

ASSIGNMENT 1
ELP-718 TELECOM SOFT. LAB

SARATH MOHAN
2021JTM2226

A Report Presented for the lab assignment on Comparison of Matrix multiplication using floating-point and fixed-point representation.



Bharti School Of
Telecommunication Technology and Management
IIT Delhi India
Aug 22, 2021

Contents

1. PROBLEM STATEMENT	2
1.1 OBJECTIVE	2
1.2 Following task has been performed	2
2. FIXED POINT AND FLOATING POINT	3
2.1 FIXED POINT REPRESENTATION.....	3
2.2 FLOATING POINT REPRESENTATION:.....	4
3. FLOW CHART OF MAIN FUNCTION.....	5
4. SCREENSHOTS.....	6
4.1 Terminal window while executing make command.....	6
4.2 Generated Data.....	7
4.3 GNU Plot.....	8
5. APPENDIX	9
5.1 C CODE	9
5.1.1 Main Function.....	9
5.1.2 DoubleToFixed.....	11
5.1.3 FixedToDouble.....	11
5.1.4 AddFixed.....	11
5.1.5 MulFixed.....	11
5.1.6 CalcTime	11
5.2 Makefile.....	12
5.3 GNU Plot instructions	12
6. COMPARISION	13
6.1 MATLAB Code	13
6.2 Comparison of Data from MATLAB and C.....	14
6.3 MATLAB Plot vs GNU Plot.....	15
7. CONCLUSION	16
8. REFERENCES	17

1. PROBLEM STATEMENT

1.1 OBJECTIVE

This introductory assignment is to learn how to use tools to improve our productivity. To make the assignment meaningful, we will look at the use of fixed-point arithmetic in the context of matrix multiplication. We will first document how performance scales with matrix size. Subsequently, we will implement Matrix Multiplication in fixed point to (potentially) improve the performance.

1.2 Following task has been performed

- Made a project structure with following folders
 - Include
 - Lib
 - Src
 - Obj
- Constructed a 'make' file
- Implemented fixed point and floating-point matrix multiplication
 - Varied the size of matrix in each iteration
 - In each iteration matrices were initialized with random floating point numbers.
 - Time taken for each matrix multiplication was calculated for floating point and fixed point implementation.
- Following values have been written into a file for each iteration
 - Log of Size of the matrix
 - Time taken for Floating point Multiplication
 - Time taken for Fixed point Multiplication
- A graph had been plotted using gnu plot
- Everything has been implemented such that if you type 'make' in the terminal window, the required graph will be generated.
- Finally, a graph has been plotted in MATLAB to compare with C program implementation
- **Maximum size of the square matrix has been limited to 1000 because of high compilation time.**

2. FIXED POINT AND FLOATING POINT

2.1 FIXED POINT REPRESENTATION

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: The sign field, integer field, and fractional field.



The advantage of using a fixed-point representation is performance and disadvantage is relatively limited range of values that they can represent. So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy.

Fixed-point numbers help control resource expense on hardware by eliminating the logic requirements for dynamic range shifts inherent in floating-point operations.

Many FPGAs either do not support floating-point arithmetic or cannot process floating-point operations efficiently. Without a dedicated Floating-Point Unit (FPU), floating-point arithmetic requires significant logic resources on hardware, increases the number of clock-cycles required per operation, and reduces the processing efficiency of the device.

Fixed-point numbers provide the following advantages:

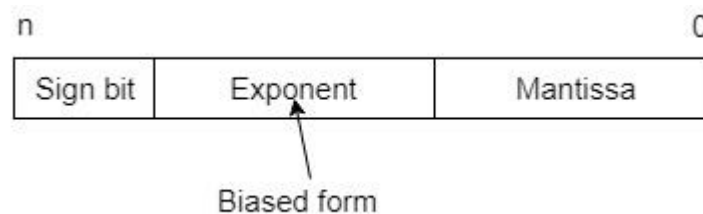
- No need for logic to support dynamic shifts—Floating-point operations require a dynamic shift, or float, of the exponent used to scale the significand, or base number, at run time. With fixed-point numbers, the exponent value is defined by the fixed-point data type and is not computed at run time. Also fixed-point numbers eliminate the need for the hardware logic to calculate and perform dynamic shifts.
- Greater control of FPGA resource costs—The fixed-point data type allows you to exercise greater control of FPGA resources by specifying non-standard bit sizes and coercion options that can conserve memory and logic resources. By specifying the word lengths and coercion behaviors for all terminals and operators in code designed to run on hardware, you optimize the performance of your device or system.

