A3-1102

Page No.
Date

Assignment:-

Title: Parallel Reduction

Problem Statement: Implement Min, Max, and Sum and Average operations using parallel reductions.

Objectives:

i) To implement min, max, sum and average operation using reduction
ii) To evaluate performance of operations mentioned above.

Outcomes:

Analysis / Results of performance of min, max, sum, avg operations implemented using parallel reduction.

Software / Hardware Requirements

4 GB RAM, 500 GB HDD, PC, Intel i-5
C++ programming, Ubuntu

Theory:

Parallel reduction refers to algorithms which combine an array of elements producing a single value as a result. Program eligible for this algorithm include those which involve operators that are associative and commutative in nature.

i) sum of array
ii) Min / Max of array.

Page No.
Date

let us understand with help of an example of finding min. (fig 1)

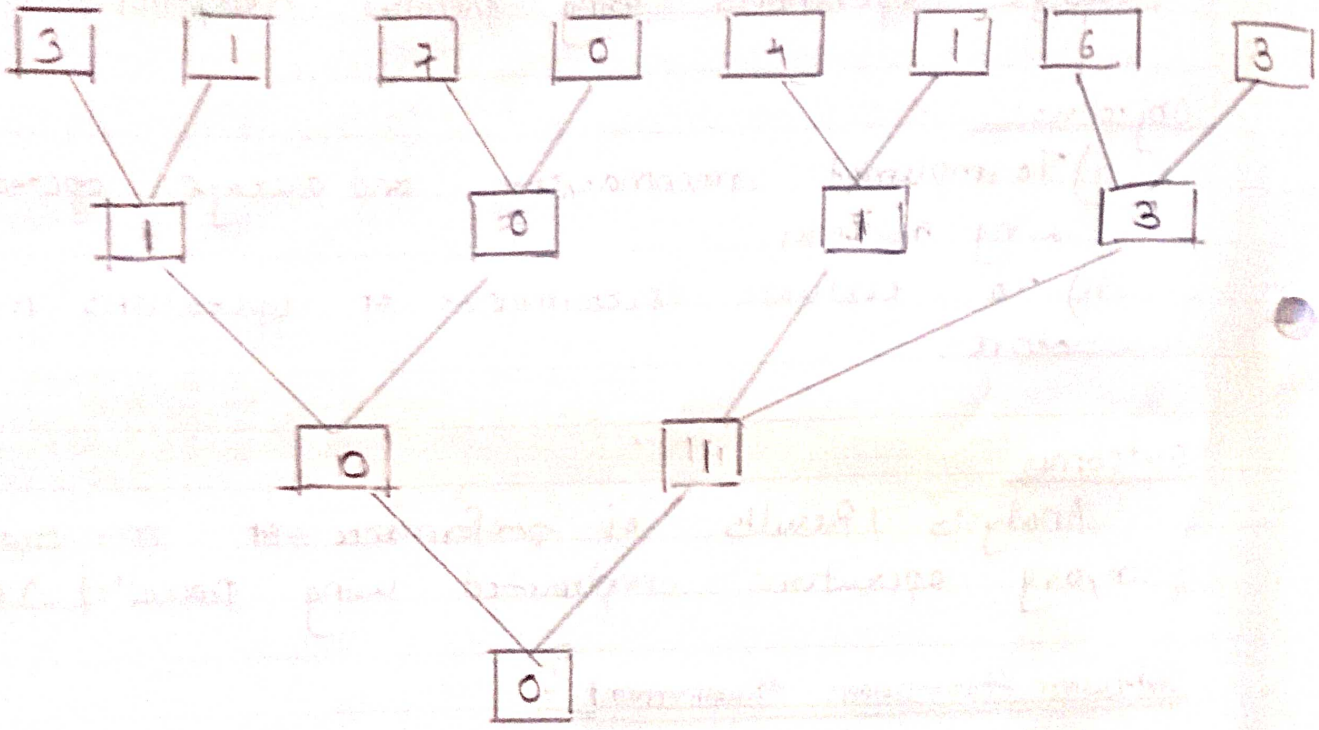
The idea is to use multiple thread blocks in GPU to reduce small portion of array. A tree based reduction is used inside each thread block and shared memory is used to achieve communication among threads of same block.

