

Homework3

November 27, 2023

1 Solution to Homework 3

1.1 Name: *SupreethMohan, YashikaPatil, SandeepKasiraju*

1.2 Problem 1

1.3 Part A

1.3.1 The C program given below will accept N as input and output the approximate value computed by the Monte Carlo method.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

double function(double x[], int dim);

double sampleInterval(double , double );

int main(int argc, char **argv)
{
    double startTime = omp_get_wtime();
    unsigned long long N = atoll(argv[1]);

    double a = -1.0;
    double b = 1.0;

    const int dimension=atoll(argv[2]);
    double x[dimension];
    double V=1024;
    double sum = 0.0;
    srand(time(NULL));
    unsigned long long i;

    for (i=0;i<N;++i)
    {
        for (int j=0;j<dimension;j++)
```

```

    {
        x[j] = sampleInterval(a,b);
    }

    double func_value = function(x,dimension);

    sum = sum + func_value;

}

double integral = (V/N) * sum;
double endTime = omp_get_wtime();
printf("%llu %f %f %f\n",N,integral,V,endTime-startTime);

return 0;
}

double function(double x[], int dim)
{
    double sum = 1.0;
    for (int i=0;i<dim;i++)
    {
        sum = sum + x[i];
    }
    return sum;
}

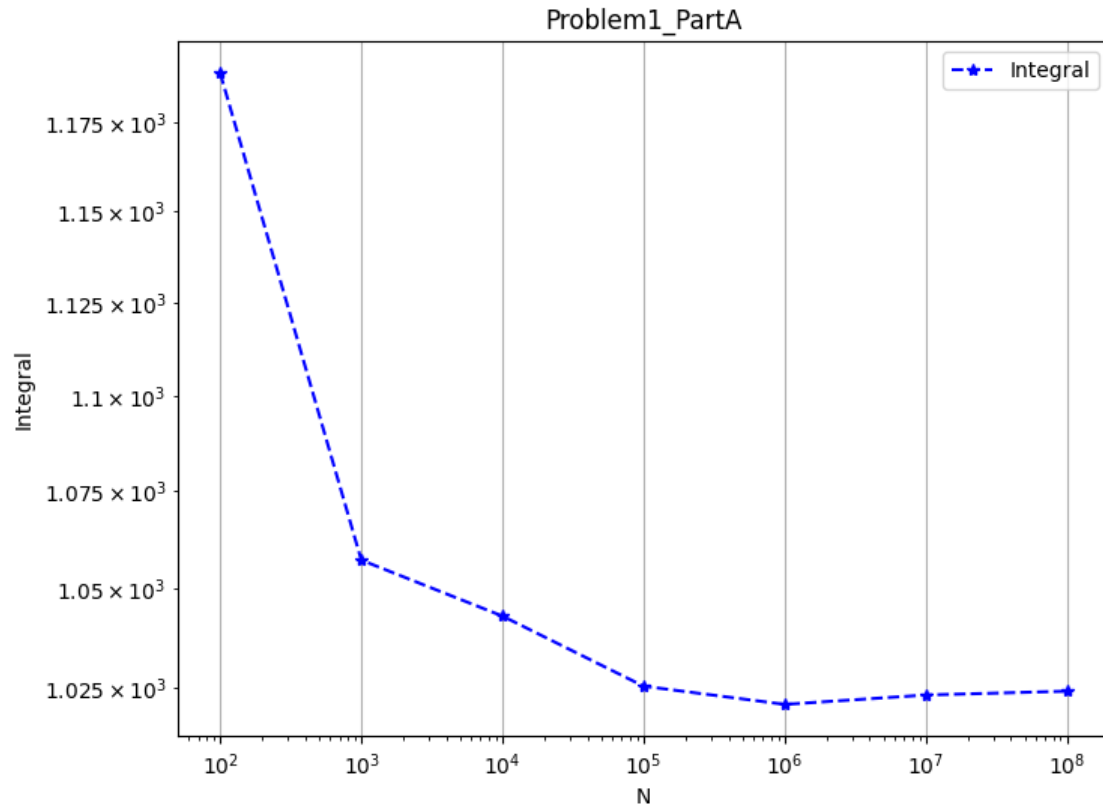
double sampleInterval(double a, double b)
{
    double x = ((double)(rand()/(double)RAND_MAX)*(b-a))+a;
    return x;
}

```

1.3.2 For this homework, we are passing dimension dynamically. Our program accepts 3 arguments, i.e., N, Dimension, and Number of Cores.

1.4 Plot of N vs Integral

1.4.1 The integral converges to 1024 at the rate of $N^{(-1/2)}$



1.5 Problem 2

1.6 Part A and Part B

1.6.1 Parallelized code using reduction and measured time using OMP_Timer and clock() in C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

double function(double x[], int dim);

double sampleInterval(double , double );

int main(int argc, char **argv)
{
```

```

unsigned int seed = (unsigned int)time(NULL);
double startTime = omp_get_wtime();
omp_set_num_threads(4);
unsigned long long N = atoll(argv[1]);
clock_t start_time = clock();
double a = -1.0;
double b = 1.0;
int dimension=atoll(argv[2]);
double x[dimension];
double V=1024;
unsigned long long i;
double sum_thread[4] = {0};
double func_value=0.0;
double sum1 = 0.0;
double integral=0.0;

#pragma omp parallel for private(x) reduction(+:integral)
for (i=0;i<N;++i)
{
    double x[dimension];
    double sum = 1.0;
    for (int j=0;j<dimension;j++)
    {
        x[j]= ((double)(rand_r(&seed))/(double)RAND_MAX)*(b-a))+a;
        sum = sum + x[j];
    }

    integral += sum;
}

double integral1 = (V/N) * integral;
double time = ((double)clock() - start_time)/CLOCKS_PER_SEC;
double endTime = omp_get_wtime();

printf("%llu %f %f %f %f\n",N,integral1,V,time,endTime-startTime);

return 0;
}

double function(double x[], int dim)
{
    double sum = 1.0;
    for (int i=0;i<dim;i++)
    {
        sum = sum + x[i];
    }
}

```

```

    return sum;

}

double sampleInterval(double a, double b)
{
    double x;
    x = ((double)(rand()/((double)RAND_MAX)*(b-a)))+a;
    return x;
}

```

1.6.2 We used 4 threads and used `omp_get_max_threads()` to print the number of threads, as seen in the image (check for last number in the terminal) below:

```

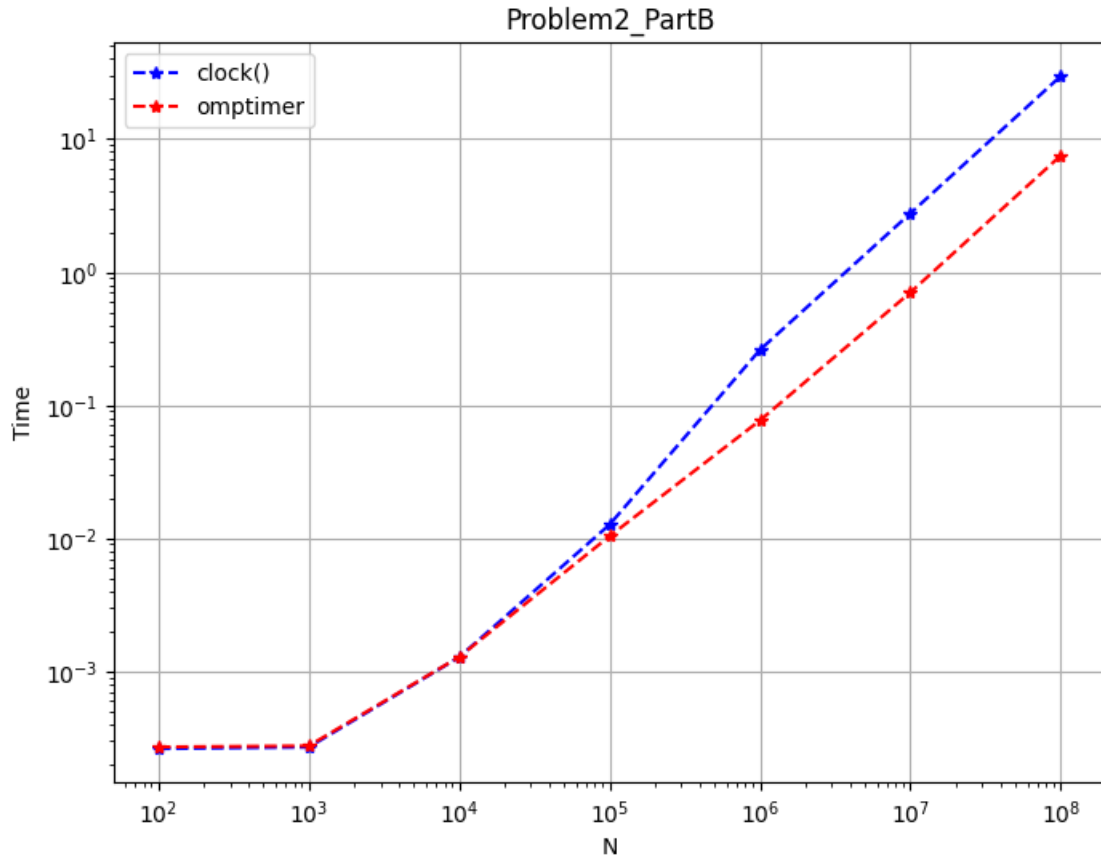
• (base) supreethbmohan@DESKTOP-QN4PIM3:~/HPSCVSCode/Homework3$ ./p2pab 100000000 10
100000000 1023.985511 1024.000000 32.133122 8.141530 4

0[||||||||||||||||||||||||||||||||||||| 82.9%] 4[|||||||||||||||||||||||||||||||||100.0%]
1[||||||||||||| 20.5%] 5[| 1.4%]
2[|||||||||||||||||||||||||||||||||100.0%] 6[| 0.7%]
3[||||| 8.0%] 7[|||||||||||||||||||||||||||||||||100.0%]
Mem[||||||||||||||||| 1.00G/3.78G] Tasks: 34, 168 thr; 5 running
Swp[||||| 0K/1.00G] Load average: 1.27 0.76 0.57
Uptime: 01:14:58

```

1.7 Part B

1.7.1 The comparison between clock time and omp_timer is shown in the image below. As expected, the clock() time is more than that of omp time.



1.8 Problem 3

1.9 Part A

1.9.1 OpenMP parallelized code implementing Rosenbrock function. We used a void function, fun(), which is a test function.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

double function(double *x, int dim);

double variable(double , double, unsigned int *);

int main(int argc, char **argv)
{
```

```

        fun();

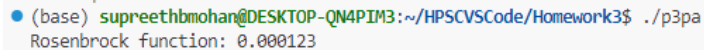
    return 0;
}

void fun()
{
    double x[10]={0};
    double val=function(x,10);
    printf("%f",val);
}

double function(double *x, int dim)
{
    double f = 0.0;
    for(int i=0; i<dim-1; i++)
    {
        f += (-1*((pow(1-x[i],2) + 100*(pow(x[i+1]- pow(x[i],2),2)))));
    }
    return exp(f);
}

```

1.9.2 Test Case:



```

(base) supreethbmohan@DESKTOP-QN4PIM3:~/HPSCVSCode/Homework3$ ./p3pa
Rosenbrock function: 0.000123

```

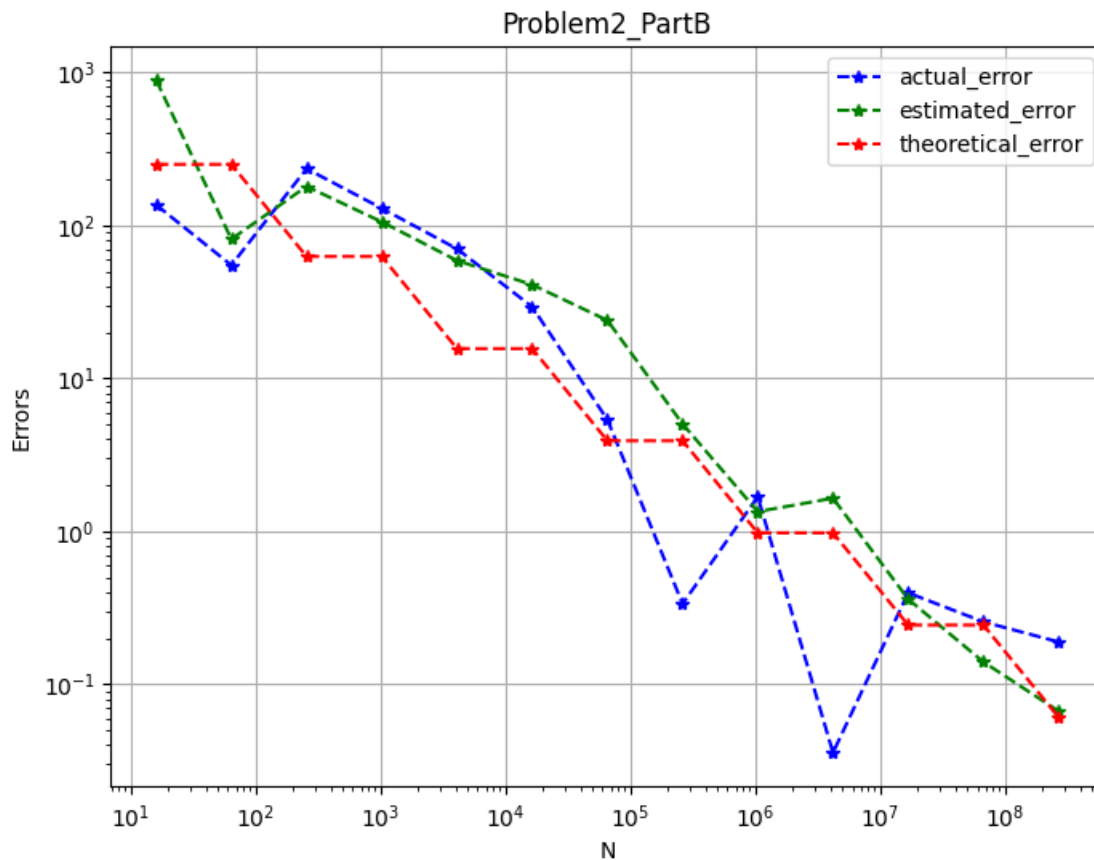
We passed $x[10] = \{0\}$, and computed the Rosenbrock function. The following image shows the output which is same as $\exp(-9)$ when calculated with the help of scientific calculator.

1.10 Part B

1.10.1 We are using reduction and critical parallelization. We used four Anvil job scripts, namely Strong scaling reduction, Strong scaling critical, Weak scaling reduction, Weak scaling critical.

1.10.2 For improving code's scaling performance,

1.10.3 1. Used `rand_r` for random number generation. 2. Used the number of cores for seed generation. 3. Implemented reduction variable with `+` operator



1.11 Part C

1.11.1 $N = 4^{20}$, using 128 cores, the integral value is 4.0108 which seems to be accurate as we got the same result multiple times by one digit accuracy.

```
x-yatil@login07.anvil:[homework3] $ cat Hw3.o3891243
109951162776 4.010807e+00 3055.472729
```


1.12 Bonus_Points

1.13 Part A

Modified the code to output the value $\hat{I}(4k) - \hat{I}(4k + 1)$ along with the value of the integral $\hat{I}(4k)$

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

double function(double x[], int dim);
double sampleInterval(double, double);

int main(int argc, char **argv)
{
    double startTime = omp_get_wtime();
    unsigned long long N = atoll(argv[1]);

    double a = -1.0;
    double b = 1.0;

    const int dimension = atoll(argv[2]);
    double x[dimension];
    double V = 1024;
    double sum = 0.0;
    srand(time(NULL));
    unsigned long long i;
    int Num=pow(4,N+1);

    const int cores=atoll(argv[3]);
    omp_set_num_threads(cores);
    double sumi =0.0;
    double absolute_error=0.0;
    double integral=0.0;
    double intg=0.0;

    for (i = 1; i <= Num; i++)
    {
        for (int j = 0; j < dimension; j++)
        {
            x[j] = sampleInterval(a, b);
        }

        double func_value = function(x, dimension);

        sum = sum + func_value;
    }
}
```

```

    for(int k=2;k<=N;k++)
    {
        if(i == pow(4,k))
        {
            integral = (V / (i)) * (sumi + sum);
            absolute_error=fabs(integral-intg);
            intg = integral;
            if(pow(4,k)-pow(4,k-1))
            printf("N=%llu  integral=%f absolute_error=%f volumn=%f\n", i, integral, absolute_error, pow(4,k)-pow(4,k-1));
            sumi+=sum;
            sum = 0;
        }
    }

}

double endTime = omp_get_wtime();

return 0;
}

double function(double x[], int dim)
{
    double sum = 1.0;
    for (int i = 0; i < dim; i++)
    {
        sum = sum + x[i];
    }
    return sum;
}

double sampleInterval(double a, double b)
{
    double x = ((double)(rand() / (double)RAND_MAX) * (b - a)) + a;
    return x;
}

```

```

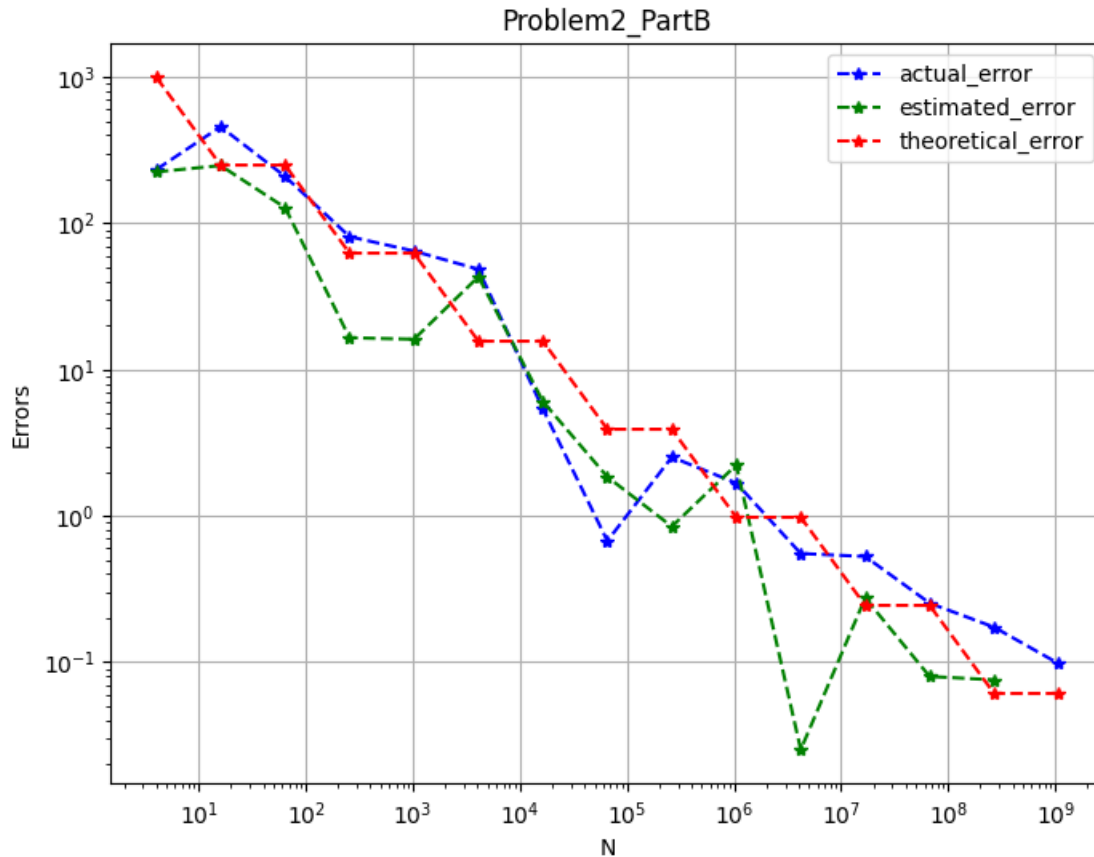
● (base) supreethmohar@DESKTOP-QN4PIM3:~/HPSCVSCode/Homework3$ ./bpa 4 10 1
N=16  integral=1275.241380 absolute_error=1275.241380 volumn=1024.000000
N=64  integral=908.244495 absolute_error=366.996886 volumn=1024.000000
N=256  integral=1018.021026 absolute_error=109.776531 volumn=1024.000000

