Name: Vikas Mane
Roll No: B1921152
Class: BE-B

Assignment 5

Title: Implement parallel reduction using Min, Max, Sum and Average Operations.

- Minimum
- Maximum
- Sum
- Average

Aim:

Implement parallel reduction using Min, Max, Sum and Average Operations.

System requirements:

64-bit Open-source Linux or its derivative Programming Languages: C/C++

Theory:

OpenMP:

OpenMP is a set of C/C++ pragmas (or FORTRAN equivalents) which provide the programmer a high-level front-end interface which get translated as calls to threads (or other similar entities). The key phrase here is "higher-level"; the goal is to better enable the programmer to "think parallel," alleviating him/her of the burden and distraction of dealing with setting up and coordinating threads. For example, the OpenMP directive. OpenMP Core

Syntax:

Most of the constructs in OpenMP are compiler directives:
#pragma omp construct [clause [clause]...]

Example:

#pragma omp parallel num_threads(4)
Function prototypes and types in the file:
#include Most OpenMP constructs apply to a “structured block”

Structured Block: A block of one or more statements surrounded by “{ }”, with one point of entry at the top and one point of exit at the bottom.

Objectives:

To study and implementation of directive based parallel programming model

Input

Array of numbers taken from user to calculate Average, Sum, Max and Min

Output:

We will understand the implementation of sequential program augmented with compiler directives to specify parallelism

Conclusion:

We have implemented parallel reduction using Min, Max, Sum and Average Operations.

Program forAverage:

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

__global__ void avg(int* input) // kernel function definition
{       const int tid = threadIdx.x;
        int avg=0; int index=0;
        int step_size = 1 ;
        int number_of_threads = blockDim.x;   // blockDim = 4 i.e. number of
threads per block = 4 while
        (number_of_threads > 0)
        {
                if (tid < number_of_threads) // still alive? {

                        const int fst = tid * step_size * 2; index in array                    //get          the

                        const int snd = fst + step_size;                                       //get          the
index in array input[fst] += input[snd]; avg = input[fst];
                        index=fst;
                        //input[fst]=input[fst]/2; }



                step_size <<= 1;          // increment step_size by 1
                number_of_threads >>= 1;       //decrement number of threads by 2
        }
        input[index]=avg/7;      // calculate average
}
```

Program for Max:

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

int main() { double arr[10];
        omp_set_num_threads(4);
        double max_val=0.0; int i;
        for( i=0; i max_val) { max_val =
        arr[i];
        } printf("\nmax_val = %f", max_val);
}
```

Program for Min:

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

```
#include < stdio.h >

#include < stdlib.h >

int main() { double arr[10];
        omp_set_num_threads(4);
        double min_val=0.0; int i;
        for( i=0; i< min_val) {
            min_val = arr[i];
        } printf("\nmin_val = %f", min_val);
 }
```

Program for Sum:

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

int main (int argc, char *argv[]) { int i, n; float
a[100], b[100], sum; /* Some initializations */ n
= 100 ;
  for (i=0; i < n; i++) {
        a[i] = b[i] = i * 1.0 ; sum = 0.0 ;
        #pragma omp parallel for reduction(+:sum) for (i=0; i <
            n; i++) { sum = sum + (a[i] * b[i]);

                                    printf(" Sum = %f\n",sum);
}
```

Output: