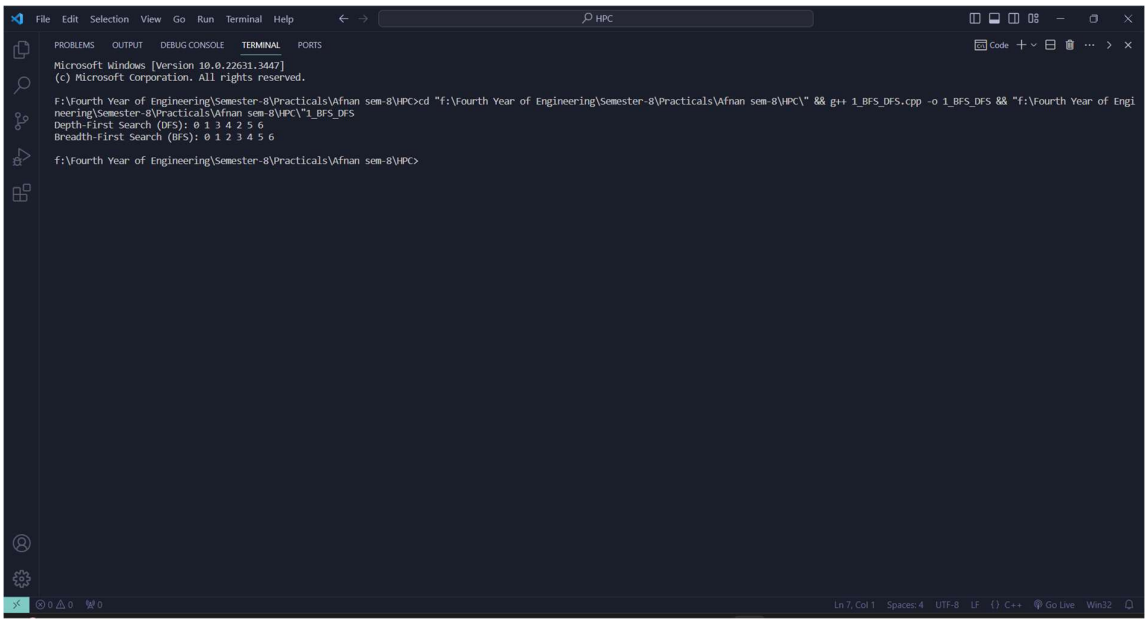
Practical-1

```
#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);

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

  return 0;
}
```

OUTPUT:

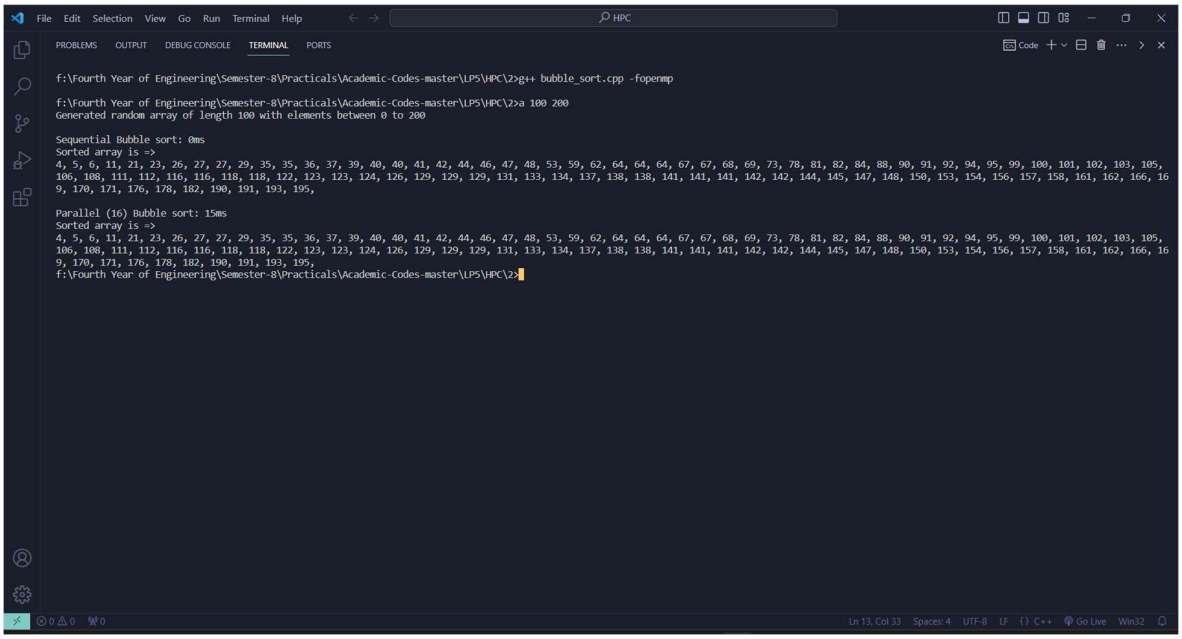


Practical-2

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

#include <array>
#include <chrono>
#include <functional>
#include <iostream>
#include <string>
#include <vector>

using std::chrono::duration_cast;
using std::chrono::high_resolution_clock;
using std::chrono::milliseconds;
using namespace std;

void s_bubble(int *, int);
void p_bubble(int *, int);
void swap(int &, int &);

void s_bubble(int *a, int n) {
  for (int i = 0; i < n; i++) {
    int first = i % 2;
    for (int j = first; j < n - 1; j += 2) {
      if (a[j] > a[j + 1]) {
        swap(a[j], a[j + 1]);
      }
    }
  }
}

void p_bubble(int *a, int n) {
  for (int i = 0; i < n; i++) {
    int first = i % 2;
#pragma omp parallel for shared(a, first)
num_threads(16)
    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;
}

std::string bench_traverse(std::function<void()>
traverse_fn) {
  auto start = high_resolution_clock::now();
  traverse_fn();
  auto stop = high_resolution_clock::now();

  // Subtract stop and start timepoints and cast it
to required unit.
  // Predefined units are nanoseconds,
microseconds, milliseconds, seconds,
  // minutes, hours. Use duration_cast() function.
  auto duration = duration_cast<milliseconds>(stop
- start);

  // To get the value of duration use the count()
member function on the
  // duration object
  return std::to_string(duration.count());
}

int main(int argc, const char **argv) {
  if (argc < 3) {
    std::cout << "Specify array length and
maximum random value\n";
    return 1;
  }
  int *a, n, rand_max;

  n = stoi(argv[1]);
  rand_max = stoi(argv[2]);
  a = new int[n];

  for (int i = 0; i < n; i++) {
    a[i] = rand() % rand_max;
  }

  int *b = new int[n];
  copy(a, a + n, b);
  cout << "Generated random array of length " << n
<< " with elements between 0 to " << rand_max
      << "\n\n";

  std::cout << "Sequential Bubble sort: " <<
bench_traverse([&] { s_bubble(a, n); }) << "ms\n";
  cout << "Sorted array is =>\n";
  for (int i = 0; i < n; i++) {
    cout << a[i] << ", ";
  }
  cout << "\n\n";

  omp_set_num_threads(16);
```

```
    std::cout << "Parallel (16) Bubble sort: " <<
bench_traverse([&] { p_bubble(b, n); }) << "ms\n";
    cout << "Sorted array is =>\n";
    for (int i = 0; i < n; i++) {
        cout << b[i] << ", ";
    }
    return 0;
}
```

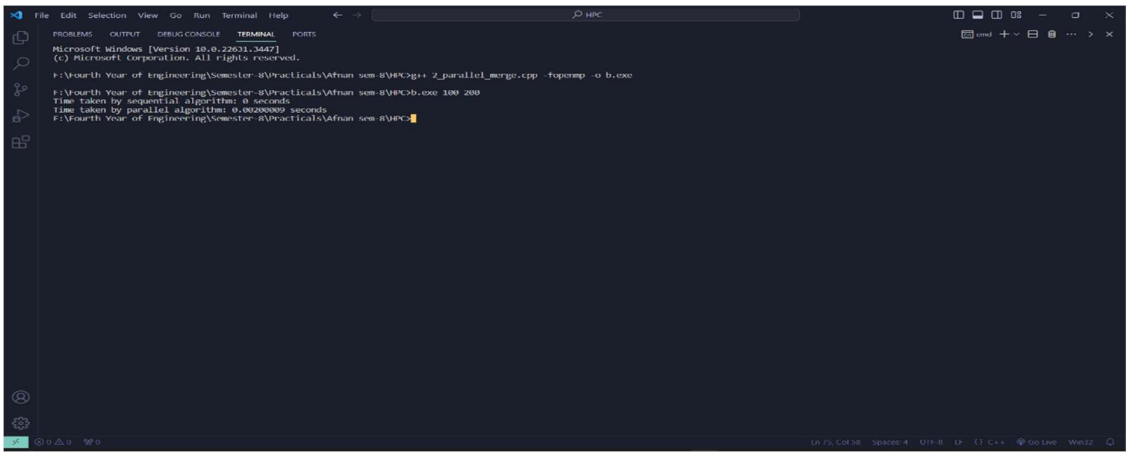
OUTPUT:



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

#include <array>
#include <chrono>
#include <functional>
#include <iostream>
#include <string>
#include <vector>

using std::chrono::duration_cast;
using std::chrono::high_resolution_clock;
using std::chrono::milliseconds;
using namespace std;

void p_mergesort(int *a, int i, int j);
void s_mergesort(int *a, int i, int j);
void merge(int *a, int i1, int j1, int i2, int j2);

void p_mergesort(int *a, int i, int j) {
  int mid;
  if (i < j) {
    if ((j - i) > 1000) {
      mid = (i + j) / 2;

#pragma omp task firstprivate(a, i, mid)
        p_mergesort(a, i, mid);
#pragma omp task firstprivate(a, mid, j)
        p_mergesort(a, mid + 1, j);

#pragma omp taskwait
        merge(a, i, mid, mid + 1, j);
    } else {
      s_mergesort(a, i, j);
    }
  }
}

void parallel_mergesort(int *a, int i, int j) {
#pragma omp parallel num_threads(16)
  {
#pragma omp single
    p_mergesort(a, i, j);
  }
}

void s_mergesort(int *a, int i, int j) {
  int mid;
  if (i < j) {
    mid = (i + j) / 2;
    s_mergesort(a, i, mid);
    s_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[2000000];
  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];
  }
}

std::string bench_traverse(std::function<void()>
traverse_fn) {
  auto start = high_resolution_clock::now();
  traverse_fn();
  auto stop = high_resolution_clock::now();

  // Subtract stop and start timepoints and cast it
to required unit.
  // Predefined units are nanoseconds,
microseconds, milliseconds, seconds,
  // minutes, hours. Use duration_cast() function.
  auto duration = duration_cast<milliseconds>(stop
- start);

  // To get the value of duration use the count()
member function on the
  // duration object
  return std::to_string(duration.count());
}

int main(int argc, const char **argv) {
  if (argc < 3) {
    std::cout << "Specify array length and
maximum random value\n";
    return 1;
  }
```

```
    int *a, n, rand_max;

    n = stoi(argv[1]);
    rand_max = stoi(argv[2]);
    a = new int[n];

    for (int i = 0; i < n; i++) {
        a[i] = rand() % rand_max;
    }

    int *b = new int[n];
    copy(a, a + n, b);
    cout << "Generated random array of length " << n
<< " with elements between 0 to " << rand_max
         << "\n\n";

    std::cout << "Sequential Merge sort: " <<
bench_traverse([&] { s_mergesort(a, 0, n - 1); })
            << "ms\n";

    cout << "Sorted array is =>\n";
    for (int i = 0; i < n; i++) {
        cout << a[i] << ", ";
    }
    cout << "\n\n";

    omp_set_num_threads(16);
    std::cout << "Parallel (16) Merge sort: "
            << bench_traverse([&] {
parallel_mergesort(b, 0, n - 1); }) << "ms\n";

    cout << "Sorted array is =>\n";
    for (int i = 0; i < n; i++) {
        cout << b[i] << ", ";
    }
    return 0;
}
```

OUTPUT:

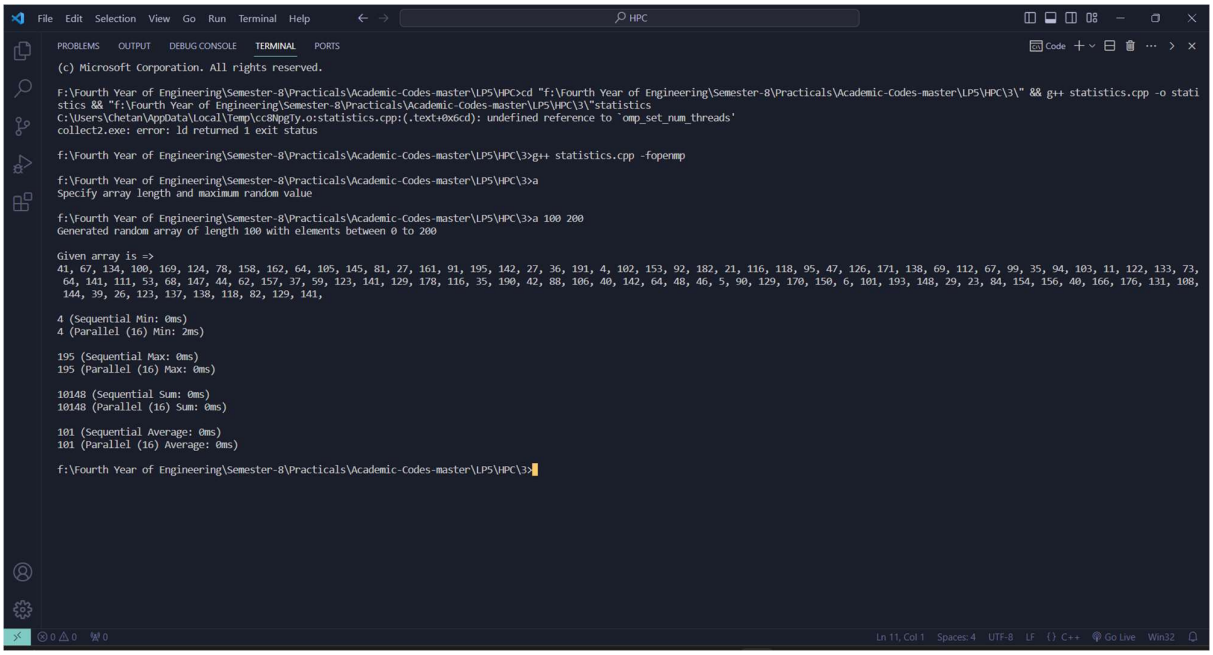


Practical-3

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

#include <array>
#include <chrono>
#include <functional>
#include <iostream>
#include <string>
#include <vector>

using std::chrono::duration_cast;
using std::chrono::high_resolution_clock;
using std::chrono::milliseconds;
using namespace std;

void s_avg(int arr[], int n) {
  long sum = 0L;
  int i;
  for (i = 0; i < n; i++) {
    sum = sum + arr[i];
  }
  cout << sum / long(n);
}

void p_avg(int arr[], int n) {
  long sum = 0L;
  int i;
#pragma omp parallel for reduction(+ : sum)
num_threads(16)
  for (i = 0; i < n; i++) {
    sum = sum + arr[i];
  }
  cout << sum / long(n);
}

void s_sum(int arr[], int n) {
  long sum = 0L;
  int i;
  for (i = 0; i < n; i++) {
    sum = sum + arr[i];
  }
  cout << sum;
}

void p_sum(int arr[], int n) {
  long sum = 0L;
  int i;
#pragma omp parallel for reduction(+ : sum)
num_threads(16)
  for (i = 0; i < n; i++) {
    sum = sum + arr[i];
  }
  cout << sum;
}

void s_max(int arr[], int n) {
  int max_val = INT_MIN;
  int i;
  for (i = 0; i < n; i++) {
    if (arr[i] > max_val) {
      max_val = arr[i];
    }
  }
  cout << max_val;
}

void p_max(int arr[], int n) {
  int max_val = INT_MIN;
  int i;
#pragma omp parallel for reduction(max : max_val)
num_threads(16)
  for (i = 0; i < n; i++) {
    if (arr[i] > max_val) {
      max_val = arr[i];
    }
  }
  cout << max_val;
}

void s_min(int arr[], int n) {
  int min_val = INT_MAX;
  int i;
  for (i = 0; i < n; i++) {
    if (arr[i] < min_val) {
      min_val = arr[i];
    }
  }
  cout << min_val;
}

void p_min(int arr[], int n) {
  int min_val = INT_MAX;
  int i;
#pragma omp parallel for reduction(min : min_val)
num_threads(16)
  for (i = 0; i < n; i++) {
    if (arr[i] < min_val) {
      min_val = arr[i];
    }
  }
  cout << min_val;
}
```

```
std::string bench_traverse(std::function<void()>
traverse_fn) {
   auto start = high_resolution_clock::now();
   traverse_fn();
   cout << " (";
   auto stop = high_resolution_clock::now();

   // Subtract stop and start timepoints and cast it
to required unit.
   // Predefined units are nanoseconds,
microseconds, milliseconds, seconds,
   // minutes, hours. Use duration_cast() function.
   auto duration = duration_cast<milliseconds>(stop
- start);

   // To get the value of duration use the count()
member function on the
   // duration object
   return std::to_string(duration.count());
}

int main(int argc, const char **argv) {
   if (argc < 3) {
      std::cout << "Specify array length and
maximum random value\n";
      return 1;
   }
   int *a, n, rand_max;

   n = stoi(argv[1]);
   rand_max = stoi(argv[2]);
   a = new int[n];

   for (int i = 0; i < n; i++) {
      a[i] = rand() % rand_max;
   }

   cout << "Generated random array of length " << n
<< " with elements between 0 to " << rand_max
       << "\n\n";
   cout << "Given array is =>\n";
   for (int i = 0; i < n; i++) {
      cout << a[i] << ", ";
   }
   cout << "\n\n";

   omp_set_num_threads(16);

   std::cout << "Sequential Min: " <<
bench_traverse([&] { s_min(a, n); }) << "ms)\n";
   std::cout << "Parallel (16) Min: " <<
bench_traverse([&] { p_min(a, n); }) << "ms)\n\n";
   std::cout << "Sequential Max: " <<
bench_traverse([&] { s_max(a, n); }) << "ms)\n";
   std::cout << "Parallel (16) Max: " <<
bench_traverse([&] { p_max(a, n); }) << "ms)\n\n";
   std::cout << "Sequential Sum: " <<
bench_traverse([&] { s_sum(a, n); }) << "ms)\n";
   std::cout << "Parallel (16) Sum: " <<
bench_traverse([&] { p_sum(a, n); }) << "ms)\n\n";
   std::cout << "Sequential Average: " <<
bench_traverse([&] { s_avg(a, n); }) << "ms)\n";
   std::cout << "Parallel (16) Average: " <<
bench_traverse([&] { p_avg(a, n); }) << "ms)\n";
   return 0;
}
```

OUTPUT:


File Edit Selection View Go Run Terminal Help
HPC
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code
(c) Microsoft Corporation. All rights reserved.
F:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC>cd "f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3\" && g++ statistics.cpp -o statistics && "f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3\"statistics
C:\Users\Chetan\AppData\Local\Temp\cc8NpgTy.o:statistics.cpp:(.text+0x6cd): undefined reference to `omp_set_num_threads'
collect2.exe: error: ld returned 1 exit status
f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3>g++ statistics.cpp -fopenmp
f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3>a
Specify array length and maximum random value
f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3>a 100 200
Generated random array of length 100 with elements between 0 to 200
Given array is =>
41, 67, 134, 100, 169, 124, 78, 158, 162, 64, 105, 145, 81, 27, 161, 91, 195, 142, 27, 36, 191, 4, 102, 153, 92, 182, 21, 116, 118, 95, 47, 126, 171, 138, 69, 112, 67, 99, 35, 94, 103, 11, 122, 133, 73, 64, 141, 111, 53, 68, 147, 44, 62, 157, 37, 59, 123, 141, 129, 178, 116, 35, 190, 42, 88, 106, 40, 142, 64, 48, 46, 5, 90, 129, 170, 150, 6, 101, 193, 148, 29, 23, 84, 154, 156, 40, 166, 176, 131, 108, 144, 39, 26, 123, 137, 138, 118, 82, 129, 141,
4 (Sequential Min: 0ms)
4 (Parallel (16) Min: 2ms)
195 (Sequential Max: 0ms)
195 (Parallel (16) Max: 0ms)
10148 (Sequential Sum: 0ms)
10148 (Parallel (16) Sum: 0ms)
101 (Sequential Average: 0ms)
101 (Parallel (16) Average: 0ms)
f:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\3>
Ln 11, Col 1 Spaces: 4 UTF-8 LF C++ Go Live Win32

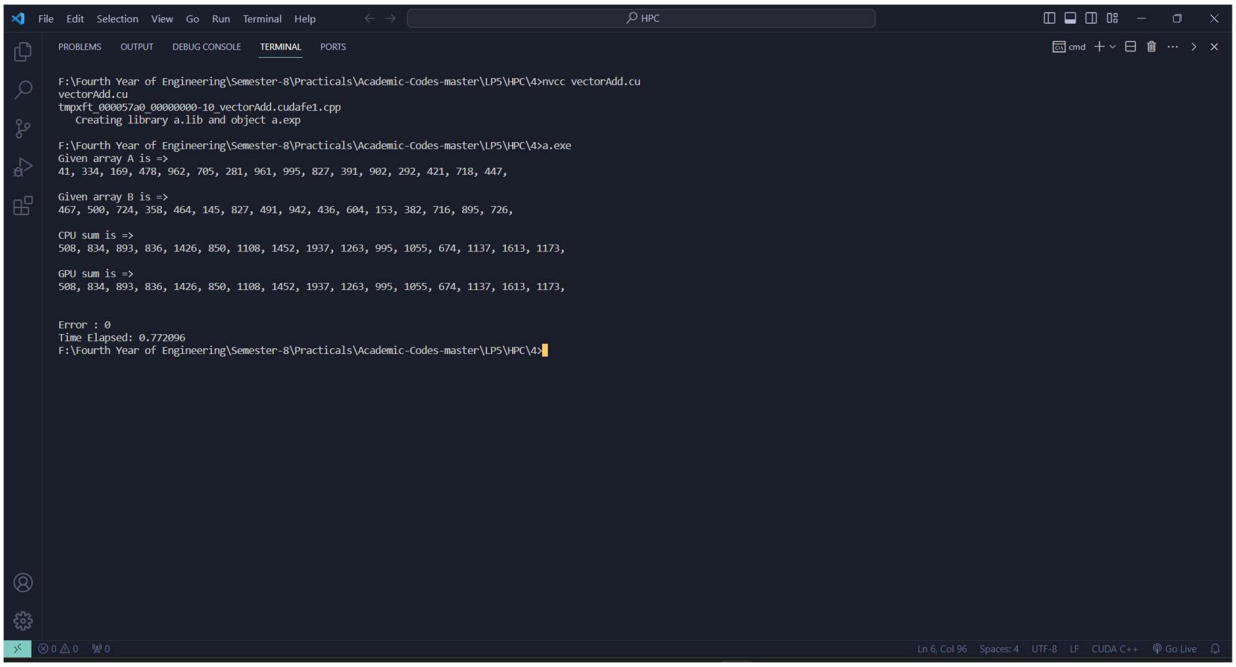
Practical-4

vectorAdd.cu

