

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.

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 A_i 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. This 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
}
```

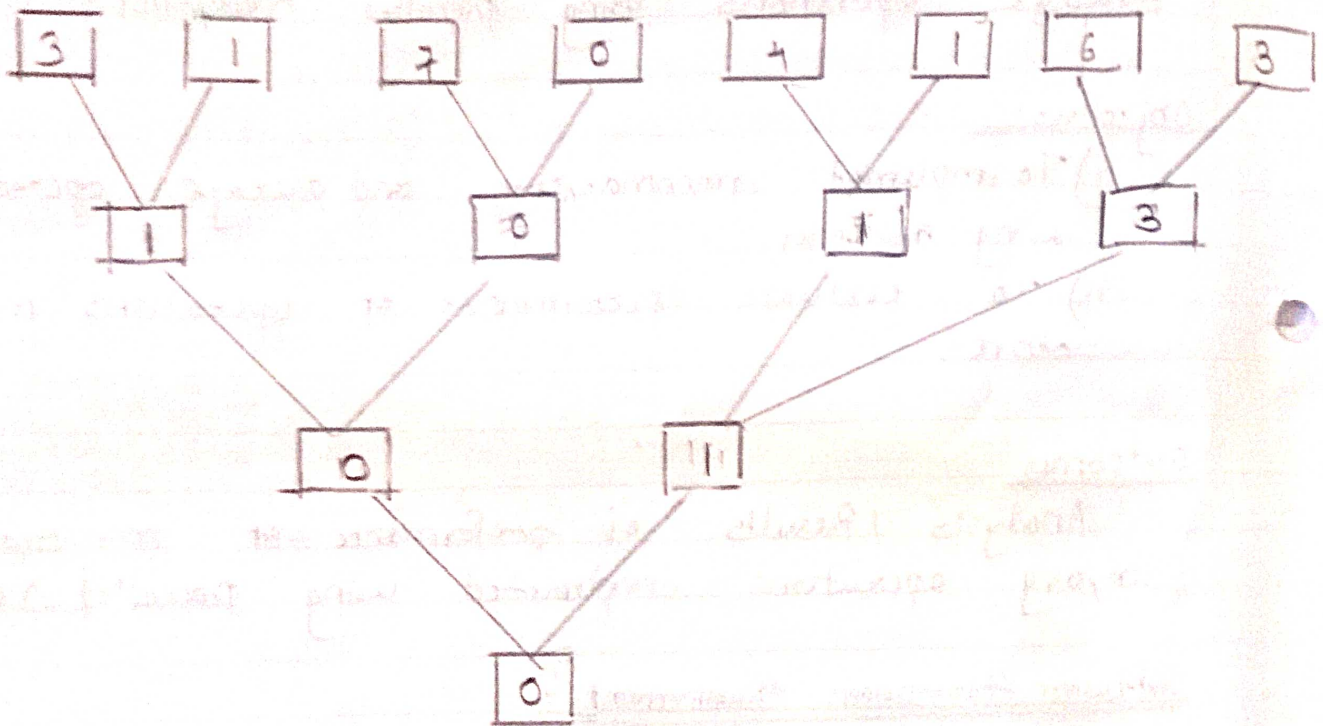


fig 1: parallel reduction to find minimum element in an array.

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

ii.) The Average can be calculated from sum i.e.
$$\text{Avg} = \text{sum} / \text{no. of. elements in array.}$$

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.

12

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:

```

70      cout<<endl;
71
72      C:\Users\DELL\Desktop\final year labs\p5\hpc\avg_min_max.exe
73
74      enter total no of elements=>200
75      41 176 103 169 74 46 21 12 28 143 77 5 166 144 112 89 181 83 3 9 30 132 83 153 91 121 135 23 20 197 20 18 98 81 60 13 14
76      0 169 151 45 161 93 167 183 186 41 66 73 113 34 20 107 100 195 138 196 80 93 193 175 55 20 80 33 87 157 196 98 169 136 1
77      0 87 11 96 49 74 132 147 72 131 132 161 165 55 63 182 142 140 62 178 60 78 104 67 110 31 38 72 15 59 9 161 16 111 192 8
78      1 108 95 116 75 197 2 191 178 39 0 44 96 72 30 67 24 116 144 44 44 138 97 11 193 14 145 57 16 97 13 153 41 118 81 181 26
79      15 132 71 65 49 110 101 96 191 22 170 166 132 23 96 29 69 47 124 128 87 35 41 51 58 21 185 29 40 83 128 166 174 102 12
80      84 171 163 40 111 53 190 87 200 66 170 37 65 110 19 126 172 119 9 39 185 100 67
81      min =0
82      Elapsed time in microseconds: 315 microsecs
83      max =200
84      Elapsed time in microseconds: 394 microsecs
85      sum =18794
86      Elapsed time in microseconds: 753 microsecs
87      avg =93
88      Elapsed time in microseconds: 414 microsecs
89
90      -----
91      Process exited after 2.607 seconds with return value 0
92      Press any key to continue . . .
93
94

```