Programming for Peformance, Assignment

1. Know your Computer (KYC)

Identify the following information with respect to the computing system you are using.

1. CPU model, generation, frequency range, number of cores, hyper threading availability,
   SIMD ISA, cache size, main memory bandwidth, etc.
2. Theoretically estimate peak FLOPS per core and per processor. Compare the peak FLOPS per core with what you get using Whetstone benchmark program. Design and
   develop a benchmark program(s) using which the peak FLOPS per core and processor
   are achieved. You have to write a parallel program to achieve peak FLOPS per for processor. Report the theoretical and empirically achieved peak FLOPS.
3. What is the main memory size? Is it DDR3 or DDR4 ? Maximum main memory bandwidth availability. Check the main memory bandwidth using the Stream benchmark program. Write a benchmark program which achieves the peak main memory
   bandwidth.
4. What is the secondary storage structure size? SSD or HDD. Read/Write bandwidth of the secondary storage structure. If you have both SSD and HDD on your system, compare their performance.

1.

CPU Model	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
Generation	10
Frequency range	800-4500 MHz
Total cores	4
Total threads	8

Hyper threading availability	Yes
SIMD ISA	mmx sse sse2 mmxext ssse3 fma sse4_1 sse4_2 avx
Cache size	L1d cache: 128 KiB L1i cache: 128 KiB L2 cache: 1 MiB L3 cache: 8 MiB
Max Memory Bandwidth	45.8 GB/s

2.

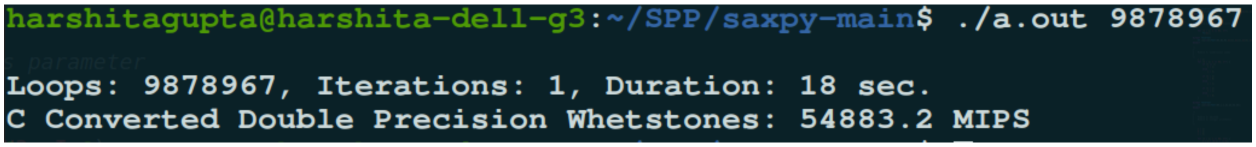
number of sockets = 1
cores per socket = 4, cpu_frequency = 2.5 GHz

cycles per second = 2.5*10e9, flops per cycle = 8
flops_per_processor = flops_per_cycle * cycles_per_second * cores_per_socket * sockets

8*2.5*10e9*4*1/10e9 = 80 GLOPS/sec
flops_per_core = flops_per_processor/cores_per_processsor

80/4 = 20 GLOPS/s

Whetstone Benchmark

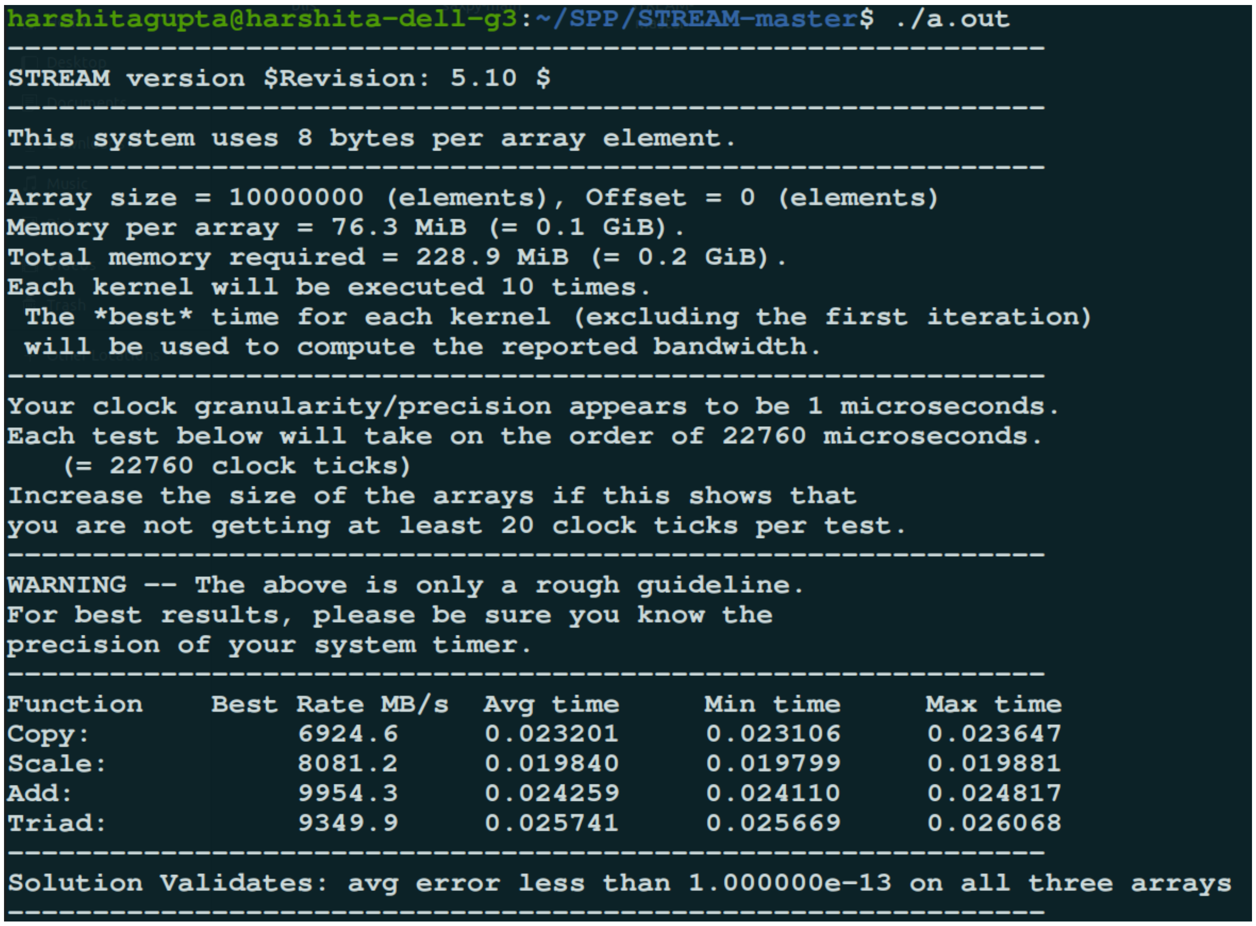

```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./a.out 9878967

Loops: 9878967, Iterations: 1, Duration: 18 sec.
C Converted Double Precision Whetstones: 54883.2 MIPS
```

FLOPS per core	20 GFLOPS/s
FLOPS per processor	80 GFLOPS/s
Wheatstone benchmark	54.883 GFLOPS/s

3.

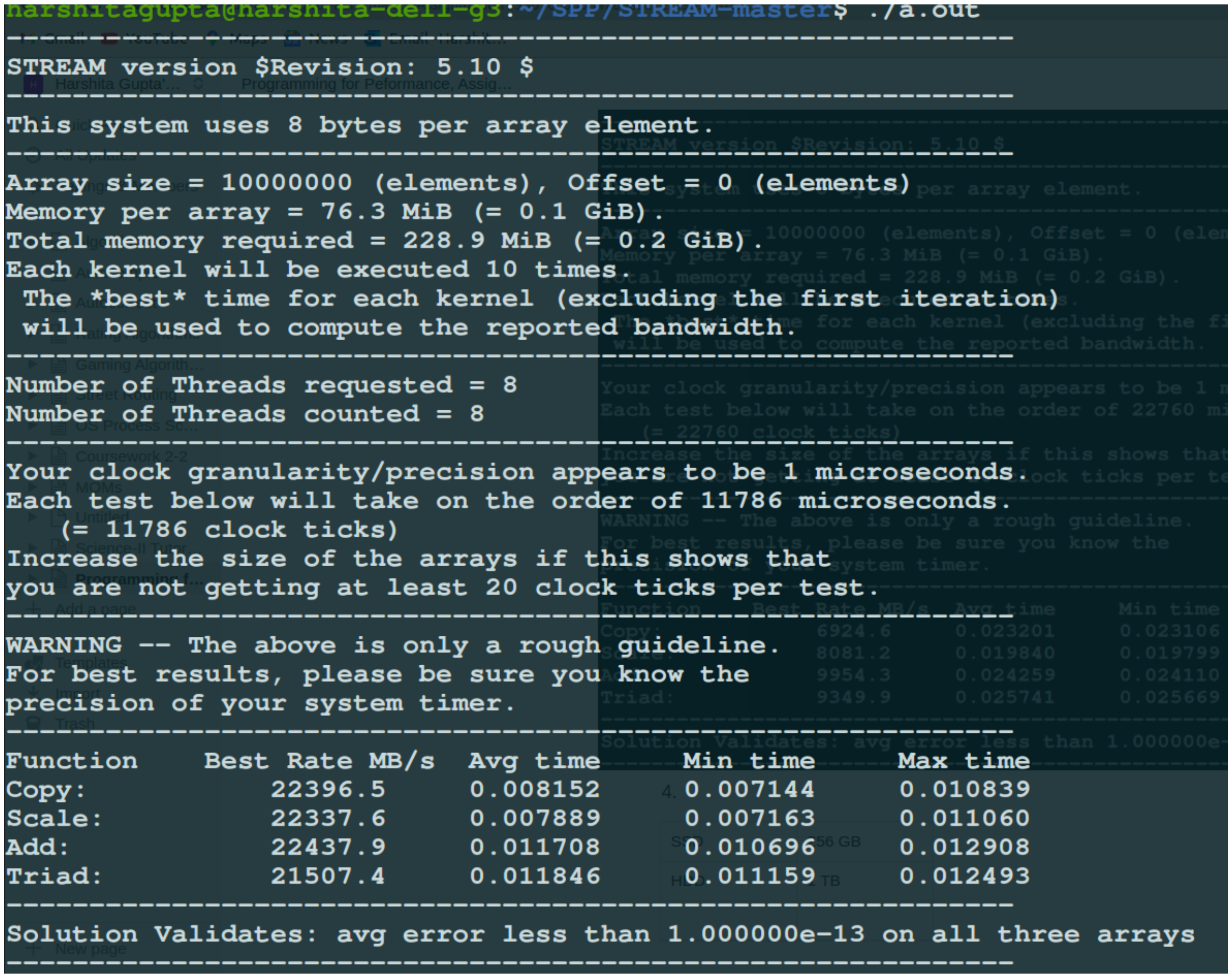
Main memory size	128 GB
Type	DDR4-2933
Max Memory Bandwidth	45.8 GB/s
Using Stream benchmark program	

gcc



```
harshitagupta@harshita-dell-g3:~/SPP/STREAM-master$ ./a.out
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 22760 microseconds.
   (= 22760 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            6924.6     0.023201     0.023106     0.023647
Scale:           8081.2     0.019840     0.019799     0.019881
Add:             9954.3     0.024259     0.024110     0.024817
Triad:           9349.9     0.025741     0.025669     0.026068
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

icc

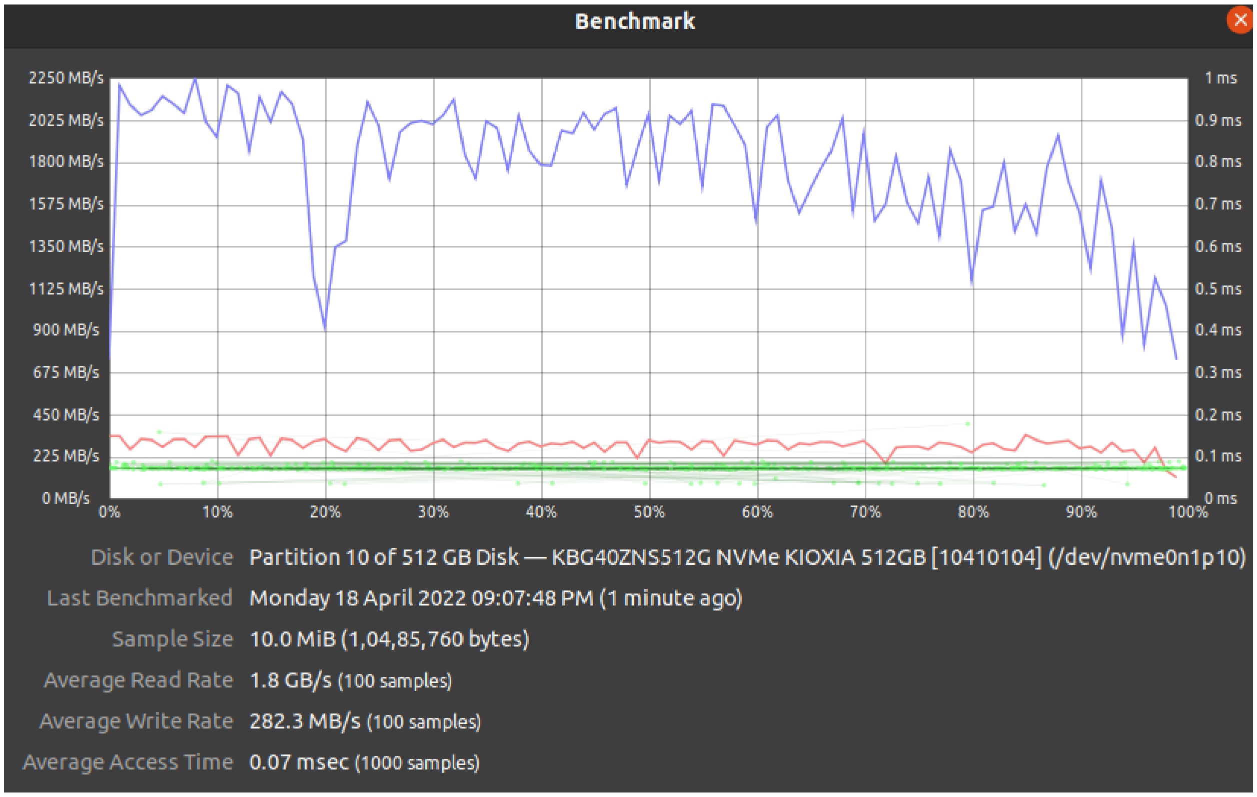


