Assignment NO-1(A)

Title :- Design & implement parallel breadth first search based on existing Algorithms using OpenMP. Use a Tree or an undirected graph for BFS.

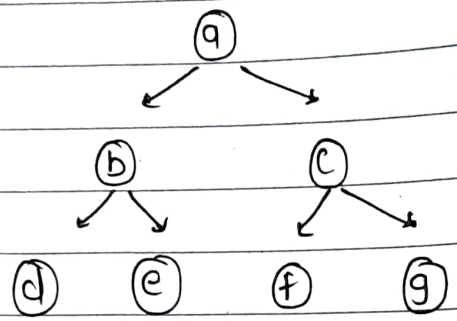
Objective of assignment :- students should be able to perform parallel Breadth first search based on existing algorithms using openMP.

Prerequisite :-
① Basic of programming language
② concept of BFS
③ Concept of parallelism.

What is BFS ?
BFS stands for breadth first search. It is a graph traversal Algo. used to explore all nodes of a graph or tree systematically, starting from root node or a specified starting pt & visiting all neighboring nodes at current depth level before moving on to next depth level.
BFS is Commonly used in many Appl^n, such as finding shortest path bet^n 2 nodes, solving puzzles & searching through a tree or graph

Example of BFS -
① take an empty queue.
② select a starting node & insert it into Queue.

③ provided that queue is not empty
④ print extracted node.



Concept of OpenMp -

OpenMP is an appl^n programming interface that supports shared-memory -llel programming in C, C++, & fortran.

It is used to writ -llel programs that can run on multicore processor, multiprocessor systems & -llel computing clusters.

OpenMP provides a set of directives & functions that can be inserted into source code of a program to parallelize its execution.

Conclusion :-

In this way we can achieve parallelism while implementing BFS.

Assignment No :- 1 (B)

Title :- Design & implement parallel Depth first search based on exisisting Algo. using openMP use a Tree or an undirected graph for DFS.

objectives :- students should be able to perform parallel Depth first search based on existing Algorithms using openMP.

Prerequisite :-
1. Basic of programming language
2. concept of DFS
3. concept of parallelism

What is DFS ?

DFS stands for Depth first search. It is a popular graph traversal Algorithm that explores as far as possible along each branch before backtracking. This algo. can be used to find the shortest path bet^n 2 Vertices or to traverse a graph in Systematic way.

The Algo. starts at root node & explores as far as possible along each branch before backtracking. DFS can be implemented using either a recursive or an interactive approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs.

The interactive approach uses a stack to keep track of nodes to be explored & is preferred for larger graphs.
DFS can also be used to detect cycles in graph.
If a cycle exists in a graph, the DFS Algorithm will eventually reach a node that has already been visited, indicated that a cycle exists.
A standard DFS implementation puts each vertex of the graph into one of 2 categories :-

① visited
② Non-visited

Conclusion :-
In this way we can achieve parallelism while implementing DFS.

Assignment NO:- 2(A)

Title :- Write a program to implement parallel Bubble sort. Use existing Algorithms & measure the performance of sequential & Parallel Algorithms.

objective of Assignment :- students should be able to write a program to implement parallel bubble sort & Can measure the performance of sequential & parallel Algorithms.

Pre requisite :-
1. Basic of programming language
2. concept of Bubble sort
3. concept of parallelism

What is Bubble sort ?
Buble Sort is a simple sorting Algo. that works by repeatedly swapping adjacent elements if they are in wrong order. It is called "bubble" sort because the Algo moves larger elements towards the end of array in a manner that resembles rising of bubbles in a liquid.

The basic Algo. of Bubble sort is as follows :-
1. start at the beginning of Array
2. Compare 1st 2 elements, if 1st element is greater than 2nd element is greater than 2nd element swap them

5. Move to next pair of elements & repeat step 2
6. Continue the process until end of array is reached
7. If any swaps were made in step 2-4, repeat the process from step 1

Example of Bubble sort :-

Let's say we want to sort a series of nos. 5, 3, 4, 1 & 2 so that they are arranged in ascending order

The sorting begins the first iteration by Comparing 1st 2 values If 1st Value is greater than second, the Algorithm pushes 1st value to index of second value

Conclusion :-

In this way we can implement Bubble Sort in parallel way using OpenMP also come to know to how to measure performance of Serial & parallel Algorithm

Assignment NO :- 2 (B)

Title :- Write a program to implement parallel merge sort. use existing Algorithms & measure the performance of sequential & IIel Algo.

Objective :- Students should be able to write a program to implement parallel merge sort & can measure the performance of sequential & parallel Algorithms.

