ES215: Computer Organisation and Architecture

Assignment 1

Guntas Singh Saran
22110089

Github Repo Link

Question 1

Implementing C++ programs to find the first 50 Fibonacci Numbers and find the CPU time taken by them using timespec struct in C.

Only the meat portion of the code has been kept for CPU time calculation.

The **timespec** *struct and* **clock_gettime** *function* (Source1) (Source2)

```
#include <time.h>

struct timespec {
    time_t     tv_sec;   /* Seconds */
    /* ... */  tv_nsec;  /* Nanoseconds [0, 999'999'999] */
};

int clock_gettime(clockid_t clockid, struct timespec *tp);

The functions clock_gettime() and clock_settime() retrieve and set
the time of the specified clock clockid.

The res and tp arguments are timespec structures, as specified in
<time.h>:

    struct timespec {
        time_t   tv_sec;        /* seconds */
        long     tv_nsec;       /* nanoseconds */
    };

The clockid argument is the identifier of the particular clock on
which to act.  A clock may be system-wide and hence visible for all
processes, or per-process if it measures time only within a single
process.

CLOCK_PROCESS_CPUTIME_ID (since Linux 2.6.12)
       This is a clock that measures CPU time consumed by this
       process (i.e., CPU time consumed by all threads in the
       process).  On Linux, this clock is not settable.
```

(a.) Using Recursion

```
Raw Recursion for first 50 Fibonacci numbers
Time taken: 112.299908s
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393
196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986
102334155 165580141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049
```

(b.) Using Loop

```
Loop for first 50 Fibonacci numbers

Time taken: 0.000020s
Time taken: 20000ns

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352
24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170
1836311903 2971215073 4807526976 7778742049
```

(c.) Using Recursion + Memoization

```
Recursion + Memoization for first 50 Fibonacci numbers

Time taken: 0.000011s
Time taken: 11000ns

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025
121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817
39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903
2971215073 4807526976 7778742049
```

(d.) Using Loop + Memoization

```
Loop + Memoization for first 50 Fibonacci numbers

Time taken: 0.000003s
Time taken: 3000ns

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025
121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817
39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903
2971215073 4807526976 7778742049
```

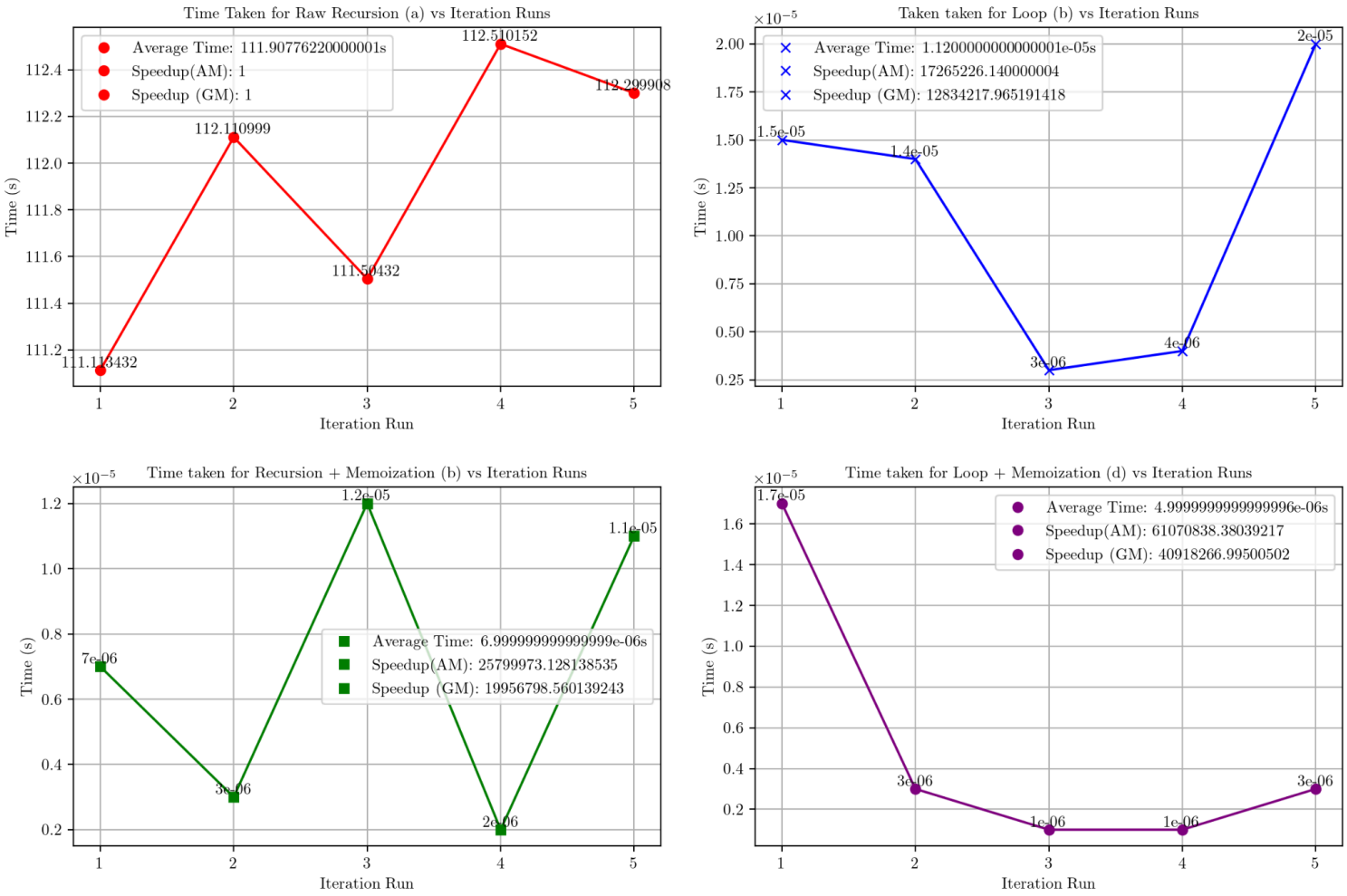
Overall, all the 4 programs were run for 5 iterations, and the speedup was calculated in the following ways:

$$\text{SpeedUp of Program A over B} = \frac{\text{Execution Time B}}{\text{Execution Time A}}$$

One way was to take the average of the speedup values for the 5 runs, and the other was to take the geometric mean of those 5 runs. Below are the plots of the 4 programs run for 5 iteration runs.

From the results, the **speed comparison** of the 4 programs can be stated as:

(a) < (b) < (c) < (d)



Final Results

Average CPU Time taken for Raw Recursion **(a)** : **111.91 s**
Average CPU Time taken for Loop **(b)** : **11.20e-06 s**
Average CPU Time taken for Recursion + Memoization **(c)** : **07.00e-06 s**
Average CPU Time taken for Loop + Memoization **(d)** : **05.00e-06 s**

Speedup (AM) for Loop (b) wrt (a): **17.26×10^6**
Speedup (GM) for Loop (b) wrt (a): **12.83×10^6**

Speedup (AM) for Recursion + Memoization (c) wrt (a): **25.80×10^6**
Speedup (GM) for Recursion + Memoization (c) wrt (a): **19.95×10^6**

Speedup (AM) for Loop + Memoization (d) wrt (a): **61.07×10^6**
Speedup (GM) for Loop + Memoization (d) wrt (a): **40.91×10^6**

Question 2

Program to find the matrix multiplication of two N x N matrices of type:
(i.) INTEGER and (ii.) DOUBLE/FLOATING POINT
With N = [64, 128, 256, 512, 1024]. Further, this program is to be written in
(i.) C++ and (ii.) Python

Now, the question asks us to find the CPU time (User CPU + System CPU).

Before going further, let us understand all these terms in detail.

StackOverflow - What do 'real' 'user' 'sys' time mean?
Ways to measure execution time
TutorialsPoint - Difference b/w User CPU and System CPU time

There are two categories of time measurements for program execution:

1. **Wall Time/Real Time**: This is simply the total time elapsed during the measurement. It's the time you can measure with a stopwatch, assuming that you are able to start and stop it exactly at the execution points you want. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example, if it is waiting for I/O to complete).
2. **CPU Time**: It refers to the time the CPU was busy processing the program's instructions. The time spent waiting for other things to complete (like I/O operations) is not included in the CPU time.

Now, within the **CPU Time**, there are two subcategories of CPU Time:

a. **User CPU Time**: It is the amount of CPU time spent in user-mode code (outside the kernel) *within* the process. This is only the actual CPU time used in executing the process. Other processes and the time the process spends blocked do not count towards this figure. For example, if you have a program written in C running on your system, the time it takes for the program to perform calculations or execute its own logic is considered user CPU time. It does not include the time spent waiting for input/output operations or system calls.
b. **System CPU Time**: It is the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls *within the kernel,* as opposed to library code, which is still running in user space. Kernel mode is a privileged mode where the operating system's core functions and services are executed. System CPU time includes the time spent executing system calls, managing hardware devices, handling interrupts, and performing other kernel–related tasks like memory management, etc.

How to obtain the Real and CPU (User and System) Time values for a program execution?

1. Using the ***"time"*** command in Unix terminal *(works for both C++ and Python)*

   > **time** ./a.out

   > ./a.out 7.38s user 0.02s system 94% cpu 7.795 total

   This tells us:

   - *User CPU Time*: **7.38s** (Not accounting for Memory Management, the time CPU took this much time in the doing the calculation part of the entire program).
   - *System CPU Time*: **0.02s** (Internal Kernel-based tasks that the CPU performed during the entire program execution).
   - *Real-Time/ Wall Time*: **7.795s** (If someone started a stopwatch when I pressed enter to find the time and stopped the watch as soon as the program was done executing, this is the time that the stopwatch would show us).

2. Using ***"clock_gettime(CLOCK_PROCESS_CPUTIME_ID, timespec& tms)"***

```
struct timespec tmStart, tmEnd;
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &tmStart);
/*
CODE HERE
*/
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &tmEnd);

double time_taken = ((tmEnd.tv_sec - tmStart.tv_sec) * 1e9 +
(tmEnd.tv_nsec - tmStart.tv_nsec)) / 1e9;
```

   Using ***"time.process_time()"*** in Python's ***time*** module.

```
start_time = time.process_time()
## Some Code
end_time = time.process_time()

time_taken = end_time - start_time
```

THE ACTUAL QUESTION

(a.) Reporting the **time** in terms of USER CPU Time + SYSTEM CPU Time
(Total Execution Time = User CPU Time + System CPU Time)

BUCKET 1 - C++
INTEGER VALUES

N = 64
CPU Time taken by the Meat Portion function: 0.005658s
./a.out **0.01s user 0.00s system** 5% cpu 0.220 total

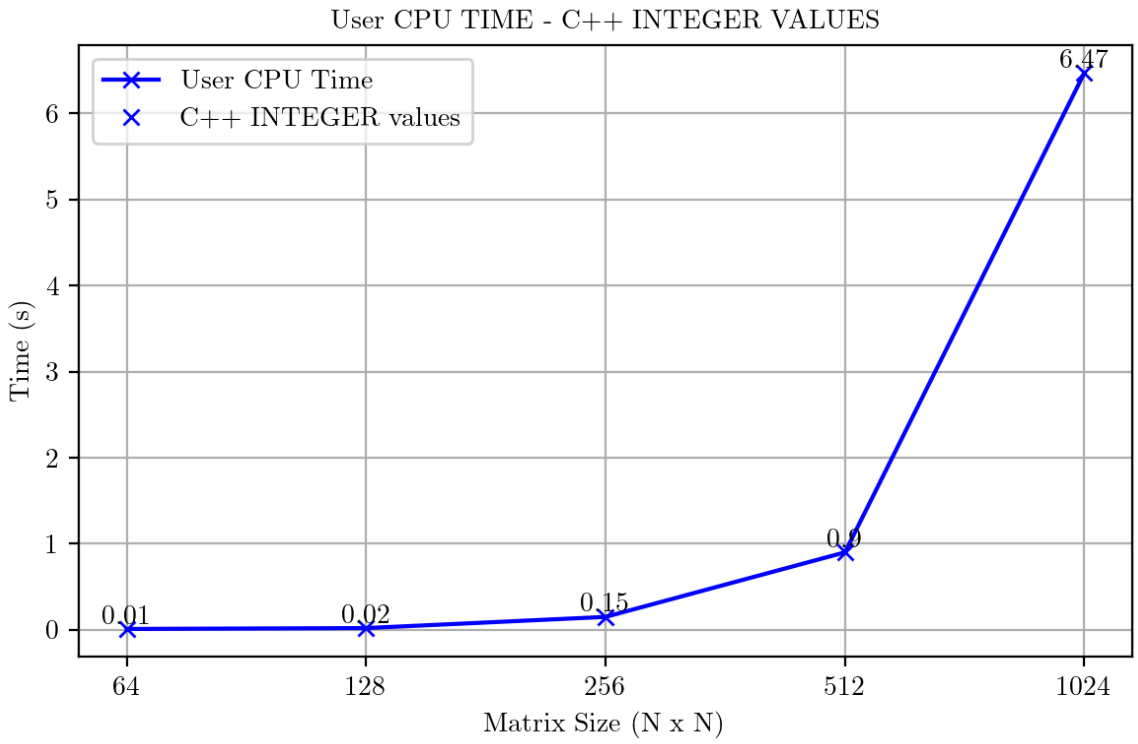
N = 128
CPU Time taken by the Meat Portion function: 0.019661s
./a.out **0.02s user 0.00s system** 12% cpu 0.185 total

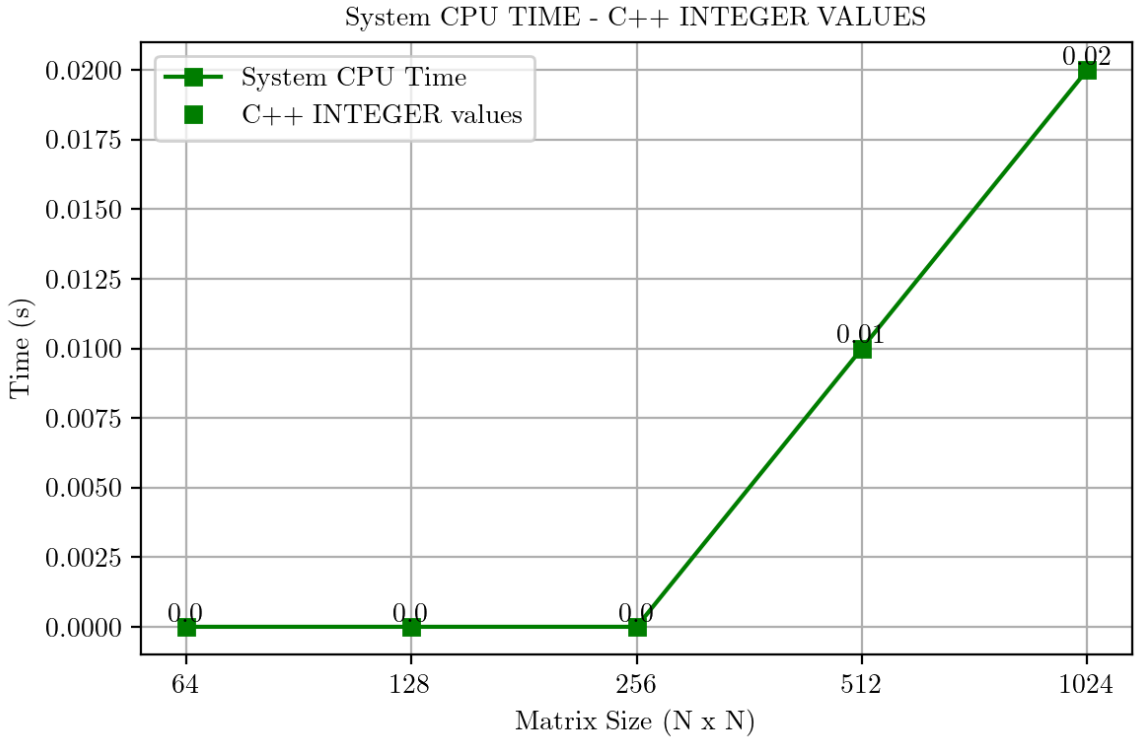
N = 256
CPU Time taken by the Meat Portion function: 0.142294s
./a.out **0.15s user 0.00s system** 48% cpu 0.319 total

N = 512
CPU Time taken by the Meat Portion function: 0.877740s
./a.out **0.90s user 0.01s system** 78% cpu 1.156 total

N = 1024
CPU Time taken by the Meat Portion function: 6.412325s
./a.out **6.47s user 0.02s system** 97% cpu 6.658 total

User CPU Time = **[0.01s, 0.02s, 0.15s, 0.9s, 6.47s]**
System CPU Time = **[0.0s, 0.0s, 0.0s, 0.01s, 0.02s]**


User CPU TIME - C++ INTEGER VALUES
User CPU Time
C++ INTEGER values
Time (s)
Matrix Size (N x N)
0.01
0.02
0.15
0.9
6.47
0
1
2
3
4
5
6
64
128
256
512
1024


System CPU TIME - C++ INTEGER VALUES
System CPU Time
C++ INTEGER values
Time (s)
Matrix Size (N x N)
0.0
0.0
0.0
0.01
0.02
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
64
128
256
512
1024

FLOATING VALUES

N = 64
CPU Time taken by the Meat Portion function: 0.005390s
./a.out **0.01s user 0.00s system** 2% cpu 0.521 total

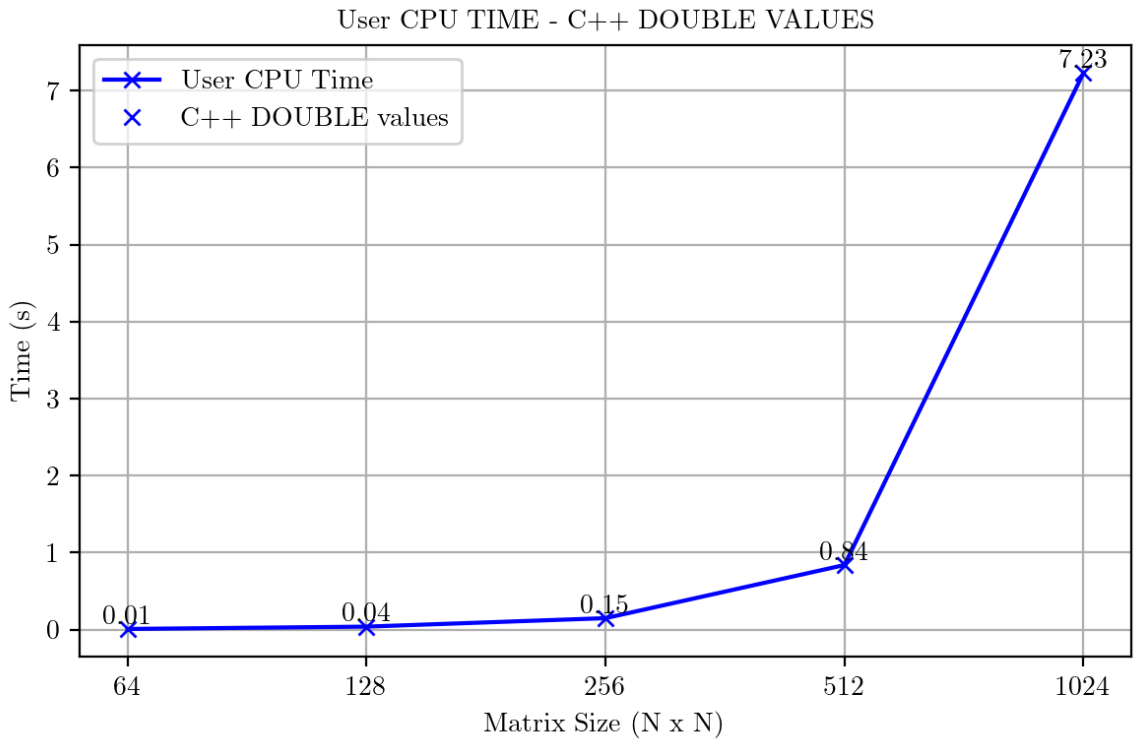
N = 128
CPU Time taken by the Meat Portion function: 0.033538s
./a.out **0.04s user 0.00s system** 19% cpu 0.209 total

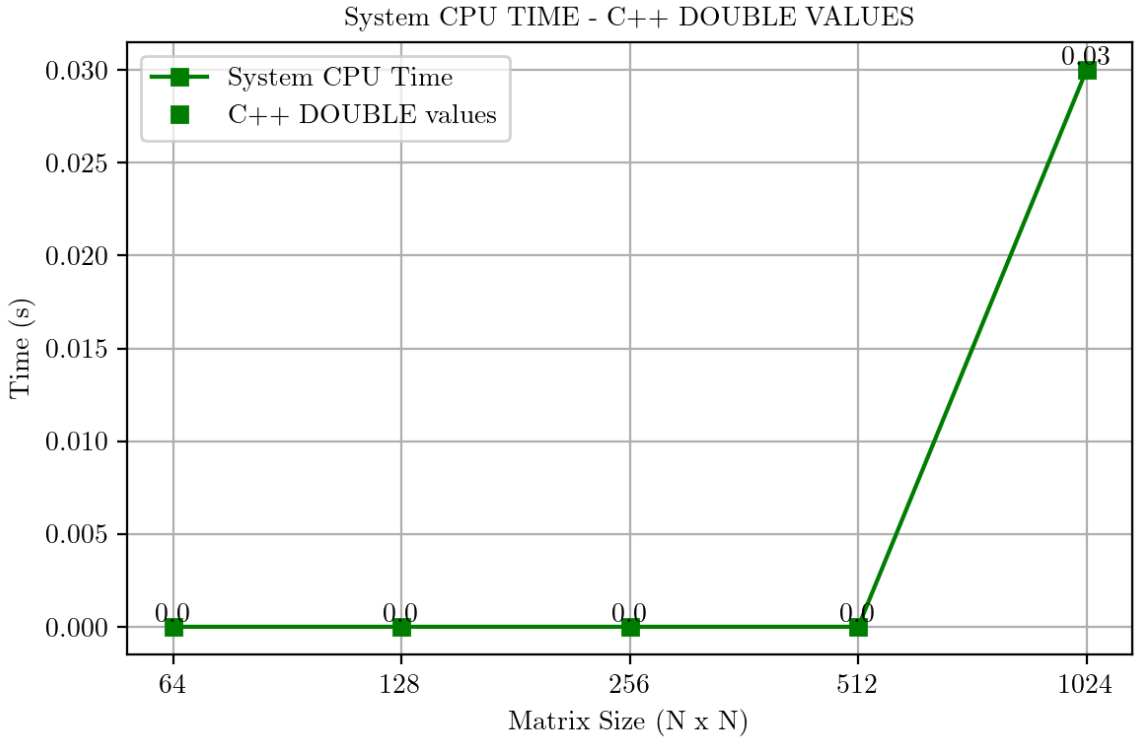
N = 256
CPU Time taken by the Meat Portion function: 0.140550s
./a.out **0.15s user 0.00s system** 47% cpu 0.319 total

N = 512
CPU Time taken by the Meat Portion function: 0.828482s
./a.out **0.84s user 0.00s system** 83% cpu 1.016 total

N = 1024
CPU Time taken by the Meat Portion function: 7.169196s
./a.out **7.23s user 0.03s system** 97% cpu 7.427 total

User CPU Time = **[0.01s, 0.04s, 0.15s, 0.84s, 7.23s]**
System CPU Time = **[0.0s, 0.0s, 0.0s, 0.0s, 0.03s]**


User CPU TIME - C++ DOUBLE VALUES
User CPU Time
C++ DOUBLE values
Time (s)
Matrix Size (N x N)
0.01
0.04
0.15
0.84
7.23
64
128
256
512
1024


System CPU TIME - C++ DOUBLE VALUES
System CPU Time
C++ DOUBLE values
Time (s)
Matrix Size (N x N)
0.0
0.0
0.0
0.0
0.03
64
128
256
512
1024