```
harshitagupta@harshita-dell-g3:~/SPP/STREAM-master$ ./a.out
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 8
Number of Threads counted = 8
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 11786 microseconds.
   (= 11786 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           22396.5     0.008152     0.007144     0.010839
Scale:          22337.6     0.007889     0.007163     0.011060
Add:            22437.9     0.011708     0.010696     0.012908
Triad:          21507.4     0.011846     0.011159     0.012493
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

4.

SSD	512 GB
HDD	1 TB
Average Read Rate (SSD)	1.8 GB/s
Average Write Rate (SSD)	282.3 MB/s

My linux system is mounted only on SSD.



Disk or Device	Partition 10 of 512 GB Disk — KBG40ZNS512G NVMe KIOXIA 512GB [10410104] (/dev/nvme0n1p10)
Last Benchmarked	Monday 18 April 2022 09:07:48 PM (1 minute ago)
Sample Size	10.0 MiB (1,04,85,760 bytes)
Average Read Rate	1.8 GB/s (100 samples)
Average Write Rate	282.3 MB/s (100 samples)
Average Access Time	0.07 msec (1000 samples)

2. Know your Cluster

The aggregate theoretical peak performance of Ada is 70.66 TFLOPS (CPU) and the Abacus cluster provides about 14 TFLOPS of peak computing power.

3. BLAS Problems

Q. BLAS stands for Basic Linear Algebra Subprogams. Please read the BLAS wiki page https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms and the NetLib page http://www.netlib.org/blas/. Please check and install Blis which is a specific implementation of BLAS problems https://github.com/flame/blis.
In this assignment we will try to develop high performance code for a subset of BLAS programs. You have to document your journey systematically, like how with each iteration
the performance of the program has improved (or decreased!) and what is your approach for
design space exploration, and finally submit a detailed final report, along with code. The following are BLAS programs you have to optimize. Consider both floating-point and

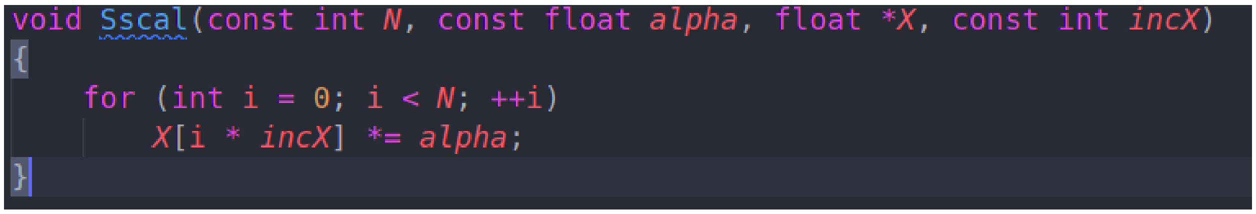
double
precision version of codes.

3.1 BLAS Level 1

BLAS Level 1 consists of Scalar-Vector and Vector-Vector operations.

1. (**xSCAL**) Scalar multiplication of a vector. x in xSCAL stands for floating or double.
   X = αX

SSCAL



```
void Sscal(const int N, const float alpha, float *X, const int incX)
{
    for (int i = 0; i < N; ++i)
        X[i * incX] *= alpha;
}
```

Operational Intensity

Dimensions of vector X = n

Number of Floating Point Operations in the operation, **X = αX = n**

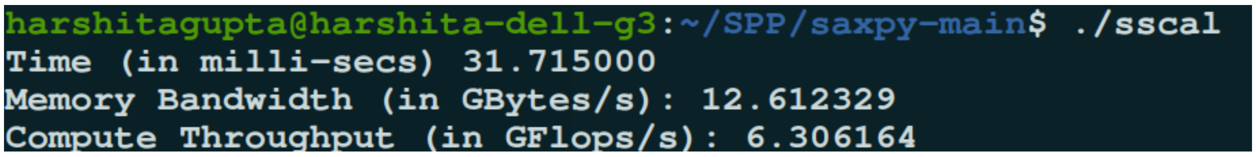
Number of bytes fetched from memory = n floating point numbers = 4.0*n bytes

a can be stored in cache and only X is fetched from memory.

Operational Intensity = n/4.0*n = **1/4.0**

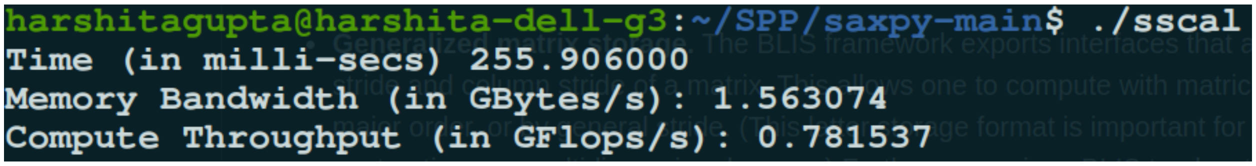
What are the execution times using gcc -O3 and icc -O3?

gcc -O3



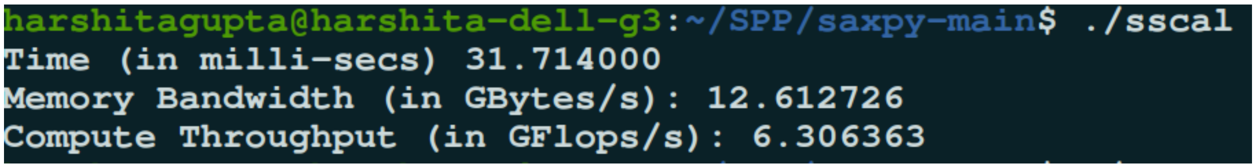
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 31.715000
Memory Bandwidth (in GBytes/s): 12.612329
Compute Throughput (in GFlops/s): 6.306164
```

gcc -O0



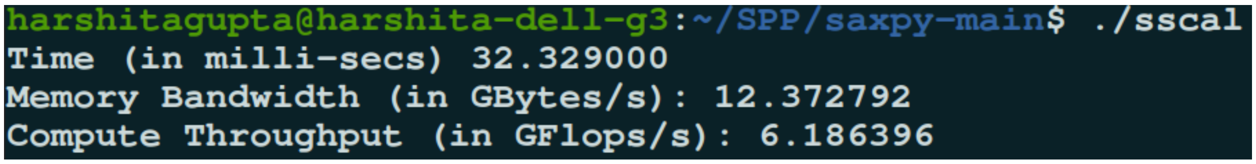
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 255.906000
Memory Bandwidth (in GBytes/s): 1.563074
Compute Throughput (in GFlops/s): 0.781537
```

#pragma ivdep



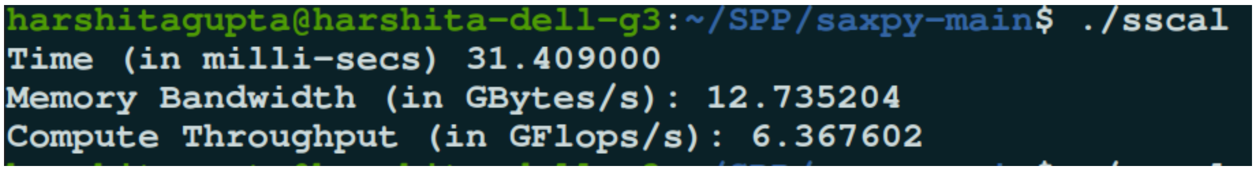
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 31.714000
Memory Bandwidth (in GBytes/s): 12.612726
Compute Throughput (in GFlops/s): 6.306363
```

#pragma omp simd



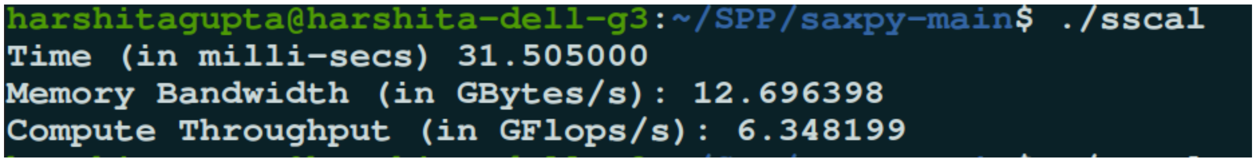
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 32.329000
Memory Bandwidth (in GBytes/s): 12.372792
Compute Throughput (in GFlops/s): 6.186396
```

#pragma omp parallel for



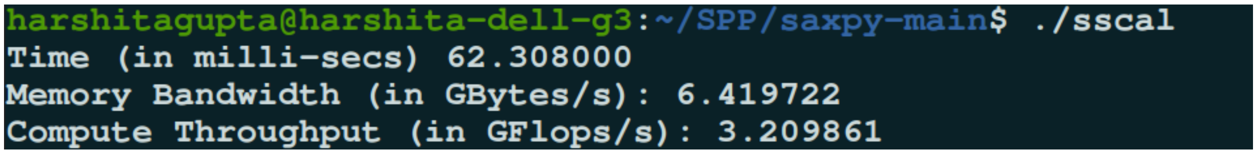
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 31.409000
Memory Bandwidth (in GBytes/s): 12.735204
Compute Throughput (in GFlops/s): 6.367602
```

#pragma vector nodynamic_align



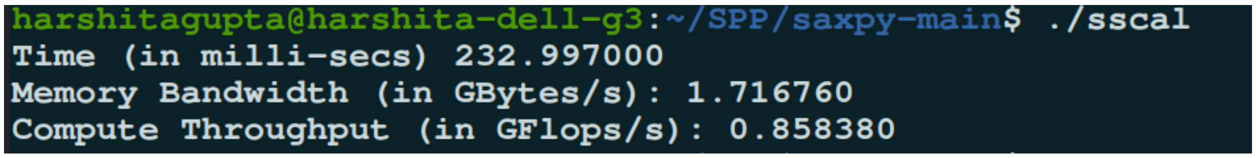
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 31.505000
Memory Bandwidth (in GBytes/s): 12.696398
Compute Throughput (in GFlops/s): 6.348199
```

icc -O3



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 62.308000
Memory Bandwidth (in GBytes/s): 6.419722
Compute Throughput (in GFlops/s): 3.209861
```

icc -O0



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./sscal
Time (in milli-secs) 232.997000
Memory Bandwidth (in GBytes/s): 1.716760
Compute Throughput (in GFlops/s): 0.858380
```

What is the baseline and best execution times you obtained ?

Baseline execution time is **255.90ms** with gcc -O0.

The best execution time is **31.40ms** with gcc -O3 and parallelization, (using #pragma omp parallel for).

What is the speedup?

speedup = 255.90/31.40 = 8.15x

What are the baseline and optimized GFlops/sec?

Baseline GFLOPS/s is 0.781 GFLOPS/sec and optimised GFLOPS/sec is 6.36.

What optimization strategies used to achieve the same?

I used the gcc compiler with -O3 flag and used the directive #pragma omp parallel for to enable parallelization.

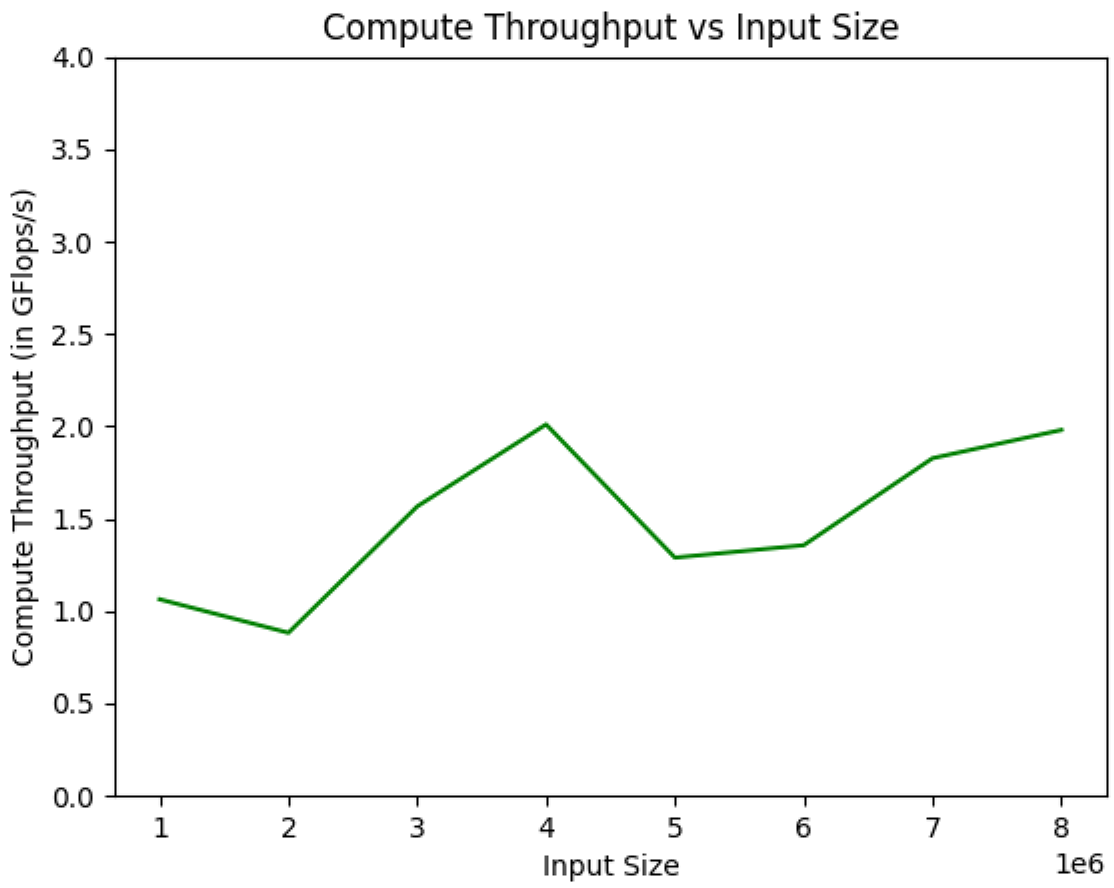
What is the memory bandwidth achieved ?

Maximum memory bandwidth achieved is 12.73 GBytes/s.

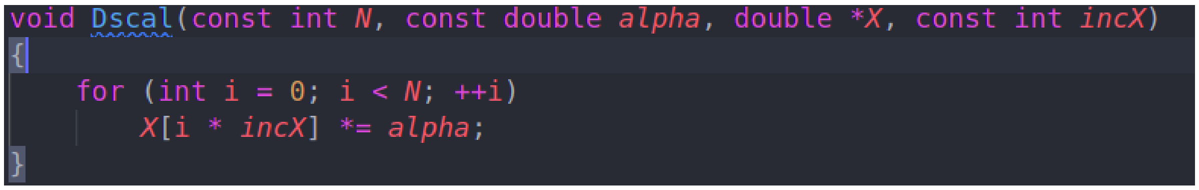
In the baseline and optimized versions, is the problem memory or compute bound ?

In both, baseline and optimised version, the problem is memory bound.

Wherever it is applicable, report the following data for varying input sizes in a graph form ?



▼ **DSCAL**



```
void Dscal(const int N, const double alpha, double *X, const int incX)
{
    for (int i = 0; i < N; ++i)
        X[i * incX] *= alpha;
}
```

Operational Intensity

Dimensions of vector X = n

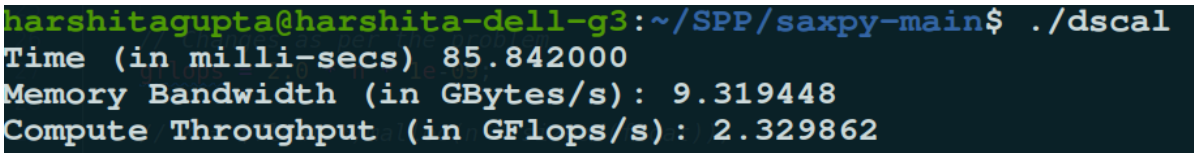
Number of Floating Point Operations in the operation, **X = αX = n**

Number of bytes fetched from memory = n double precision point numbers = 16.0*n bytes

a can be stored in cache and only X is fetched from memory.

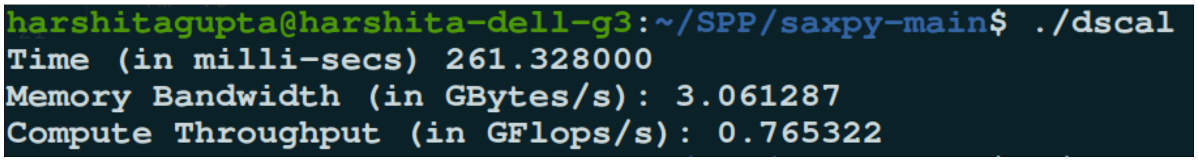
What are the execution times using gcc -O3 and icc -O3 ?

gcc -O3



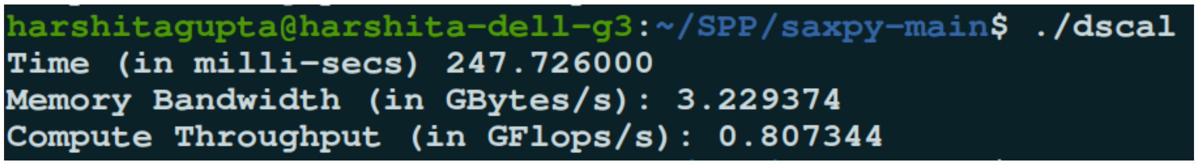
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 85.842000
Memory Bandwidth (in GBytes/s): 9.319448
Compute Throughput (in GFlops/s): 2.329862
```

gcc -O0



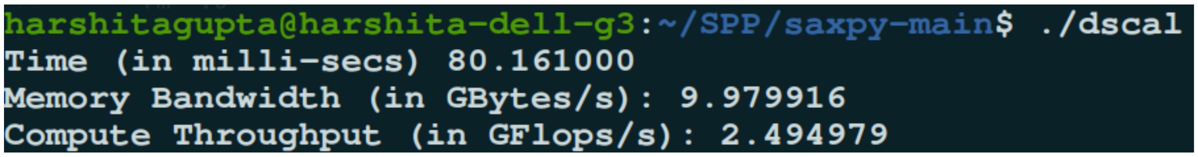
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 261.328000
Memory Bandwidth (in GBytes/s): 3.061287
Compute Throughput (in GFlops/s): 0.765322
```

icc -O0



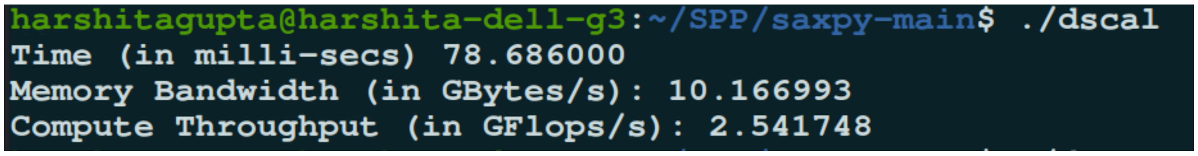
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 247.726000
Memory Bandwidth (in GBytes/s): 3.229374
Compute Throughput (in GFlops/s): 0.807344
```

icc -O3



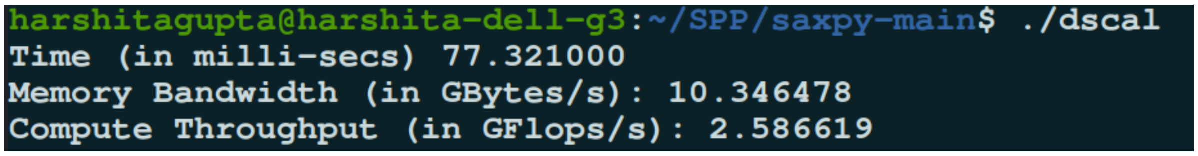
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 80.161000
Memory Bandwidth (in GBytes/s): 9.979916
Compute Throughput (in GFlops/s): 2.494979
```

#pragma ivdep

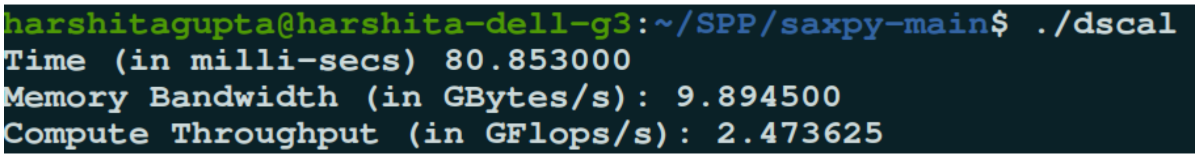


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 78.686000
Memory Bandwidth (in GBytes/s): 10.166993
Compute Throughput (in GFlops/s): 2.541748
```

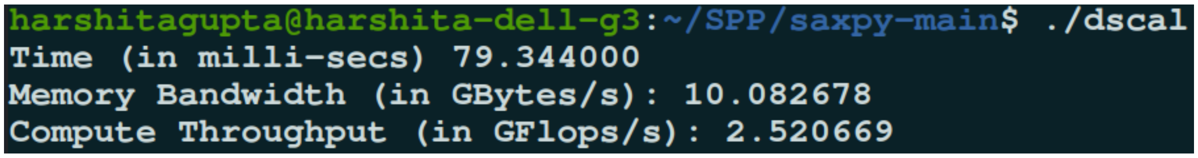
#pragma omp simd


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 77.321000
Memory Bandwidth (in GBytes/s): 10.346478
Compute Throughput (in GFlops/s): 2.586619
```

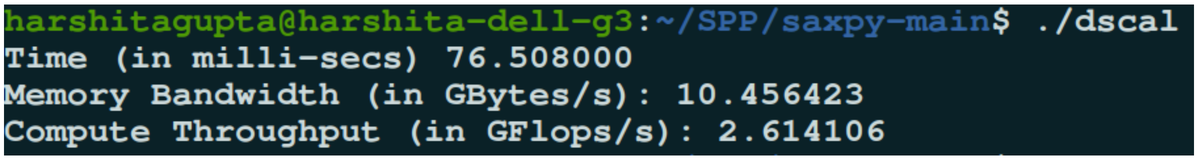
#pragma omp parallel for


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 80.853000
Memory Bandwidth (in GBytes/s): 9.894500
Compute Throughput (in GFlops/s): 2.473625
```

#pragma vector nodynamic_align


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 79.344000
Memory Bandwidth (in GBytes/s): 10.082678
Compute Throughput (in GFlops/s): 2.520669
```

```
# pragma ivdep
# pragma omp simd
# pragma omp parallel for
# pragma vector nodynamic_align
```


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./dscal
Time (in milli-secs) 76.508000
Memory Bandwidth (in GBytes/s): 10.456423
Compute Throughput (in GFlops/s): 2.614106
```

What is the baseline and best execution times you obtained ?

Baseline execution time is **261.32ms** with gcc -O0.

The best execution time is **77.32ms** with gcc -O3 and parallelization, (using #pragma omp parallel for).

What is the speedup?

speedup = 261.32/77.32 = 3.8x

What are the baseline and optimized GFlops/sec?

Baseline GFLOPS/s is 0.76 GFLOPS/sec and optimised GFLOPS/sec is 2.58.

What optimization strategies used to achieve the same?

I used the gcc compiler with -O3 flag and used the directive #pragma omp parallel for to enable parallelization.

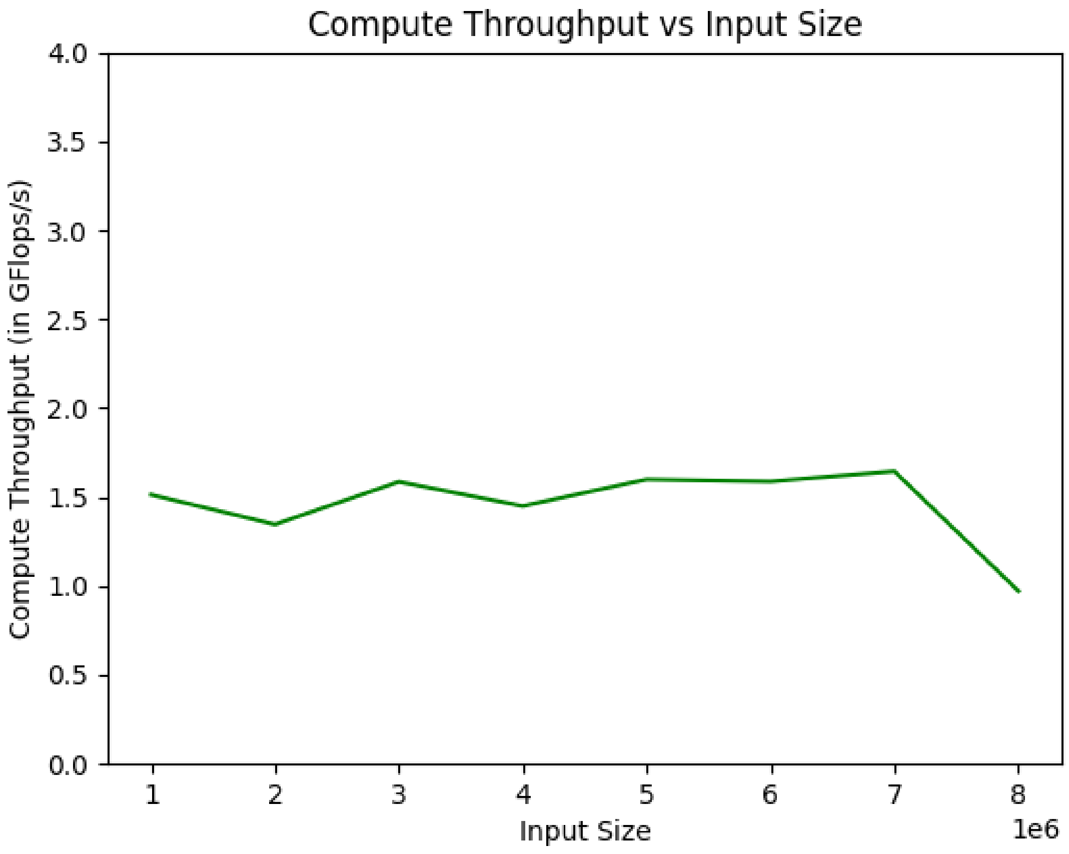
What is the memory bandwidth achieved ?

Maximum memory bandwidth achieved is 10.34 GBytes/s.

In the baseline and optimized versions, is the problem memory or compute bound ?

In both, baseline and optimised version, the problem is memory bound.

Wherever it is applicable, report the following data for varying input sizes in a graph form?



2. (**xAXPY**) Scalar Multiplication of a vector followed by addition with another vector.
   Y = αX + Y

▼ **SAXPY**

Operational intensity Y = αX + Y

Number of floating point operations

aX, n multiplications

aX + Y. n additions

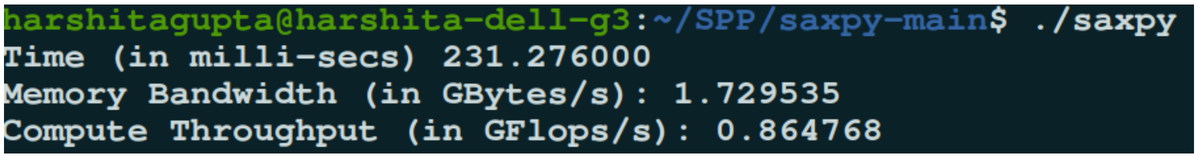
total, 2n operations

Number of bytes fetched from memory, a can be stored in cache, X and Y are fetched from memory, that is 2*n floating point numbers i.e 8*n bytes.

Operational intensity = 2*n/8*n = 1/4

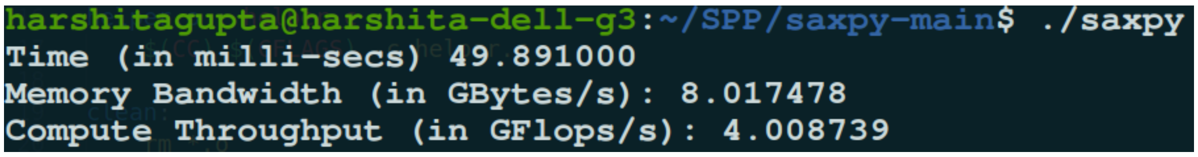
What are the execution times using gcc -O3 and icc -O3?

gcc -O0



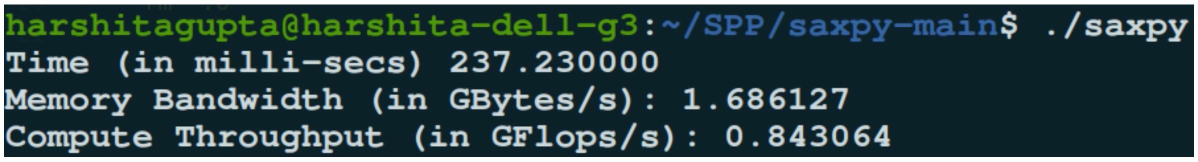
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 231.276000
Memory Bandwidth (in GBytes/s): 1.729535
Compute Throughput (in GFlops/s): 0.864768
```

gcc -O3



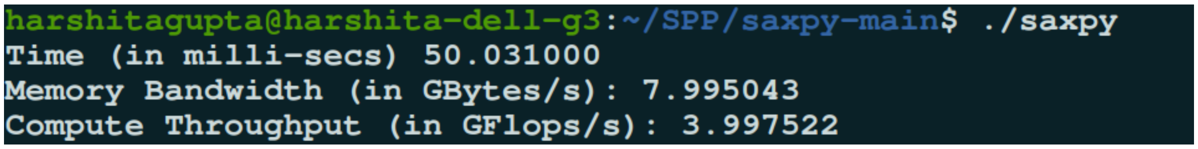
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 49.891000
Memory Bandwidth (in GBytes/s): 8.017478
Compute Throughput (in GFlops/s): 4.008739
```

icc -00



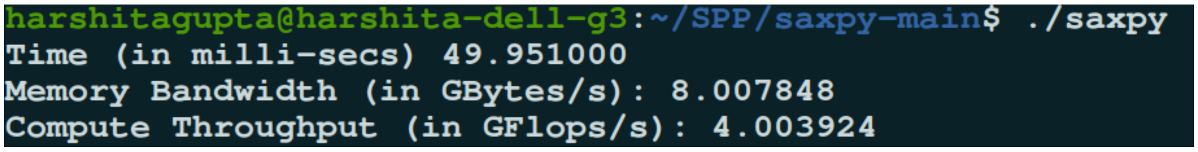
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 237.230000
Memory Bandwidth (in GBytes/s): 1.686127
Compute Throughput (in GFlops/s): 0.843064
```

icc -03



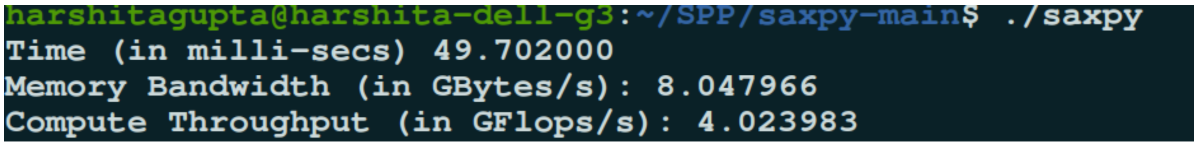
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 50.031000
Memory Bandwidth (in GBytes/s): 7.995043
Compute Throughput (in GFlops/s): 3.997522
```

#pragma ivdep



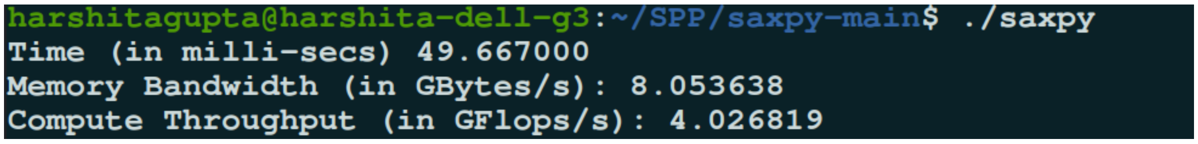
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 49.951000
Memory Bandwidth (in GBytes/s): 8.007848
Compute Throughput (in GFlops/s): 4.003924
```

#pragma omp simd



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 49.702000
Memory Bandwidth (in GBytes/s): 8.047966
Compute Throughput (in GFlops/s): 4.023983
```

#pragma omp parallel for



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Time (in milli-secs) 49.667000
Memory Bandwidth (in GBytes/s): 8.053638
Compute Throughput (in GFlops/s): 4.026819
```

What is the baseline and best execution times you obtained ?

Baseline execution time is **237.23ms** with gcc -O0.

The best execution time is **49.66ms** with gcc -O3 and parallelization, (using #pragma omp parallel for).

What is the speedup?

speedup = 5.5x

What are the baseline and optimized GFlops/sec?

Baseline GFLOPS/s is 0.843 GFLOPS/sec and optimised GFLOPS/sec is 4.02.

What optimization strategies used to achieve the same?

I used the gcc compiler with -O3 flag and used the directive #pragma omp parallel for to enable parallelization.

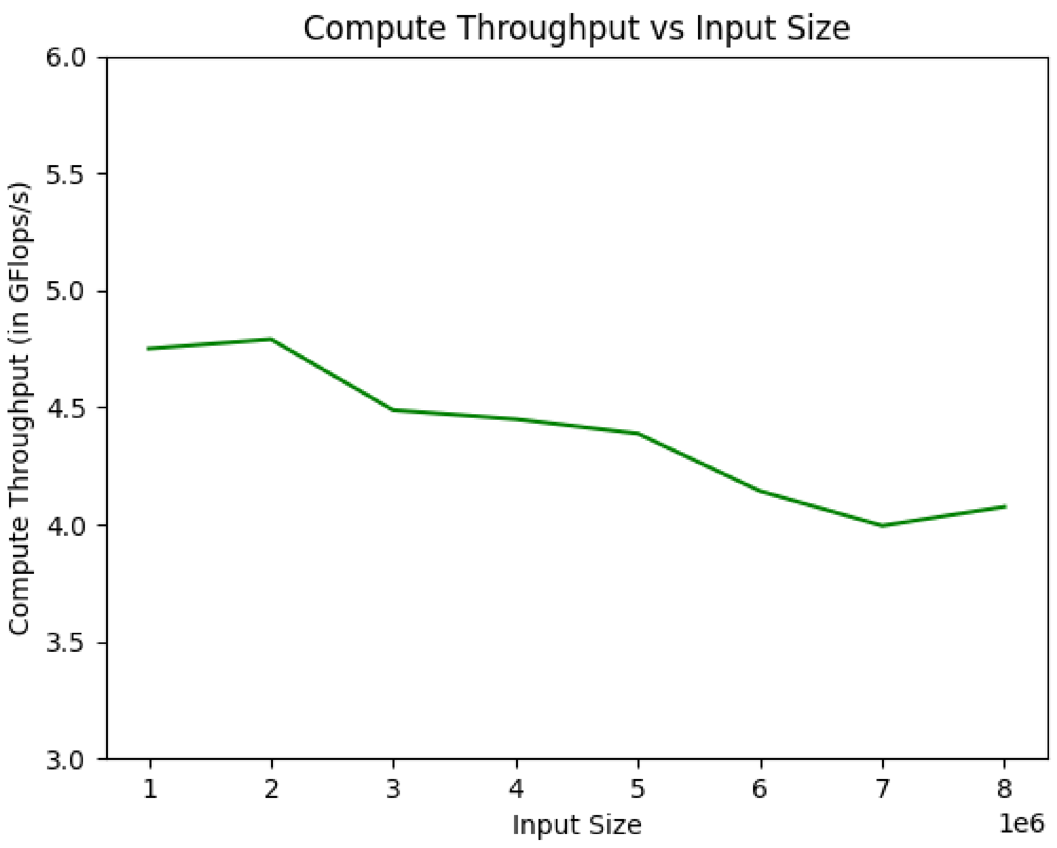
What is the memory bandwidth achieved ?

Maximum memory bandwidth achieved is 8.04 GBytes/s.

In the baseline and optimized versions, is the problem memory or compute bound ?

In both, baseline and optimised version, the problem is memory bound.

Wherever it is applicable, report the following data for varying input sizes in a graph form ?



▼ DAXPY

Operational intensity Y = αX + Y

Number of floating point operations

aX, n multiplications

aX + Y. n additions

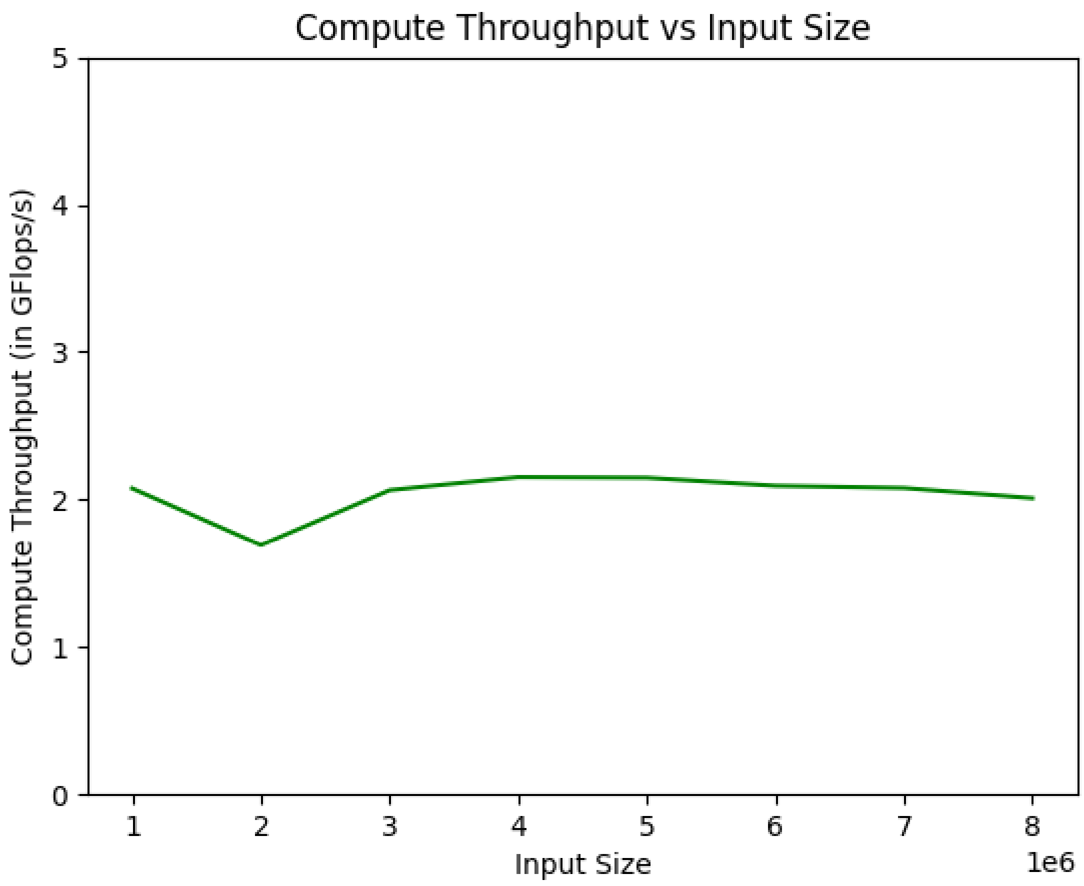
total, 2n operations

Number of bytes fetched from memory, a can be stored in cache, X and Y are fetched from memory, that is 2*n double precision point numbers i.e 16*n bytes.

Operational intensity = 2*n/8*n = 1/4

icc -O3

gcc -O3



So we figure out that #pragma omp parallel which is used to parallelise the for code block is the best optimal approach along with -O3 turns all optimasations of the compiler. A lot of parallel threads executing the loop iteration will be run. #pragma omp simd also works well as it parallelises the data and same instruction is run on many data. #pragma vector nodynamic_align loops is vectorised which disables dynamic alignment optimization for the loop which does not make use of the fact that the vector elements are nearby in memory in an array. This implies it does not perform that well. #pragma ivdep is assuming that there is no dependency and running and might lead to errornus code but here the loop doesnt have dependency so it works allright but its performance is still not good.

3.2 BLAS Level 2

BLAS Level 2 consists of matrix-vector operations.

1. (**xGEMV**) Matrix times a vector.
   Y = αAX + βY or **Y = αAT X + βY**

SGEMV

Operational Intensity

Dimensions of matrix A = m*n

Size of vector X = n*1

Size of vector Y = m*1

Number of Floating Point Operations in the operation, Y = αAX + βY or Y = αAT X + βY

AX = m scalar-vector multiplications

1 scalar vector multiplication = n multiplications + n-1 additions = 2n-1 operations

aAX = m scalar multiplications

bY = m scalar multiplications

finally, aAX + bY = m additions

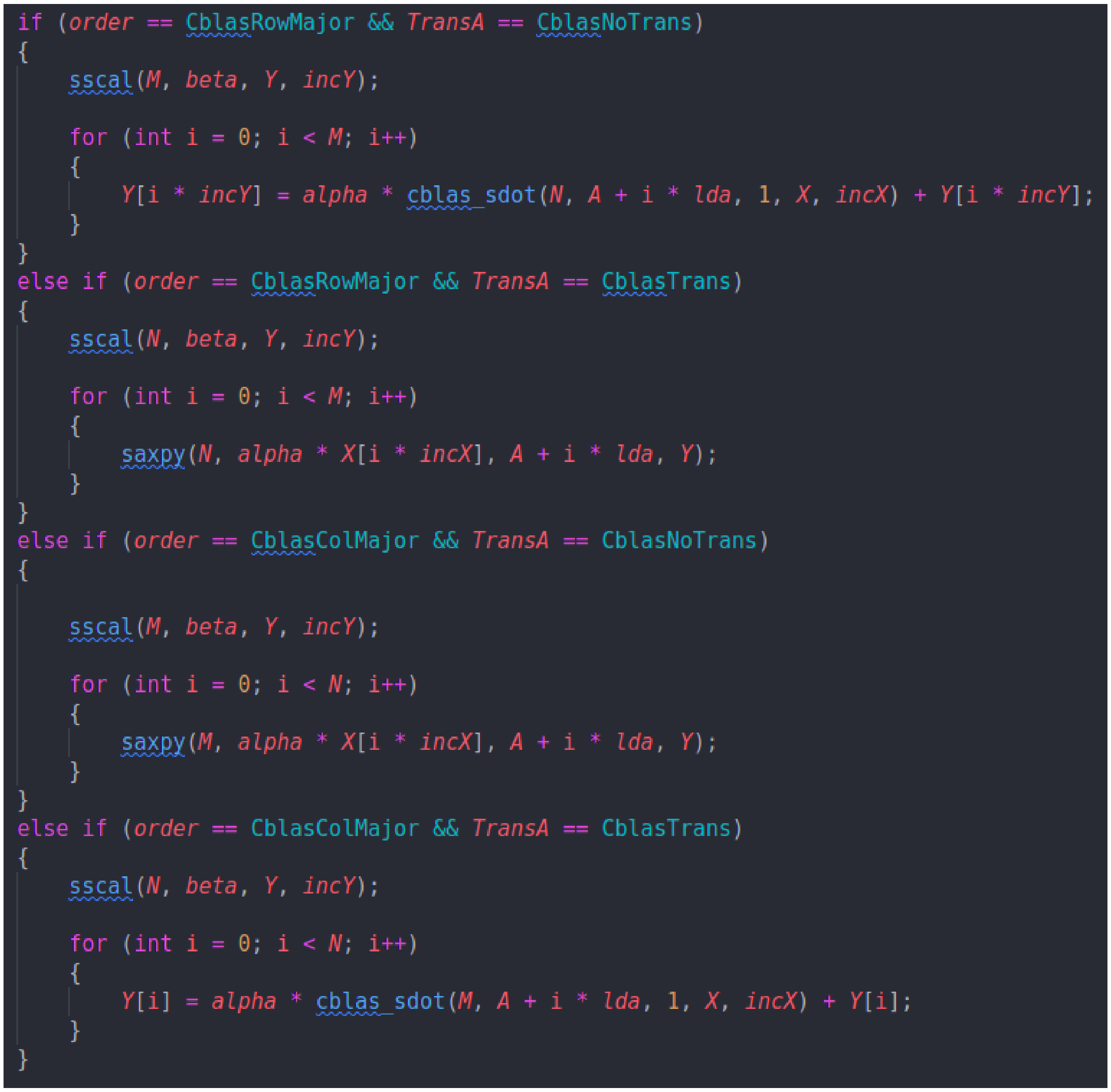
Total operations = m + m + m + m*(2n-1) = 3m + 2mn - m = 2m + 2mn = **2*m*(n+1)**

Number of bytes fetched from memory

a and b can be stored in cache/register

We fetch A, X and Y = (m*n + m + n) floating point numbers i.e. 4*(m*n + m + n) bytes

Operational Intensity = 2*m*(n+1)/4*(m+n+m*n) approx 0.5

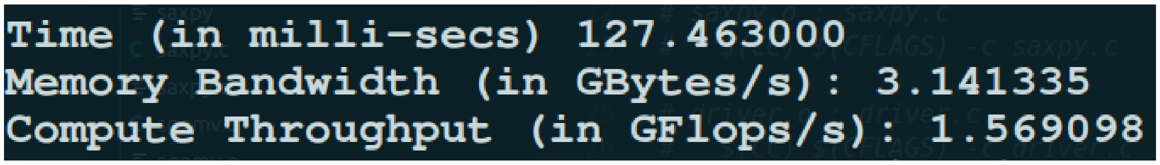


```
if (order == CblasRowMajor && TransA == CblasNoTrans)
{
    sscal(M, beta, Y, incY);

    for (int i = 0; i < M; i++)
    {
        Y[i * incY] = alpha * cblas_sdot(N, A + i * lda, 1, X, incX) + Y[i * incY];
    }
}
else if (order == CblasRowMajor && TransA == CblasTrans)
{
    sscal(N, beta, Y, incY);

    for (int i = 0; i < M; i++)
    {
        saxpy(N, alpha * X[i * incX], A + i * lda, Y);
    }
}
else if (order == CblasColMajor && TransA == CblasNoTrans)
{

    sscal(M, beta, Y, incY);

    for (int i = 0; i < N; i++)
    {
        saxpy(M, alpha * X[i * incX], A + i * lda, Y);
    }
}
else if (order == CblasColMajor && TransA == CblasTrans)
{
    sscal(N, beta, Y, incY);

    for (int i = 0; i < N; i++)
    {
        Y[i] = alpha * cblas_sdot(M, A + i * lda, 1, X, incX) + Y[i];
    }
}
```

Performance Analysis

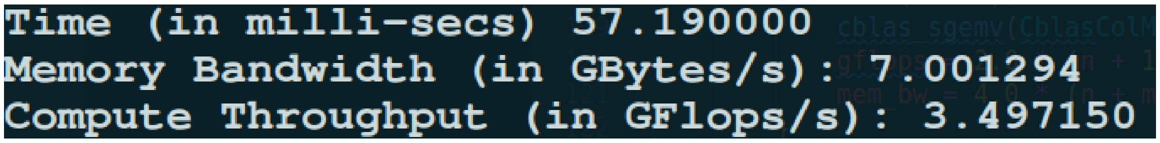
Execution times using gcc -O3

Row major and No transpose



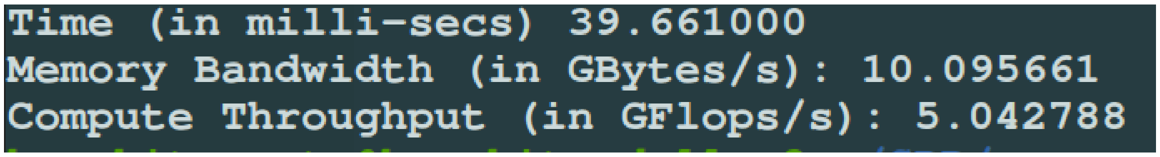
```
Time (in milli-secs) 127.463000
Memory Bandwidth (in GBytes/s): 3.141335
Compute Throughput (in GFlops/s): 1.569098
```

