

Assignment – 1

Title: Implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP.

Problem Statement: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.

Objectives: Students should be able to perform Parallel Breadth First Search based on existing algorithms using OpenMP.

Theory:

BFS: BFS stands for Breadth-First Search. It is a graph traversal algorithm used to explore all the nodes of a graph or tree systematically, starting from the root node or a specified starting point, and visiting all the neighbouring nodes at the current depth level before moving on to the next depth level. The algorithm uses a queue data structure to keep track of the nodes that need to be visited, and marks each visited node to avoid processing it again. The basic idea of the BFS algorithm is to visit all the nodes at a given level before moving on to the next level, which ensures that all the nodes are visited in breadth-first order. BFS is commonly used in many applications, such as finding the shortest path between two nodes, solving puzzles, and searching through a tree or graph.

Example of BFS:

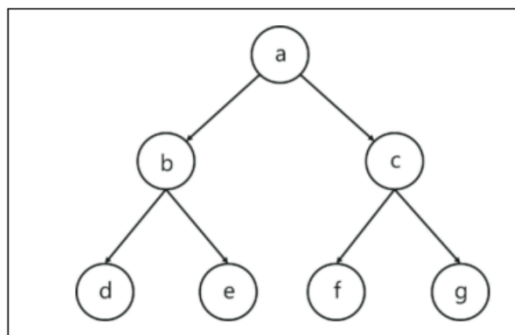
Now let us look at the steps involved in traversing a graph by using Breadth-First Search:

Step 1: Take an Empty Queue.

Step 2: Select a starting node (visiting a node) and insert it into the Queue.

Step 3: Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue.

Step 4: Print the extracted node.



Example of DFS:

To implement DFS traversal, you need to take the following stages.

Step 1: Create a stack with the total number of vertices in the graph as the size.

Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.

Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

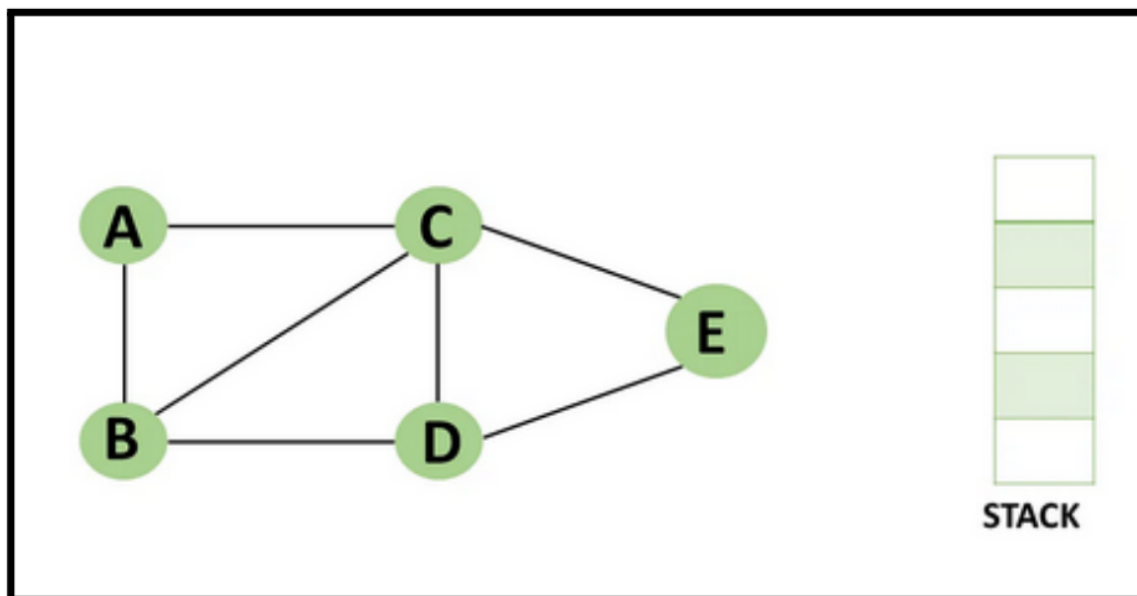
Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.

Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.

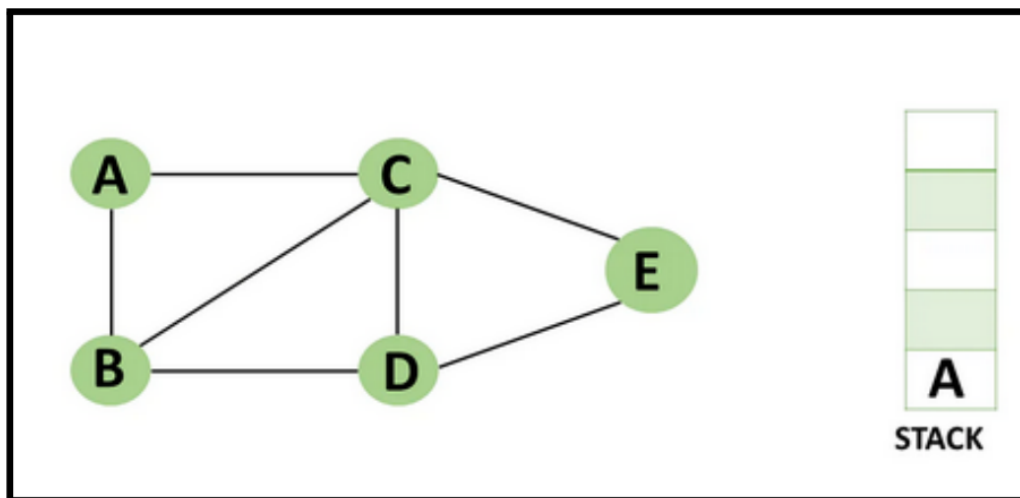
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

Consider the following graph as an example of how to use the DFS algorithm.



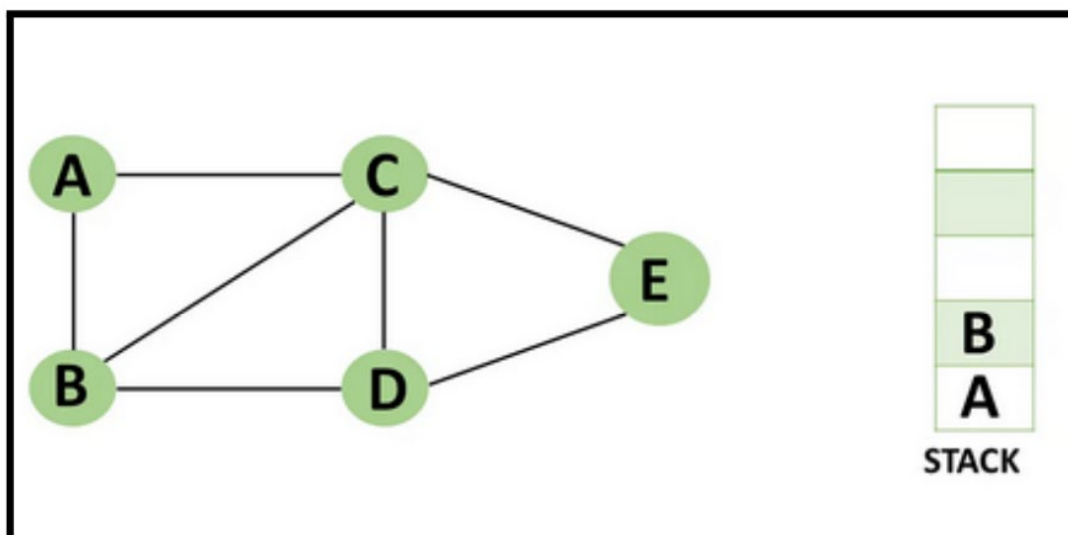
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

- You should push vertex A to the top of the stack.



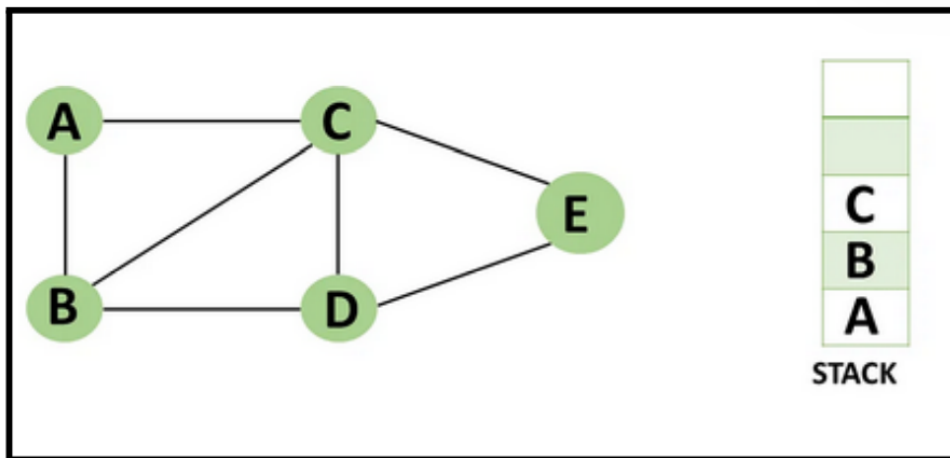
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

- You should push vertex B to the top of the stack.



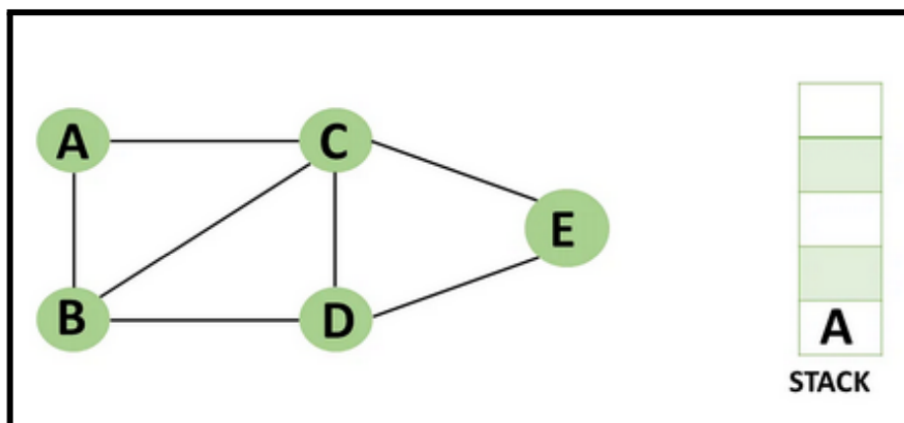
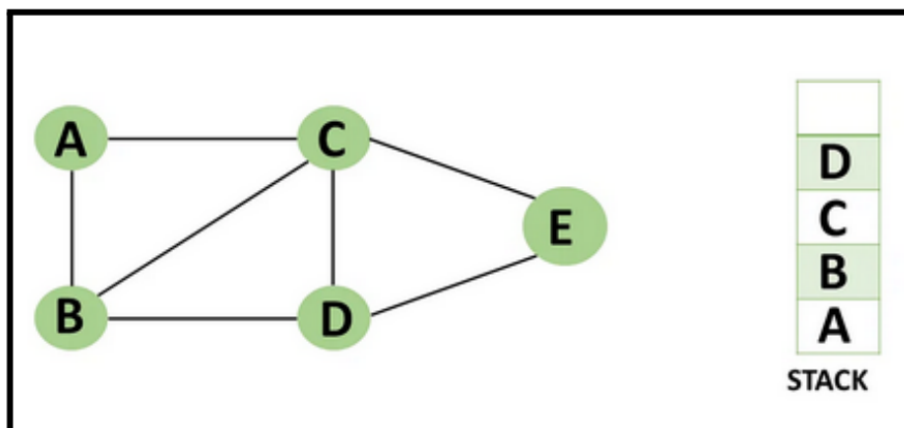
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

- Vertex C is pushed to the top of the stack.