1. A small portion of array A1 is read from global memory by each thread and stored in shared memory.
2. The Array is reduced in shared memory. i.e. min value for chunk A_i is calculated
3. The min value is stored in global memory.
4. The final global minimum is then extracted from global memory.

OpenMp is API that supports shared memory parallel programming in C/C++. It provides directives that are easy to use with loops, sections, functions and other program constructs.

1) Syntax for parallel operation to find minimum:

```
int min_val = INT_MAX
#pragma omp parallel for reduction(min:min_val)
for ( 1 ——→ n )
{
    // logic to find min
}
```



~~fig 1~~: parallel reduction to find minimum element in an array.

Page No.
Date

Page No.
Date

ii) Similar, is the syntax for finding max, sum reduction

iii) The Average can be calculated from sum i.e. Avg = sum / no. of. elements in array.

Operations:

i) Min: returns the minimum from array of given numbers.

ii) Max: returns maximum from input array

iii) Sum: calculates the sum of each number in input array

iv) Average: Returns arithmetic mean of input array

Conclusion:

Thus, we successfully implemented Min, Max, Sum and Average operations using parallel reduction and evaluated their performances.

Code

```
//#include <iostream>
//#include <omp.h>
//#include <climits>
//#include <cstdlib>
//#include <chrono>
//#include <ctime>
#include<bits/stdc++.h>
using namespace std;


int min_reduction(int arr[], int n) {

  int min_value = INT_MAX;
  #pragma omp parallel for reduction(min: min_value)
  for (int i = 0; i < n; i++) {
            if (arr[i] < min_value) {
            min_value = arr[i];
            }
  }
  return min_value;
}

int max_reduction(int arr[], int n) {
  int max_value = INT_MIN;
  #pragma omp parallel for reduction(max: max_value)
  for (int i = 0; i < n; i++) {
            if (arr[i] > max_value) {
            max_value = arr[i];
```

```
            }
  }
  return max_value;
}

int sum_reduction(int arr[], int n) {
  int sum = 0;
   #pragma omp parallel for reduction(+: sum)
   for (int i = 0; i < n; i++) {
            sum += arr[i];
  }
  return sum;
}

int average_reduction(int arr[], int n) {
  int sum = 0;
  #pragma omp parallel for reduction(+: sum)
  for (int i = 0; i < n; i++) {
            sum += arr[i];
  }
  return sum/n;
}

int main() {
    int *arr,n;
    cout<<"\n enter total no of elements=>";
    cin>>n;

    int lb=0;
    int ub=n;
            arr=new int[n];
```

```
   for(int i=0;i<n;i++)
   {
          arr[i]=(rand() % (ub - lb + 1));
   }
          for(int i=0;i<n;i++)
   {
                    cout<<arr[i]<<" ";
   }
   cout<<endl;

   auto start = chrono::steady_clock::now();
          cout<<"min ="<< min_reduction(arr, n)<<endl;
   auto end = chrono::steady_clock::now();
          cout << "Elapsed time in microseconds: " <<
chrono::duration_cast<chrono::microseconds>(end - start).count()<< " microsecs" << endl;

          start = chrono::steady_clock::now();
                    cout<<"max ="<< max_reduction(arr, n)<<endl;
          end = chrono::steady_clock::now();
          cout << "Elapsed time in microseconds: " <<
chrono::duration_cast<chrono::microseconds>(end - start).count()<< " microsecs" << endl;

          start = chrono::steady_clock::now();
                    cout<<"sum ="<<sum_reduction(arr, n)<<endl;
          end = chrono::steady_clock::now();
          cout << "Elapsed time in microseconds: " <<
chrono::duration_cast<chrono::microseconds>(end - start).count()<< " microsecs" << endl;

          start = chrono::steady_clock::now();
```

```
        cout<<"avg ="<<average_reduction(arr, n)<<endl;

    end = chrono::steady_clock::now();

    cout << "Elapsed time in microseconds: " <<
chrono::duration_cast<chrono::microseconds>(end - start).count()<< " microsecs" << endl;


}
```

output: