Ex-5	Variable types & Parallel Prefix sum
12/08/2025	

Aim:

To

1. Write a C++ program in which, inside the parallel region, change the datatype of variables private, firstprivate, lastprivate, shared. Explore the default behaviour inside a parallel region.

2. Write a C++ program with OpenMP directives to find the parallel prefix sum. Compare with the serial version of the code for array sizes changing from {10,100,1000,10000,100000,1000000}.

3. Find an application with a while loop which can be parallelised using the slave model. Compare the speedup you achieve post-parallelisation.

Algorithm:

1)

1) start

2) declare variables for shared,private,firstprivate and last private

3) open openmp section and modify and access the variables

4) print the changes in the variables

5) stop

2)

1) start

2) implement functions for prefix sum in parallel and prefix sum in serial

3) In main function call both function and measure time

4) stop

3)

1) start

2) implement functions for performing monte carlo estimation of pi using serial and parallel methods

3) In main functions call the functions and compare by measuring time

4) stop

CODE:

1)

```
#include <iostream>
#include <omp.h>

int main() {
    int shared_var=10;
    int private_var=20;
    int firstprivate_var=30;
    int lastprivate_var=40;

    std::cout<<"Before parallel region:"<<std::endl;
    std::cout<<"shared_var = "<<shared_var<<std::endl;
    std::cout<<"private_var = "<<private_var<<std::endl;
    std::cout<<"firstprivate_var = "<<firstprivate_var<<std::endl;
    std::cout<<"lastprivate_var = "<<lastprivate_var<<std::endl;

    #pragma omp parallel for \
        shared(shared_var) \
        private(private_var) \
        firstprivate(firstprivate_var) \
        lastprivate(lastprivate_var)
    for(int i=0;i<omp_get_max_threads();i++){
        int default_var=100;

        shared_var+=i;
        private_var+=i;
        firstprivate_var+=i;
        lastprivate_var=i;
        default_var+=i;

        #pragma omp critical
        {
            std::cout<<"Thread "<<omp_get_thread_num()<<": "
                     <<"shared_var = "<<shared_var<<", "
                     <<"private_var = "<<private_var<<", "
                     <<"firstprivate_var = "<<firstprivate_var<<", "
                     <<"lastprivate_var = "<<lastprivate_var<<", "
                     <<"default_var = "<<default_var<<std::endl;
        }
    }

    std::cout << "\nAfter parallel region:" << std::endl;
    std::cout << "shared_var = " << shared_var << std::endl;
    std::cout << "private_var = " << private_var << " (unchanged in master thread)" << std::endl;
    std::cout << "firstprivate_var = " << firstprivate_var << " (unchanged in master thread)" <<
std::endl;
    std::cout << "lastprivate_var = " << lastprivate_var << " (value from last iteration/thread)" <<
std::endl;

    return 0;
}
```

2)

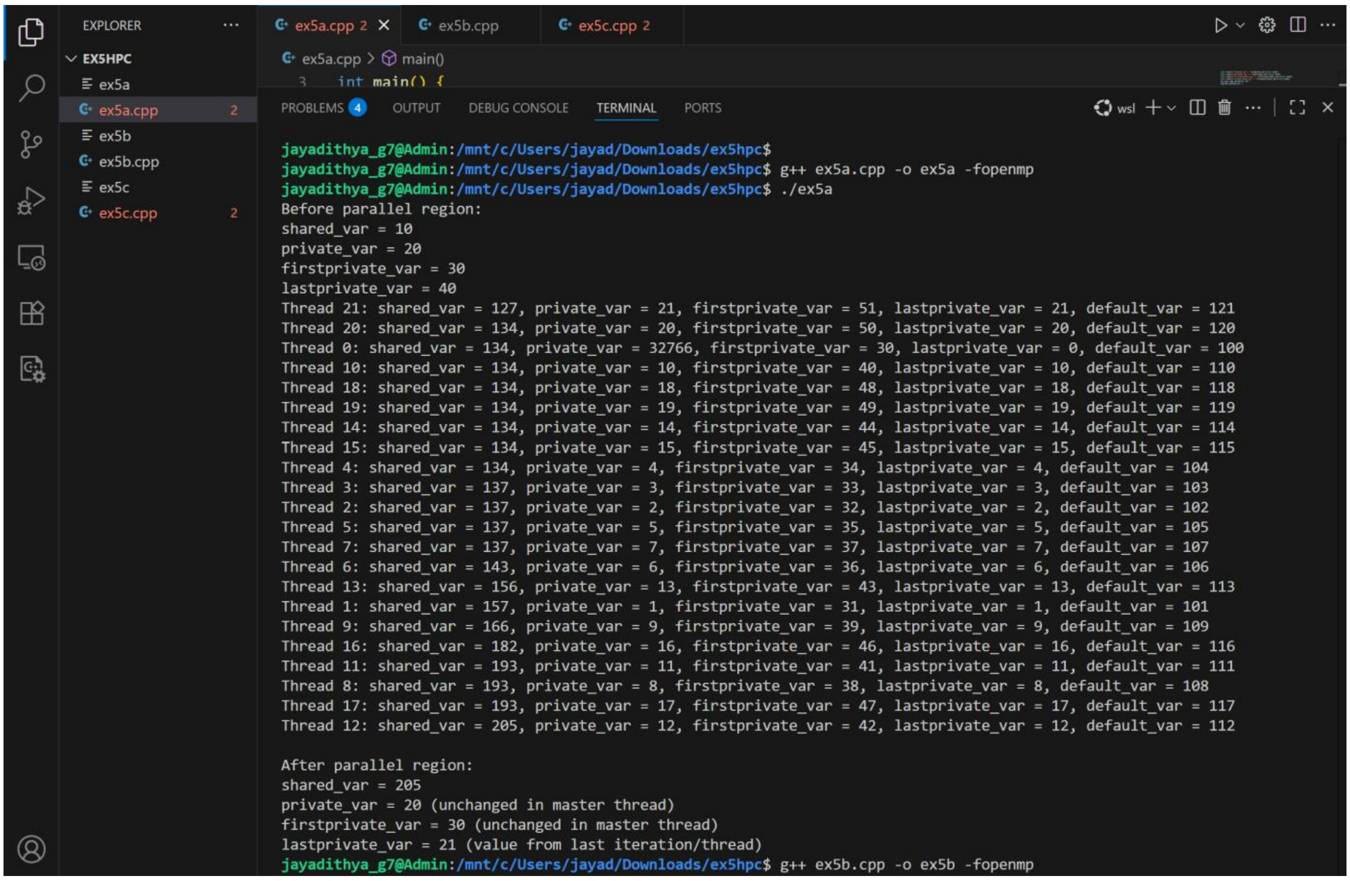
```
#include <iostream>
#include <vector>
#include <chrono>
#include <omp.h>
#include <cassert>


void prefix_sum_serial(const std::vector<int>& input, std::vector<int>& output) {
    output[0]=input[0];
    for (size_t i=1;i<input.size();i++) {
        output[i]=output[i-1]+input[i];
    }

}


void prefix_sum_parallel(const std::vector<int>& input, std::vector<int>& output) {
    int n=input.size();
    int num_threads=1;

    #pragma omp parallel
    {
        #pragma omp single
        num_threads=omp_get_num_threads();
    }

    std::vector<int> partial_sums(num_threads+1,0);

    #pragma omp parallel
    {
        int tid=omp_get_thread_num();
        int chunk_size=(n+num_threads-1)/num_threads;
        int start=tid*chunk_size;
        int end=std::min(start+chunk_size,n);

        if(start<n){
            output[start]=input[start];

            for (int i=start+1;i<end;i++) {
                output[i]=output[i-1]+input[i];
            }
            partial_sums[tid+1]=output[end-1]; // store sum of this chunk
        }
    }
    for (int i=1;i<num_threads+1;i++){
        partial_sums[i]+=partial_sums[i - 1];
    }


    #pragma omp parallel
    {
        int tid=omp_get_thread_num();
        int chunk_size=(n+num_threads-1)/num_threads;
        int start=tid*chunk_size;
        int end=std::min(start+chunk_size,n);

        if(tid>0&&start<n){
            int add_value=partial_sums[tid];
            for(int i=start;i<end;i++){
                output[i]+=add_value;
            }
        }
    }
}

int main() {
    std::vector<int> sizes={10,100,1000,10000,100000,1000000};

    for (int n : sizes){
        std::vector<int> input(n);
        std::vector<int> output_serial(n);
        std::vector<int> output_parallel(n);


        for (int i=0;i<n;i++) {
            input[i] = i+1;
        }


        auto start_serial=std::chrono::high_resolution_clock::now();
        prefix_sum_serial(input, output_serial);
        auto end_serial=std::chrono::high_resolution_clock::now();


        auto start_parallel=std::chrono::high_resolution_clock::now();
        prefix_sum_parallel(input, output_parallel);
        auto end_parallel=std::chrono::high_resolution_clock::now();


        bool correct=true;
        for (int i=0;i<n;i++) {
            if(output_serial[i]!=output_parallel[i]) {
                correct=false;
                break;
            }
        }

        auto serial_time=std::chrono::duration<double, std::milli>(end_serial - start_serial).count();
        auto parallel_time=std::chrono::duration<double, std::milli>(end_parallel -
start_parallel).count();

        std::cout<<"Array size: "<<n<<"\n";
        std::cout<<"Serial time: "<<serial_time<<" ms\n";
        std::cout<<"Parallel time: "<<parallel_time<<" ms\n";
        std::cout<<"Results match? "<<(correct ? "Yes" : "No")<<"\n\n";
    }

    return 0;
}
```

3)

```
#include <iostream>
#include <random>
#include <chrono>
#include <omp.h>

double monte_carlo_serial(long long num_points) {
    std::mt19937_64 rng(42);
    std::uniform_real_distribution<double> dist(0.0, 1.0);
    long long inside_circle=0;
    long long i=0;

    while(i<num_points){
        double  x=dist(rng);
        double y=dist(rng);

        if(x*x+y*y<=1.0)
            inside_circle++;
        i++;
    }

    return 4.0*inside_circle/num_points;
}

double monte_carlo_parallel(long long num_points, int num_threads) {
    long long inside_circle = 0;

    #pragma omp parallel num_threads(num_threads)
    {
        std::mt19937_64 rng(42 + omp_get_thread_num());
        std::uniform_real_distribution<double> dist(0.0, 1.0);
        long long local_count=0;

        #pragma omp for
        for (long long i=0;i<num_points;i++) {
            double x=dist(rng);
            double y=dist(rng);

            if (x*x+y*y<=1.0)
                local_count++;
        }

        #pragma omp atomic
        inside_circle+=local_count;
    }

    return 4.0*inside_circle/num_points;
}

int main(){
    long long num_points=1e8;
    int num_threads=4;

    std::cout<<"Estimating pi using "<<num_points<<" points\n";
    std::cout<<"Using "<<num_threads<<" threads for parallel version\n\n";


    auto start_serial=std::chrono::high_resolution_clock::now();
    double pi_serial=monte_carlo_serial(num_points);
    auto end_serial=std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> time_serial=end_serial-start_serial;

    std::cout<<"Serial pi ≈ "<<pi_serial<<"\n";
    std::cout<<"Serial Time: "<<time_serial.count()<<" seconds\n\n";


    auto start_parallel=std::chrono::high_resolution_clock::now();
    double pi_parallel=monte_carlo_parallel(num_points, num_threads);
    auto end_parallel=std::chrono::high_resolution_clock::now();
    std::chrono::duration<double>  time_parallel=end_parallel-start_parallel;

    std::cout<<"Parallel pi ≈ "<<pi_parallel<<"\n";
    std::cout<<"Parallel Time: "<<time_parallel.count()<<" seconds\n";


    double speedup=time_serial.count()/time_parallel.count();
    std::cout<<"\nSpeedup: "<<speedup<<"×\n";

    return 0;
}
```

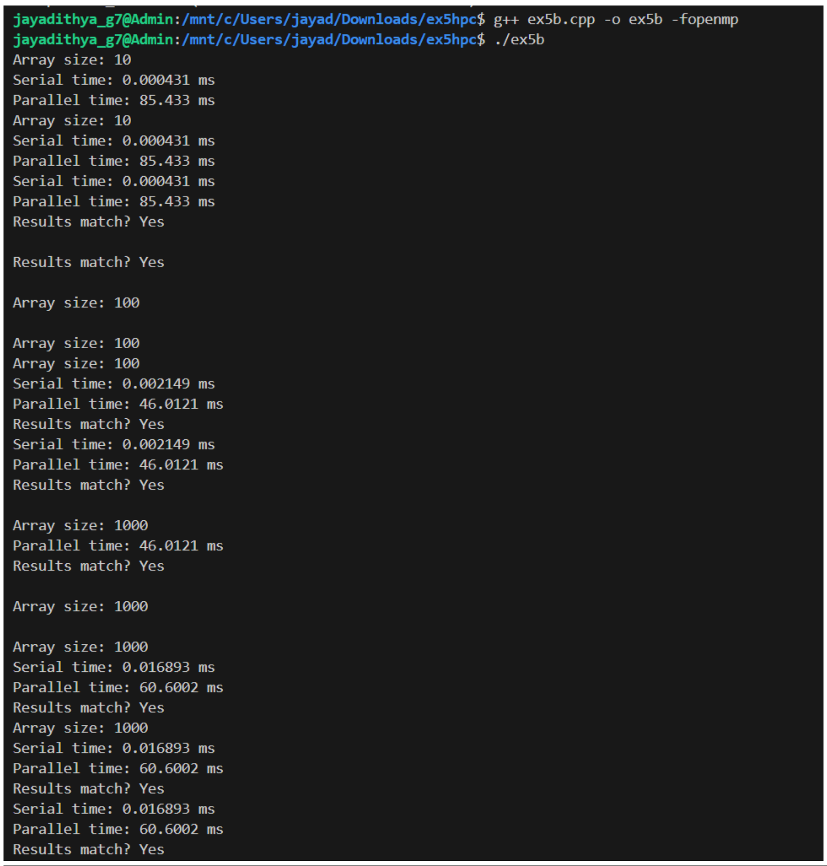
OUTPUT:

1)