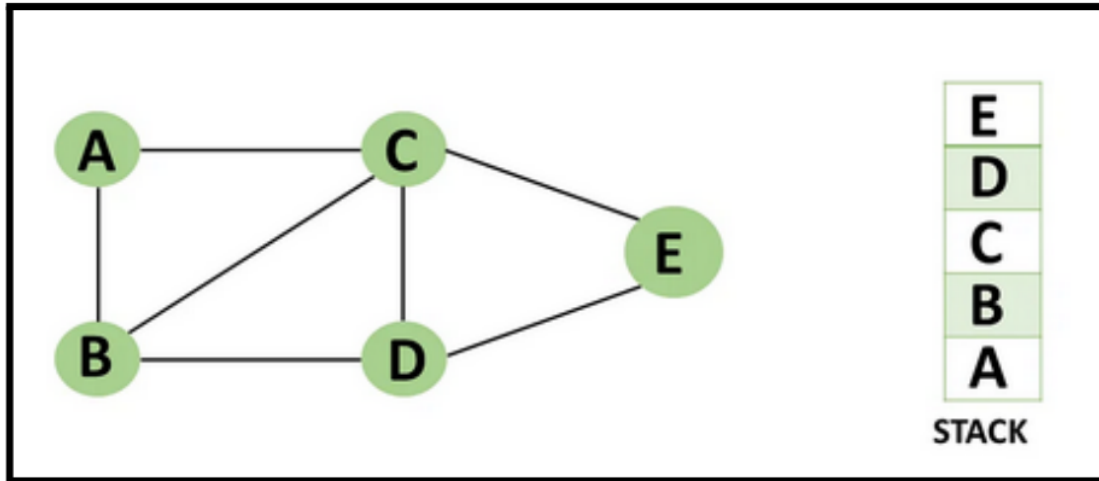
Step 4: You can visit any nearby unvisited vertices of vertex C; you need to select vertex D and designate it as a visited vertex.

- Vertex D is pushed to the top of the stack.

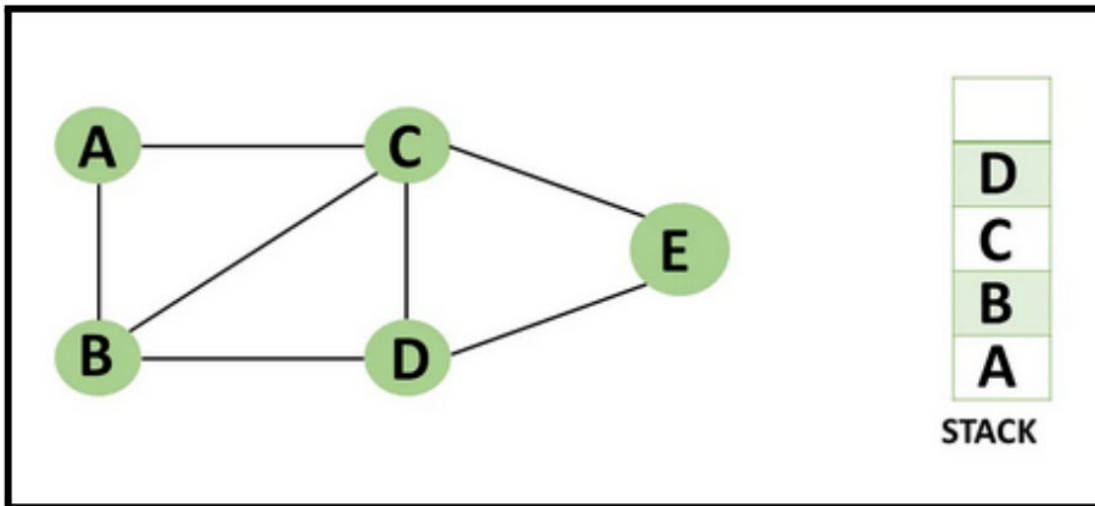


Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

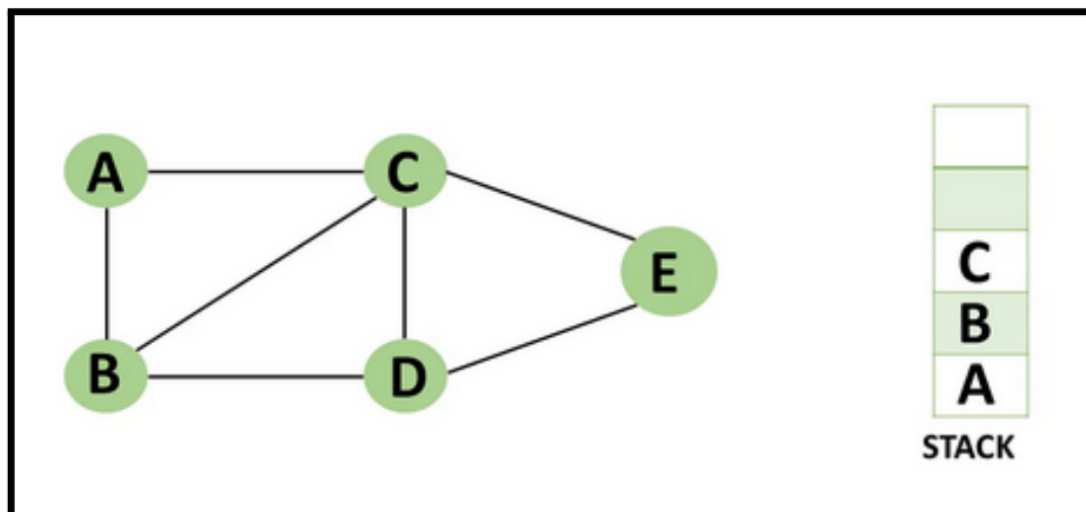
- Vertex E should be pushed to the top of the stack.



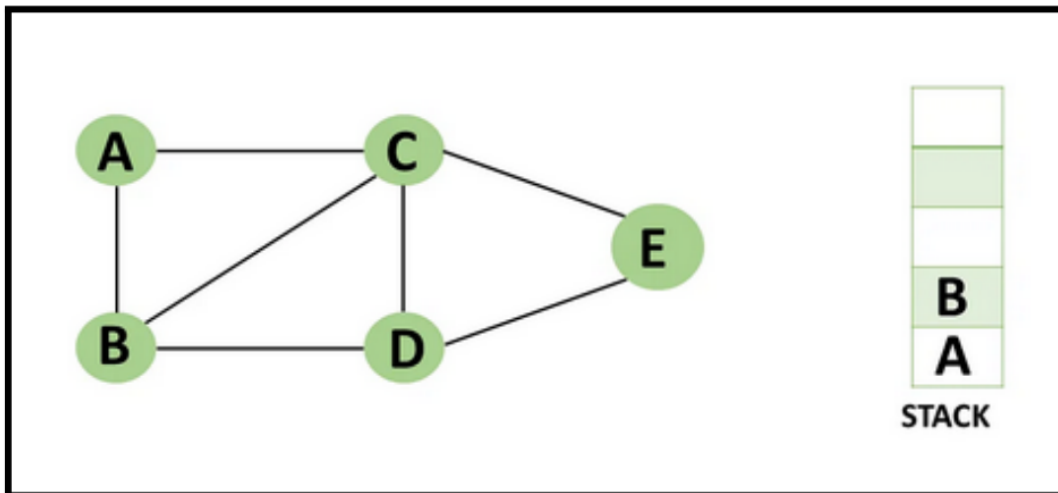
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.



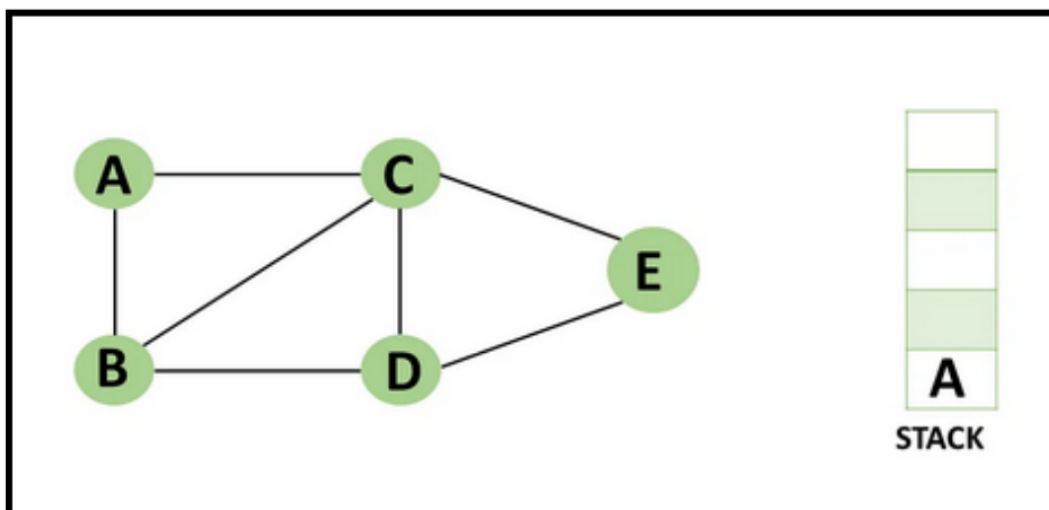
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.



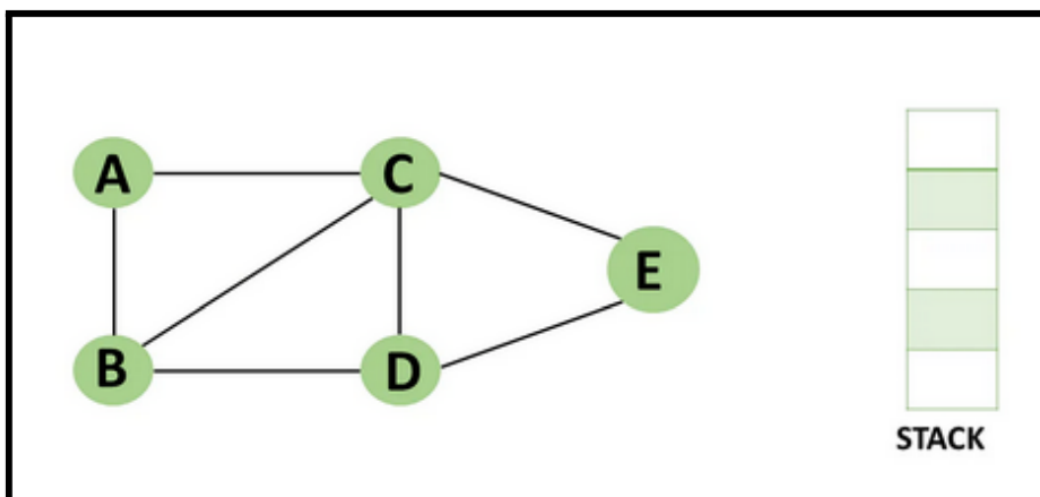
Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.



Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.



Step 10: All the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.



Algorithm:**Parallel Breadth First Search (BFS):****Initialize:**

- Create a queue data structure to store nodes to be visited.
- Mark all nodes as not visited.
- Enqueue the starting node into the queue and mark it as visited.

Parallel Exploration:**While the queue is not empty:**

- Use OpenMP parallel constructs to parallelize the exploration of nodes at each level of the BFS tree.
- Each thread dequeues a node from the queue and explores its neighbours in parallel.
- Mark each neighbour node as visited and enqueue it into the queue if it has not been visited before.

Terminate:

- When all nodes have been visited, terminate the BFS algorithm.

Parallel Depth First Search (DFS):**Initialize:**

- Create a stack data structure to store nodes to be visited.
- Mark all nodes as not visited.
- Push the starting node onto the stack and mark it as visited.