2.2 FLOATING POINT REPRESENTATION:

This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead, it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two parts: The first part represents a signed fixed-point number called mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. The fixed-point mantissa may be fraction or an integer. Floating point is always interpreted to represent a number in the following form: $M \times r^e$

Only the mantissa M and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that it uses **base 2** for the exponent.



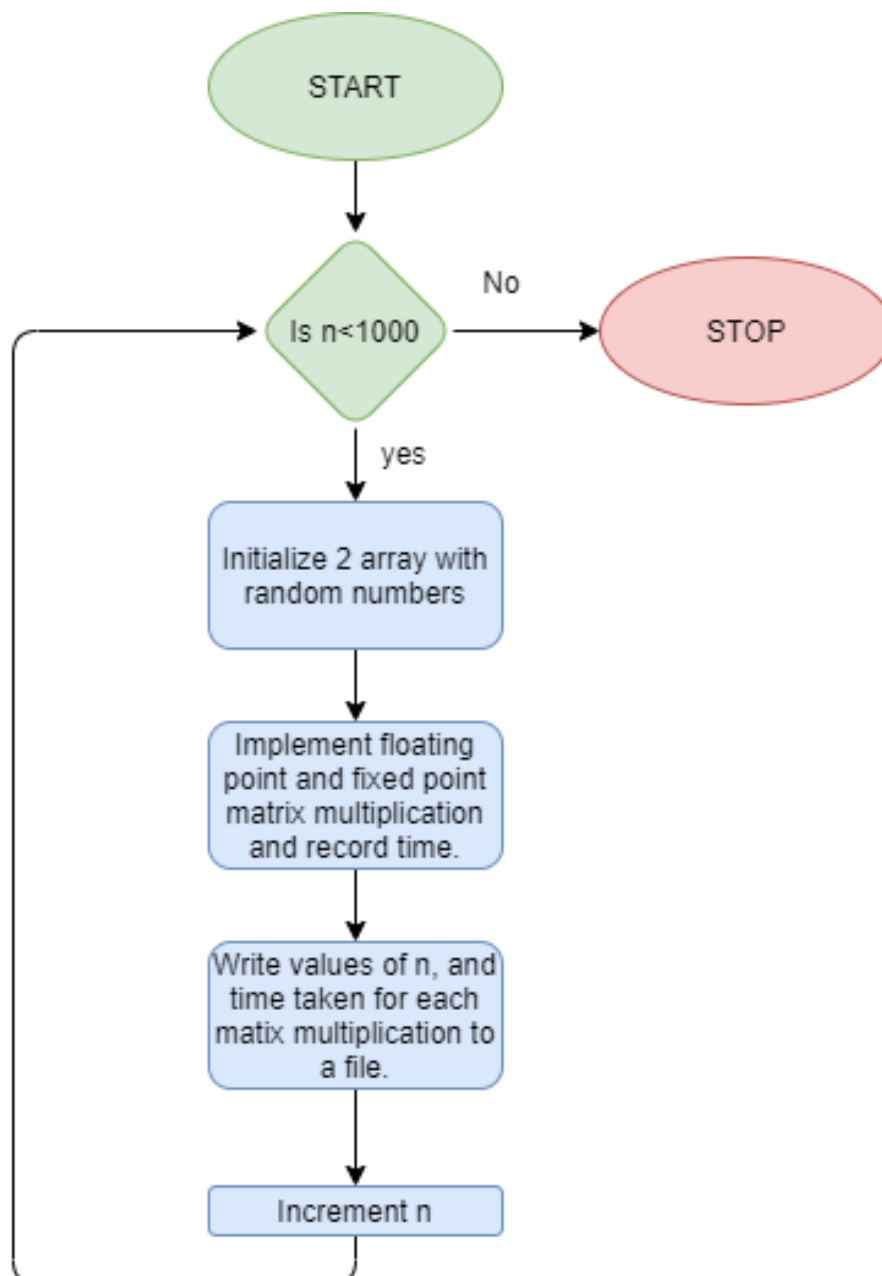
Example:

Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a “hidden bit”.

Then -53.5 is normalized as $-53.5 = (-110101.1)_2 = (-1.101011) \times 2^5$, which is represented as following below,

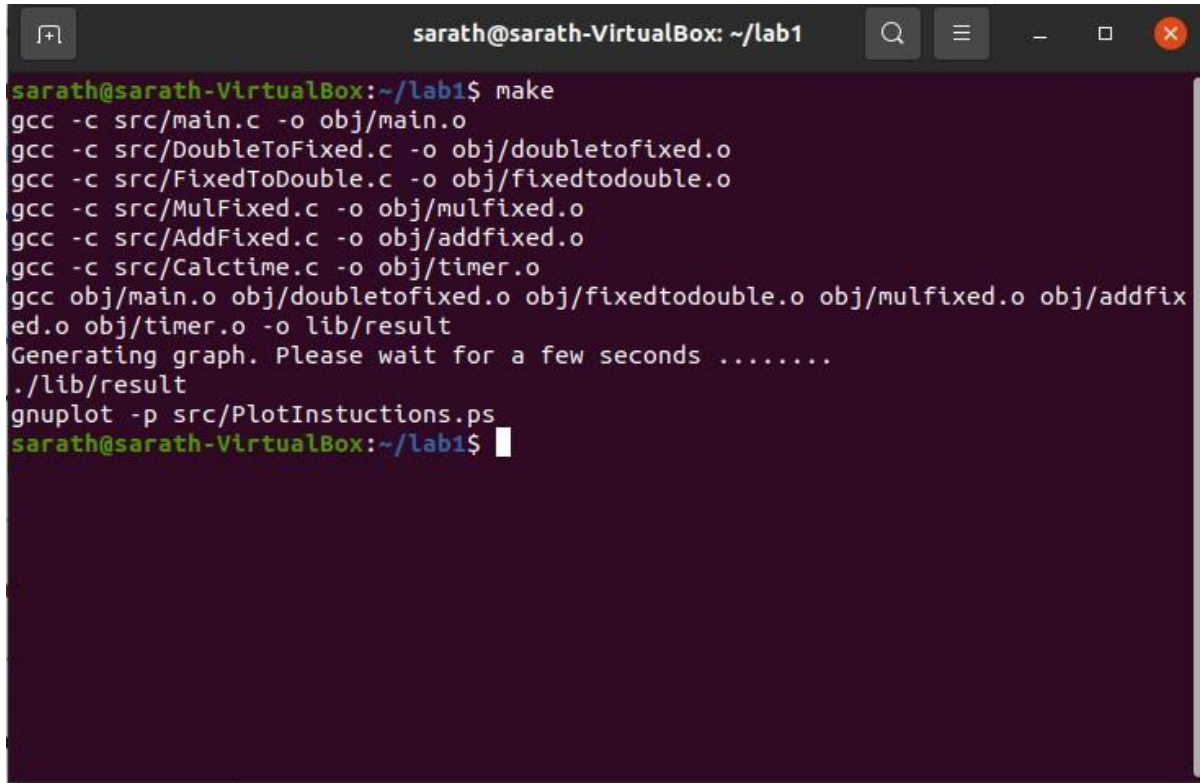
	1	00000101	101011000000000000000000
	Sign bit	Exponent part	Mantissa part
Smallest	0	10000010	000000000000000000000000
	Sign bit	Exponent part	Mantissa part
Largest	0	01111111	111111111111111111111111
	Sign bit	Exponent part	Mantissa part