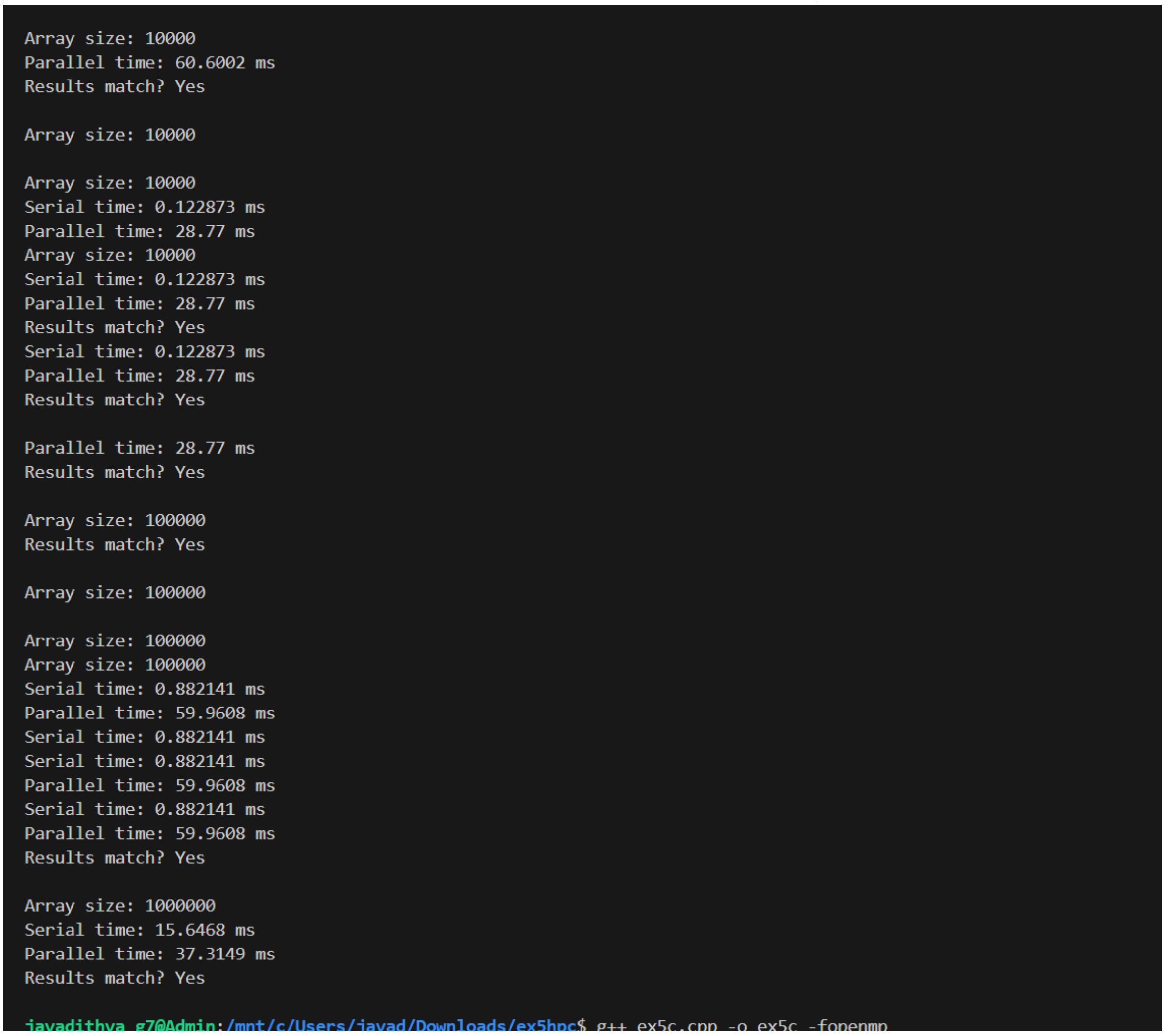


```
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ g++ ex5a.cpp -o ex5a -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ ./ex5a
Before parallel region:
shared_var = 10
private_var = 20
firstprivate_var = 30
lastprivate_var = 40
Thread 21: shared_var = 127, private_var = 21, firstprivate_var = 51, lastprivate_var = 21, default_var = 121
Thread 20: shared_var = 134, private_var = 20, firstprivate_var = 50, lastprivate_var = 20, default_var = 120
Thread 0: shared_var = 134, private_var = 32766, firstprivate_var = 30, lastprivate_var = 0, default_var = 100
Thread 10: shared_var = 134, private_var = 10, firstprivate_var = 40, lastprivate_var = 10, default_var = 110
Thread 18: shared_var = 134, private_var = 18, firstprivate_var = 48, lastprivate_var = 18, default_var = 118
Thread 19: shared_var = 134, private_var = 19, firstprivate_var = 49, lastprivate_var = 19, default_var = 119
Thread 14: shared_var = 134, private_var = 14, firstprivate_var = 44, lastprivate_var = 14, default_var = 114
Thread 15: shared_var = 134, private_var = 15, firstprivate_var = 45, lastprivate_var = 15, default_var = 115
Thread 4: shared_var = 134, private_var = 4, firstprivate_var = 34, lastprivate_var = 4, default_var = 104
Thread 3: shared_var = 137, private_var = 3, firstprivate_var = 33, lastprivate_var = 3, default_var = 103
Thread 2: shared_var = 137, private_var = 2, firstprivate_var = 32, lastprivate_var = 2, default_var = 102
Thread 5: shared_var = 137, private_var = 5, firstprivate_var = 35, lastprivate_var = 5, default_var = 105
Thread 7: shared_var = 137, private_var = 7, firstprivate_var = 37, lastprivate_var = 7, default_var = 107
Thread 6: shared_var = 143, private_var = 6, firstprivate_var = 36, lastprivate_var = 6, default_var = 106
Thread 13: shared_var = 156, private_var = 13, firstprivate_var = 43, lastprivate_var = 13, default_var = 113
Thread 1: shared_var = 157, private_var = 1, firstprivate_var = 31, lastprivate_var = 1, default_var = 101
Thread 9: shared_var = 166, private_var = 9, firstprivate_var = 39, lastprivate_var = 9, default_var = 109
Thread 16: shared_var = 182, private_var = 16, firstprivate_var = 46, lastprivate_var = 16, default_var = 116
Thread 11: shared_var = 193, private_var = 11, firstprivate_var = 41, lastprivate_var = 11, default_var = 111
Thread 8: shared_var = 193, private_var = 8, firstprivate_var = 38, lastprivate_var = 8, default_var = 108
Thread 17: shared_var = 193, private_var = 17, firstprivate_var = 47, lastprivate_var = 17, default_var = 117
Thread 12: shared_var = 205, private_var = 12, firstprivate_var = 42, lastprivate_var = 12, default_var = 112

After parallel region:
shared_var = 205
private_var = 20 (unchanged in master thread)
firstprivate_var = 30 (unchanged in master thread)
lastprivate_var = 21 (value from last iteration/thread)
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ g++ ex5b.cpp -o ex5b -fopenmp
```