BUCKET 2 - PYTHON
FLOATING POINT VALUES

N = 64
CPU Time taken by the Meat Portion function: 0.029775000000000003s
python3 Matrix.py **0.06s user 0.01s system** 86% cpu 0.081 total

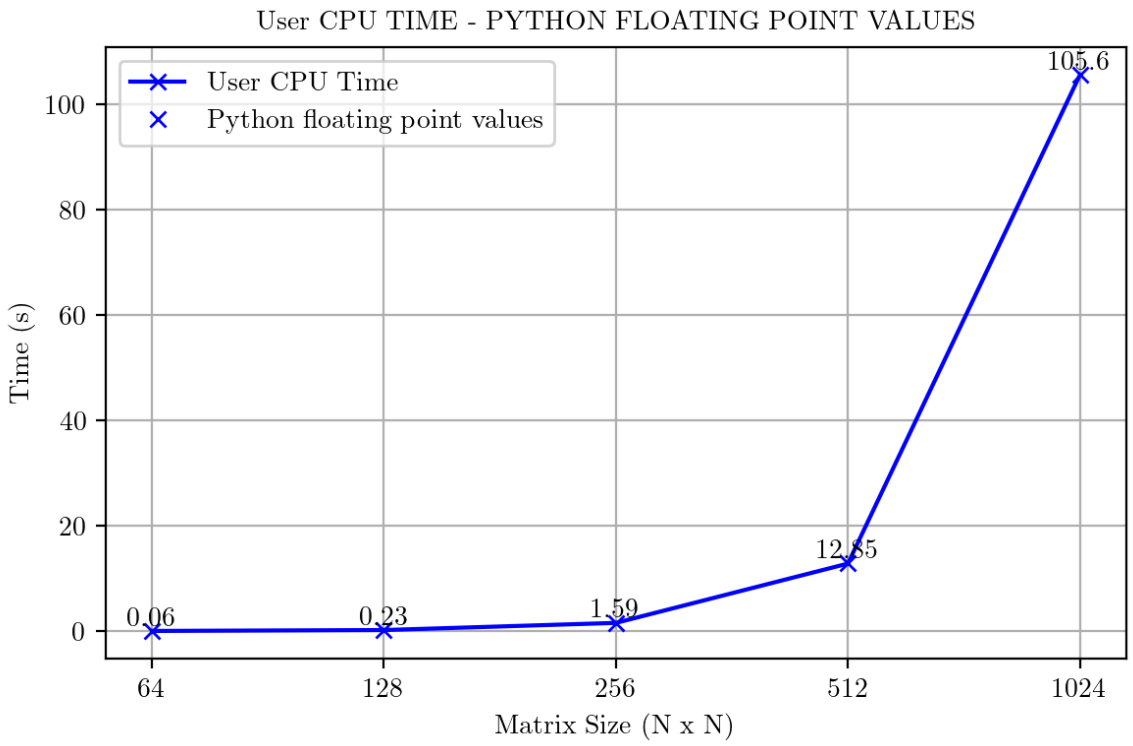
N = 128
CPU Time taken by the Meat Portion function: 0.196865s
python3 Matrix.py **0.23s user 0.01s system** 91% cpu 0.263 total

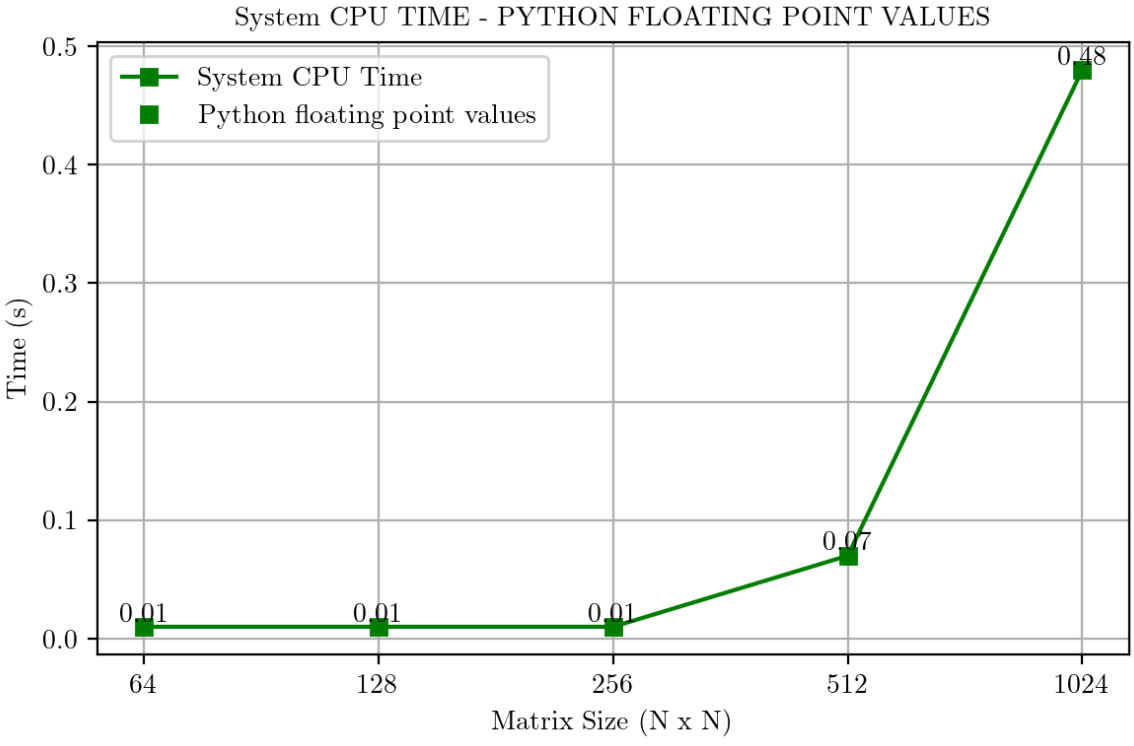
N = 256
CPU Time taken by the Meat Portion function: 1.544774s
python3 Matrix.py **1.59s user 0.01s system** 99% cpu 1.619 total

N = 512
CPU Time taken by the Meat Portion function: 12.780401000000001s
python3 Matrix.py **12.85s user 0.07s system** 99% cpu 12.949 total

N = 1024
CPU Time taken by the Meat Portion function: 105.682336s
python3 Matrix.py **105.60s user 0.48s system** 98% cpu 1:47.60 total

User CPU Time = **[0.06s, 0.23s, 1.59s, 12.85s, 105.6s]**
System CPU Time = **[0.01s, 0.01s, 0.01s, 0.07s, 0.48s]**


User CPU TIME - PYTHON FLOATING POINT VALUES
User CPU Time
Python floating point values
Time (s)
Matrix Size (N x N)
0.06
0.23
1.59
12.85
105.6
0
20
40
60
80
100
64
128
256
512
1024


System CPU TIME - PYTHON FLOATING POINT VALUES
System CPU Time
Python floating point values
Time (s)
Matrix Size (N x N)
0.01
0.01
0.01
0.07
0.48
0.0
0.1
0.2
0.3
0.4
0.5
64
128
256
512
1024

INTEGER VALUES

N = 64
CPU Time taken by the Meat Portion function: 0.033104s
python3 Matrix.py **0.07s user 0.01s system** 97% cpu 0.080 total

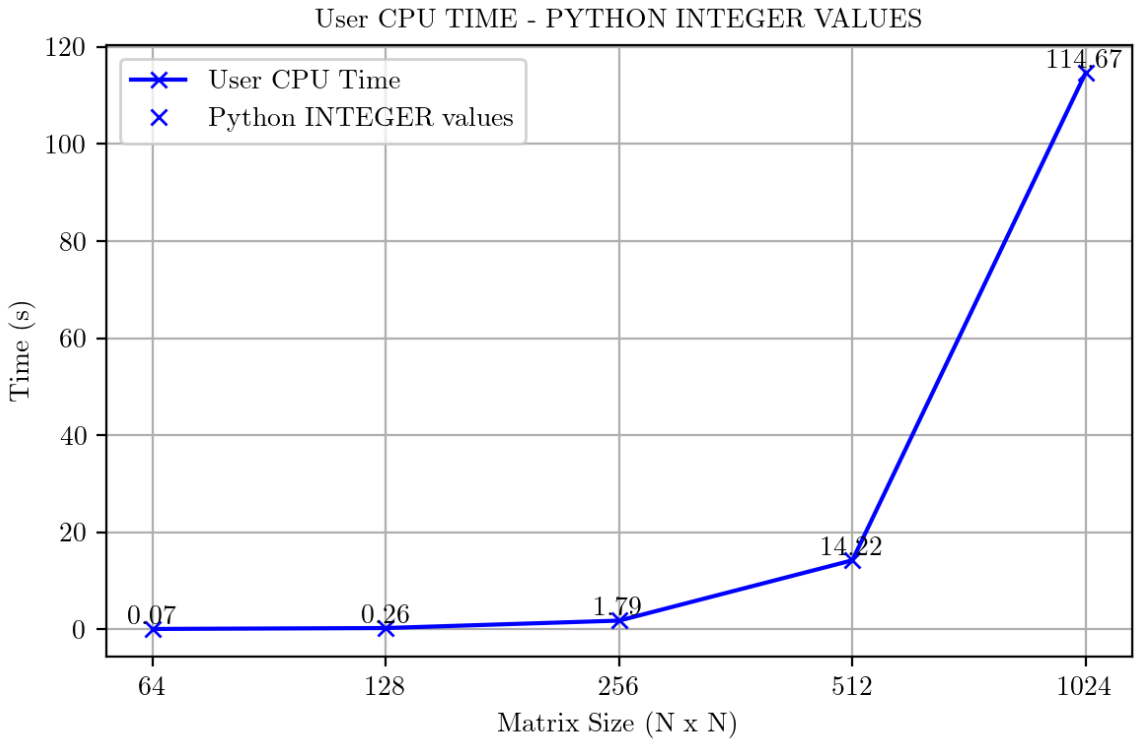
N = 128
CPU Time taken by the Meat Portion function: 0.212662s
python3 Matrix.py **0.26s user 0.01s system** 98% cpu 0.277 total

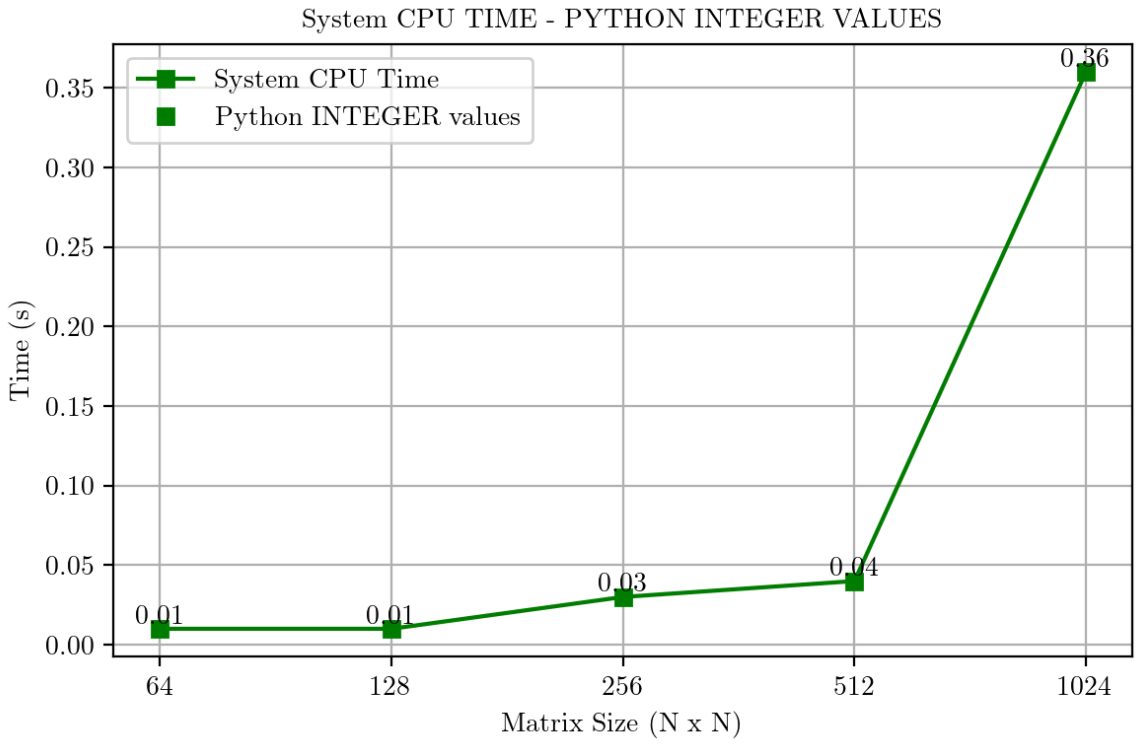
N = 256
CPU Time taken by the Meat Portion function: 1.708992s
python3 Matrix.py **1.79s user 0.03s system** 97% cpu 1.854 total