3. FLOW CHART OF MAIN FUNCTION



4. SCREENSHOTS

4.1 Terminal window while executing make command



```
sarath@sarath-VirtualBox: ~/lab1$ make
gcc -c src/main.c -o obj/main.o
gcc -c src/DoubleToFixed.c -o obj/doubletofixed.o
gcc -c src/FixedToDouble.c -o obj/fixedtodouble.o
gcc -c src/MulFixed.c -o obj/mulfixed.o
gcc -c src/AddFixed.c -o obj/addfixed.o
gcc -c src/Calctime.c -o obj/timer.o
gcc obj/main.o obj/doubletofixed.o obj/fixedtodouble.o obj/mulfixed.o obj/addfixed.o obj/timer.o -o lib/result
Generating graph. Please wait for a few seconds .....
./lib/result
gnuplot -p src/PlotInstructions.ps
sarath@sarath-VirtualBox:~/lab1$
```

4.2 Generated Data

Col 1: Index

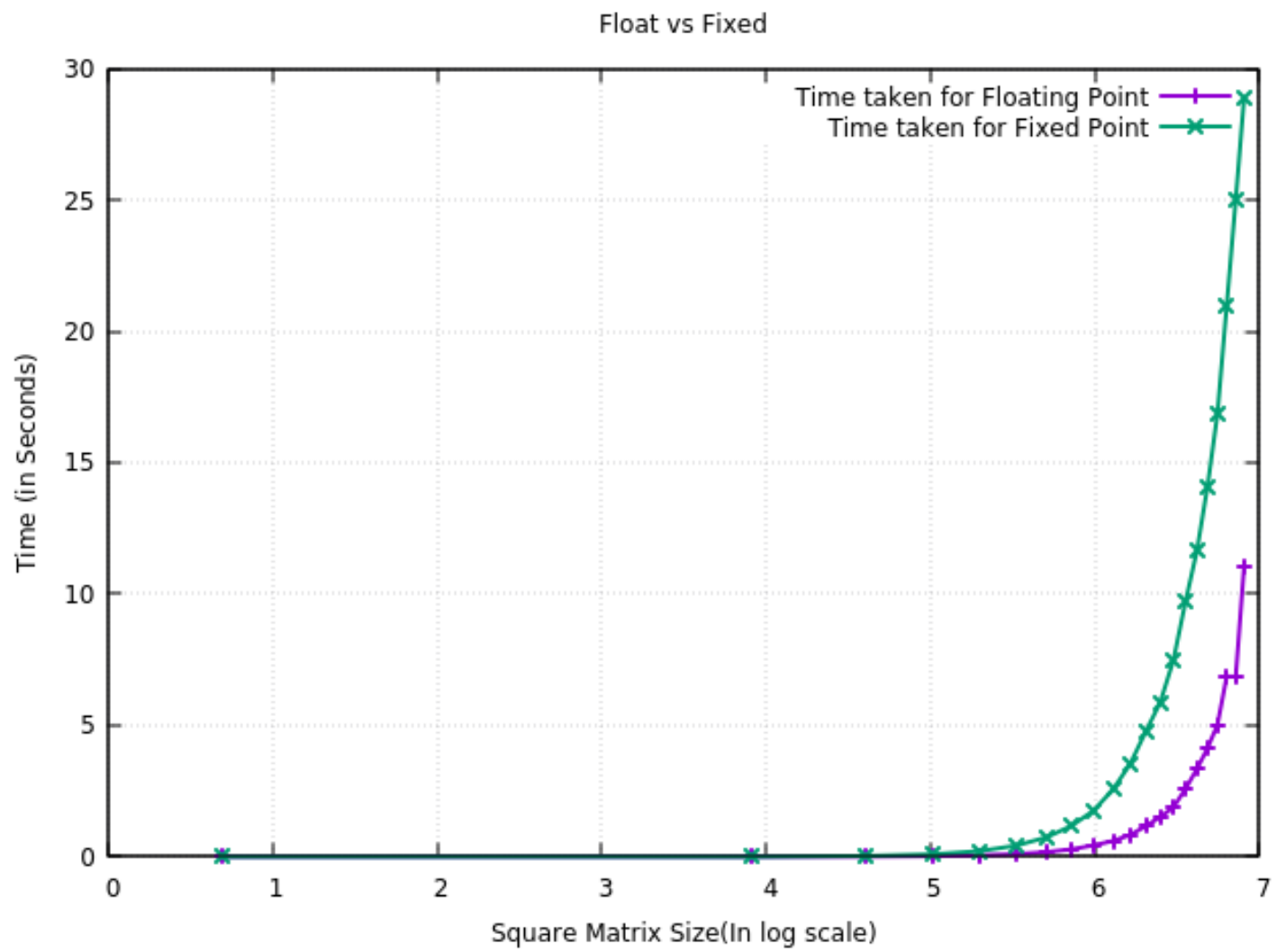
Col 2: Value of n

Col 3: Time taken for Floating Point (in seconds)

Col 4: Time taken for fixed point (in seconds)

plotdata.dat			
1	2	0.000001	0.000001
2	50	0.001432	0.003552
3	100	0.007013	0.027649
4	150	0.022768	0.094259
5	200	0.058841	0.221026
6	250	0.107112	0.433766
7	300	0.187952	0.767081
8	350	0.324137	1.262649
9	400	0.425431	1.801305
10	450	0.632624	2.542315
11	500	0.876995	3.392382
12	550	1.131342	4.662006
13	600	1.442412	6.002328
14	650	2.112292	7.622237
15	700	2.686764	9.610033
16	750	3.134074	11.743550
17	800	4.174913	14.139502
18	850	5.143877	17.398883
19	900	7.036686	20.753308
20	950	7.252889	24.911735
21	1000	10.861375	28.703596

4.3 GNU Plot



5. APPENDIX

5.1 C CODE

5.1.1 Main Function

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include "../include/fixedarithmetic.h" // For float to fixed conversion
#include "../include/timer.h" // Including function to calculate time
const int shift = 8; //Setting the Fixed Point

void main()
{
    int i,n=1;
    FILE *fp; // Initializing file pointer
    fp = fopen("src/plotdata.dat","w+");
    srand(n); // Setting different seed values in each iteration
    double a = rand()*10.0; //Scale factor
    while(n<=1000){
        //Dynamic initialization of 2D array
        double **arr1 = (double **)malloc(n * sizeof(double *)); //Array 1
        for (i=0; i<n; i++)
            arr1[i] = (double *)malloc(n * sizeof(double));

        double **arr2 = (double **)malloc(n * sizeof(double *)); // Array 2
        for (i=0; i<n; i++)
            arr2[i] = (double *)malloc(n * sizeof(double));

        double **result = (double **)malloc(n * sizeof(double *)); //Result
        for (i=0; i<n; i++)
            result[i] = (double *)malloc(n * sizeof(double));

        long long **temp1 = (long long **)malloc(n * sizeof(long long *)); //Temp array to
        store Fixed point Array 1
        for (i=0; i<n; i++)
            temp1[i] = (long long *)malloc(n * sizeof(long long));

        long long **temp2 = (long long **)malloc(n * sizeof(long long *)); //Temp array to
        store Fixed point Array 2
        for (i=0; i<n; i++)
            temp2[i] = (long long *)malloc(n * sizeof(long long));

        // Generating Floating point number using rand()
        for(int i =0;i < n;i++){
            for(int j =0;j<n;j++){
                arr1[i][j]=(float)rand()/(float)(RAND_MAX)* a;
                arr2[i][j]=(float)rand()/(float)(RAND_MAX)* a;
            }
        }
    }
}

```

```

    }
}

// Floating point Matix Multiplication
clock_t start1 = clock();    // Start timer 1
for(int i=0;i <n;i++){
    for(int j=0;j<n;j++){
        double sum=0.0;
        for(int k=0;k<n;k++){
            sum+=arr1[i][k] * arr2[k][j];
        }
        result[i][j]=sum;
    }
}
clock_t stop1 = clock();    //Stop timer 1
double time_spent1 = calctime(start1,stop1); // Function to calculate time

//Fixed Point Matrix Multiplication
//Conversion of floating point to fixed point
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        temp1[i][j]=DoubleToFixed(arr1[i][j],shift);
        temp2[i][j]=DoubleToFixed(arr2[i][j],shift);
    }
}
clock_t start2 = clock();    // Start timer 2
for(int i=0;i <n;i++){
    for(int j=0;j<n;j++){
        long long sum=0;
        for(int k=0;k<n;k++){
            long long c = MulFixed(temp1[i][k],temp2[k][j],shift);
            sum = AddFixed(sum,c,shift);
        }
        result[i][j]=FixedToDouble(sum,shift);
    }
}
clock_t stop2 = clock();    //Stop timer 2
double time_spent2 = calctime(start2,stop2);
fprintf(fp,"%d\t%f\t%f\n",n,time_spent1,time_spent2); //Writing to a file
if(n==1) n=0;
n+=50;
}
fclose(fp);
}

```

5.1.2 DoubleToFixed

```

long long DoubleToFixed(double Double,int shift)
{
    long long Fixed = Double * (1<<shift);
    return Fixed;
}

```

5.1.3 FixedToDouble

```

double FixedToDouble(long long Fixed,int shift)
{
    double Double = Fixed>>shift;
    return Double;
}

```

5.1.4 AddFixed

```

long long AddFixed(long long a,long long b, int shift)
{
    return a+b;
}

```

5.1.5 MulFixed

```

long long MulFixed(long long a , long long b, int shift)
{
    long long mul = (a*b)>>shift;
    return mul;
}

```

5.1.6 CalcTime

```

#include<time.h>

double calctime(double start , double stop)
{
    return (double)(stop-start) / CLOCKS_PER_SEC;
}

```

5.2 Makefile

```
all : result

result: main.o doubletofixed.o fixedtodouble.o mulfixed.o addfixed.o
timer.o
    gcc obj/main.o obj/doubletofixed.o obj/fixedtodouble.o
obj/mulfixed.o obj/addfixed.o obj/timer.o -o lib/result
    @echo Generating graph. Please wait for a few seconds .....
    ./lib/result
    gnuplot -p src/PlotInstructions.ps

main.o: src/main.c
    gcc -c src/main.c -o obj/main.o

doubletofixed.o: src/DoubleToFixed.c
    gcc -c src/DoubleToFixed.c -o obj/doubletofixed.o

fixedtodouble.o: src/FixedToDouble.c
    gcc -c src/FixedToDouble.c -o obj/fixedtodouble.o

mulfixed.o: src/MulFixed.c
    gcc -c src/MulFixed.c -o obj/mulfixed.o

addfixed.o: src/AddFixed.c
    gcc -c src/AddFixed.c -o obj/addfixed.o

timer.o: src/Calctime.c
    gcc -c src/Calctime.c -o obj/timer.o

clean:
    rm -rf obj/*.o
    rm -f lib/result
    rm -f src/plotdata.dat
```

5.3 GNU Plot instructions

```
set title "Float vs Fixed "
set xlabel "Square Matrix Size(In log scale)"
set ylabel "Time (in Seconds)"
set grid
plot "src/plotdata.dat" using (log($1)):2 with linespoints title "Time
taken for Floating Point" lw 2,"src/plotdata.dat" using (log($1)):3 with
linespoints title "Time taken for Fixed Point" lw 2
```

6. COMPARISON

Same procedure for Fixed point vs Floating point has been implemented in MATLAB to compare with the results that are obtained from C.

6.1 MATLAB Code

```
n=2;i=1;
format long; %To generate more precise random numbers

while (n<=1000)

    %Random matrix initialization
    a=100.*randn(n,n); %random numbers from normal dnx
    b=100.*randn(n,n);

    %Floating point multiplication
    tic %Start timer
    c=a*b;
    t1(i)=toc; %Stop Timer2 of iteration i
    a=round(a,2); %Fixing a decimal place
    b=round(b,2);

    %Fixed point multiplication
    tic %Start timer2
    c=a*b;
    t2(i)=toc; %Stop Timer2 of iteration i
    arsz(i)=n; %Storing each n value
    if n==2, n=0; end
    n=n+50;
    i=i+1;
end

%Writing data to a file
A=[arsz;t1;t2];
fileID = fopen('data.txt','w+');
fprintf(fileID,'%d\t%f\t%f\n',A);
fclose(fileID);

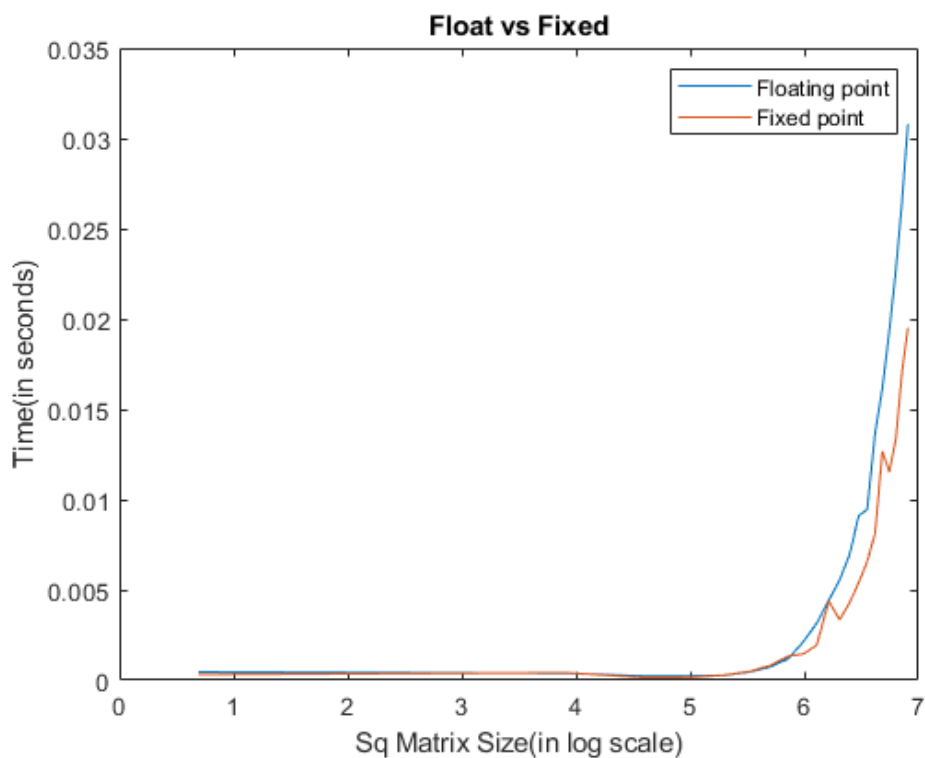
%plot
arsz=log(arsz);
figure
plot(arsz,t1,arsz,t2)
title('Float vs Fixed')
xlabel('Sq Matrix Size(in log scale)')
ylabel('Time(in seconds)')
legend('Floating point','Fixed point')
```

6.2 Comparison of Data from MATLAB and C

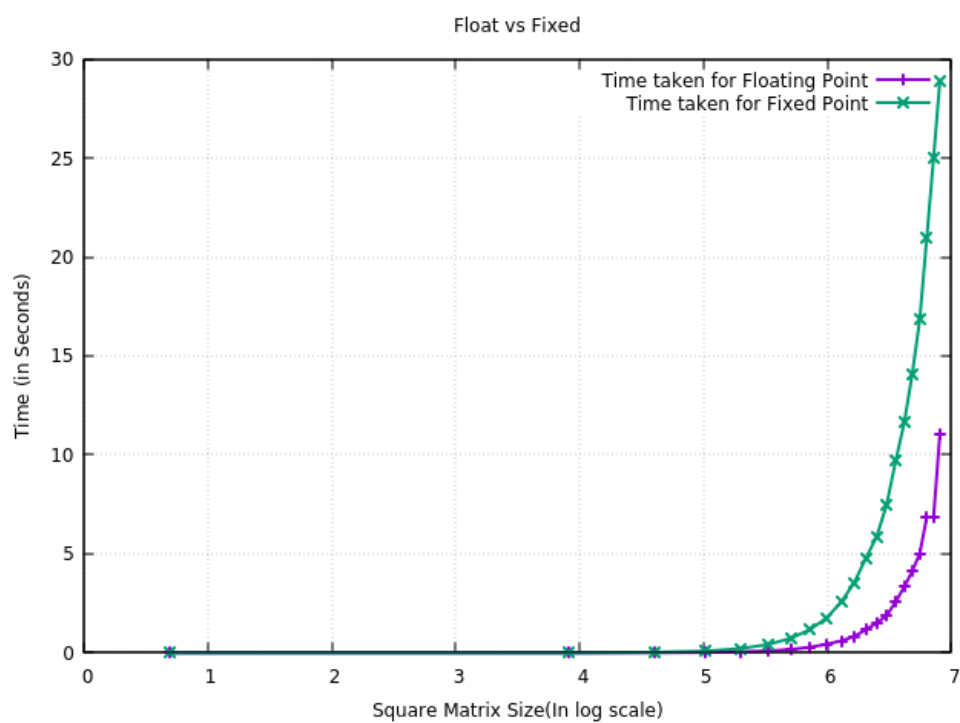
VALUE OF N	DATA FROM C (Time in seconds)		DATA FROM MATLAB (Time in seconds)	
	FLOAT	FIXED	FLOAT	FIXED
2	0.000002	0.000001	0.000899	0.000061
50	0.000645	0.003534	0.000178	0.000118
100	0.008512	0.028531	0.000420	0.000166
150	0.025618	0.093415	0.000262	0.000248
200	0.055809	0.219384	0.000621	0.000566
250	0.104647	0.425970	0.000576	0.000613
300	0.178535	0.736976	0.001067	0.000845
350	0.288266	1.180078	0.001303	0.000709
400	0.436653	1.749475	0.002107	0.001402
450	0.605353	2.561168	0.003087	0.003400
500	0.832732	3.539025	0.003549	0.002609
550	1.172710	4.784563	0.005600	0.003360
600	1.513302	5.869811	0.007024	0.004359
650	1.885594	7.486120	0.009023	0.009155
700	2.579288	9.693191	0.010738	0.006524
750	3.333454	11.632023	0.013545	0.007978
800	4.127273	14.077047	0.013379	0.009374
850	5.019548	16.838892	0.019009	0.011347
900	6.866147	20.996582	0.017971	0.013074
950	6.843670	24.994429	0.026417	0.015811
1000	11.042231	28.864217	0.030753	0.027702

6.3 MATLAB Plot vs GNU Plot

MATLAB Plot:



GNU Plot:



7. CONCLUSION

From the above comparison, the following results has been observed

- Overall, Matrix multiplication is a lot faster in MATLAB than C
 - Reason: MATLAB uses highly optimized libraries for matrix multiplication which is why the plain MATLAB matrix multiplication is so fast.
- MATLAB:
 - Floating point matrix multiplication is taking more time than fixed point
- C:
 - Fixed point matrix multiplication is taking more time than floating point.
- REASON:

MATLAB has inbuilt functions for setting the precision for a floating-point number which can be directly used to convert it to a fixed-point representation. Whereas in C, we needed to write some user defined functions which will slow down the conversion and multiplication process. This resulted in the slow fixed point matrix multiplication for the program in C.

8. REFERENCES

- <https://embeddedartistry.com/blog/2018/07/12/simple-fixed-point-conversion-in-c>
- <https://www.ni.com/documentation/en/labview/latest/data-types/advantages-fixed-point-numbers/>
- <https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/>
- <https://www.geeksforgeeks.org/basics-file-handling-c/>
- <http://www.gnuplot.info/>
- <https://www.i-programmer.info/programming/cc/13669-applying-c-fixed-point-arithmetic.html>
- <https://stackoverflow.com/questions/5248915/execution-time-of-c-program>
- <https://www.tutorialspoint.com/fixed-point-and-floating-point-number-representations>
- <https://stackoverflow.com/questions/13408990/how-to-generate-random-float-number-in-c>
- <https://www.youtube.com/watch?v=S12qx1DwjVk&t=1885s>
- <https://www.youtube.com/watch?v=Y37DTIF-kRQ&t=1097s>
- <https://stackoverflow.com/questions/6058139/why-is-matlab-so-fast-in-matrix-multiplication>
- <https://in.mathworks.com/help/matlab/ref/fprintf.html>
- <https://in.mathworks.com/matlabcentral/answers/39017-calculating-the-time>