Assignment No: 1

- Title : Design and implement parallel BFS and DFS based on existing algorithm using OpenMP. Use a tree or an undirected graph BFS & DFS

- BFS :

① To design and implement parallel BFS you will need to divide the graph into smaller sub graphs & assign each sub-graph to different processor or thread.

② Each process will then perform a BFS on its assigned sub-graph concurrently with othe processors.

③ Two methods : Vertex by vertex or level by level.

Parallel DFS :

- Different subtrees can be searched concurrently.
- Subtrees can be very different in size.
- Estimate the size of subtree rooted at a node
- Dynamic load balancing is required.

Parallel DFS :

- When a processor runs out of work, it gets more work from another processor.
- This is done using work request and response in shared address space machines.

- Load Balancing Schemes :
- Asynchronous round robin : Each processor maintains a counter and makes request in round robin fashion.
- Global round robin : The system maintain a global counter and request are made in a RR fashion

- Analyzing DFS :
- We can't compute, analytically, the serial work or parallel time
- for dynamic load balancing, idling time is subsumed by communication.
- Termination Detection : Processor P_0 has all work and weight of one is associated with it. When its work is partitioned and sent to another processor.
- If P_i the recipient processor and w_i is the weight at processor P_i, then after the first work transfer
- Termination is signaled when the weight w_0 at processor P_0 becomes one and processor P_0 has finished its work

- Conclusion :-

Designed and implemented parallel BFS & DFS based on existing using openMP.

Assignment no : 2

- Title : write a program to implement parallel bubble sort and Merge sort using openMP. use existing algorithms and measure the performance once of sequential and parallel algo.

- Theory :

Bubble Sort -

i) The complexity of bubble sort is $O(n^2)$

2) Bubble sort is difficult to parallelize since the algorithm has no concurrency

3) A simple variant, through, uncovers the concurrency sequential odd-even transposition sort algorithm.

ⓐ After n phases of odd-even exchanges.

ⓑ Each phase of algorithm required $O(n)$ comparisons

ⓒ Serial complexity is $O(n^2)$

Parallel odd-Even transposition :

ⓐ Consider the one item per processor case.

ⓑ There are n iterations, in each iteration, each processor does one compare - exchange.

ⓒ This is cost optimal with respect to base serial algorithm but not the optimal one.

Parallel formulation of odd-even transposition

ⓐ Consider a block of n/p element per processor

ⓑ The first step is a local sort

ⓒ In each subsequent step, the compare exchange operation is replaced by the compare split operation

Algorithm : [Parallel bubble sort]

① This program uses openMP to parallelize the bubble sort algorithm.

② The omp parallel for directive tells compiler to create a leam of threads to execute for loop.

③ The bubblesort function takes in an array and it sort it using the bubble sort algorithm. The outer loop iterates from 0 to n-i-1

④ The main function create a sample array & calls the bubblesort function to sort it.

⑤ It is worth nothing such that bubble sort is not an efficient sorting algorithm, specially for large inputs.

⑥ In this implementation, the bubble sort odd even function takes in an array & sort it using the odd_even transposition algorithm.
The router while loop continues untill the array is sorted
Each thread performs the swap operation in parallel.

⑦ The two #progma amp parallel for inside while loop one for even indexes and one for odd indexes, allows each thread to sort the even and odd indexed element simultaneously

- Parallel merge sort -
A odd an even are defined as the set of elements of a with odd & even indices resp.

- Similarly, let a set of element A odd= $\{a, \ldots a_3, a' \ldots\}$ and A even= $\{\sim, a_4, a_6, \ldots\}$ regarding a set of elements A= $\{a \ldots a_n\}$

Merge(A,B) $= \{a, b, \sim, b_2, a_3, b_3, \ldots, a_n, b_n\}$ for ex
if $A = \{1,2,3,4\}$ and $B = \{5,6,7,8\}$
then Merge $(\{1,2,3,4\}, \{5,6,7,8\}) = (1,5,2,6,3,7,4,8)$
Join(A,B) = {Merge(A,B) , odd - even(A,B)}

Algorithm : odd-Even (A,B,S)

```
begin
if A and B are of length 1
    then
Merge A & B using one compare and exchange operation
else
begin
compute Sodd and Seven IN parallel do
Sodd = Merge (Aodd, Bodd)
Seven = Merge (Aeven, Beven)
Sodd - Seven = Join (Sodd, Seven)
end
end if
end
```

Conclusion :-

Hence, we learn how sequential and parallel algorithm works.

Assignment no : 3

Title : Implement min, max, sum and average operation using parallel reduction

Implementation :-

① The min-reduction function finds the minimum value in the input array using # pragma omp parallel for reduction direction and divides loop iteration among the available thread.
Each thread performs the comparison operation in parallel and update the min-value variable if smaller value is found.

② Similarly, the max-reduction function takes maximum value in the array, sum-reduction function finds the sum of elemen of array and average-reduction function finds the average of the element of array by dividing the sum by the size of array.

③ The reduction clause is used to combine the results of multiple threads into a signal value, which is then returned by the function. The min & max operators are used for the min-reduction & max-reduction function resp. and the +operator is used for sum reduction and average-reduction function.
In the main function, it creates a vector and calls the

functions min-reduction, max-reduction, sum-reduction and average-reduction to compute the values of min, max, sum & average resp.

Conclusion :-

The implementation of min, max, sum and average operations using parallel reduction not only showcases the benifits of parallelization in optimizing common mathematical computation

Assignment no : 4

- Title : Implement HPC application for AI or ML domain.

- Theory :- Parallel computing for AI or ML

- Parallel Computing is a computing architecture that enables multiple processors to perform computation simultaneously, allowing for faster processing times and improved performed. In the field of AI and ML, parallel computing can be used to speed up the training of complex models make faster prediction and process large amount of data.

    In AI and ML, parallel computing is typically achieved by distributing the computation across multiple GPU or CPU cores.

    Another area where parallel computing is used in AI and ML is in data processing. In many AI and ML applications large amount of data need to be processed.

    Parallel computing can also be used to speed up the prediction process in AI and ML.

- Implementation :

    Here is an example program in python at implements parallel programming for data processing in the AI/ML domain using multiprocessing module.

- Algorithm :

In this program, we first define a function process_data that represent our data processing logic. This function take chunk of data as input, processes it & processed data.

We then create a pool of worker processes using the multiprocessing pool class. The number of processes is set to the number of available CPU cores using the multiprogessing, CPU_count() function.

Finally, we concatenate the processed chunks into a single list and print the first elements of the processed data.

Note that this is a simplified example and in practice, you may need to consider additional factor such as data communication, load balancing and synchronization for data processing in AI/ML domain.

- Conclusion :

In conclusion, parallel computing is key technology for improving performance for AI/ML application. The use of parallel computing in AI/ML is leading to significant advancement in these field making it possible to develop more complex and sophisticated model.