N = 512
CPU Time taken by the Meat Portion function: 13.956954s
python3 Matrix.py **14.22s user 0.04s system** 99% cpu 14.262 total

N = 1024
CPU Time taken by the Meat Portion function: 113.959915s
python3 Matrix.py **114.67s user 0.36s system** 99% cpu 1:55.41 total

User CPU Time = **[0.07s, 0.26s, 1.79s, 14.22s, 114.67s]**
System CPU Time = **[0.01s, 0.01s, 0.03s, 0.04s, 0.36s]**


User CPU TIME - PYTHON INTEGER VALUES
User CPU Time
Python INTEGER values
Time (s)
Matrix Size (N x N)
0.07
0.26
1.79
14.22
114.67
64
128
256
512
1024


System CPU TIME - PYTHON INTEGER VALUES
System CPU Time
Python INTEGER values
Time (s)
Matrix Size (N x N)
0.01
0.01
0.03
0.04
0.36
64
128
256
512
1024

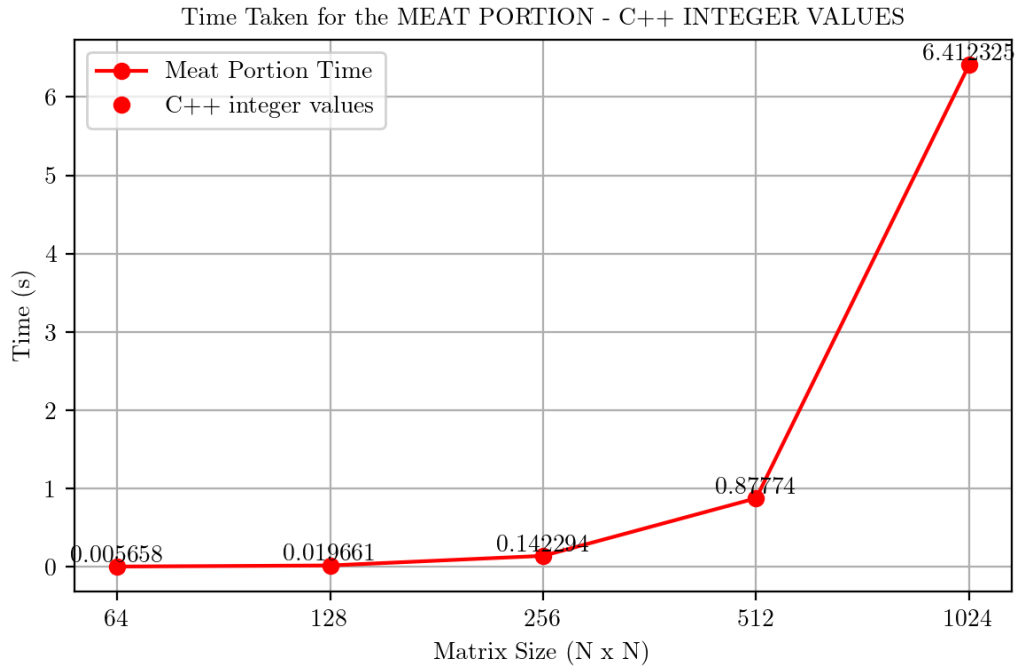
(b.) Time of Meat Portion of the program and the total execution time of the program and their proportion with increasing value of N

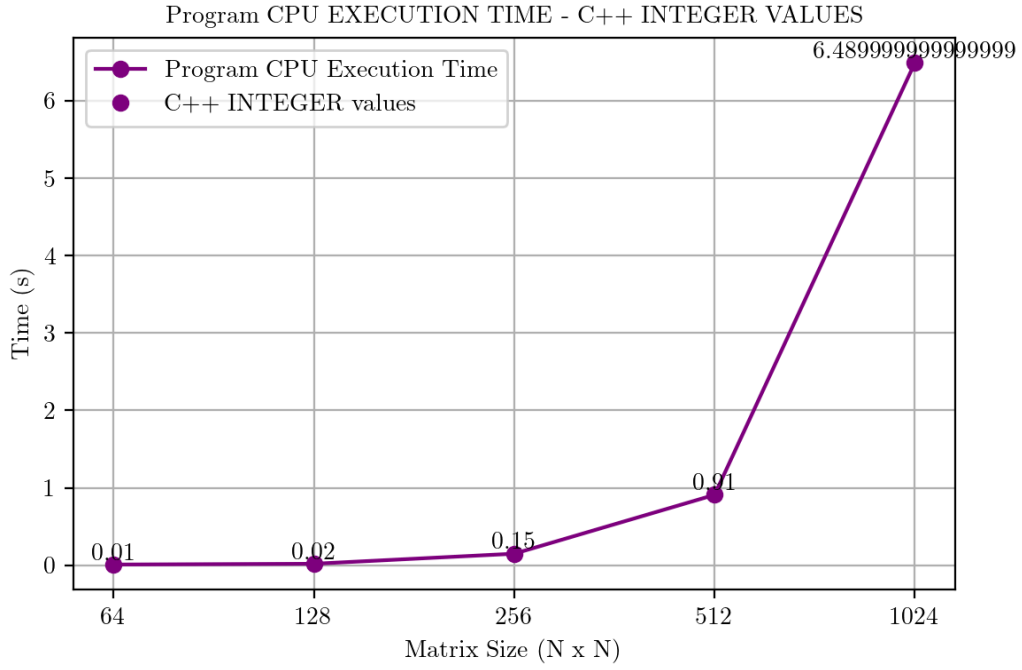
BUCKET 1 - C++

INTEGER VALUES

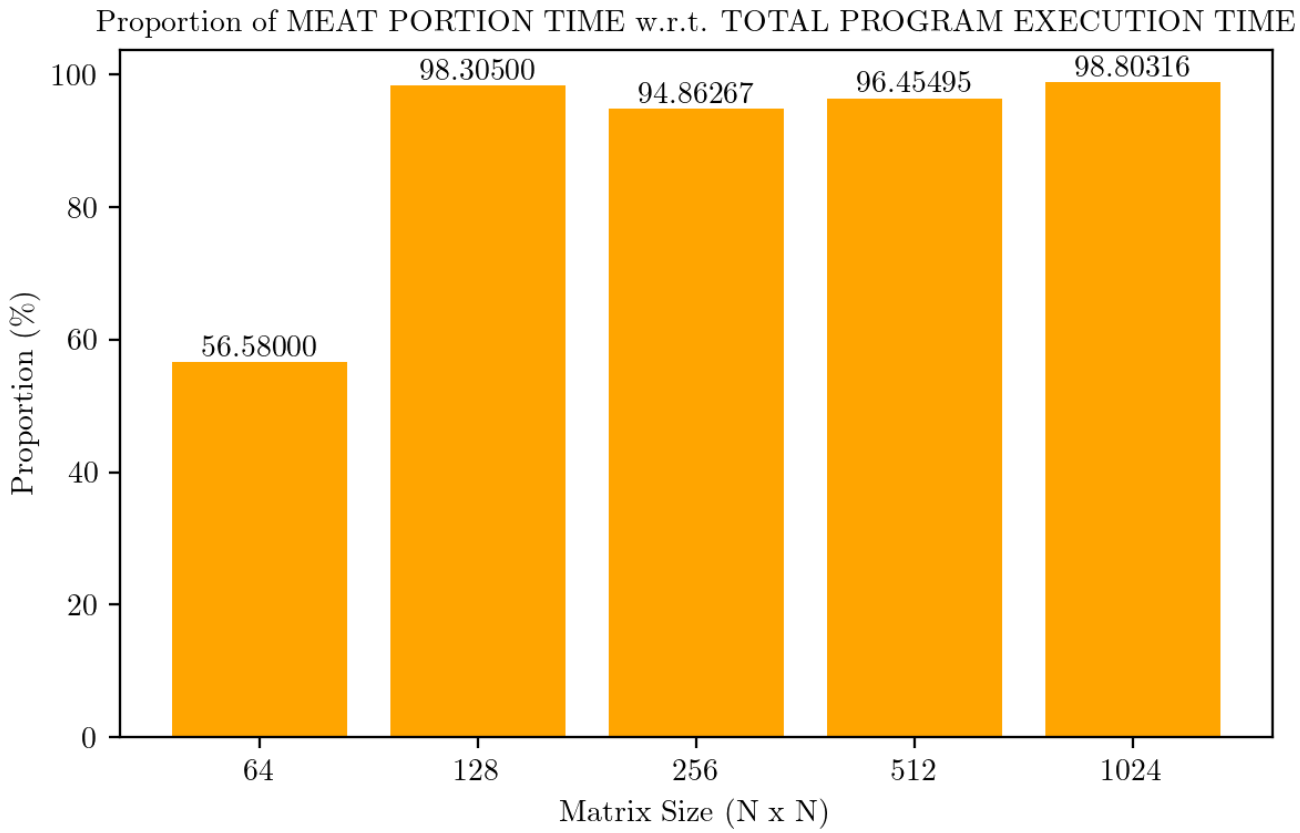
Meat Portion = **[0.005658s, 0.019661s, 0.142294s, 0.87774s, 6.412325s]**

Total Execution Time = **[0.01s, 0.002s, 0.15s, 0.91s, 6.49s]**





Proportion of Time Taken by Meat Portion w.r.t Total Program Execution Time

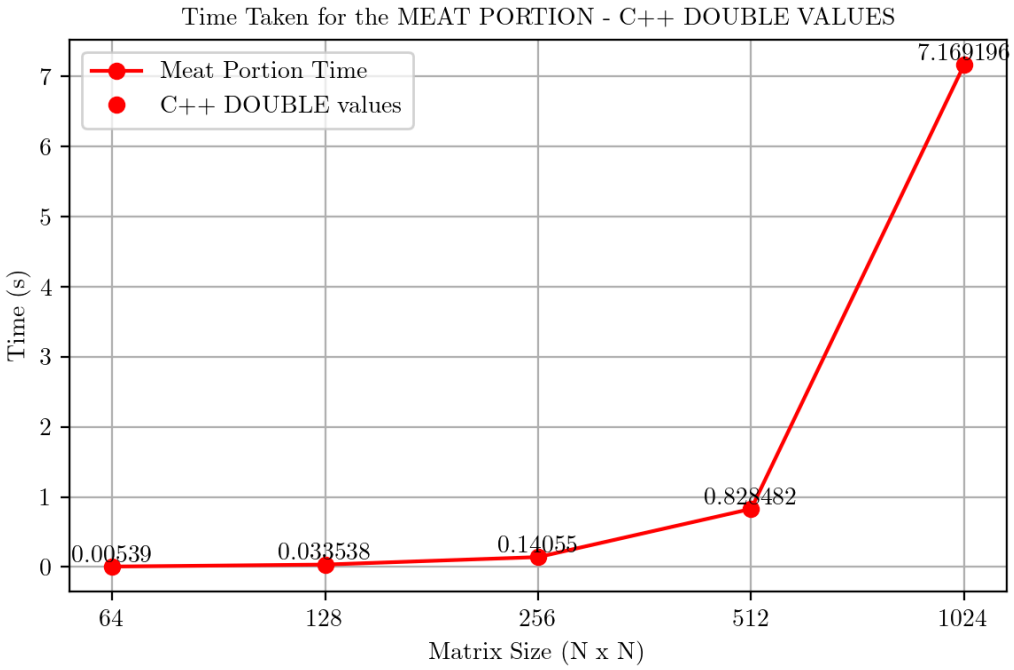


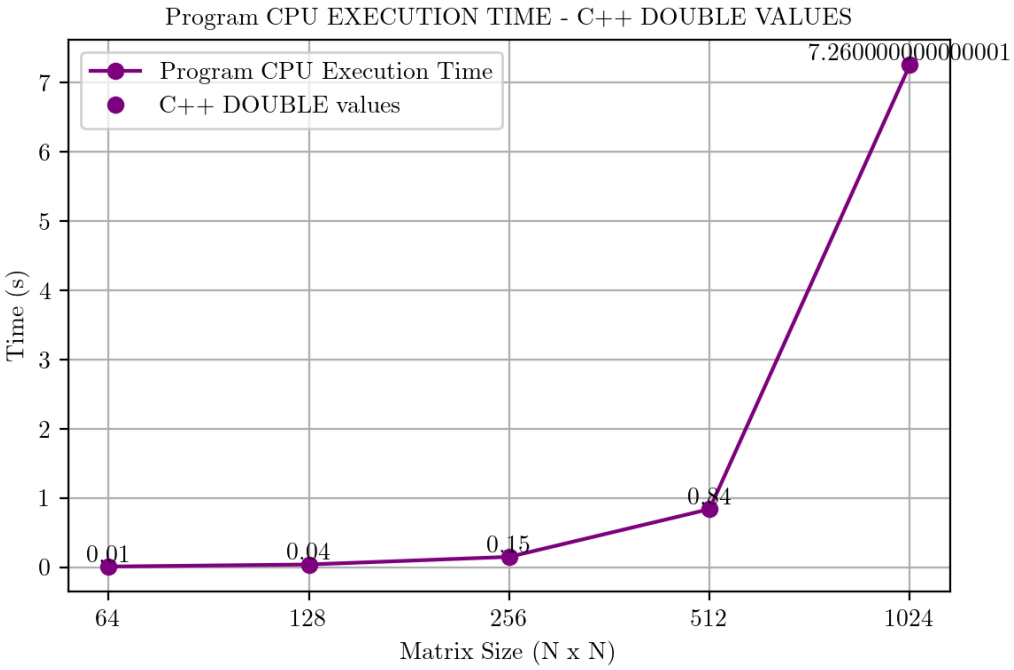
	64	128	256	512	1024
Meat Portion Time	0.005658	0.019661	0.142294	0.87774	6.412325
User Time	0.010000	0.020000	0.150000	0.90000	6.470000
System Time	0.000000	0.000000	0.000000	0.01000	0.020000
CPU Time	0.010000	0.020000	0.150000	0.91000	6.490000

DOUBLE VALUES

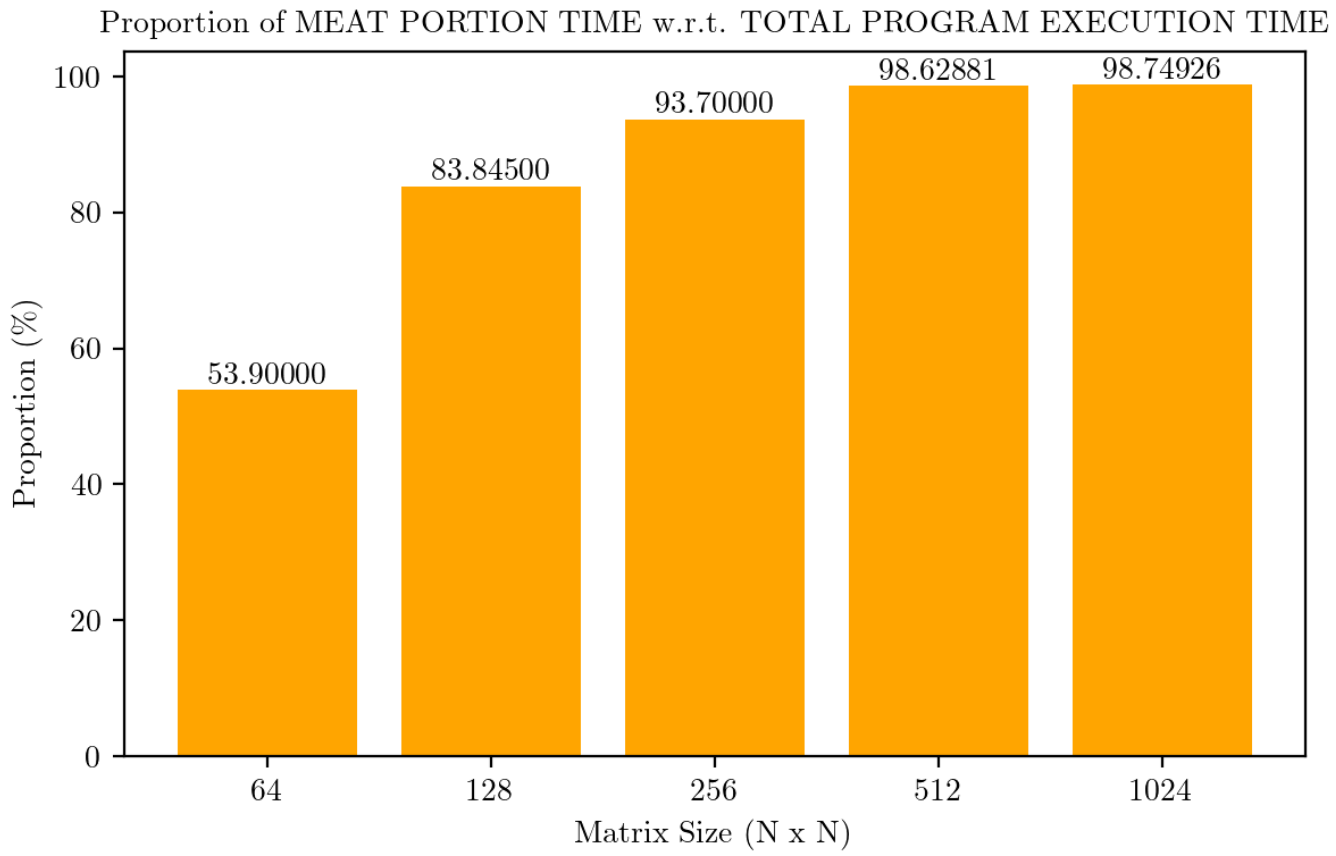
Meat Portion = **[0.00539s, 0.033538s, 0.14055s, 0.828482s, 7.169196s]**

Total Execution Time = **[0.01s, 0.04s, 0.15s, 0.84s, 7.26s]**





Proportion of Time Taken by Meat Portion w.r.t Total Program Execution Time



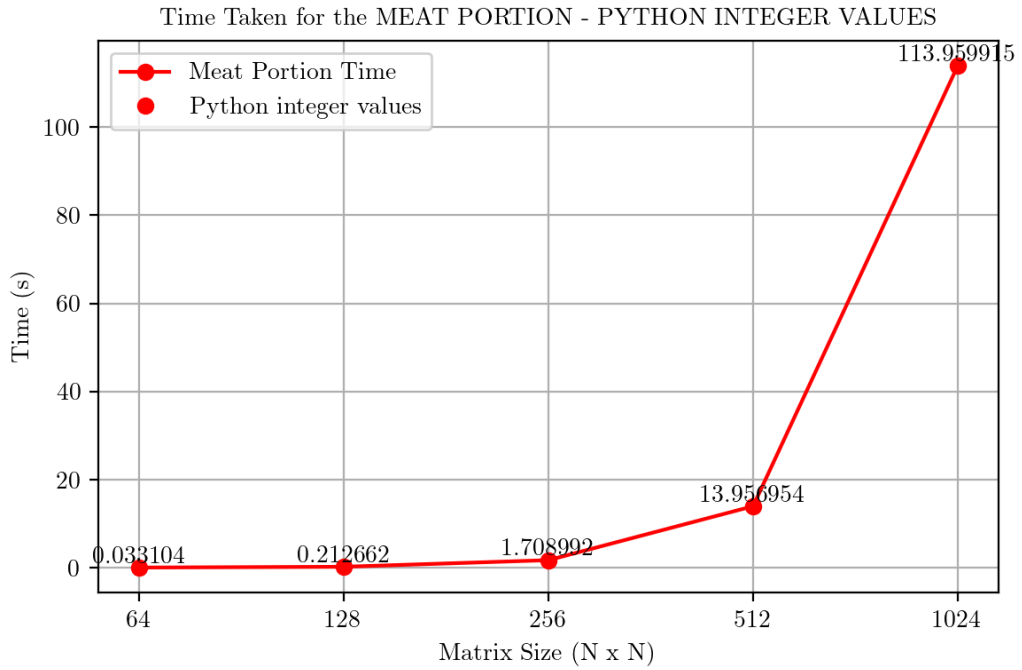
	64	128	256	512	1024
Meat Portion Time	0.00539	0.033538	0.14055	0.828482	7.169196
User Time	0.01000	0.040000	0.15000	0.840000	7.230000
System Time	0.00000	0.000000	0.00000	0.000000	0.030000
CPU Time	0.01000	0.040000	0.15000	0.840000	7.260000

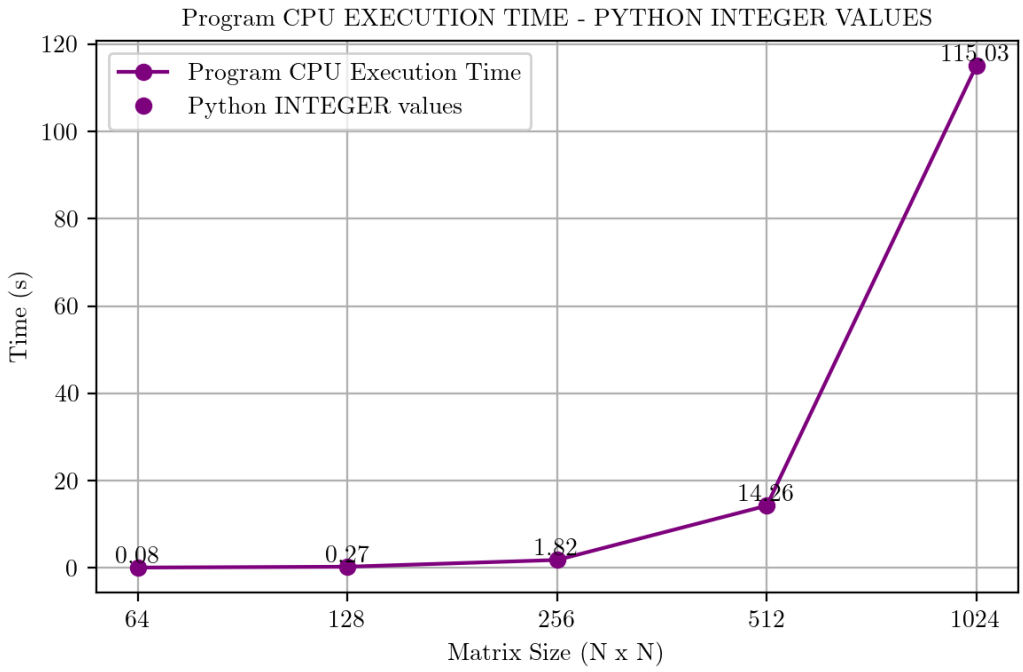
BUCKET 2 - PYTHON

INTEGER VALUES

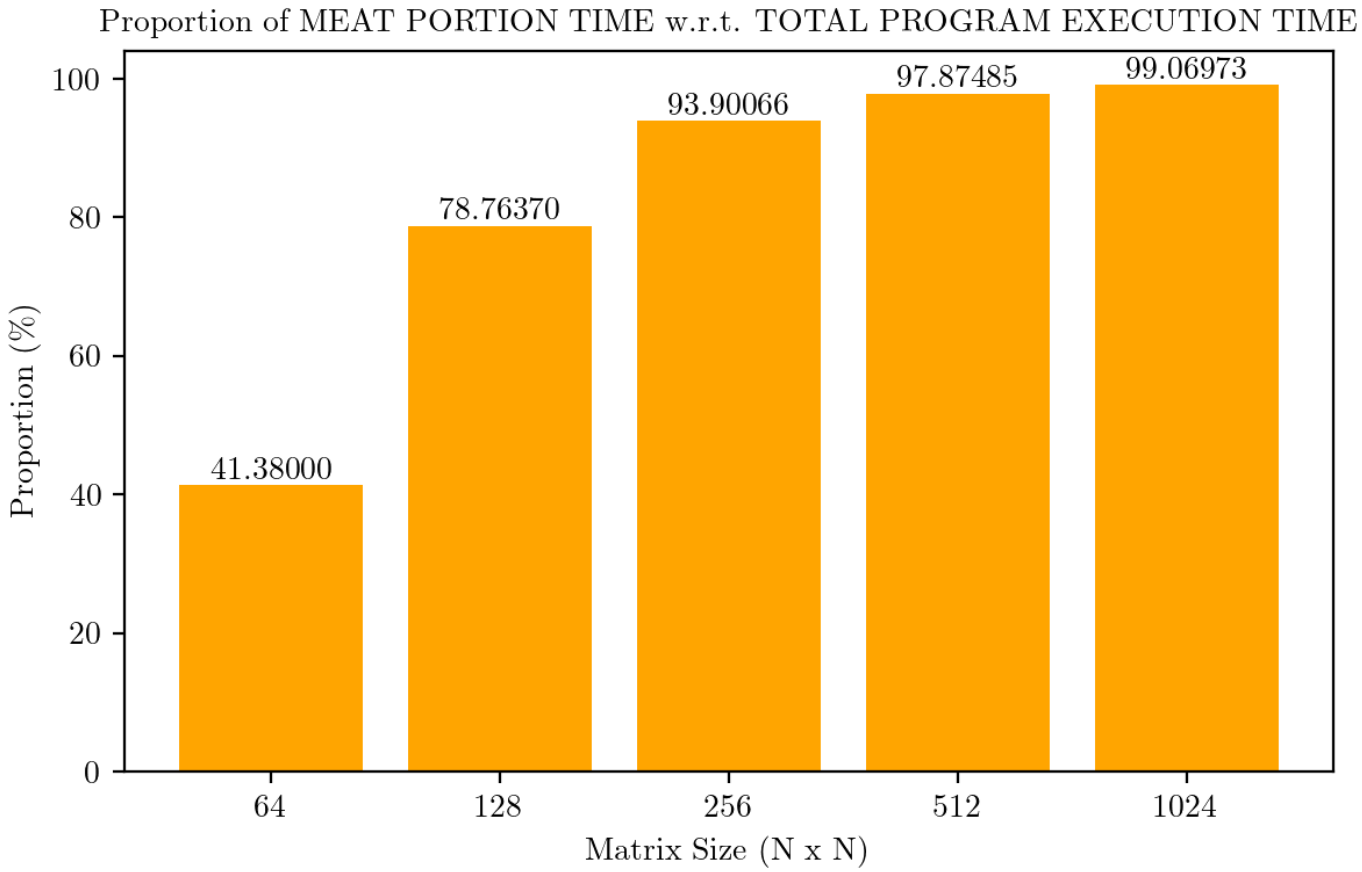
Meat Portion = **[0.033104s, 0.212662s, 1.71s, 13.957s, 113.96s]**

Total Execution Time = **[0.08s, 0.270s, 1.82s, 14.26s, 115.03s]**





Proportion of Time Taken by Meat Portion w.r.t Total Program Execution Time

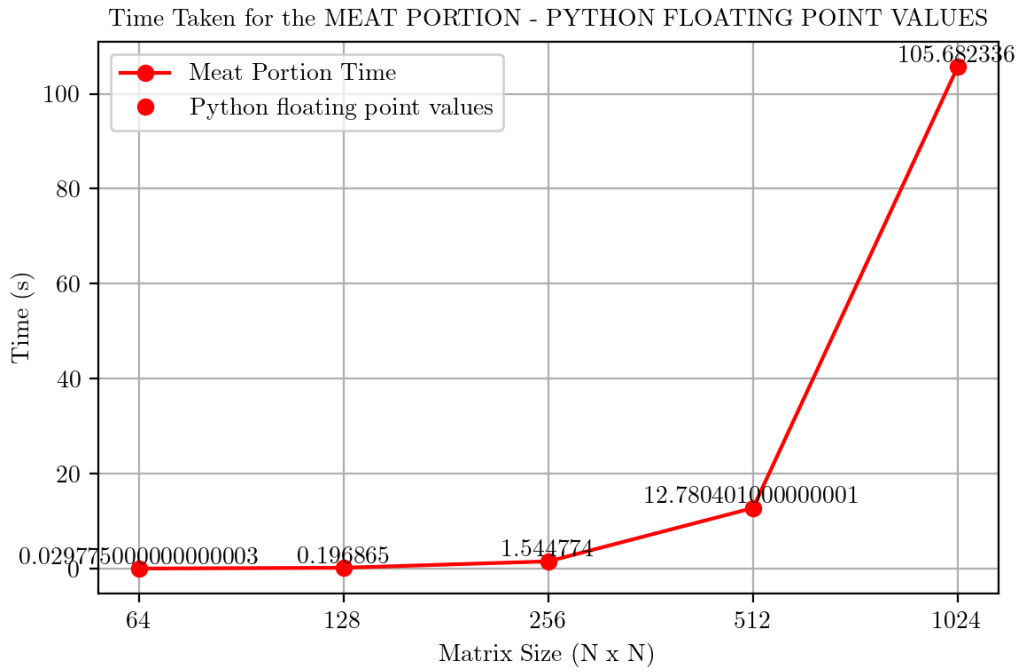


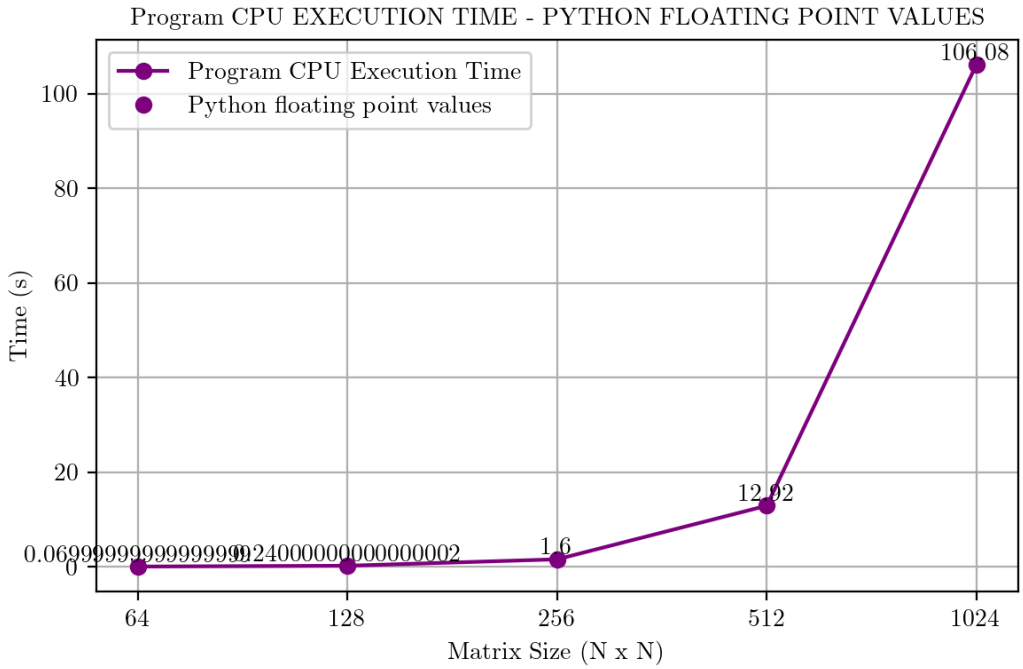
	64	128	256	512	1024
Meat Portion Time	0.033104	0.212662	1.708992	13.956954	113.959915
User Time	0.070000	0.260000	1.790000	14.220000	114.670000
System Time	0.010000	0.010000	0.030000	0.040000	0.360000
CPU Time	0.080000	0.270000	1.820000	14.260000	115.030000

FLOATING POINT VALUES

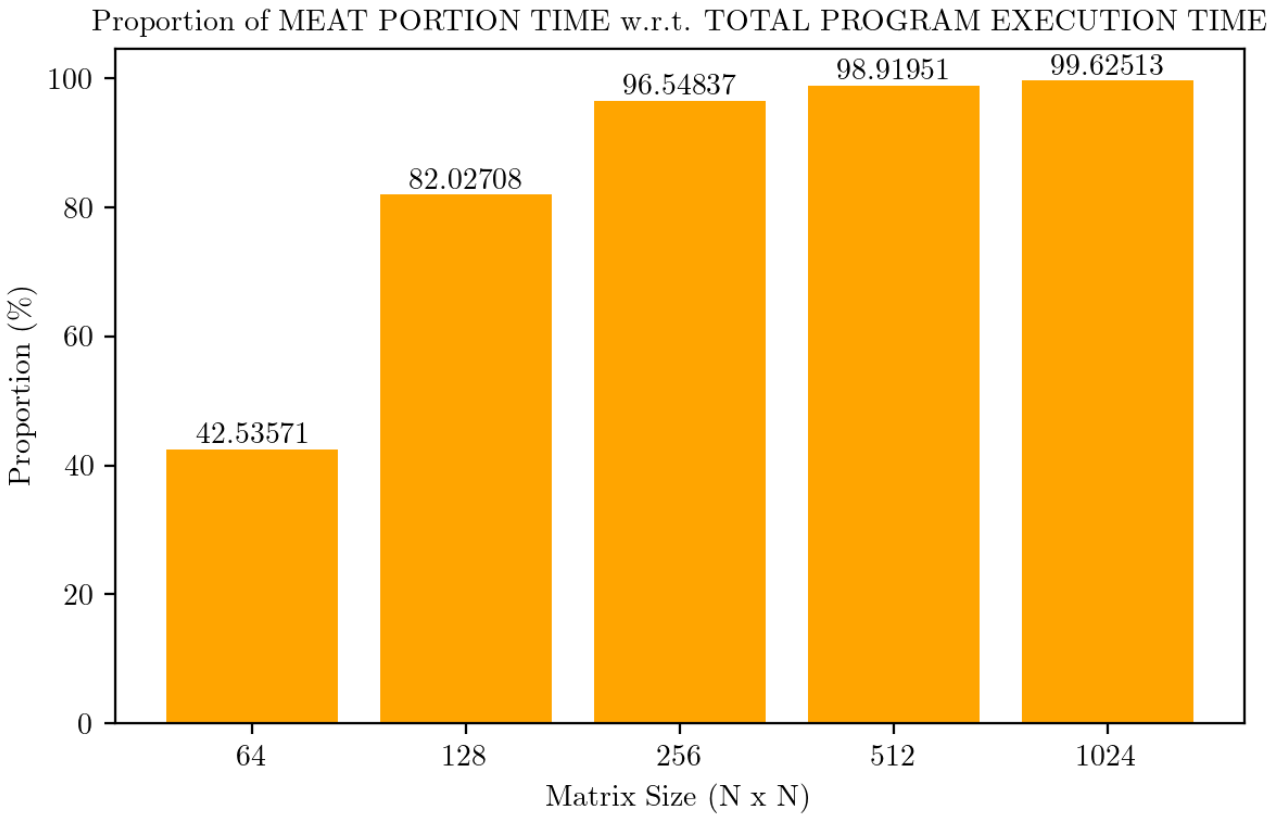
Meat Portion = **[0.03s, 0.197s, 1.544774s, 12.78s, 105.682336s]**

Total Execution Time = **[0.07s, 0.240s, 1.6s, 12.92s, 106.08s]**





Proportion of Time Taken by Meat Portion w.r.t Total Program Execution Time

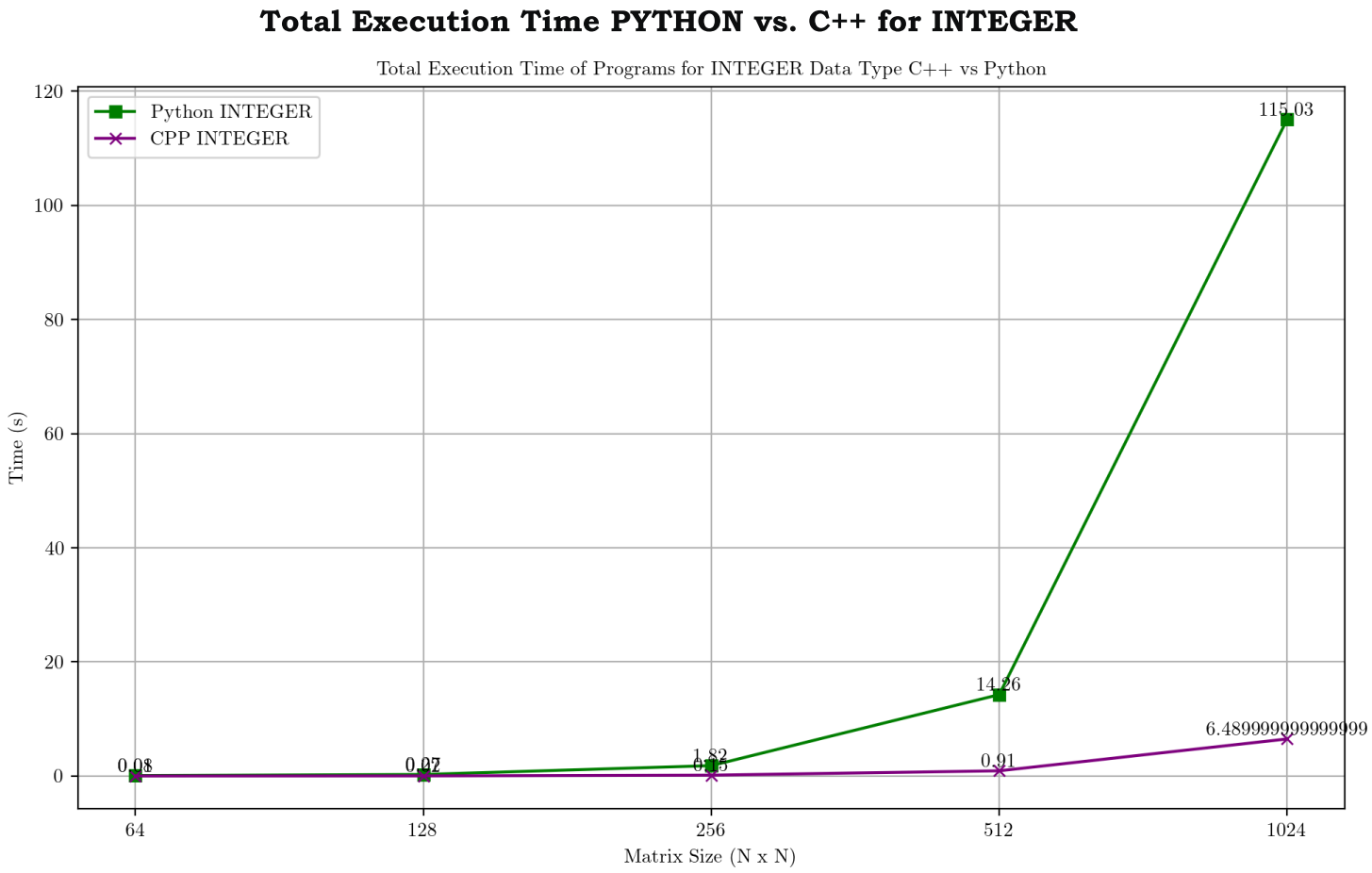


	64	128	256	512	1024
Meat Portion Time	0.029775	0.196865	1.544774	12.780401	105.682336
User Time	0.060000	0.230000	1.590000	12.850000	105.600000
System Time	0.010000	0.010000	0.010000	0.070000	0.480000
CPU Time	0.070000	0.240000	1.600000	12.920000	106.080000

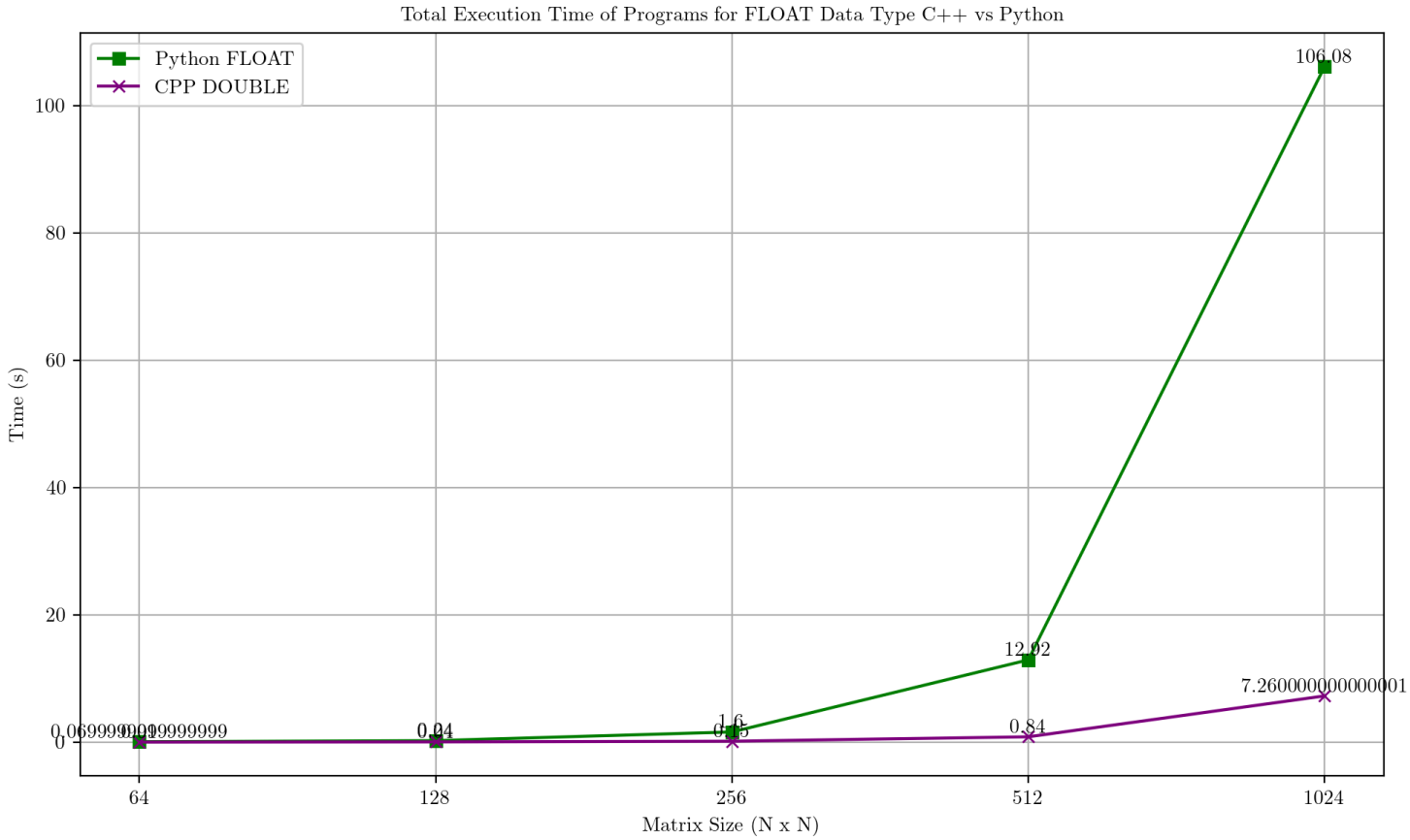
REASON FOR PROPORTION OF TIME INCREASING FOR MEAT PORTION WRT TOTAL EXECUTION TIME

Since the value of N increases, the major part of the CPU time is utilized in the heavy computation of the matrix multiplication of N x N matrices, and the remaining processes, like printing and initialization, all now consume negligible time as compared to the huge multiplication that it has to now do.

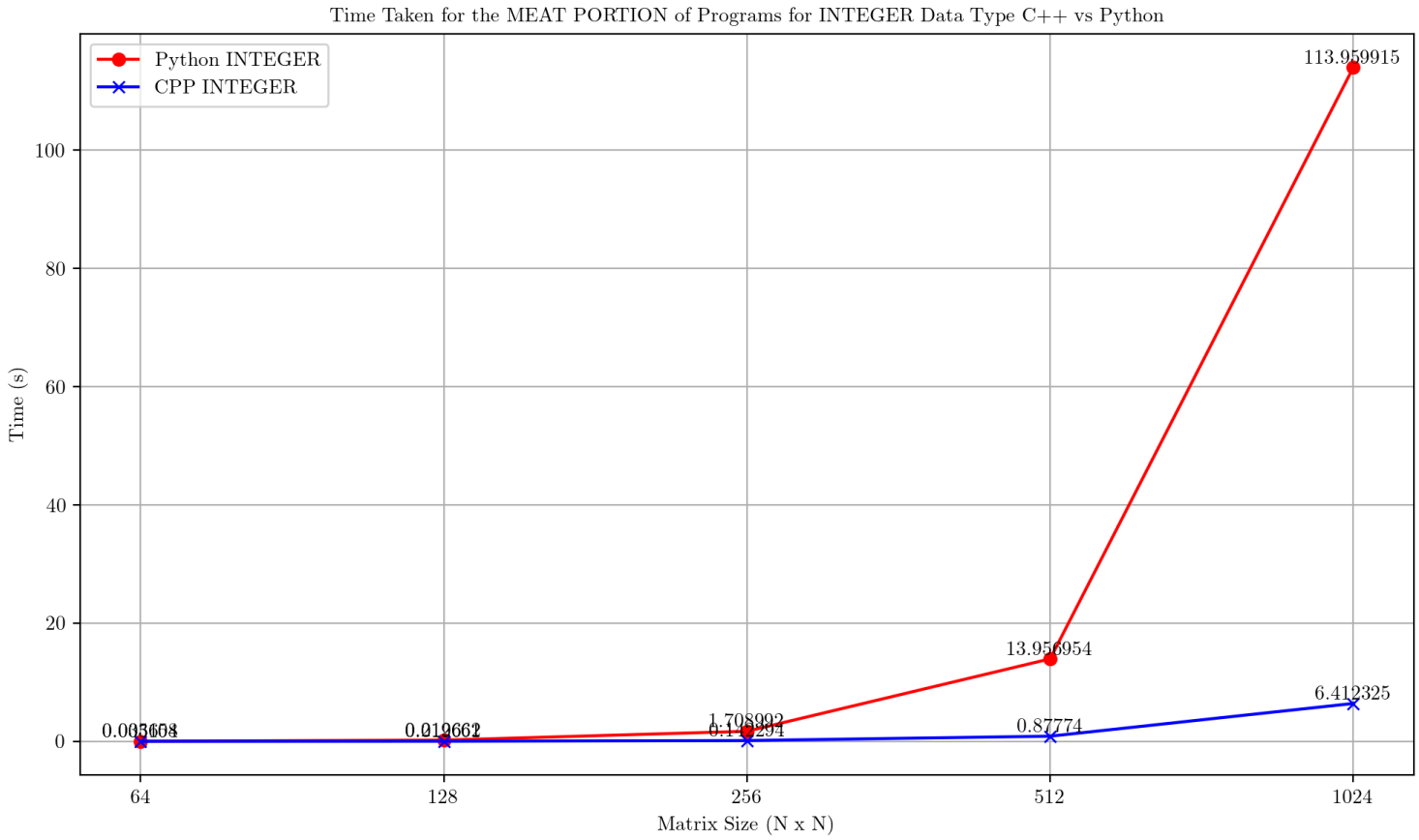
(c.) Comparison of Total Program Execution Times for Bucket 1 and Bucket 2 for cases INTEGER (a) and FLOAT (b) values for various N values