Column major and No transpose



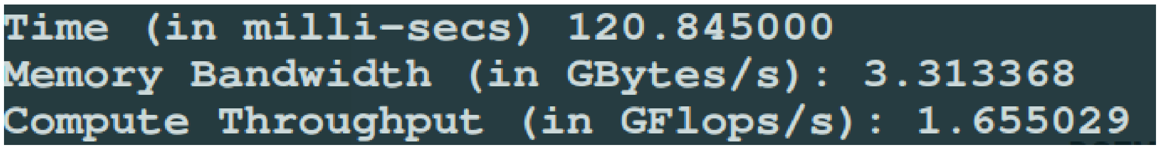
```
Time (in milli-secs) 57.190000
Memory Bandwidth (in GBytes/s): 7.001294
Compute Throughput (in GFlops/s): 3.497150
```

Row major and Transpose



```
Time (in milli-secs) 39.661000
Memory Bandwidth (in GBytes/s): 10.095661
Compute Throughput (in GFlops/s): 5.042788
```

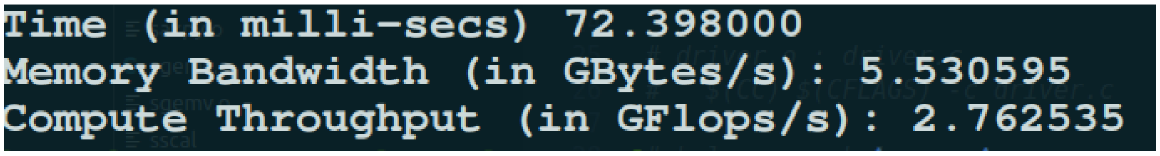
Column major and Transpose



```
Time (in milli-secs) 120.845000
Memory Bandwidth (in GBytes/s): 3.313368
Compute Throughput (in GFlops/s): 1.655029
```

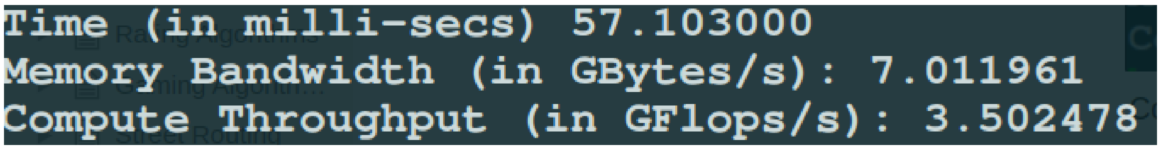
Execution times using icc -O3

Row major and No transpose



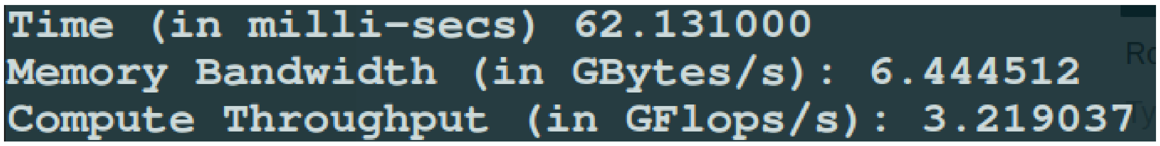
```
Time (in milli-secs) 72.398000
Memory Bandwidth (in GBytes/s): 5.530595
Compute Throughput (in GFlops/s): 2.762535
```

Column major and No transpose



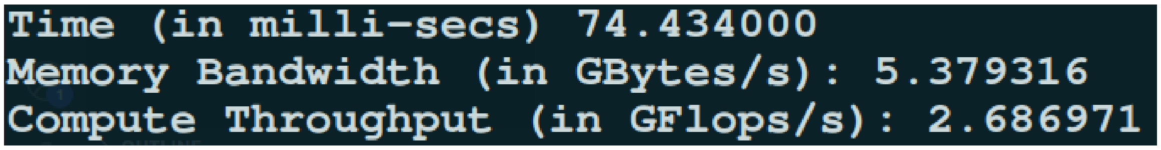
```
Time (in milli-secs) 57.103000
Memory Bandwidth (in GBytes/s): 7.011961
Compute Throughput (in GFlops/s): 3.502478
```

Row major and Transpose



```
Time (in milli-secs) 62.131000
Memory Bandwidth (in GBytes/s): 6.444512
Compute Throughput (in GFlops/s): 3.219037
```

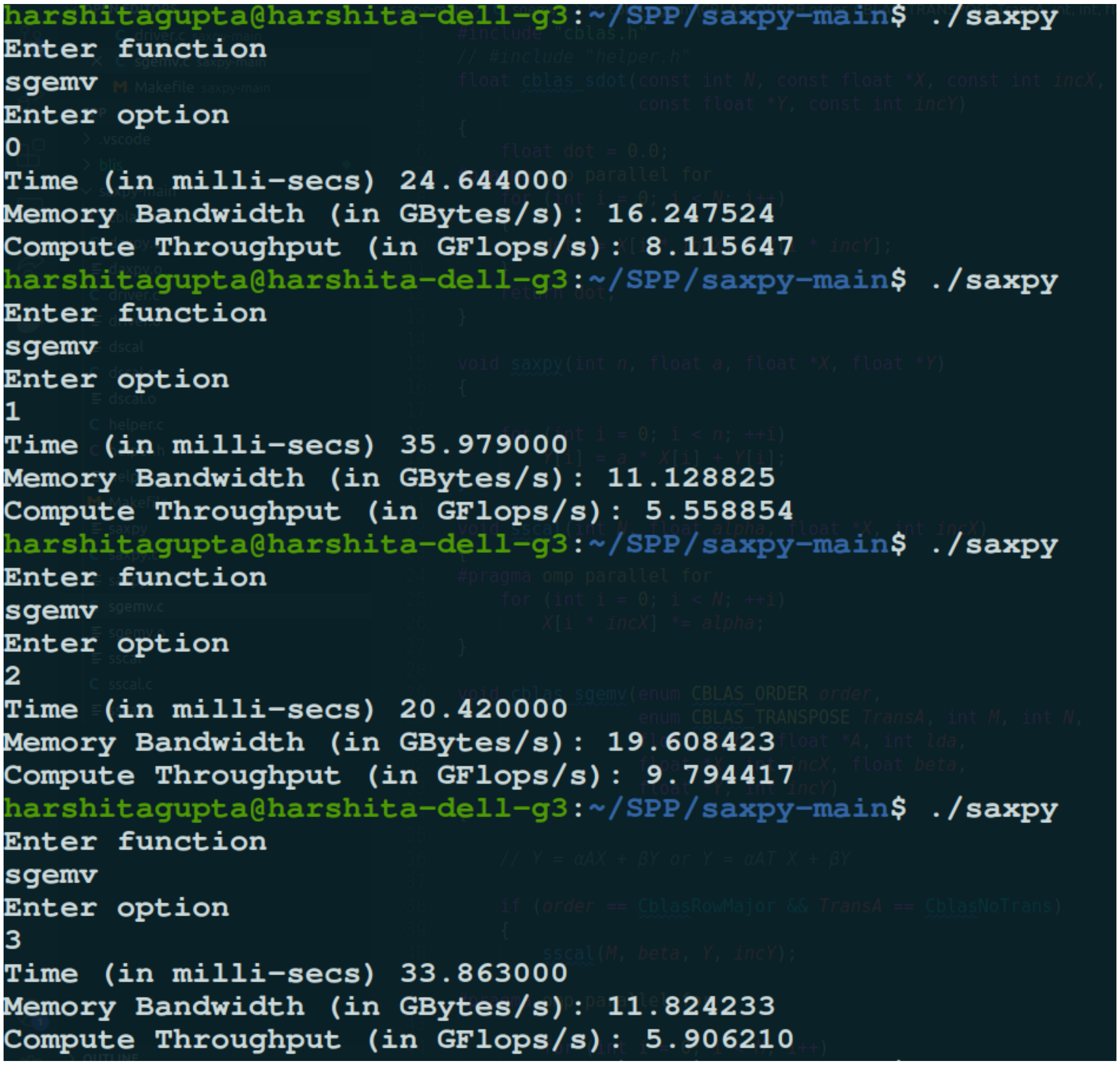
Column major and Transpose



```
Time (in milli-secs) 74.434000
Memory Bandwidth (in GBytes/s): 5.379316
Compute Throughput (in GFlops/s): 2.686971
```

After **parallelization,**

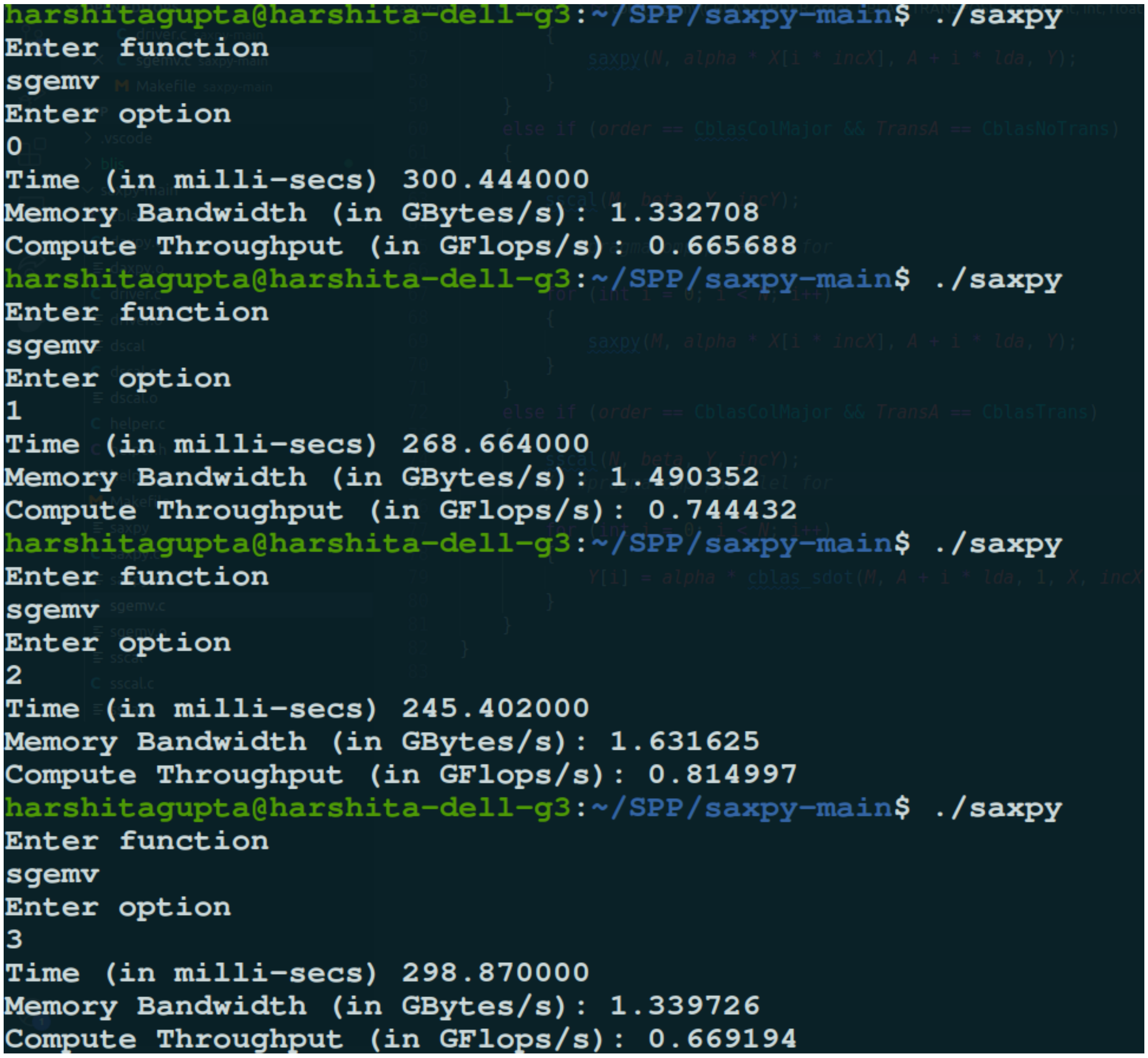
that is after adding #pragma omp parallel for directive before the sscal, cblas_dot and sgemv for loops.


```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
0
Time (in milli-secs) 24.644000
Memory Bandwidth (in GBytes/s): 16.247524
Compute Throughput (in GFlops/s): 8.115647
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
1
Time (in milli-secs) 35.979000
Memory Bandwidth (in GBytes/s): 11.128825
Compute Throughput (in GFlops/s): 5.558854
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
2
Time (in milli-secs) 20.420000
Memory Bandwidth (in GBytes/s): 19.608423
Compute Throughput (in GFlops/s): 9.794417
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
3
Time (in milli-secs) 33.863000
Memory Bandwidth (in GBytes/s): 11.824233
Compute Throughput (in GFlops/s): 5.906210
```

What is the baseline and best execution times you obtained?