```

We passed $k=4$, and printed the integrals, $\hat{I}(4k)$, along with absolute error, $\hat{I}(4k) - \hat{I}(4k+1)$

1.14 Part B

1.14.1 plot of actual error, $\hat{I}(4k) - 2^{10}$, the estimated error, and the theoretically expected error



1.15 Part C

1.15.1 Parallelized the program using OpenMP reduction. To make one thread print the integral, we restructured the code and used omp critical, so that only one thread will have access to compute and print the integral.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

double function(double x[], int dim);
double variable(double a, double b, unsigned int *seed);

int main(int argc, char **argv) {
    double startTime = omp_get_wtime();
    unsigned long long N = atoll(argv[1]);
```

```

double a = -1.0;
double b = 1.0;

const int dimension = atoll(argv[2]);
double x[dimension];
double V = 1024;
unsigned int seed = (unsigned int)time(NULL);
unsigned long long i;
int Num = pow(4, N);
const int cores = atoll(argv[3]);
omp_set_num_threads(cores);
double sumi = 0.0;
double integral = 0.0;
double intg = 0.0;

#pragma omp parallel private(x, seed, i) reduction(+ : sumi, integral, intg)
{
    double sum = 0.0;

    #pragma omp for
    for (i = 1; i <= Num; i++) {
        for (int j = 0; j < dimension; j++) {
            x[j] = variable(a, b, &seed);
        }

        double func_value = function(x, dimension);

        sum += func_value;

        for (int k = 2; k <= N; k++) {
            if (i == pow(4, k)) {
                #pragma omp critical
                {
                    integral = (V / (i)) * (sumi + sum);
                    intg = integral;
                    printf("%llu %f %f\n", i, integral, V);
                    sumi += sum;
                    sum = 0.0; // Reset private sum for each thread
                }
            }
        }
    }
}

double endTime = omp_get_wtime();
return 0;
}

```

```

double function(double x[], int dim) {
    double sum = 1.0;
    for (int i = 0; i < dim; i++) {
        sum = sum + x[i];
    }
    return sum;
}

double variable(double a, double b, unsigned int *seed) {
    double x = ((double)(rand_r(seed) / (double)RAND_MAX) * (b - a)) + a;
    return x;
}

```

```

• (base) supreethmohan@DESKTOP-QN4PIM3:~/HPSCVCode/Homework3$ ./bpcss 15 10 4
MC Points = 4, Integral = 529.564377, Estimated Error (for i = 0) = 529.564377, V = 1024.000000
MC Points = 16, Integral = 900.182524, Estimated Error (for i = 4) = 370.618147, V = 1024.000000
MC Points = 64, Integral = 1024.111816, Estimated Error (for i = 16) = 123.929293, V = 1024.000000
MC Points = 256, Integral = 800.299654, Estimated Error (for i = 64) = 223.812162, V = 1024.000000
MC Points = 1024, Integral = 1015.942800, Estimated Error (for i = 256) = 215.643145, V = 1024.000000
MC Points = 4096, Integral = 1036.486726, Estimated Error (for i = 1024) = 20.543927, V = 1024.000000
MC Points = 16384, Integral = 1021.977383, Estimated Error (for i = 4096) = 14.509343, V = 1024.000000
MC Points = 65536, Integral = 1006.598770, Estimated Error (for i = 16384) = 15.378612, V = 1024.000000
MC Points = 262144, Integral = 1017.995476, Estimated Error (for i = 65536) = 11.396705, V = 1024.000000
MC Points = 1048576, Integral = 1024.937894, Estimated Error (for i = 262144) = 6.942418, V = 1024.000000
MC Points = 4194304, Integral = 1024.998427, Estimated Error (for i = 1048576) = 0.060533, V = 1024.000000
MC Points = 16777216, Integral = 1024.151512, Estimated Error (for i = 4194304) = 0.846915, V = 1024.000000
MC Points = 67108864, Integral = 1024.000000, Estimated Error (for i = 16777216) = 0.151512, V = 1024.000000
MC Points = 268435456, Integral = 1024.000000, Estimated Error (for i = 67108864) = 0.000000, V = 1024.000000
○ (base) supreethmohan@DESKTOP-QN4PIM3:~/HPSCVCode/Homework3$

```