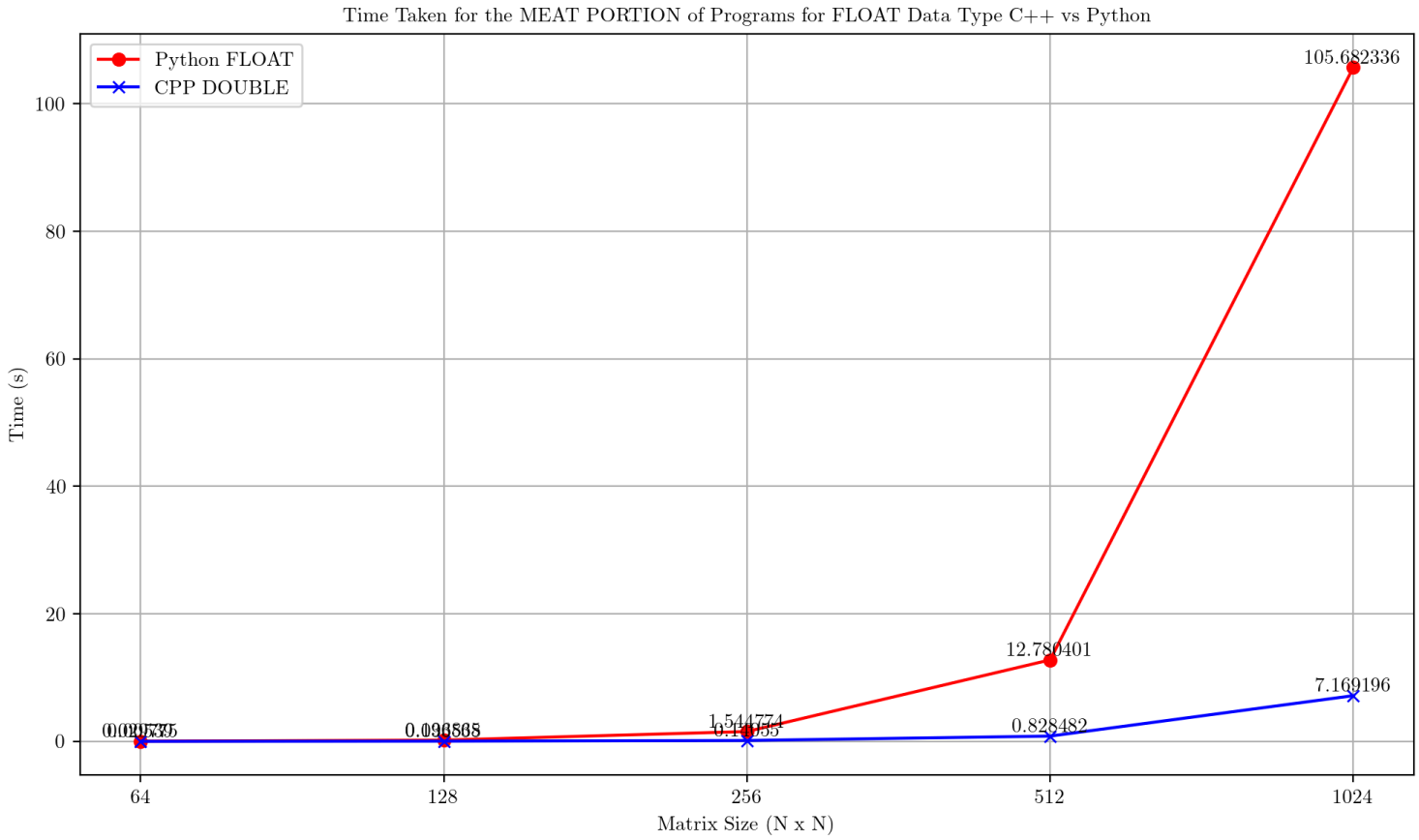
Total Execution Time PYTHON vs. C++ for DOUBLE



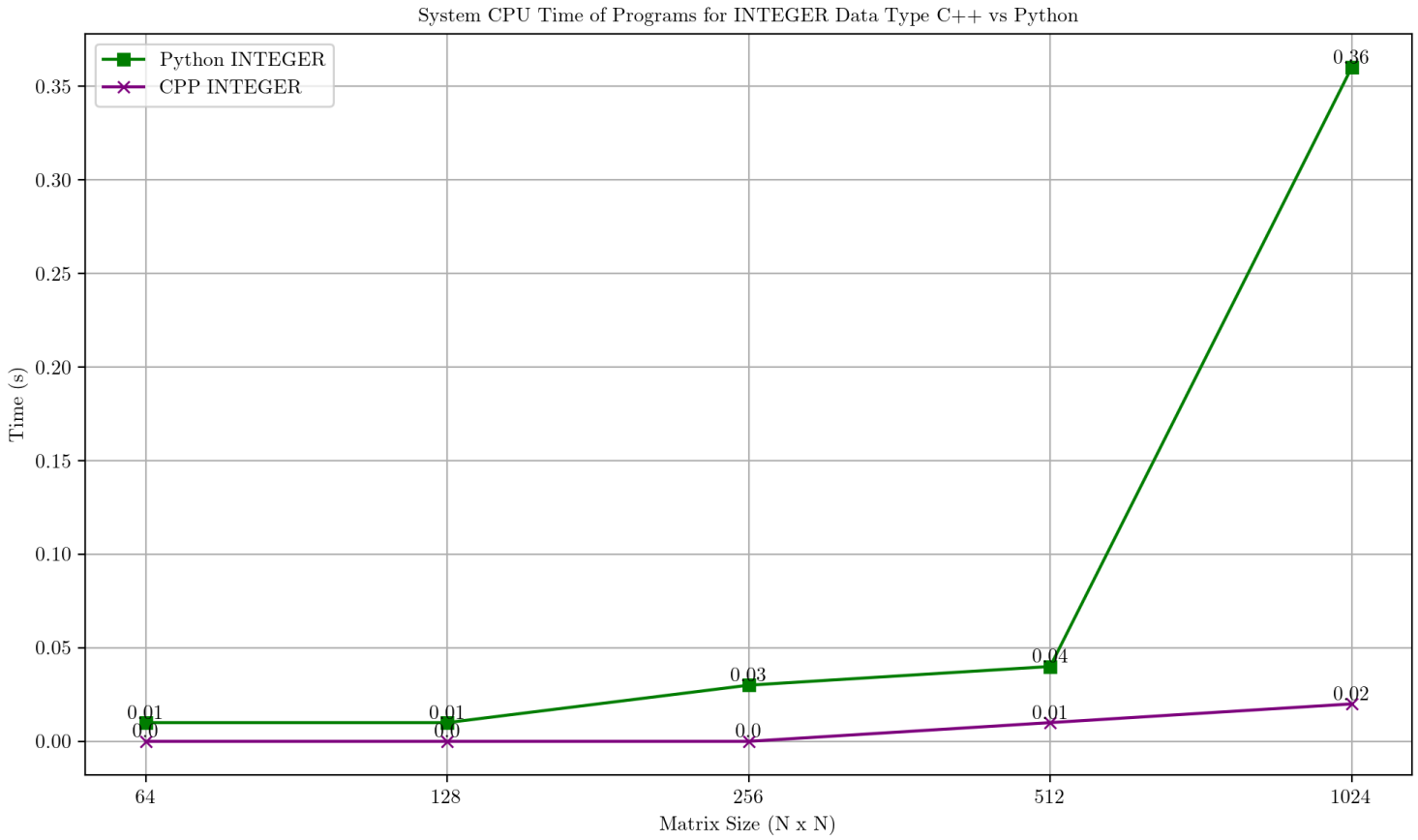
Meat Portion Time PYTHON vs. C++ for INTEGER



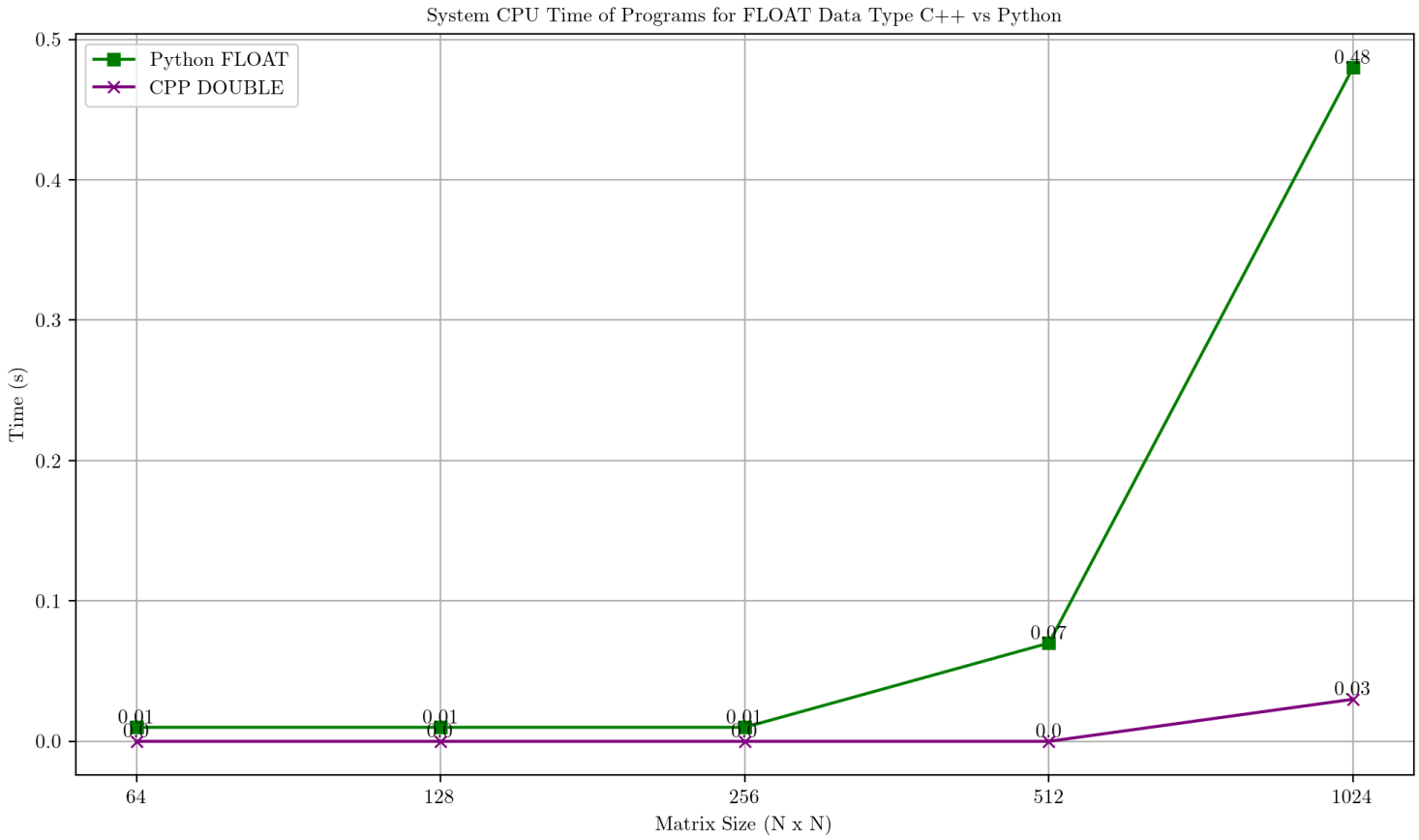
Meat Portion Time PYTHON vs. C++ for FLOATING



System CPU Time PYTHON vs. C++ for INTEGER



System CPU Time PYTHON vs. C++ for FLOATING



OBSERVATIONS

As the value of **N increases**, the TOTAL EXECUTION CPU TIME for both buckets C++ and Python **increases.** This is evident from the fact that a greater number of clock cycles are utilized since the input size increases $O(N^3)$. Now, across Bucket 1 (C++) and Bucket 2 (Python), the TOTAL EXECUTION CPU TIME of Python is greater than that of C++. And even the SYSTEM CPU TIME is greater for Python than C++.

C++ BEATS PYTHON IN ALL PERFORMANCE ASPECTS