Prerequisite :-
1. Basic of programming language.
2. Concept of merge sort
3. Concept of parallelism

What is Merge sort ?
Merge sort is a sorting Algorithm that uses a divide & conquer Approach to sort an array or a list of elements.
the Algorithm works by recursively dividing the i/p array into 2 halves, sorting each half, & then merging sorted halves to produce a sorted o/p.

How to measure the performance of sequential & Parallel Algorithms ?

There are several metrices that can be used to measure the performance of sequential & parallel merge sort Algorithms :-

① Execution Time :-
Execution time is the amount of time it takes by the Algorithm to complete its sorting operation. This metric can be used to compare the speed of sequential & parallel merge sort Algorithms.

② Speedup :-
speedup is the ratio of execution time of sequential merge sort Algo to execution time of parallel merge sort Algo.

③ Efficiency :-
Efficiency is the ratio of speedup to no. of processors or cores used in parallel Algorithm.

④ scalability :-
Scalability is ability of Algorithm to maintain its performance as i/p size & no. of processors or cores increase.

Conclusion :-
In this way we can implement merge sort in parallel way using openMP also come to know how to measure performance of serial & parallel Algorithm.

Assignment NO:-3

Title :- Implement min, max, sum & Average operation using parallel reduction.

Objective :- To Understand the concept of parallel reduction & how it can be used to perform basic mathematical operations on given data sets.

Prerequisite :-
1. parallel Computing Architectures
2. parallel programming models
3. Proficiency in programming languages.

Parallel Reduction :-
Here's function-wise manual on how to understand & run the sample c++ program that demonstrates how to implement min, max, sum & Average operations using parallel reduction.

1. Min-Reduction function
The function takes in a vector of integers as i/p & finds the min value in the vection using ||el reduction.
The min value found by each thread is reduced to overall min. value of entire array.

② Max - Reduction function
The max. value found by each thread is reduced to the overall max. value of the entire array.

③ sum - Reduction function.
The OpenMP reduction clause is used with "+" operator to find the sum across all threads. The sum found by each thread is reduced to overall sum of entire array.

④ Average - Reduction function.
The sum found by each thread is reduced to overall sum of entire array. The final sum is divided by the size of array to find the average.

⑤ Main function:-
The function calls min-reduction, max-reduction, sum-reduction & average-reduction functions on the i/p vector to find corresponding values.

Conclusion :- We have implemented min, max, sum & Average operations using 11ed reduction in c++ with openMP, 11ed reduction is a powerful technique that allows us to perform these operations on large arrays more efficiently by dividing the work among multiple threads running in 11el. We presented a code ex. that demonstrates implementation of these operations using 11el redn in c++ with openMP. we also provided a manual for running openMP programs on the ubuntu platform.

Assignment No :- 4

Title :- Write a CUDA program for :-
① Addition of two large vectors
② matrix multiplication using CUDAC.

Objective :- students should be able to write a program to implement addition of 2 vectors & matrix multiplication using CUDAC.

Theory :- Addition OF 2 Vectors :
- In this program, the 'addvectors' kernal takes in the two i/P vectors 'A' & 'B' the o/p vector 'c', & size OF vectors 'n'.
- The kernel uses the 'blockIdx.x' & 'threadIdx.x' variables to calculate the index 'i' of current thread. If the index is less than 'n', the kernel performs the addition operation 'C[i]=A[i]+B[i]'.
- In the main function, the program first allocates memory for the i/P & o/P vectors on the host & initializes them. Then it allocates memory for the vectors on the device & copies the data from host to the device using 'cudamemcpy'.

Matrix-matrix multiplication

Consider 2 n×n matrices A & B partitioned into P blocks Ai,j & Bi,j ($0 \le i, j < \sqrt{P}$) OF size each.
$(n/\sqrt{P}) \times (n/\sqrt{P})$

Process Pij initially stores Aij & Bij & computes block Cij of result matrix.

Computing submatrix Cij requires all submetrices Aijk & Bkj for $0 \leq k < \sqrt{P}$.

All-to-all broadcast blocks of A along rows & B along columns.

perform local submatrix multiplication.

In the main function, the program first allocates memory for i/p & o/p matrices on host & initializes them.

Next, program launches the kernel with the appropriate grid & block dimensions. The kernel uses a 2D grid of thread blocks to perform the matrix multiplication in parallel.

Finally, it copies data from device to host using cudamemcpy & prints the result using nested for loop. And it also frees the memory used.

Conclusion :-
In this way, we have implemented vector addition & matrix multiplication using CUDAC