2)

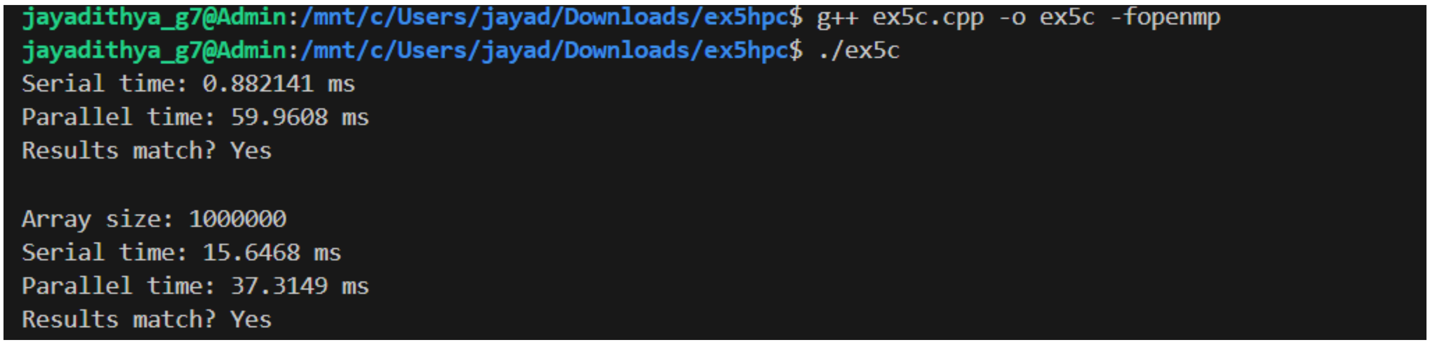


```
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ g++ ex5b.cpp -o ex5b -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ ./ex5b
Array size: 10
Serial time: 0.000431 ms
Parallel time: 85.433 ms
Array size: 10
Serial time: 0.000431 ms
Parallel time: 85.433 ms
Serial time: 0.000431 ms
Parallel time: 85.433 ms
Results match? Yes

Results match? Yes

Array size: 100

Array size: 100
Array size: 100
Serial time: 0.002149 ms
Parallel time: 46.0121 ms
Results match? Yes
Serial time: 0.002149 ms
Parallel time: 46.0121 ms
Results match? Yes

Array size: 1000
Parallel time: 46.0121 ms
Results match? Yes

Array size: 1000

Array size: 1000
Serial time: 0.016893 ms
Parallel time: 60.6002 ms
Results match? Yes
Array size: 1000
Serial time: 0.016893 ms
Parallel time: 60.6002 ms
Results match? Yes
Serial time: 0.016893 ms
Parallel time: 60.6002 ms
Results match? Yes
```



```
Array size: 10000
Parallel time: 60.6002 ms
Results match? Yes

Array size: 10000

Array size: 10000
Serial time: 0.122873 ms
Parallel time: 28.77 ms
Array size: 10000
Serial time: 0.122873 ms
Parallel time: 28.77 ms
Results match? Yes
Serial time: 0.122873 ms
Parallel time: 28.77 ms
Results match? Yes

Parallel time: 28.77 ms
Results match? Yes

Array size: 100000
Results match? Yes

Array size: 100000

Array size: 100000
Array size: 100000
Serial time: 0.882141 ms
Parallel time: 59.9608 ms
Serial time: 0.882141 ms
Serial time: 0.882141 ms
Parallel time: 59.9608 ms
Serial time: 0.882141 ms
Parallel time: 59.9608 ms
Results match? Yes

Array size: 1000000
Serial time: 15.6468 ms
Parallel time: 37.3149 ms
Results match? Yes

jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ g++ ex5c.cpp -o ex5c -fopenmp
```

3)



```
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ g++ ex5c.cpp -o ex5c -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex5hpc$ ./ex5c
Serial time: 0.882141 ms
Parallel time: 59.9608 ms
Results match? Yes

Array size: 1000000
Serial time: 15.6468 ms
Parallel time: 37.3149 ms
Results match? Yes
```

Result:-

Hence the c++ programs are executed successfully and output has been verified.