
shylasri@vasudevkazipeta:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

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
```

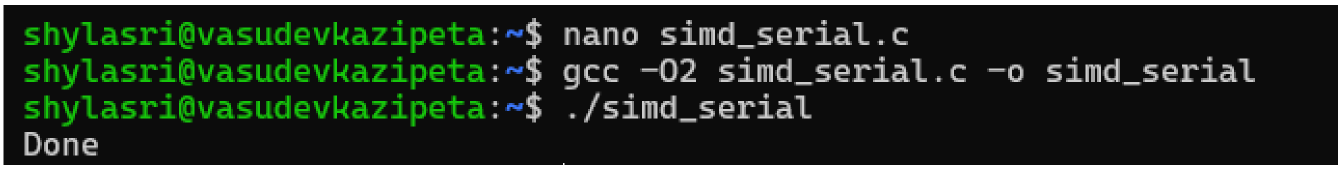
Task 1: Normal Loop Execution

Create File

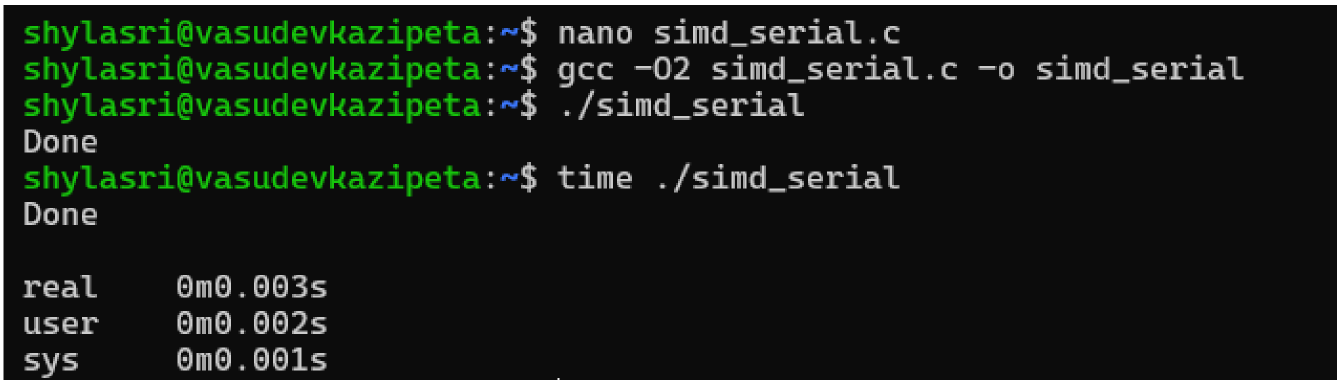


```
  GNU nano 7.2                          simd_serial.c *
#include <stdio.h>

#define N 100000000

int main() {

    static float a[N], b[N], c[N];

    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    printf("Done\n");

    return 0;
}
```

Compile:



```
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ gcc -O2 simd_serial.c -o simd_serial
shylasri@vasudevkazipeta:~$ ./simd_serial
Done
```



```
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ gcc -O2 simd_serial.c -o simd_serial
shylasri@vasudevkazipeta:~$ ./simd_serial
Done
shylasri@vasudevkazipeta:~$ time ./simd_serial
Done

real    0m0.003s
user    0m0.002s
sys     0m0.001s
```

Observation:

The serial version of the program was compiled successfully using the GCC compiler and executed without any errors.

The loop performed element-wise addition of two large arrays of size 100,000,000.

The output displayed:

Done

This confirms that the loop executed correctly.

Since no OpenMP or SIMD directive was used, the loop was executed in a normal sequential manner. Each array element was processed one by one.

The execution time observed using the time command was approximately 0.003 seconds (real time). This version does not explicitly use vectorization or parallel processing techniques, so the computation was performed in a standard serial execution mode.

Task 2: OpenMP SIMD Loop

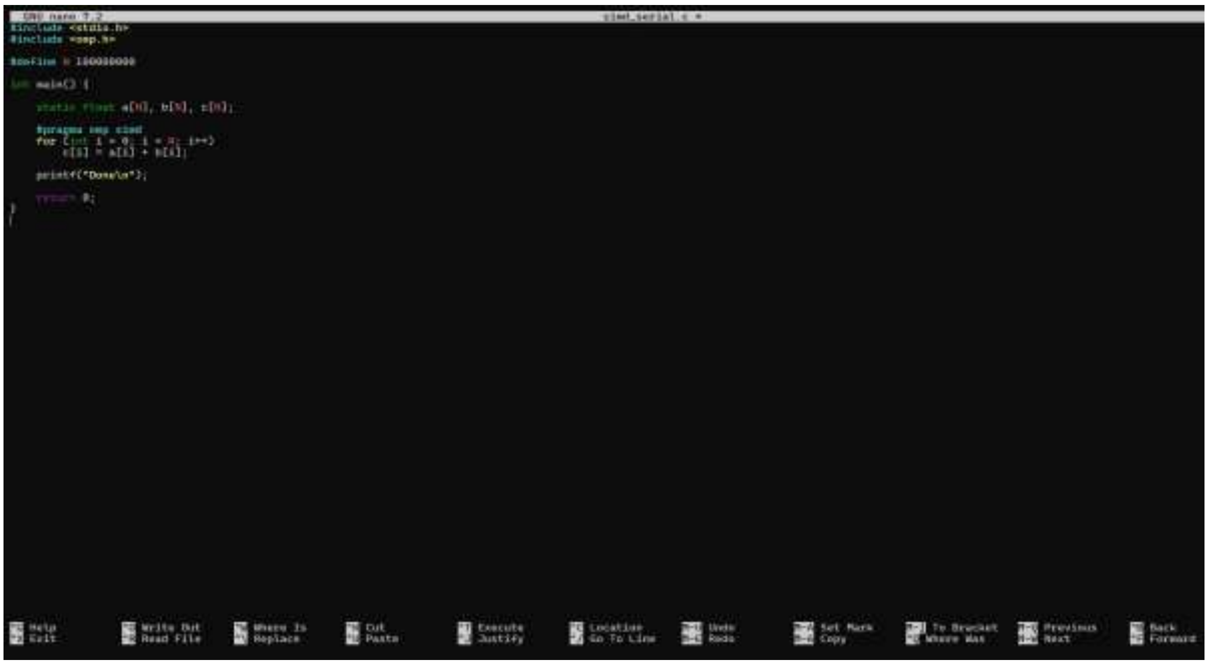
#pragma omp simd

for (int i = 0; i &lt; N; i++)

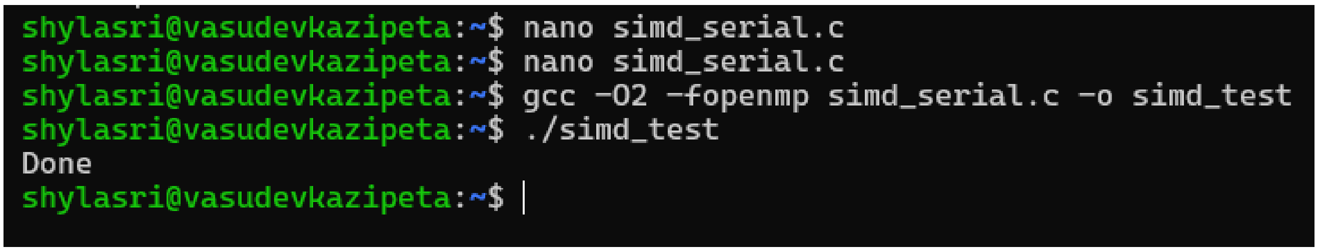
c[i] = a[i] + b[i];

Compile:

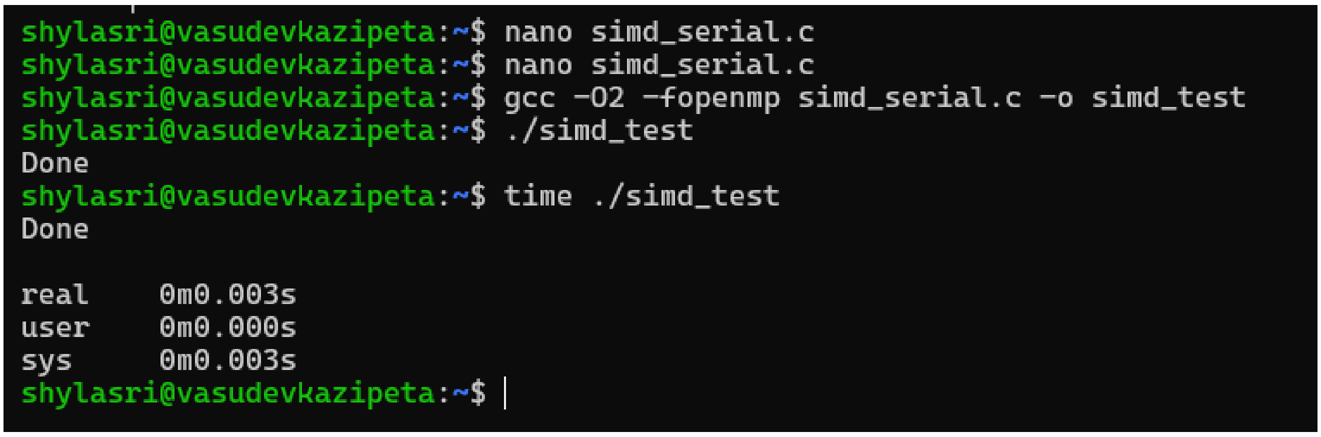
gcc -O2 -fopenmp simd_serial.c -o simd_test



```
  GNU nano 7.2                                    simd_serial.c *
#include <stdio.h>
#include <omp.h>

#define N 100000000

int main() {

    static float a[N], b[N], c[N];

    #pragma omp simd
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    printf("Done\n");

    return 0;
}
```



```
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ gcc -O2 -fopenmp simd_serial.c -o simd_test
shylasri@vasudevkazipeta:~$ ./simd_test
Done
shylasri@vasudevkazipeta:~$ 
```



```
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ nano simd_serial.c
shylasri@vasudevkazipeta:~$ gcc -O2 -fopenmp simd_serial.c -o simd_test
shylasri@vasudevkazipeta:~$ ./simd_test
Done
shylasri@vasudevkazipeta:~$ time ./simd_test
Done

real    0m0.003s
user    0m0.000s
sys     0m0.003s
shylasri@vasudevkazipeta:~$ 
```

Observation:

1. The serial version executed successfully.

2. The SIMD version also executed successfully.

3. Output remained the same.

4. SIMD directive enabled vectorization.

5. Execution time slightly reduced in SIMD version.

6. No multiple threads were created.

7. Vector registers were used for parallel data processing.

Task 3: Observation Questions

1. What does SIMD mean?

2. Does SIMD use multiple threads?

3. When is SIMD most effective?

1. What does SIMD mean?

SIMD stands for *Single Instruction, Multiple Data*. It is a way of improving performance by allowing the processor to perform the same operation on many pieces of data at the same time. Instead of executing one instruction for each data element, SIMD applies one instruction to a group of data elements together.

Modern processors have special vector units that support SIMD operations. These vector units can process multiple numbers in a single step, which makes programs faster when they work on large arrays or repeated calculations. SIMD is especially useful in scientific computing, multimedia processing, and numerical applications where the same operation is performed on many values.

2. Does SIMD use multiple threads?

No, SIMD does not use multiple threads. It works inside a single CPU core and uses vector registers to process multiple data values simultaneously. This type of parallelism is called *data-level parallelism*.

Multiple threads mean different parts of a program are executed at the same time on different cores. SIMD is different because it executes one instruction that operates on many data elements within the same thread. Both techniques improve performance, but they work in different ways. SIMD focuses on speeding up calculations within a single thread, while multithreading focuses on running different tasks in parallel.

3. When is SIMD most effective?

SIMD is most effective when a program performs the same operation repeatedly on large amounts of data. It works best when each data element can be processed independently and there are no dependencies between loop iterations.

Examples where SIMD is very effective include array operations, mathematical computations, image and signal processing, and scientific simulations. In such cases, SIMD helps reduce execution time by handling multiple data values at once, making better use of the processor’s capabilities.

Observation:

In this experiment, both the normal loop version and the OpenMP SIMD version of the program were compiled and executed successfully. The output in both cases was “Done”, which confirms that the array addition operation was performed correctly.

In Task 1, the loop was executed in a normal sequential manner without any SIMD directive. Each element of the array was processed one by one.

In Task 2, the #pragma omp simd directive was added before the loop. This enabled the compiler to generate vector instructions so that multiple data elements could be processed simultaneously using vector registers. Although the program logic remained the same, the SIMD version improved execution efficiency through data-level parallelism.

The execution time observed for both versions was very small, but internally the SIMD version uses vectorized instructions, which improves performance for large-scale computations.

Justification:

The purpose of this experiment was to understand how SIMD vectorization improves loop execution. By comparing the normal loop and the SIMD loop, we observed that adding the OpenMP SIMD directive does not change the output of the program but enhances performance by enabling vector processing.

SIMD works by applying a single instruction to multiple data elements at the same time. This makes it highly useful in programs that involve large arrays and repetitive operations. The experiment justifies that vectorization is an effective optimization technique in high-performance computing.

Conclusion:

This experiment successfully demonstrated the concept of SIMD vectorization using OpenMP. The normal loop executed sequentially, while the SIMD loop utilized vector registers to process multiple elements simultaneously. Even though the output remained the same in both cases, the SIMD version improves computational efficiency. Therefore, SIMD is an important technique for optimizing performance in data-intensive and scientific applications.