Parallel Exploration:**While the stack is not empty:**

- Use OpenMP parallel constructs to parallelize the exploration of nodes at each level of the DFS tree.
- Each thread pops a node from the stack and explores its neighbours in parallel.
- Mark each neighbour's node as visited and push it onto the stack if it has not been visited before.

Terminate:

- When all nodes have been visited, terminate the DFS algorithm.

Concept of OpenMP:

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.

- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.
- OpenMP is widely used in scientific computing, engineering, and other fields that require high-performance computing. It is supported by most modern compilers and is available on a wide range of platforms, including desktops, servers, and supercomputers.

Conclusion:

In this way we have implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP.

CODE:

```
#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

using namespace std;

// Graph class representing the adjacency list
class Graph {
    int V; // Number of vertices
    vector<vector<int>>> adj; // Adjacency list

public:
    Graph(int V) : V(V), adj(V) {}

    // Add an edge to the graph
    void addEdge(int v, int w) {
        adj[v].push_back(w);
    }

    // Parallel Depth-First Search
    void parallelDFS(int startVertex) {
        vector<bool> visited(V, false);
        parallelDFSUtil(startVertex, visited);
    }

    // Parallel DFS utility function
    void parallelDFSUtil(int v, vector<bool>& visited) {
        visited[v] = true;
        cout << v << " ";
```

```

#pragma omp parallel for
for (int i = 0; i < adj[v].size(); ++i) {
    int n = adj[v][i];
    if (!visited[n])
        parallelDFSUtil(n, visited);
}
}

// Parallel Breadth-First Search
void parallelBFS(int startVertex) {
    vector<bool> visited(V, false);
    queue<int> q;

    visited[startVertex] = true;
    q.push(startVertex);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";

#pragma omp parallel for
for (int i = 0; i < adj[v].size(); ++i) {
    int n = adj[v][i];
    if (!visited[n]) {
        visited[n] = true;
        q.push(n);
    }
}
}
}

```

```

    }
};

int main() {
    // Create a graph
    Graph g(7);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 6);

    /*
        0 ----->1
        |      /\
        |      /  \
        |      /   \
        v      v    v
        2 ----> 3    4
        |    |
        |    |
        v    v
        5    6
    */

    cout << "Depth-First Search (DFS): ";
    g.parallelDFS(0);
    cout << endl;

    cout << "Breadth-First Search (BFS): ";

```

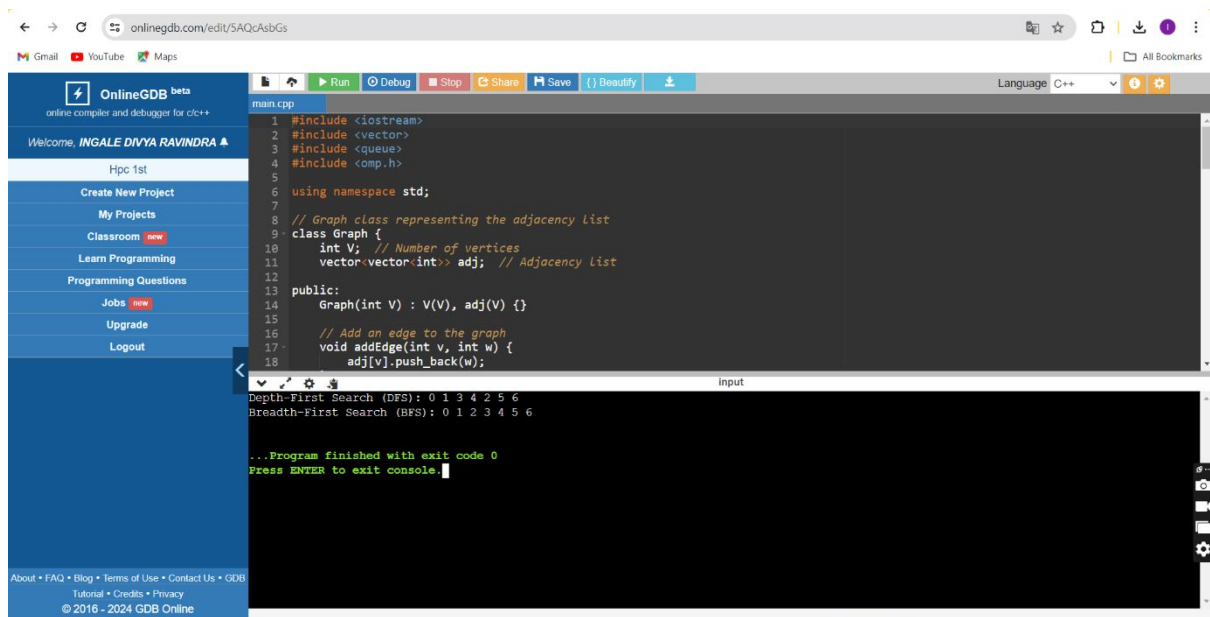
```
g.parallelBFS(0);
```

```
cout << endl;
```

```
return 0;
```

```
}
```

OUTPUT:



The screenshot shows the OnlineGDB interface. The code in `main.cpp` defines a `Graph` class with an adjacency list and a `parallelBFS` method. The output shows the results of a Depth-First Search (DFS) and a Breadth-First Search (BFS) on a graph with 6 vertices.

```
1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 #include <omp.h>
5
6 using namespace std;
7
8 // Graph class representing the adjacency list
9 class Graph {
10     int V; // Number of vertices
11     vector<vector<int>> adj; // Adjacency list
12
13 public:
14     Graph(int V) : V(V), adj(V) {}
15
16     // Add an edge to the graph
17     void addEdge(int v, int w) {
18         adj[v].push_back(w);
19     }
20
21     void parallelBFS(int s) {
22         vector<int> visited(V, 0);
23         queue<int> q;
24         q.push(s);
25         visited[s] = 1;
26         while (!q.empty()) {
27             int u = q.front(); q.pop();
28             for (int v : adj[u]) {
29                 if (!visited[v]) {
30                     visited[v] = 1;
31                     q.push(v);
32                 }
33             }
34         }
35     }
36
37     void DFSUtil(int v, bool visited[]) {
38         visited[v] = true;
39         cout << v << " ";
40         for (int i : adj[v]) {
41             if (!visited[i]) {
42                 DFSUtil(i, visited);
43             }
44         }
45     }
46
47     void DFS() {
48         bool visited[V];
49         for (int i = 0; i < V; i++) {
50             if (!visited[i]) {
51                 DFSUtil(i, visited);
52             }
53         }
54     }
55
56     void BFS() {
57         bool visited[V];
58         for (int i = 0; i < V; i++) {
59             visited[i] = false;
60         }
61         parallelBFS(0);
62     }
63
64     void printAdjList() {
65         for (int i = 0; i < V; i++) {
66             cout << i << " : ";
67             for (int j : adj[i]) {
68                 cout << j << " ";
69             }
70             cout << endl;
71         }
72     }
73
74     void printVisited() {
75         for (int i = 0; i < V; i++) {
76             cout << i << " : " << visited[i] << " ";
77             if (i % 5 == 4) {
78                 cout << endl;
79             }
80         }
81     }
82
83     void printVisitedDFS() {
84         for (int i = 0; i < V; i++) {
85             cout << i << " : " << visited[i] << " ";
86             if (i % 5 == 4) {
87                 cout << endl;
88             }
89         }
90     }
91
92     void printVisitedBFS() {
93         for (int i = 0; i < V; i++) {
94             cout << i << " : " << visited[i] << " ";
95             if (i % 5 == 4) {
96                 cout << endl;
97             }
98         }
99     }
100
101     void printVisitedDFSAndBFS() {
102         DFS();
103         BFS();
104     }
105
106     void printVisitedDFSAndBFSAndAdjList() {
107         DFS();
108         BFS();
109         printAdjList();
110     }
111
112     void printVisitedDFSAndBFSAndAdjListAndVisited() {
113         DFS();
114         BFS();
115         printAdjList();
116         printVisited();
117         printVisitedDFS();
118         printVisitedBFS();
119     }
120
121     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFS() {
122         DFS();
123         BFS();
124         printAdjList();
125         printVisited();
126         printVisitedDFS();
127         printVisitedBFS();
128         DFS();
129         BFS();
130     }
131
132     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjList() {
133         DFS();
134         BFS();
135         printAdjList();
136         printVisited();
137         printVisitedDFS();
138         printVisitedBFS();
139         DFS();
140         BFS();
141         printAdjList();
142     }
143
144     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisited() {
145         DFS();
146         BFS();
147         printAdjList();
148         printVisited();
149         printVisitedDFS();
150         printVisitedBFS();
151         DFS();
152         BFS();
153         printAdjList();
154         printVisited();
155         printVisitedDFS();
156         printVisitedBFS();
157     }
158
159     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFS() {
160         DFS();
161         BFS();
162         printAdjList();
163         printVisited();
164         printVisitedDFS();
165         printVisitedBFS();
166         DFS();
167         BFS();
168         printAdjList();
169         printVisited();
170         printVisitedDFS();
171         printVisitedBFS();
172         DFS();
173         BFS();
174         printAdjList();
175         printVisited();
176         printVisitedDFS();
177         printVisitedBFS();
178     }
179
180     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjList() {
181         DFS();
182         BFS();
183         printAdjList();
184         printVisited();
185         printVisitedDFS();
186         printVisitedBFS();
187         DFS();
188         BFS();
189         printAdjList();
190         printVisited();
191         printVisitedDFS();
192         printVisitedBFS();
193         DFS();
194         BFS();
195         printAdjList();
196         printVisited();
197         printVisitedDFS();
198         printVisitedBFS();
199         DFS();
200         BFS();
201         printAdjList();
202         printVisited();
203         printVisitedDFS();
204         printVisitedBFS();
205     }
206
207     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisited() {
208         DFS();
209         BFS();
210         printAdjList();
211         printVisited();
212         printVisitedDFS();
213         printVisitedBFS();
214         DFS();
215         BFS();
216         printAdjList();
217         printVisited();
218         printVisitedDFS();
219         printVisitedBFS();
220         DFS();
221         BFS();
222         printAdjList();
223         printVisited();
224         printVisitedDFS();
225         printVisitedBFS();
226         DFS();
227         BFS();
228         printAdjList();
229         printVisited();
230         printVisitedDFS();
231         printVisitedBFS();
232         DFS();
233         BFS();
234         printAdjList();
235         printVisited();
236         printVisitedDFS();
237         printVisitedBFS();
238     }
239
240     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFS() {
241         DFS();
242         BFS();
243         printAdjList();
244         printVisited();
245         printVisitedDFS();
246         printVisitedBFS();
247         DFS();
248         BFS();
249         printAdjList();
250         printVisited();
251         printVisitedDFS();
252         printVisitedBFS();
253         DFS();
254         BFS();
255         printAdjList();
256         printVisited();
257         printVisitedDFS();
258         printVisitedBFS();
259         DFS();
260         BFS();
261         printAdjList();
262         printVisited();
263         printVisitedDFS();
264         printVisitedBFS();
265         DFS();
266         BFS();
267         printAdjList();
268         printVisited();
269         printVisitedDFS();
270         printVisitedBFS();
271         DFS();
272         BFS();
273         printAdjList();
274         printVisited();
275         printVisitedDFS();
276         printVisitedBFS();
277     }
278
279     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjList() {
280         DFS();
281         BFS();
282         printAdjList();
283         printVisited();
284         printVisitedDFS();
285         printVisitedBFS();
286         DFS();
287         BFS();
288         printAdjList();
289         printVisited();
290         printVisitedDFS();
291         printVisitedBFS();
292         DFS();
293         BFS();
294         printAdjList();
295         printVisited();
296         printVisitedDFS();
297         printVisitedBFS();
298         DFS();
299         BFS();
300         printAdjList();
301         printVisited();
302         printVisitedDFS();
303         printVisitedBFS();
304         DFS();
305         BFS();
306         printAdjList();
307         printVisited();
308         printVisitedDFS();
309         printVisitedBFS();
310         DFS();
311         BFS();
312         printAdjList();
313         printVisited();
314         printVisitedDFS();
315         printVisitedBFS();
316         DFS();
317         BFS();
318         printAdjList();
319         printVisited();
320         printVisitedDFS();
321         printVisitedBFS();
322         DFS();
323         BFS();
324         printAdjList();
325         printVisited();
326         printVisitedDFS();
327         printVisitedBFS();
328     }
329
330     void printVisitedDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisitedAndDFSAndBFSAndAdjListAndVisited() {
331         DFS();
332         BFS();
333         printAdjList();
334         printVisited();
335         printVisitedDFS();
336         printVisitedBFS();
337         DFS();
338         BFS();
339         printAdjList();
340         printVisited();
341         printVisitedDFS();
342         printVisitedBFS();
343         DFS();
344         BFS();
345         printAdjList();
346         printVisited();
347         printVisitedDFS();
348         printVisitedBFS();
349         DFS();
350         BFS();
351         printAdjList();
352         printVisited();
353         printVisitedDFS();
354         printVisitedBFS();
355         DFS();
356         BFS();
357         printAdjList();
358         printVisited();
359         printVisitedDFS();
360         printVisitedBFS();
361         DFS();
362         BFS();
363         printAdjList();
364         printVisited();
365         printVisitedDFS();
366         printVisitedBFS();
367         DFS();
368         BFS();
369         printAdjList();
370         printVisited();
371         printVisitedDFS();
372         printVisitedBFS();
373         DFS();
374         BFS();
375         printAdjList();
376         printVisited();
377         printVisitedDFS();
378         printVisitedBFS();
379         DFS();
380         BFS();
381         printAdjList();
382         printVisited();
383         printVisitedDFS();
384         printVisitedBFS();
385         DFS();
386         BFS();
387         printAdjList();
388         printVisited();
389         printVisitedDFS();
390         printVisitedBFS();
391         DFS();
392         BFS();
393         printAdjList();
394         printVisited();
395         printVisitedDFS();
396         printVisitedBFS();
397         DFS();
398         BFS();
399         printAdjList();
400         printVisited();
401         printVisitedDFS();
402         printVisitedBFS();
403         DFS();
404         BFS();
405         printAdjList();
406         printVisited();
407         printVisitedDFS();
408         printVisitedBFS();
409         DFS();
410         BFS();
411         printAdjList();
412         printVisited();
413         printVisitedDFS();
414         printVisitedBFS();
415         DFS();
416         BFS();
417         printAdjList();
418         printVisited();
419         printVisitedDFS();
420         printVisitedBFS();
421         DFS();
422         BFS();
423         printAdjList();
424         printVisited();
425         printVisitedDFS();
426         printVisitedBFS();
427         DFS();
428         BFS();
429         printAdjList();
430         printVisited();
431         printVisitedDFS();
432         printVisitedBFS();
433         DFS();
434         BFS();
435         printAdjList();
436         printVisited();
437         printVisitedDFS();
438         printVisitedBFS();
439         DFS();
440         BFS();
441         printAdjList();
442         printVisited();
443         printVisitedDFS();
444         printVisitedBFS();
445         DFS();
446         BFS();
447         printAdjList();
448         printVisited();
449         printVisitedDFS();
450         printVisitedBFS();
451         DFS();
452         BFS();
453         printAdjList();
454         printVisited();
455         printVisitedDFS();
456         printVisitedBFS();
457         DFS();
458         BFS();
459         printAdjList();
460         printVisited();
461         printVisitedDFS();
462         printVisitedBFS();
463         DFS();
464         BFS();
465         printAdjList();
466         printVisited();
467         printVisitedDFS();
468         printVisitedBFS();
469         DFS();
470         BFS();
471         printAdjList();
472         printVisited();
473         printVisitedDFS();
474         printVisitedBFS();
475         DFS();
476         BFS();
477         printAdjList();
478         printVisited();
479         printVisitedDFS();
480         printVisitedBFS();
481         DFS();
482         BFS();
483         printAdjList();
484         printVisited();
485         printVisitedDFS();
486         printVisitedBFS();
487         DFS();
488         BFS();
489         printAdjList();
490         printVisited();
491         printVisitedDFS();
492         printVisitedBFS();
493         DFS();
494         BFS();
495         printAdjList();
496         printVisited();
497         printVisitedDFS();
498         printVisitedBFS();
499         DFS();
500         BFS();
501         printAdjList();
502         printVisited();
503         printVisitedDFS();
504         printVisitedBFS();
505         DFS();
506         BFS();
507         printAdjList();
508         printVisited();
509         printVisitedDFS();
510         printVisitedBFS();
511         DFS();
512         BFS();
513         printAdjList();
514         printVisited();
515         printVisitedDFS();
516         printVisitedBFS();
517         DFS();
518         BFS();
519         printAdjList();
520         printVisited();
521         printVisitedDFS();
522         printVisitedBFS();
523         DFS();
524         BFS();
525         printAdjList();
526         printVisited();
527         printVisitedDFS();
528         printVisitedBFS();
529         DFS();
530         BFS();
531         printAdjList();
532         printVisited();
533         printVisitedDFS();
534         printVisitedBFS();
535         DFS();
536         BFS();
537         printAdjList();
538         printVisited();
539         printVisitedDFS();
540         printVisitedBFS();
541         DFS();
542         BFS();
543         printAdjList();
544         printVisited();
545         printVisitedDFS();
546         printVisitedBFS();
547         DFS();
548         BFS();
549         printAdjList();
550         printVisited();
551         printVisitedDFS();
552         printVisitedBFS();
553         DFS();
554         BFS();
555         printAdjList();
556         printVisited();
557         printVisitedDFS();
558         printVisitedBFS();
559         DFS();
560         BFS();
561         printAdjList();
562         printVisited();
563         printVisitedDFS();
564         printVisitedBFS();
565         DFS();
566         BFS();
567         printAdjList();
568         printVisited();
569         printVisitedDFS();
570         printVisitedBFS();
571         DFS();
572         BFS();
573         printAdjList();
574         printVisited();
575         printVisitedDFS();
576         printVisitedBFS();
577         DFS();
578         BFS();
579         printAdjList();
580         printVisited();
581         printVisitedDFS();
582         printVisitedBFS();
583         DFS();
584         BFS();
585         printAdjList();
586         printVisited();
587         printVisitedDFS();
588         printVisitedBFS();
589         DFS();
590         BFS();
591         printAdjList();
592         printVisited();
593         printVisitedDFS();
594         printVisitedBFS();
595         DFS();
596         BFS();
597         printAdjList();
598         printVisited();
599         printVisitedDFS();
600         printVisitedBFS();
601         DFS();
602         BFS();
603         printAdjList();
604         printVisited();
605         printVisitedDFS();
606         printVisitedBFS();
607         DFS();
608         BFS();
609         printAdjList();
610         printVisited();
611         printVisitedDFS();
612         printVisitedBFS();
613         DFS();
614         BFS();
615         printAdjList();
616         printVisited();
617         printVisitedDFS();
618         printVisitedBFS();
619         DFS();
620         BFS();
621         printAdjList();
622         printVisited();
623         printVisitedDFS();
624         printVisitedBFS();
625         DFS();
626         BFS();
627         printAdjList();
628         printVisited();
629         printVisitedDFS();
630         printVisitedBFS();
631         DFS();
632         BFS();
633         printAdjList();
634         printVisited();
635         printVisitedDFS();
636         printVisitedBFS();
637         DFS();
638         BFS();
639         printAdjList();
640         printVisited();
641         printVisitedDFS();
642         printVisitedBFS();
643         DFS();
644         BFS();
645         printAdjList();
646         printVisited();
647         printVisitedDFS();
648         printVisitedBFS();
649         DFS();
650         BFS();
651         printAdjList();
652         printVisited();
653         printVisitedDFS();
654         printVisitedBFS();
655         DFS();
656         BFS();
657         printAdjList();
658         printVisited();
659         printVisitedDFS();
660         printVisitedBFS();
661         DFS();
662         BFS();
663         printAdjList();
664         printVisited();
665         printVisitedDFS();
666         printVisitedBFS();
667         DFS();
668         BFS();
669         printAdjList();
670         printVisited();
671         printVisitedDFS();
672         printVisitedBFS();
673         DFS();
674         BFS();
675         printAdjList();
676         printVisited();
677         printVisitedDFS();
678         printVisitedBFS();
679         DFS();
680         BFS();
681         printAdjList();
682         printVisited();
683         printVisitedDFS();
684         printVisitedBFS();
685         DFS();
686         BFS();
687         printAdjList();
688         printVisited();
689         printVisitedDFS();
690         printVisitedBFS();
691         DFS();
692         BFS();
693         printAdjList();
694         printVisited();
695         printVisitedDFS();
696         printVisitedBFS();
697         DFS();
698         BFS();
699         printAdjList();
700         printVisited();
701         printVisitedDFS();
702         printVisitedBFS();
703         DFS();
704         BFS();
705         printAdjList();
706         printVisited();
707         printVisitedDFS();
708         printVisitedBFS();
709         DFS();
710         BFS();
711         printAdjList();
712         printVisited();
713         printVisitedDFS();
714         printVisitedBFS();
715         DFS();
716         BFS();
717         printAdjList();
718         printVisited();
719         printVisitedDFS();
720         printVisitedBFS();
721         DFS();
722         BFS();
723         printAdjList();
724         printVisited();
725         printVisitedDFS();
726         printVisitedBFS();
727         DFS();
728         BFS();
729         printAdjList();
730         printVisited();
731         printVisitedDFS();
732         printVisitedBFS();
733         DFS();
734         BFS();
735         printAdjList();
736         printVisited();
737         printVisitedDFS();
738         printVisitedBFS();
739         DFS();
740         BFS();
741         printAdjList();
742         printVisited();
743         printVisitedDFS();
744         printVisitedBFS();
745         DFS();
746         BFS();
747         printAdjList();
748         printVisited();
749         printVisitedDFS();
750         printVisitedBFS();
751         DFS();
752         BFS();
753         printAdjList();
754         printVisited();
755         printVisitedDFS();
756         printVisitedBFS();
757         DFS();
758         BFS();
759         printAdjList();
760         printVisited();
761         printVisitedDFS();
762         printVisitedBFS();
763         DFS();
764         BFS();
765         printAdjList();
766         printVisited();
767         printVisitedDFS();
768         printVisitedBFS();
769         DFS();
770         BFS();
771         printAdjList();
772         printVisited();
773         printVisitedDFS();
774         printVisitedBFS();
775         DFS();
776         BFS();
777         printAdjList();
778         printVisited();
779         printVisitedDFS();
780         printVisitedBFS();
781         DFS();
782         BFS();
783         printAdjList();
784         printVisited();
785         printVisitedDFS();
786         printVisitedBFS();
787         DFS();
788         BFS();
789         printAdjList();
790         printVisited();
791         printVisitedDFS();
792         printVisitedBFS();
793         DFS();
794         BFS();
795         printAdjList();
796         printVisited();
797         printVisitedDFS();
798         printVisitedBFS();
799         DFS();
800         BFS();
801         printAdjList();
802         printVisited();
803         printVisitedDFS();
804         printVisitedBFS();
805         DFS();
806         BFS();
807         printAdjList();
808         printVisited();
809         printVisitedDFS();
810         printVisitedBFS();
811         DFS();
812         BFS();
813         printAdjList();
814         printVisited();
815         printVisitedDFS();
816         printVisitedBFS();
817         DFS();
818         BFS();
819         printAdjList();
820         printVisited();
821         printVisitedDFS();
822         printVisitedBFS();
823         DFS();
824         BFS();
825         printAdjList();
826         printVisited();
827         printVisitedDFS();
828         printVisitedBFS();
829         DFS();
830         BFS();
831         printAdjList();
832         printVisited();
833         printVisitedDFS();
834         printVisitedBFS();
835         DFS();
836         BFS();
837         printAdjList();
838         printVisited();
839         printVisitedDFS();
840         printVisitedBFS();
841         DFS();
842         BFS();
843         printAdjList();
844         printVisited();
845         printVisitedDFS();
846         printVisitedBFS();
847         DFS();
848         BFS();
849         printAdjList();
850         printVisited();
851         printVisitedDFS();
852         printVisitedBFS();
853         DFS();
854         BFS();
855         printAdjList();
856         printVisited();
857         printVisitedDFS();
858         printVisitedBFS();
859         DFS();
860         BFS();
861         printAdjList();
862         printVisited();
863         printVisitedDFS();
864         printVisitedBFS();
865         DFS();
866         BFS();
867         printAdjList();
868         printVisited();
869         printVisitedDFS();
870         printVisitedBFS();
871         DFS();
872         BFS();
873         printAdjList();
874         printVisited();
875         printVisitedDFS();
876         printVisitedBFS();
877         DFS();
878         BFS();
879         printAdjList();
880         printVisited();
881         printVisitedDFS();
882         printVisitedBFS();
883         DFS();
884         BFS();
885         printAdjList();
886         printVisited();
887         printVisitedDFS();
888         printVisitedBFS();
889         DFS();
890         BFS();
891         printAdjList();
892         printVisited();
893         printVisitedDFS();
894         printVisitedBFS();
895         DFS();
896         BFS();
897         printAdjList();
898         printVisited();
899         printVisitedDFS();
900         printVisitedBFS();
901         DFS();
902         BFS();
903         printAdjList();
904         printVisited();
905         printVisitedDFS();
906         printVisitedBFS();
907         DFS();
908         BFS();
909         printAdjList();
910         printVisited();
911         printVisitedDFS();
912         printVisitedBFS();
913         DFS();
914         BFS();
915         printAdjList();
916         printVisited();
917         printVisitedDFS();
918         printVisitedBFS();
919         DFS();
920         BFS();
921         printAdjList();
922         printVisited();
923         printVisitedDFS();
924         printVisitedBFS();
925         DFS();
926         BFS();
927         printAdjList();
928         printVisited();
929         printVisitedDFS();
930         printVisitedBFS();
931         DFS();
932         BFS();
933         printAdjList();
934         printVisited();
935         printVisitedDFS();
936         printVisitedBFS();
937         DFS();
938         BFS();
939         printAdjList();
940         printVisited();
941         printVisitedDFS();
942         printVisitedBFS();
943         DFS();
944         BFS();
945         printAdjList();
946         printVisited();
947         printVisitedDFS();
948         printVisitedBFS();
949         DFS();
950         BFS();
951         printAdjList();
952         printVisited();
953         printVisitedDFS();
954         printVisitedBFS();
955         DFS();
956         BFS();
957         printAdjList();
958         printVisited();
959         printVisitedDFS();
960         printVisitedBFS();
961         DFS();
962         BFS();
963         printAdjList();
964         printVisited();
965         printVisitedDFS();
966         printVisitedBFS();
967         DFS();
968         BFS();
969         printAdjList();
970         printVisited();
971         printVisitedDFS();
972         printVisitedBFS();
973         DFS();
974         BFS();
975         printAdjList();
976         printVisited();
977         printVisitedDFS();
978         printVisitedBFS();
979         DFS();
980         BFS();
981         printAdjList();
982         printVisited();
983         printVisitedDFS();
984         printVisitedBFS();
985         DFS();
986         BFS();
987         printAdjList();
988         printVisited();
989         printVisitedDFS();
990         printVisitedBFS();
991         DFS();
992         BFS();
993         printAdjList();
994         printVisited();
995         printVisitedDFS();
996         printVisitedBFS();
997         DFS();
998         BFS();
999         printAdjList();
1000         printVisited();
1001         printVisitedDFS();
1002         printVisitedBFS();
1003         DFS();
1004         BFS();
1005         printAdjList();
1006         printVisited();
1007         printVisitedDFS();
1008         printVisitedBFS();
1009         DFS();
1010         BFS();
1011         printAdjList();
1012         printVisited();
1013         printVisitedDFS();
1014         printVisitedBFS();
1015         DFS();
1016         BFS();
1017         printAdjList();
1018         printVisited();
1019         printVisitedDFS();
1020         printVisitedBFS();
1021         DFS();
1022         BFS();
1023         printAdjList();
1024         printVisited();
1025         printVisitedDFS();
1026         printVisitedBFS();
1027         DFS();
1028         BFS();
1029         printAdjList();
1030         printVisited();
1031         printVisitedDFS();
1032         printVisitedBFS();
1033         DFS();
1034         BFS();
1035         printAdjList();
1036         printVisited();
1037         printVisitedDFS();
1038         printVisitedBFS();
1039         DFS();
1040         BFS();
1041         printAdjList();
1042         printVisited();
1043         printVisitedDFS();
1044         printVisitedBFS();
1045         DFS();
1046         BFS();
1047         printAdjList();
1048         printVisited();
1049         printVisitedDFS();
1050         printVisitedBFS();
1051         DFS();
1052         BFS();
1053         printAdjList();
1054         printVisited();
1055         printVisitedDFS();
1056         printVisitedBFS();
1057         DFS();
1058         BFS();
1059         printAdjList();
1060         printVisited();
1061         printVisitedDFS();
1062         printVisitedBFS();
1063         DFS();
1064         BFS();
1065         printAdjList();
1066         printVisited();
1067         printVisitedDFS();
1068         printVisitedBFS();
1069         DFS();
1070         BFS();
1071         printAdjList();
1072         printVisited();
1073         printVisitedDFS();
1074         printVisitedBFS();
1075         DFS();
1076         BFS();
1077         printAdjList();
1078         printVisited();
1079         printVisitedDFS();
1080         printVisitedBFS();
1081         DFS();
1082         BFS();
1083         printAdjList();
1084         printVisited();
1085         printVisitedDFS();
1086         printVisitedBFS();
1087         DFS();
1088         BFS();
1089         printAdjList();
1090         printVisited();
1091         printVisitedDFS();
1092         printVisitedBFS();
1093         DFS();
1094         BFS();
1095         printAdjList();
1096         printVisited();
1097         printVisitedDFS();
1098         printVisitedBFS();
1099         DFS();
1100         BFS();
1101         printAdjList();
1102         printVisited();
1103         printVisitedDFS();
1104         printVisitedBFS();
1105         DFS();
1106         BFS();
1107         printAdjList();
1108         printVisited();
1109         printVisitedDFS();
1110         printVisitedBFS();
1111         DFS();
1112         BFS();
1113         printAdjList();
1114         printVisited();
1115         printVisitedDFS();
1116         printVisitedBFS();
1117         DFS();
1118         BFS();
1119         printAdjList();
1120         printVisited();
1121         printVisitedDFS();
1122         printVisitedBFS();
1123         DFS();
1124         BFS();
1125         printAdjList();
1126         printVisited();
1127         printVisitedDFS();
1128         printVisitedBFS();
1129         DFS();
1130         BFS();
1131         printAdjList();
1132         printVisited();
1133         printVisitedDFS();
1134         printVisitedBFS();
1135         DFS();
1136         BFS();
1137         printAdjList();
1138         printVisited();
1139         printVisitedDFS();
1140         printVisitedBFS();
1141         DFS();
1142         BFS();
1143         printAdjList();
1144         printVisited();
1145         printVisitedDFS();
1146         printVisitedBFS();
1147         DFS();
1148         BFS();
1149         printAdjList();
1150         printVisited();
1151         printVisitedDFS();
1152         printVisitedBFS();
1153         DFS();
1154         BFS();
1155         printAdjList();
1156         printVisited();
1157         printVisitedDFS();
1158         printVisitedBFS();
1159         DFS();
1160         BFS();
1161         printAdjList();
1162         printVisited();
1163         printVisitedDFS();
1164         printVisitedBFS();
1165         DFS();
1166         BFS();
1167         printAdjList();
1168         printVisited();
1169         printVisitedDFS();
1170         printVisitedBFS();
1171         DFS();
1172         BFS();
1173         printAdjList();
1174         printVisited();
1175         printVisitedDFS();
1176         printVisitedBFS();
1177         DFS();
1178         BFS();
1179         printAdjList();
1180         printVisited();
1181         printVisitedDFS();
1182         printVisitedBFS();
1183         DFS();
1184         BFS();
1185         printAdjList();
1186         printVisited();
1187         printVisitedDFS();
1188         printVisitedBFS();
1189         DFS();
1190         BFS();
1191         printAdjList();
1192         printVisited();
1193         printVisitedDFS();
1194         printVisitedBFS();
1195         DFS();
1196         BFS();
1197         printAdjList();
1198         printVisited();
1199         printVisitedDFS();
1200         printVisitedBFS();
1201         DFS();
1202         BFS();
1203         printAdjList();
1204         printVisited();
1205         printVisitedDFS();
1206         printVisitedBFS();
1207         DFS();
1208         BFS();
1209         printAdjList();
1210         printVisited();
1211         printVisitedDFS();
1212         printVisitedBFS();
1213         DFS();
1214         BFS();
1215         printAdjList();
1216         printVisited();
1217         printVisitedDFS();
1218         printVisitedBFS();
1219         DFS();
1220         BFS();
1221         printAdjList();
1222         printVisited();
1223         printVisitedDFS();
1224         printVisitedBFS();
1225         DFS();
1226         BFS();
1227         printAdjList();
1228         printVisited();
1229         printVisitedDFS();
1230         printVisitedBFS();
1231         DFS();
1232         BFS();
1233         printAdjList();
1234         printVisited();
1235         printVisitedDFS();
1236         printVisitedBFS();
1237         DFS();
1238         BFS();
1239         printAdjList();
1240         printVisited();
1241         printVisitedDFS();
1242         printVisitedBFS();
1243         DFS();
1244         BFS();
1245         printAdjList();
1246         printVisited();
1247         printVisitedDFS();
1248         printVisitedBFS();
1249         DFS();
1250         BFS();
1251         printAdjList();
1252         printVisited();
1253         printVisitedDFS();
1254         printVisitedBFS();
1255         DFS();
1256         BFS();
1257         printAdjList();
1258         printVisited();
1259         printVisitedDFS();
1260         printVisitedBFS();
1261         DFS();
1262         BFS();
1263         printAdjList();
1264         printVisited();
1265         printVisitedDFS();
1266         printVisitedBFS();
1267         DFS();
1268         BFS();
1269         printAdjList();
1270         printVisited();
1271         printVisitedDFS();
1272         printVisitedBFS();
1273         DFS();
1274         BFS();
1275         printAdjList();
1276         printVisited();
1277         printVisitedDFS();
1278         printVisitedBFS();
1279         DFS();
1280         BFS();
1281         printAdjList();
1282         printVisited();
1283         printVisitedDFS();
1284         printVisitedBFS();
1285         DFS();
1286         BFS();
1287         printAdjList();
1288         printVisited();
1289         printVisitedDFS();
1290         printVisitedBFS();
1291         DFS();
1292         BFS();
1293         printAdjList();
1294         printVisited();
1295         printVisitedDFS();
1296         printVisitedBFS();
1297         DFS();
1298         BFS();
1299         printAdjList();
1300         printVisited();
1301         printVisitedDFS();
1302         printVisitedBFS();
1303         DFS();
1304         BFS();
1305         printAdjList();
1306         printVisited();
1307         printVisitedDFS();
1308         printVisitedBFS();
1309         DFS();
1310         BFS();
1311         printAdjList();
1312         printVisited();
1313         printVisitedDFS();
1314         printVisitedBFS();
1315         DFS();
1316         BFS();
1317         printAdjList();
1318         printVisited();
1319         printVisitedDFS();
1320         printVisitedBFS();
1321         DFS();
1322         BFS();
1323         printAdjList();
1324         printVisited();
1325         printVisitedDFS();
1326         printVisitedBFS();
1327         DFS();
1328         BFS();
1329         printAdjList();
1330         printVisited();
1331         printVisitedDFS();
1332         printVisitedBFS();
1333         DFS();
1334         BFS();
1335         printAdjList();
1336         printVisited();
1337         printVisitedDFS();
1338         printVisitedBFS();
1339         DFS();
1340         BFS();
1341         printAdjList();
1342         printVisited();
1343         printVisitedDFS();
1344         printVisitedBFS();
1345         DFS();
1346         BFS();
1347         printAdjList();
1348         printVisited();
1349         printVisitedDFS();
1350         printVisitedBFS();
1351         DFS();
1352         BFS();
1353         printAdjList();
1354         printVisited();
1355         printVisitedDFS();
1356         printVisitedBFS();
1357         DFS();
1358         BFS();
1359         printAdjList();
1360        
```

Assignment – 2

Title: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Problem Statement: Design and implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objectives: Students should be able to Write a program to implement Parallel Bubble Sort and Merge Sort and can measure the performance of sequential and parallel algorithms.

Theory:

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

The basic algorithm of Bubble Sort is as follows:

1. Start at the beginning of the array.
2. Compare the first two elements. If the first element is greater than the second element, swap them.
3. Move to the next pair of elements and repeat step 2.
4. Continue the process until the end of the array is reached.
5. If any swaps were made in step 2-4, repeat the process from step 1.

The time complexity of Bubble Sort is $O(n^2)$, which makes it inefficient for large lists. However, it has the advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small datasets.

Bubble Sort has limited practical use in modern software development due to its inefficient time complexity of $O(n^2)$ which makes it unsuitable for sorting large datasets. However, Bubble Sort has some advantages and use cases that make it a valuable algorithm to understand, such as:

1. **Simplicity:** Bubble Sort is one of the simplest sorting algorithms, and it is easy to understand and implement. It can be used to introduce the concept of sorting to beginners and as a basis for more complex sorting algorithms.
2. **Educational purposes:** Bubble Sort is often used in academic settings to teach the principles of sorting algorithms and to help students understand how algorithms work.
3. **Small datasets:** For very small datasets, Bubble Sort can be an efficient sorting algorithm, as its overhead is relatively low.
4. **Partially sorted datasets:** If a dataset is already partially sorted, Bubble Sort can be very efficient. Since Bubble Sort only swaps adjacent elements that are in the wrong order, it has a low number of operations for a partially sorted dataset.

5. Performance optimization: Although Bubble Sort itself is not suitable for sorting large datasets, some of its techniques can be used in combination with other sorting algorithms to optimize their performance. For example, Bubble Sort can be used to optimize the performance of Insertion Sort by reducing the number of comparisons needed.

How Parallel Bubble Sort Work:

- Parallel Bubble Sort is a modification of the classic Bubble Sort algorithm that takes advantage of

parallel processing to speed up the sorting process.

- In parallel Bubble Sort, the list of elements is divided into multiple sub lists that are sorted concurrently by multiple threads. Each thread sorts its sub list using the regular Bubble Sort algorithm. When all sub lists have been sorted, they are merged to form the final sorted list.

- The parallelization of the algorithm is achieved using OpenMP, a programming API that supports

parallel processing in C++, Fortran, and other programming languages. OpenMP provides a set of

compiler directives that allow developers to specify which parts of the code can be executed in parallel.

- In the parallel Bubble Sort algorithm, the main loop that iterates over the list of elements is divided into multiple iterations that are executed concurrently by multiple threads. Each thread sorts a subset of the list, and the threads synchronize their work at the end of each iteration to ensure that the elements are properly ordered.

- Parallel Bubble Sort can provide a significant speedup over the regular Bubble Sort algorithm, especially when sorting large datasets on multi-core processors. However, the speedup is limited by the overhead of thread creation and synchronization, and it may not be worth the effort for small datasets or when using a single-core processor.

Merge Sort: Merge sort is a sorting algorithm that uses a divide-and-conquer approach to sort an array or a list of elements. The algorithm works by recursively dividing the input array into two halves, sorting each half, and then merging the sorted halves to produce a sorted output.

The merge sort algorithm can be broken down into the following steps:

1. Divide the input array into two halves.

2. Recursively sort the left half of the array.

3. Recursively sort the right half of the array.

4. Merge the two sorted halves into a single sorted output array.

- The merging step is where the bulk of the work happens in merge sort. The algorithm compares the first elements of each sorted half, selects the smaller element, and appends it to

the output array. This process continues until all elements from both halves have been appended to the output array.

- The time complexity of merge sort is $O(n \log n)$, which makes it an efficient sorting algorithm for large input arrays. However, merge sort also requires additional memory to store the output array, which can make it less suitable for use with limited memory resources.
- In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.
- One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of $O(n \log n)$, which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.
- Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

How Parallel Merge Sort Work:

- Parallel merge sort is a parallelized version of the merge sort algorithm that takes advantage of multiple processors or cores to improve its performance. In parallel merge sort, the input array is divided into smaller subarrays, which are sorted in parallel using multiple processors or cores. The sorted subarrays are then merged together in parallel to produce the final sorted output.
- The parallel merge sort algorithm can be broken down into the following steps:
 - Divide the input array into smaller subarrays.
 - Assign each subarray to a separate processor or core for sorting.
 - Sort each subarray in parallel using the merge sort algorithm.
 - Merge the sorted subarrays together in parallel to produce the final sorted output.
- The merging step in parallel merge sort is performed in a similar way to the merging step in the sequential merge sort algorithm. However, because the subarrays are sorted in parallel, the merging step can also be performed in parallel using multiple processors or cores. This can significantly reduce the time required to merge the sorted subarrays and produce the final output.
- Parallel merge sort can provide significant performance benefits for large input arrays with many elements, especially when running on hardware with multiple processors or cores. However, it also requires additional overhead to manage the parallelization, and may not always provide performance improvements for smaller input sizes or when run on hardware with limited parallel processing capabilities.

How to measure the performance of sequential and parallel algorithms:

There are several metrics that can be used to measure the performance of sequential and parallel merge sort algorithms:

1. **Execution time:** Execution time is the amount of time it takes for the algorithm to complete its sorting operation. This metric can be used to compare the speed of sequential and parallel merge sort algorithms.
2. **Speedup:** Speedup is the ratio of the execution time of the sequential merge sort algorithm to the execution time of the parallel merge sort algorithm. A speedup of greater than 1 indicates that the parallel algorithm is faster than the sequential algorithm.
3. **Efficiency:** Efficiency is the ratio of the speedup to the number of processors or cores used in the parallel algorithm. This metric can be used to determine how well the parallel algorithm is utilizing the available resources.
4. **Scalability:** Scalability is the ability of the algorithm to maintain its performance as the input size and number of processors or cores increase. A scalable algorithm will maintain a consistent speedup and efficiency as more resources are added.

Concept of OpenMP:

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

Conclusion: In this way we can implement Bubble Sort and Merge Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm.

BUBBLE SORT:

CODE:

```
#include<iostream>

#include<stdlib.h>

#include<omp.h>

using namespace std;

void bubble(int *, int);

void swap(int &, int &);

void bubble(int *a, int n)
{
    for( int i = 0; i < n; i++ )
    {
        int first = i % 2;

        #pragma omp parallel for shared(a,first)
        for( int j = first; j < n-1; j += 2 )
        {
            if( a[ j ] > a[ j+1 ] )
            {
                swap( a[ j ], a[ j+1 ] );
            }
        }
    }
}

void swap(int &a, int &b)
{

```

```
int test;

test=a;

a=b;

b=test;

}

int main()
{

    int *a,n;

    cout<<"\n enter total no of elements=>";

    cin>>n;

    a=new int[n];

    cout<<"\n enter elements=>";

    for(int i=0;i<n;i++)

    {

        cin>>a[i];

    }

    bubble(a,n);

    cout<<"\n sorted array is=>";

    for(int i=0;i<n;i++)

    {

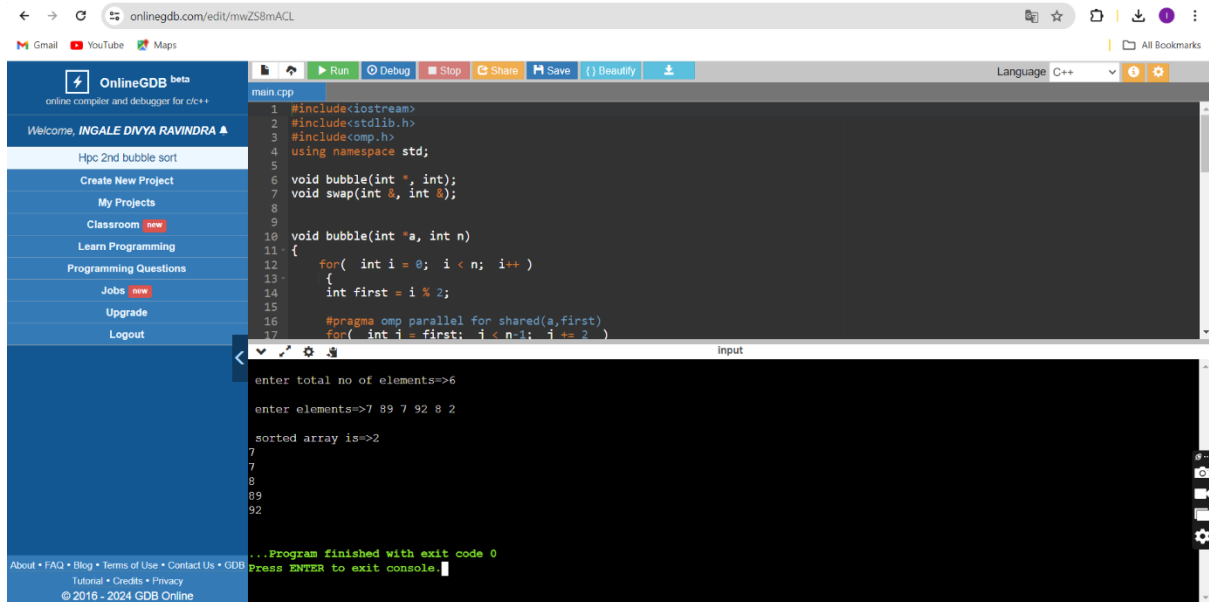
        cout<<a[i]<<endl;

    }

    return 0;
```

}

OUTPUT:



The screenshot shows the OnlineGDB web interface. The left sidebar contains navigation links like 'Welcome, INGALE DIVYA RAVINDRA', 'Hpc 2nd bubble sort', 'Create New Project', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Jobs', 'Upgrade', and 'Logout'. The main editor displays a C++ file named 'main.cpp' with the following code:

```
1 #include<iostream>
2 #include<stdlib.h>
3 #include<omp.h>
4 using namespace std;
5
6 void bubble(int *, int);
7 void swap(int &, int &);
8
9
10 void bubble(int *a, int n)
11 {
12     for( int i = 0; i < n; i++)
13     {
14         int first = i % 2;
15
16         #pragma omp parallel for shared(a,first)
17         for( int i = first; i < n-1; i += 2 )
```

The output window at the bottom shows the program's execution:

```
enter total no of elements=>6
enter elements=>7 89 7 92 8 2
sorted array is=>2
7
7
8
89
92
...Program finished with exit code 0
Press ENTER to exit console.
```

MERGE SORT:

CODE:

```
#include<iostream>
```

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

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

```
using namespace std;
```

```
void mergesort(int a[],int i,int j);
```

```
void merge(int a[],int i1,int j1,int i2,int j2);
```

```
void mergesort(int a[],int i,int j)
```

```
{
```

```
    int mid;
```

```
    if(i<j)
```

```
    {
```

```
mid=(i+j)/2;
```

```
#pragma omp parallel sections
```

```
{
```

```
#pragma omp section
```

```
{
```

```
    mergesort(a,i,mid);
```

```
}
```

```
#pragma omp section
```

```
{
```

```
    mergesort(a,mid+1,j);
```

```
}
```

```
}
```

```
merge(a,i,mid,mid+1,j);
```

```
}
```

```
}
```

```
void merge(int a[],int i1,int j1,int i2,int j2)
```

```
{
```

```
    int temp[1000];
```

```
    int i,j,k;
```

```
    i=i1;
```

```
    j=i2;
```

```
    k=0;
```

```
    while(i<=j1 && j<=j2)
```

```
    {
```

```

        if(a[i]<a[j])
        {
            temp[k++]=a[i++];
        }
        else
        {
            temp[k++]=a[j++];
        }
    }

    while(i<=j1)
    {
        temp[k++]=a[i++];
    }

    while(j<=j2)
    {
        temp[k++]=a[j++];
    }

    for(i=i1,j=0;i<=j2;i++,j++)
    {
        a[i]=temp[j];
    }
}

```

```

int main()
{
    int *a,n,i;

    cout<<"\n enter total no of elements=>";

```

```

    cin>>n;

    a= new int[n];

    cout<<"\n enter elements=>";

    for(i=0;i<n;i++)

    {

        cin>>a[i];

    }

    //    start=.....

//#pragma omp.....

    mergesort(a, 0, n-1);

//    stop.....

    cout<<"\n sorted array is=>";

    for(i=0;i<n;i++)

    {

        cout<<"\n"<<a[i];

    }

    // Cout<<Stop-Start

    return 0;

}

```

OUTPUT:

The screenshot shows the OnlineGDB interface. The code in main.cpp is as follows:

```

1 #include<iostream>
2 #include<stdlib.h>
3 #include<omp.h>
4 using namespace std;
5
6
7 void mergesort(int a[],int i,int j);
8 void merge(int a[],int i1,int j1,int i2,int j2);
9
10 void mergesort(int a[],int i,int j)
11 {
12     int mid;
13     if(i<j)

```

The output console shows the following interaction:

```

enter total no of elements=>7
enter elements=>4 5 6 7 8 2 1
sorted array is=>
1
2
4
5
6
7
8
...Program finished with exit code 0
Press ENTER to exit console.

```

Assignment – 3

Title: Implement Min, Max, Sum, and Average operations using Parallel Reduction.

Problem Statement: Design and Implement Min, Max, Sum and Average operations using Parallel Reduction.

Objectives: To understand the concept of parallel reduction and how it can be used to perform basic mathematical operations on given data sets.

Theory:

Parallel Reduction.

Here's a function-wise manual on how to understand and run the sample C++ program that demonstrates

how to implement Min, Max, Sum, and Average operations using parallel reduction.

1. Min_Reduction function

- The function takes in a vector of integers as input and finds the minimum value in the vector using parallel reduction.
- The OpenMP reduction clause is used with the "min" operator to find the minimum value across all threads.
- The minimum value found by each thread is reduced to the overall minimum value of the entire array.
- The final minimum value is printed to the console.

2. Max_Reduction function

- The function takes in a vector of integers as input and finds the maximum value in the vector using parallel reduction.
- The OpenMP reduction clause is used with the "max" operator to find the maximum value across all threads.
- The maximum value found by each thread is reduced to the overall maximum value of the entire array.
- The final maximum value is printed to the console.

3. Sum_Reduction function

- The function takes in a vector of integers as input and finds the sum of all the values in the vector using parallel reduction.
- The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
- The sum found by each thread is reduced to the overall sum of the entire array.

- The final sum is printed to the console.

4. Average_Reduction function

- The function takes in a vector of integers as input and finds the average of all the values in the vector using parallel reduction.
- The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
- The sum found by each thread is reduced to the overall sum of the entire array.
- The final sum is divided by the size of the array to find the average.
- The final average value is printed to the console.

5. Main Function

- The function initializes a vector of integers with some values.
- The function calls the min_reduction, max_reduction, sum_reduction, and average_reduction functions on the input vector to find the corresponding values.
- The final minimum, maximum, sum, and average values are printed to the console.

6. Compiling and running the program

Compile the program: You need to use a C++ compiler that supports OpenMP, such as g++ or clang. Open a terminal and navigate to the directory where your program is saved. Then, compile the program using the following command:

```
$ g++ -fopenmp program.cpp -o program
```

This command compiles your program and creates an executable file named "program". The "-fopenmp" flag tells the compiler to enable OpenMP.

Run the program: To run the program, simply type the name of the executable file in the terminal and press Enter:

```
$ ./program
```

Conclusion: We have implemented the Min, Max, Sum, and Average operations using parallel reduction in C++ with OpenMP. Parallel 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 parallel. We presented a code example that demonstrates the implementation of these operations using parallel reduction in C++ with OpenMP. We also provided a manual for running OpenMP programs on the Ubuntu platform.

CODE:

```
#include <iostream>

#include <vector>

#include <omp.h>

#include <climits>

using namespace std;
```

```
void 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];  
        }  
    }  
    cout << "Minimum value: " << min_value << endl;  
}
```

```
void 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];  
        }  
    }  
    cout << "Maximum value: " << max_value << endl;  
}
```

```
void 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];  
    }  
    cout << "Sum: " << sum << endl;  
}
```

```
void 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];  
    }  
    cout << "Average: " << (double)sum / (n-1) << endl;  
}
```

```
int main() {  
    int *arr,n;  
    cout<<"\n enter total no of elements=>";  
    cin>>n;  
    arr=new int[n];  
    cout<<"\n enter elements=>";  
    for(int i=0;i<n;i++)  
    {  
        cin>>arr[i];  
    }
```

```
// int arr[] = {5, 2, 9, 1, 7, 6, 8, 3, 4};  
// int n = size(arr);
```

```
    min_reduction(arr, n);  
    max_reduction(arr, n);  
    sum_reduction(arr, n);  
    average_reduction(arr, n);  
}
```

OUTPUT:

The screenshot displays the OnlineGDB web interface. The browser address bar shows the URL `onlinegdb.com/edit/RLAuFO641`. The interface includes a sidebar with navigation links such as "Welcome, INGALE DIVYA RAVINDRA", "Hpc 3rd", "Create New Project", "My Projects", "Classroom", "Learn Programming", "Programming Questions", "Jobs", "Upgrade", and "Logout". The main editor area shows a C++ program named `main.cpp` with the following code:

```
1 #include <iostream>
2 #include <vector>
3 #include <omp.h>
4 #include <limits>
5 using namespace std;
6 void min_reduction(int arr[], int n) {
7     int min_value = INT_MAX;
8     #pragma omp parallel for reduction(min: min_value)
9     for (int i = 0; i < n; i++) {
10         if (arr[i] < min_value) {
11             min_value = arr[i];
12         }
13     }
14 }
```

The console output shows the program's execution results:

```
enter total no of elements=>5
enter elements=>6 7 2 4 5
Minimum value: 2
Maximum value: 7
Sum: 24
Average: 6

...Program finished with exit code 0
Press ENTER to exit console.
```

Assignment – 4

Title: Write a CUDA Program for:

1. Addition of two large vectors
2. Matrix Multiplication using CUDA C

Problem Statement: Write a CUDA Program for:

1. Addition of two large vectors
2. Matrix Multiplication using CUDA C

Objectives: Students should be able to perform CUDA Program for Addition of two large vectors.

Theory:

CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use the power of NVIDIA graphics processing units (GPUs) to accelerate computation tasks in various applications, including scientific computing, machine learning, and computer vision. CUDA provides a set of programming APIs, libraries, and tools that enable developers to write and execute parallel code on NVIDIA GPUs. It supports popular programming languages like C, C++, and Python, and provides a simple programming model that abstracts away much of the low-level details of GPU architecture. Using CUDA, developers can exploit the massive parallelism and high computational power of GPUs to accelerate computationally intensive tasks, such as matrix operations, image processing, and deep learning. CUDA has become an important tool for scientific research and is widely used in fields like physics, chemistry, biology, and engineering.

Steps for Addition of two large vectors using CUDA

1. Define the size of the vectors: In this step, you need to define the size of the vectors that you want to add. This will determine the number of threads and blocks you will need to use to parallelize the addition operation.
2. Allocate memory on the host: In this step, you need to allocate memory on the host for the two vectors that you want to add and for the result vector. You can use the C malloc function to allocate memory.
3. Initialize the vectors: In this step, you need to initialize the two vectors that you want to add on the host. You can use a loop to fill the vectors with data.
4. Allocate memory on the device: In this step, you need to allocate memory on the device for the two vectors that you want to add and for the result vector. You can use the CUDA function cudaMalloc to allocate memory.
5. Copy the input vectors from host to device: In this step, you need to copy the two input vectors from the host to the device memory. You can use the CUDA function cudaMemcpy to copy the vectors.

6. Launch the kernel: In this step, you need to launch the CUDA kernel that will perform the addition operation. The kernel will be executed by multiple threads in parallel. You can use the `<<>>` syntax to specify the number of blocks and threads to use.

7. Copy the result vector from device to host: In this step, you need to copy the result vector from the device memory to the host memory. You can use the CUDA function `cudaMemcpy` to copy the result vector.

8. Free memory on the device: In this step, you need to free the memory that was allocated on the device. You can use the CUDA function `cudaFree` to free the memory.

9. Free memory on the host: In this step, you need to free the memory that was allocated on the host. You can use the C free function to free the memory.

Execution of Program over CUDA Environment Here are the steps to run a CUDA program for adding two large vectors: 1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.

2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the `PATH` and `LD_LIBRARY_PATH` environment variables to the appropriate directories.

3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a `.cu` extension.

4. Compile the CUDA program: You need to compile the CUDA program using the `nvcc` compiler that comes with the CUDA Toolkit.

The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named `program_name`. Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

This will execute the program and perform the addition of two large vectors.

Steps for Matrix Multiplication using CUDA Here are the steps for implementing matrix multiplication using CUDA C:

1. Matrix Initialization: The first step is to initialize the matrices that you want to multiply. You can use standard C or CUDA functions to allocate memory for the matrices and initialize their values. The matrices are usually represented as 2D arrays.

2. Memory Allocation: The next step is to allocate memory on the host and the device for the matrices. You can use the standard C `malloc` function to allocate memory on the host and the CUDA function `cudaMalloc()` to allocate memory on the device.

3. Data Transfer: The third step is to transfer data between the host and the device. You can use the CUDA function `cudaMemcpy()` to transfer data from the host to the device or vice versa.

4. Kernel Launch: The fourth step is to launch the CUDA kernel that will perform the matrix multiplication on the device. You can use the `<<>>` syntax to specify the number of blocks and threads to use. Each thread in the kernel will compute one element of the output matrix.

5. Device Synchronization: The fifth step is to synchronize the device to ensure that all kernel executions have completed before proceeding. You can use the CUDA function `cudaDeviceSynchronize()` to synchronize the device.

6. Data Retrieval: The sixth step is to retrieve the result of the computation from the device to the host. You can use the CUDA function `cudaMemcpy()` to transfer data from the device to the host.

7. Memory Deallocation: The final step is to deallocate the memory that was allocated on the host and the device. You can use the C free function to deallocate memory on the host and the CUDA function `cudaFree()` to deallocate memory on the device.

Execution of Program over CUDA Environment

1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.

2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the `PATH` and `LD_LIBRARY_PATH` environment variables to the appropriate directories.

3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a `.cu` extension.

4. Compile the CUDA program: You need to compile the CUDA program using the `nvcc` compiler that comes with the CUDA Toolkit. The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named `program_name`. Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

This will execute the program and perform the Matrix Multiplication using CUDA.

Conclusion: In this way we can implement the addition of two large vectors and the Matrix Multiplication using CUDA.

Program for Addition:

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

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <omp.h>

using namespace std;

#define MAX 100

int main()
{
    int a[MAX], b[MAX], c[MAX], i;

    printf("\n First Vector:\t");

    #pragma omp parallel for
        for (i = 0; i < MAX; i++)
        {
            a[i] = rand() % 1000;
        }

    for (i = 0; i < MAX; i++)
    {
        printf("%d\t", a[i]);
    }

    printf("\n Second Vector:\t");

    #pragma omp parallel for
        for (i = 0; i < MAX; i++)
        {
            b[i] = rand() % 1000;
        }
```



```
for (i = 0; i < MAX; i++)
```

```
{
```

```
    printf("%d\t", b[i]);
```

```
}
```

```
printf("\n Parallel-Vector Addition:(a,b,c)\t");
```

```
#pragma omp parallel for
```

```
for (i = 0; i < MAX; i++)
```

```
{
```

```
    c[i] = a[i] + b[i];
```

```
}
```

```
for (i = 0; i < MAX; i++)
```

```
{
```

```
    printf("\n%d\t%d\t%d\t", a[i], b[i], c[i]);
```

```
}
```

```
return 0;
```

```
}
```

Output:

```
1 #include <stdio.h>
2 #include <iostream>
3 #include <stdlib.h>
4 #include <omp.h>
5
6 using namespace std;
7
8 #define MAX 100
9
10 int main()
11 {
12     int a[MAX], b[MAX], c[MAX];
13     int i;
14
15     // Generating random numbers for vector a
16     for (i = 0; i < MAX; i++)
17         a[i] = rand() % 1000;
18
19     // Generating random numbers for vector b
20     for (i = 0; i < MAX; i++)
21         b[i] = rand() % 1000;
22
23     // Parallel-Vector Addition
24     #pragma omp parallel for
25     for (i = 0; i < MAX; i++)
26         c[i] = a[i] + b[i];
27
28     // Displaying the result
29     printf("\n Parallel-Vector Addition:(a,b,c)\t");
30
31     // Printing vector a
32     printf("\n First Vector:");
33     for (i = 0; i < MAX; i++)
34         printf("%d\t", a[i]);
35
36     // Printing vector b
37     printf("\n Second Vector:");
38     for (i = 0; i < MAX; i++)
39         printf("%d\t", b[i]);
40
41     // Printing vector c
42     printf("\n Parallel-Vector Addition:");
43     for (i = 0; i < MAX; i++)
44         printf("%d\t", c[i]);
45
46     return 0;
47 }
```

Output:

```
First Vector: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926
540 426 172 736 211 368 567 429 782 530 862 123 67 135 929 802 22
58 69 167 393 456 11 42 229 373 421 919 784 537 198 324 315 370
413 526 91 980 956 873 862 170 996 281 305 925 84 327 336 505 846
729 313 857 124 895 582 545 814 367 434 364 43 750 87 808 276 178 78
584 403 651 754 399 932 60 676 368 739 12 226 586 94 539
Second Vector: 795 570 434 378 467 601 97 902 317 492 652 756 301 280 286 441
865 689 444 619 440 729 31 117 97 771 481 675 709 927 567 856 497
353 586 965 306 683 219 624 528 871 732 829 503 19 270 368 708 715
340 149 796 723 618 245 846 451 921 555 379 488 764 432 551 437 228 27
500 34 764 124 914 987 856 743 491 227 365 859 936 432 551 437 228 27
5 407 474 121 858 395 29 237 235 793 818 428 143 11 928 529
Parallel-Vector Addition: (a,b,c)
383 795 1178
886 570 1456
777 434 1211
915 378 1293
793 467 1260
335 601 936
386 97 483
492 902 1394
649 317 966
421 492 913
362 652 1014
27 756 783
```

Program for Multiplication:

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

```
#include<iostream>
```

```
#include<cstdlib>
```

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

```
using namespace std;
```

```
int main()
```

```
{
```

```
int m=3,n=2;
```

```
int mat[m][n], vec[n],out[m];
```

```
//matrix of size 3x2
```

```
for(int row=0;row<m;row++)
```

```
{
```

```
for(int col=0;col<n;col++)
```

```
{
```

```
mat[row][col]=1;
```

```
}
```

```
}
```

```
//display matrix
```

```
cout<<"Input Matrix"<<endl;
```

```
for(int row=0;row<m;row++)
```

```
{
```

```
for(int col=0;col<n;col++)
```

```
{
```

```
cout<<"\t"<<mat[row][col];
```

```
}
```

```
cout<<"\n"<<endl;
```

```
}
```

```
//column vector of size 2x1
```

```
for(int row=0;row<n;row++)
```

```
{
```

```
vec[row]=2;
```

```
}
```

```
//display vector
```

```
cout<<"Input Col-Vector"<<endl;
```

```
for(int row=0;row<n;row++)
```

```
{
```

```
cout<<vec[row]<<endl;
```

```
}
```

```
{
```

```
#pragma omp parallel for
```

```
for(int row=0;row<m;row++)
```

```
{
```

```
out[row]=0;
```

```
for(int col=0;col<n;col++)
```

```
{ out[row] +=mat[row][col]*vec[col];
```

```
}
```

```
}
```

```
}
```

```
//display resultant vector
```

```
cout<< "Resultant Col-Vector"<<endl;
```

```
for (int row=0; row<m;row++)
```

```
{
```

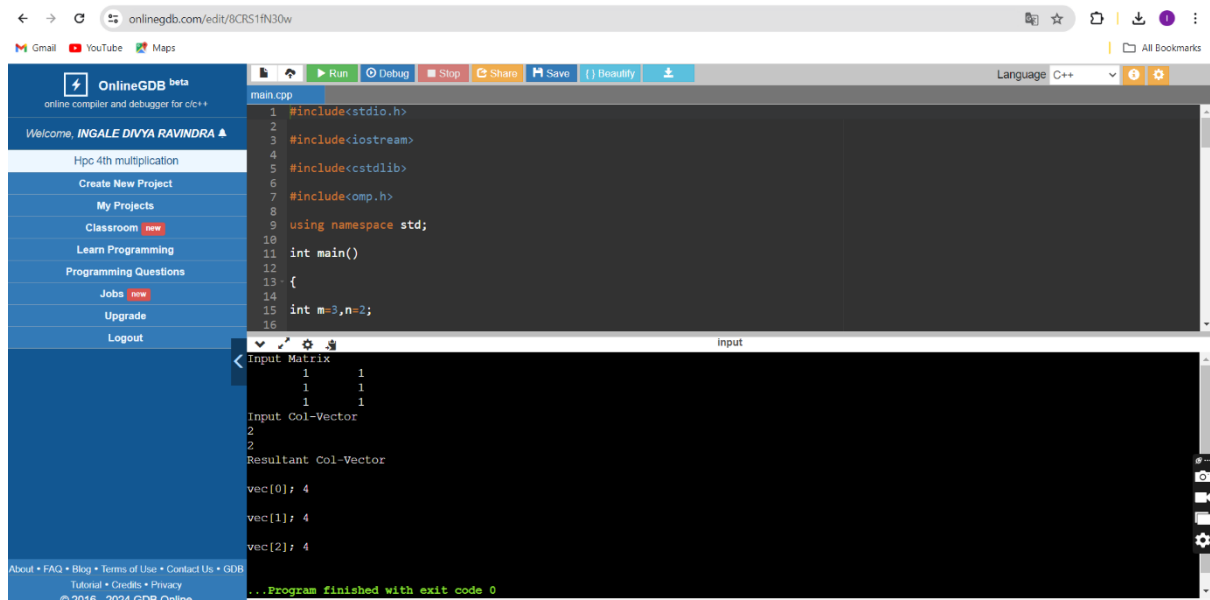
```
cout<<"\nvec["<<row<<"]; "<<out[row]<<endl;
```

```
}
```

```
return 0;
```

```
}
```

Output:



The screenshot displays the OnlineGDB web interface. The browser address bar shows the URL `onlinegdb.com/edit/8CRS1fN30w`. The interface includes a sidebar with navigation links such as "Welcome, INGALE DIVYA RAVINDRA", "Hpc 4th multiplication", "Create New Project", "My Projects", "Classroom", "Learn Programming", "Programming Questions", "Jobs", "Upgrade", and "Logout". The main editor area shows a C++ program for matrix multiplication. The code includes headers for `stdio.h`, `iostream`, `cstdlib`, and `omp.h`, and uses the `std` namespace. The `main` function defines `m=3` and `n=2`. The output window shows the input matrix as a 3x2 grid of ones, the input column vector as a 2x1 grid of ones, and the resultant column vector as a 3x1 grid with values 4, 4, and 4. The program concludes with the message "...Program finished with exit code 0".

```
1 #include<stdio.h>
2 #include<iostream>
3 #include<cstdlib>
4 #include<omp.h>
5 using namespace std;
6
7
8
9
10
11 int main()
12 {
13     int m=3,n=2;
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Input Matrix

1	1
1	1
1	1

Input Col-Vector

2
2

Resultant Col-Vector

vec[0]: 4
vec[1]: 4
vec[2]: 4

...Program finished with exit code 0



Bharati Vidyapeeth's College of Engineering for Women, Pune-43

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 1

TITLE: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS .

NAME OF LABORATORY: LPV

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

NAME OF STUDENT: Divya Ravindra Ingale

ROLL NO: 4225

SEMESTER: 8th Semester

YEAR: FOURTH YEAR

NAME OF FACULTY: Prof. Anjali Kadam

MARKS/ GRADE OBTAINED:

Signature of faculty



Bharati Vidyapeeth's College of Engineering for Women, Pune-43

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 2

TITLE: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

NAME OF LABORATORY: LPV

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

NAME OF STUDENT: Divya Ravindra Ingale

ROLL NO: 4225

SEMESTER: 8th Semester

YEAR: FOURTH YEAR

NAME OF FACULTY: Prof. Anjali Kadam

MARKS/ GRADE OBTAINED:

Signature of faculty



Bharati Vidyapeeth's College of Engineering for Women, Pune-43
DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 3

TITLE: Implement Min, Max, Sum and Average operations using Parallel Reduction.

NAME OF LABORATORY: LPV

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

NAME OF STUDENT: Divya Ravindra Ingale

ROLL NO: 4225

SEMESTER: 8th Semester

YEAR: FOURTH YEAR

NAME OF FACULTY: Prof. Anjali Kadam

MARKS/ GRADE OBTAINED:

Signature of faculty



Bharati Vidyapeeth's College of Engineering for Women, Pune-43

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 4

TITLE: Write a CUDA Program for : 1. Addition of two large vectors 2. Matrix Multiplication using CUDA C

NAME OF LABORATORY: LPV

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

NAME OF STUDENT: Divya Ravindra Ingale

ROLL NO: 4225

SEMESTER: 8th Semester

YEAR: FOURTH YEAR

NAME OF FACULTY: Prof. Anjali Kadam

MARKS/ GRADE OBTAINED:

Signature of faculty



Bharati Vidyapeeth's College of Engineering for Women, Pune-43
DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 5

TITLE: Project Report

NAME OF LABORATORY: LPV

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

NAME OF STUDENT: Divya Ravindra Ingale

ROLL NO: 4225

SEMESTER: 8th Semester

YEAR: FOURTH YEAR

NAME OF FACULTY: Prof. Anjali Kadam

MARKS/ GRADE OBTAINED:

Signature of faculty