Average best execution time we obtain is around 28ms

Baseline time we obtain is **without parallelization and with gcc -O0**



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
0
Time (in milli-secs) 300.444000
Memory Bandwidth (in GBytes/s): 1.332708
Compute Throughput (in GFlops/s): 0.665688
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
1
Time (in milli-secs) 268.664000
Memory Bandwidth (in GBytes/s): 1.490352
Compute Throughput (in GFlops/s): 0.744432
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
2
Time (in milli-secs) 245.402000
Memory Bandwidth (in GBytes/s): 1.631625
Compute Throughput (in GFlops/s): 0.814997
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemv
Enter option
3
Time (in milli-secs) 298.870000
Memory Bandwidth (in GBytes/s): 1.339726
Compute Throughput (in GFlops/s): 0.669194
```

Compute ThroughputAverage baseline execution time = 277.75ms

What is the speedup?

Speedup = 277.75/28 = 9.9

What are the baseline and optimized GFlops/sec?

Average baseline GFLOPS/sec = 0.7175

Average optimised GLOPS/sec = 7.339

What optimization strategies used to achieve the same?

I used icc compiler instead of gcc compiler and used flag -O3 in place of -O0. In addition to this I added parallelization that is added #pragma omp parallel for directive before the sscal, cblas_dot and sgemv for loops.

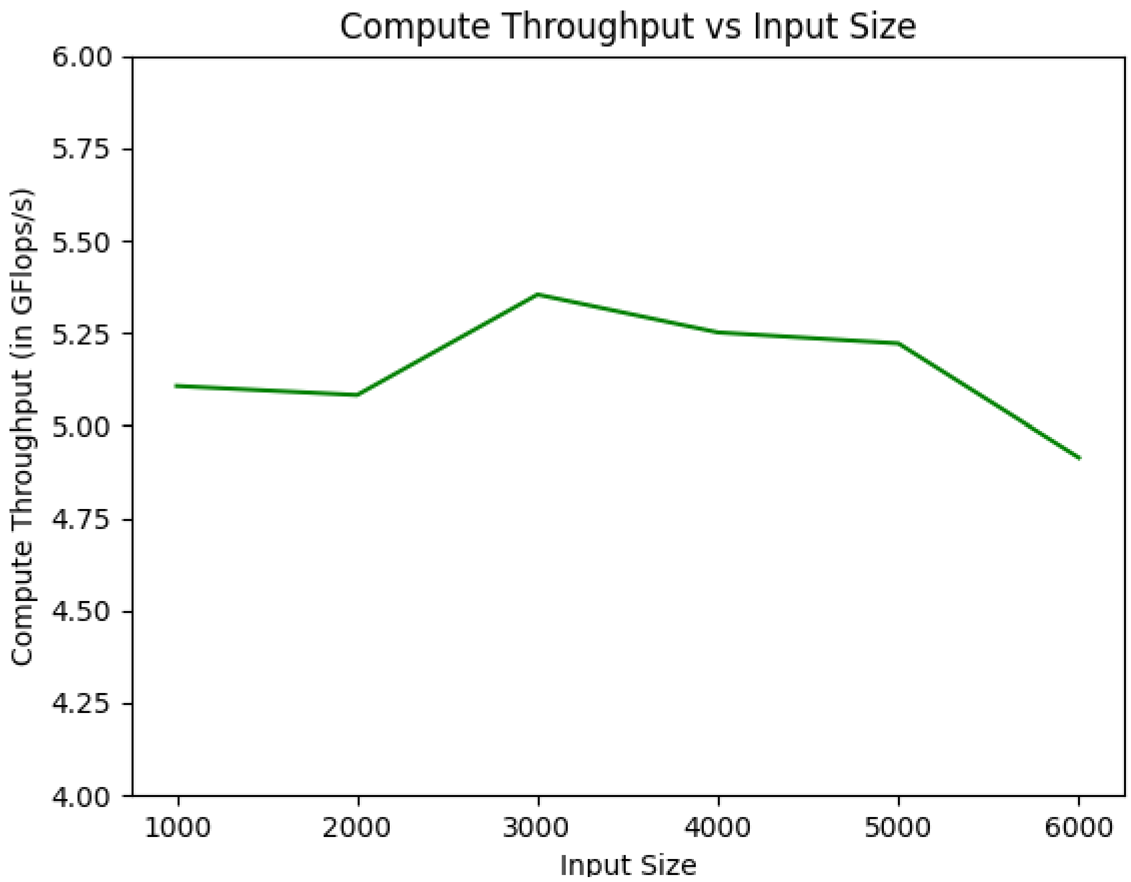
What is the memory bandwidth achieved ?

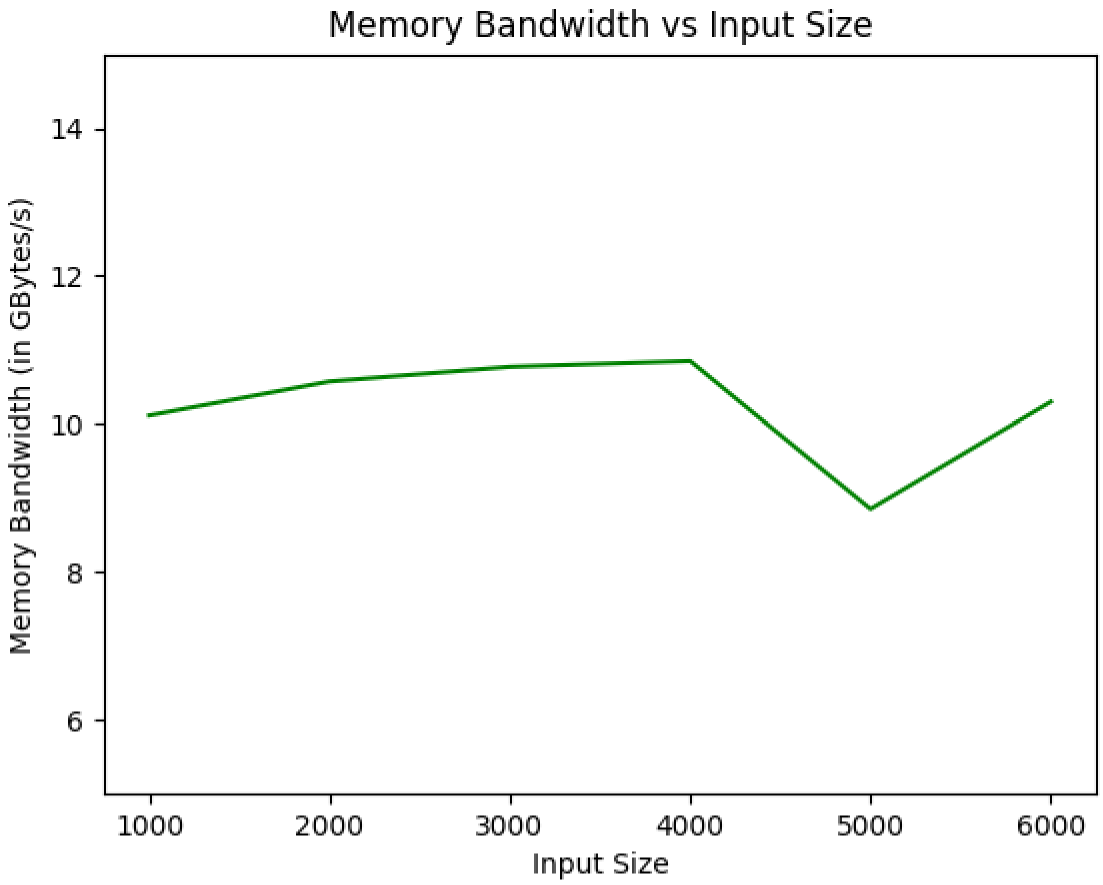
Optimized memory bandwidth achieved is 14.69 GBytes/s.

In the baseline and optimized versions, is the problem memory or compute bound?

In both, baseline and optimised version, the problem is memory bound.

Wherever it is applicable, report the following data for varying input sizes in a graph form ?





SGEMM

C = αop(A)op(B) + βC where op(A) = A or A^T

Operational Intensity

Dimensions of matrices A, B and C = n*n

Number of Floating Point Operations in the operation

AB = For calculating an element in AB, we do a vector-vector multiplication i.e n multiplications + (n-1) additions.

Total operations for calculating AB = n*n*(n+n-1)

For bC, number of operations = n*n

For aAB, number of operations = n*n

For bC + aAB, number of operations = n*n

Total operations = n*n + n*n + n*n + n*n*(2n-1) = 2*n*n*n + 2*n*n = 2*n*n(n +1)

Number of bytes fetched from memory

a and b are stored in cache

A, B, C are fetched from memory = n*n*3 floating point numbers

i.e. 12*n*n bytes

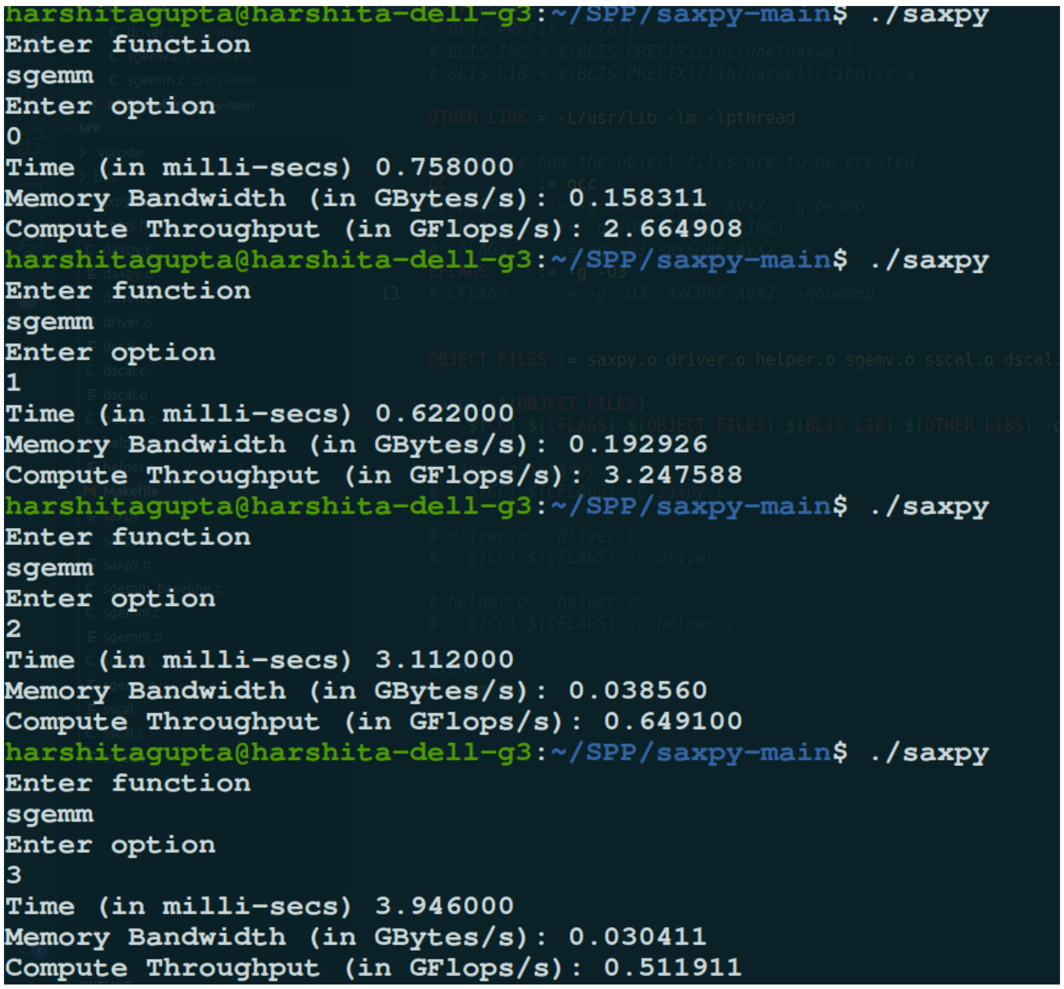
Operational Intensity = 2*(n + 1)/12 = (n + 1)/6

gcc -O1

What are the execution times using gcc -O3 and icc -O3 ?

gcc -O3

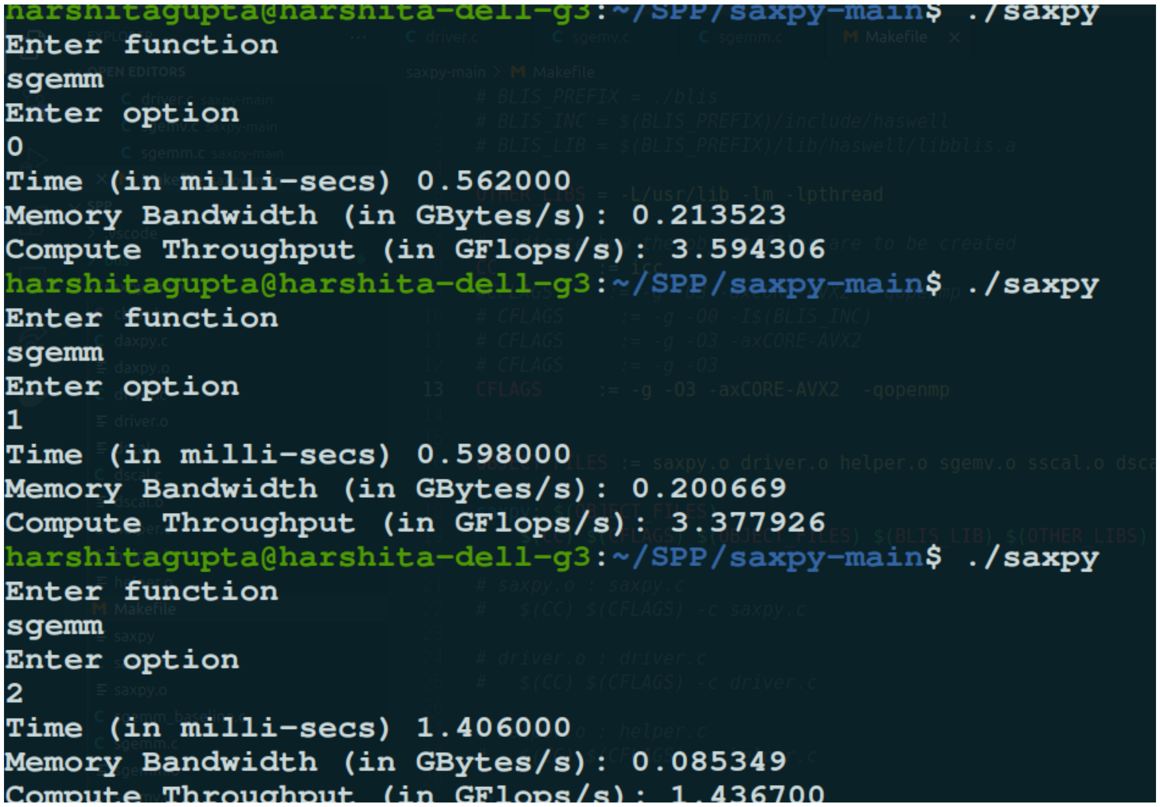
using #pragma omp parallel for



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
0
Time (in milli-secs) 0.758000
Memory Bandwidth (in GBytes/s): 0.158311
Compute Throughput (in GFlops/s): 2.664908
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
1
Time (in milli-secs) 0.622000
Memory Bandwidth (in GBytes/s): 0.192926
Compute Throughput (in GFlops/s): 3.247588
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
2
Time (in milli-secs) 3.112000
Memory Bandwidth (in GBytes/s): 0.038560
Compute Throughput (in GFlops/s): 0.649100
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
3
Time (in milli-secs) 3.946000
Memory Bandwidth (in GBytes/s): 0.030411
Compute Throughput (in GFlops/s): 0.511911
```

icc -O3

removing #pragma omp parallel for



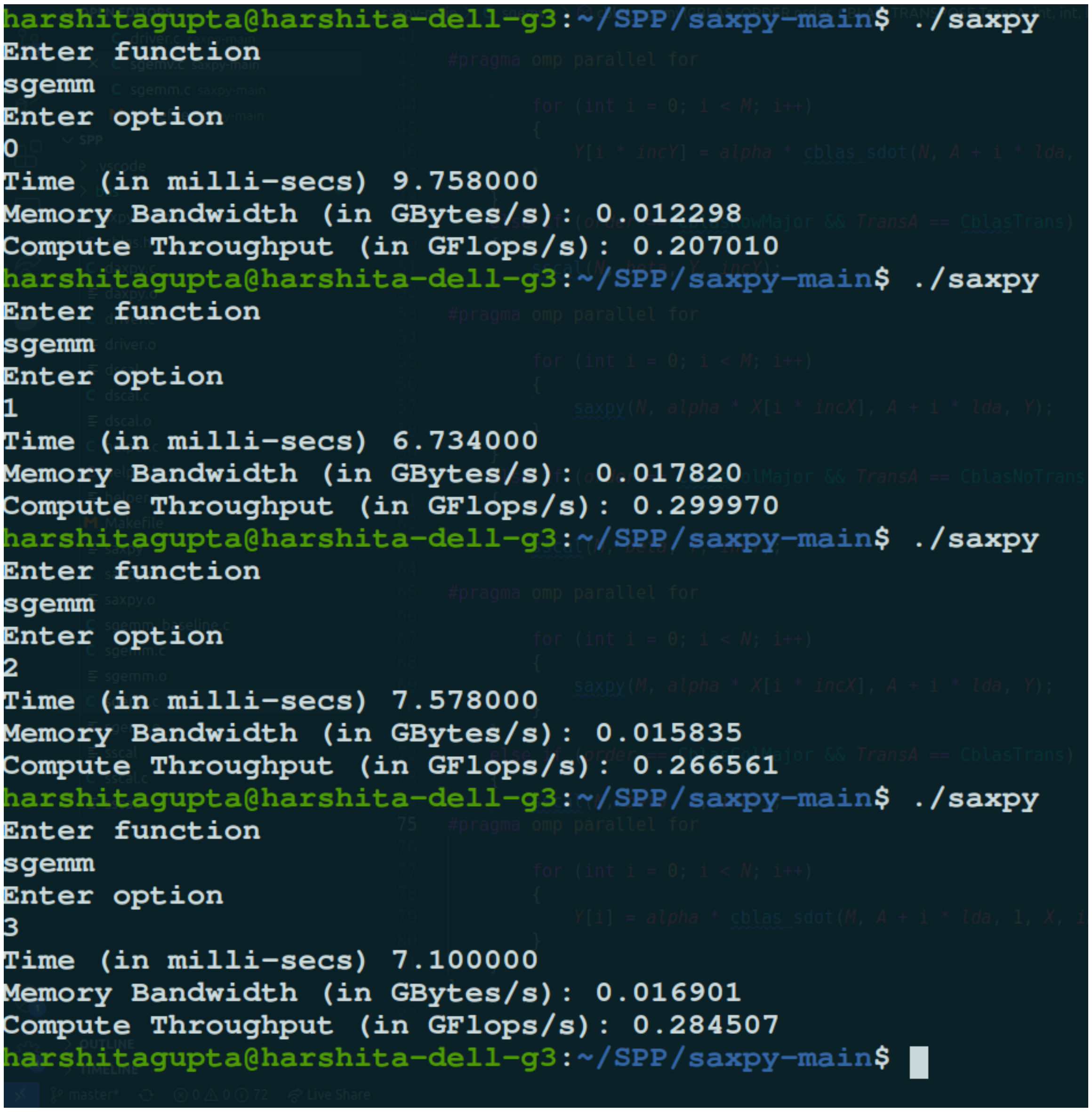
```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
0
Time  (in milli-secs) 0.562000
Memory Bandwidth (in GBytes/s): 0.213523
Compute Throughput (in GFlops/s): 3.594306
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
1
Time  (in milli-secs) 0.598000
Memory Bandwidth (in GBytes/s): 0.200669
Compute Throughput (in GFlops/s): 3.377926
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
2
Time  (in milli-secs) 1.406000
Memory Bandwidth (in GBytes/s): 0.085349
Compute Throughput (in GFlops/s): 1.436700
```

What is the baseline and best execution times you obtained?

The baseline execution time I obtained was 9.75ms

The best execution time I obtained was 0.562ms

gcc -O0



```
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
0
Time (in milli-secs) 9.758000
Memory Bandwidth (in GBytes/s): 0.012298
Compute Throughput (in GFlops/s): 0.207010
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
1
Time (in milli-secs) 6.734000
Memory Bandwidth (in GBytes/s): 0.017820
Compute Throughput (in GFlops/s): 0.299970
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
2
Time (in milli-secs) 7.578000
Memory Bandwidth (in GBytes/s): 0.015835
Compute Throughput (in GFlops/s): 0.266561
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$ ./saxpy
Enter function
sgemm
Enter option
3
Time (in milli-secs) 7.100000
Memory Bandwidth (in GBytes/s): 0.016901
Compute Throughput (in GFlops/s): 0.284507
harshitagupta@harshita-dell-g3:~/SPP/saxpy-main$
```

What is the speedup ?

Speedup = 9.75/0.562 = 17.34

What are the baseline and optimized GFlops/sec?

Average baseline GFLOPS/sec = 0.207 with gcc -O0

Average optimised GLOPS/sec = 3.59 with icc -O3

What optimization strategies used to achieve the same?

I used icc compiler instead of icc compiler and used flag -O3 in place of -O0.

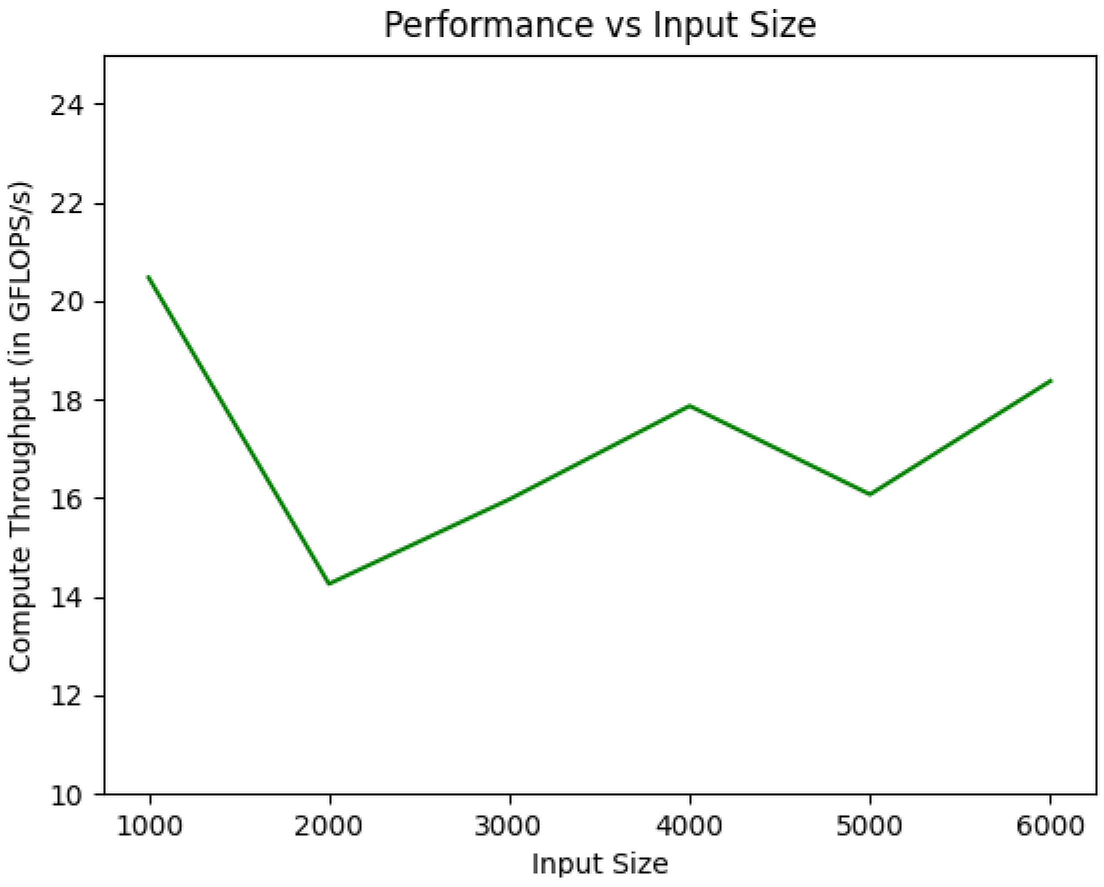
What is the memory bandwidth achieved ?

Optimized memory bandwidth achieved is 0.38 GBytes/s.

In the baseline and optimized versions, is the problem memory or compute bound?

In both, baseline and optimised version, the problem is memory bound.

Wherever it is applicable, report the following data for varying input sizes in a graph form ?


Performance vs Input Size
Compute Throughput (in GFLOPS/s)
Input Size