```
#include <cstdlib>
#include <iostream>

#define checkCudaErrors(call)
\
   do {
\
      cudaError_t err = call;
\
      if (err != cudaSuccess) {
\
         printf("CUDA error at %s %d: %s\n",
__FILE__, __LINE__, cudaGetErrorString(err)); \
         exit(EXIT_FAILURE);
\
      }
\
   } while (0)

using namespace std;

// VectorAdd parallel function
__global__ void vectorAdd(int *a, int *b, int
*result, int n) {
   int tid = threadIdx.x + blockIdx.x * blockDim.x;
   if (tid < n) {
      result[tid] = a[tid] + b[tid];
   }
}

int main() {
   int *a, *b, *c;
   int *a_dev, *b_dev, *c_dev;
   int n = 1 << 4;

   a = new int[n];
   b = new int[n];
   c = new int[n];
   int *d = new int[n];
   int size = n * sizeof(int);
   checkCudaErrors(cudaMalloc(&a_dev, size));
   checkCudaErrors(cudaMalloc(&b_dev, size));
   checkCudaErrors(cudaMalloc(&c_dev, size));

   // Array initialization..You can use Randon
function to assign values
   for (int i = 0; i < n; i++) {
      a[i] = rand() % 1000;
      b[i] = rand() % 1000;
      d[i] = a[i] + b[i];  // calculating serial addition
   }
   cout << "Given array A is =>\n";
   for (int i = 0; i < n; i++) {
      cout << a[i] << ", ";
   }
   cout << "\n\n";

   cout << "Given array B is =>\n";
   for (int i = 0; i < n; i++) {
      cout << b[i] << ", ";
   }
   cout << "\n\n";

   cudaEvent_t start, end;

   checkCudaErrors(cudaEventCreate(&start));
   checkCudaErrors(cudaEventCreate(&end));

   checkCudaErrors(cudaMemcpy(a_dev, a, size,
cudaMemcpyHostToDevice));
   checkCudaErrors(cudaMemcpy(b_dev, b, size,
cudaMemcpyHostToDevice));
   int threads = 1024;
   int blocks = (n + threads - 1) / threads;
   checkCudaErrors(cudaEventRecord(start));

   // Parallel addition program
   vectorAdd<<<blocks, threads>>>(a_dev, b_dev,
c_dev, n);

   checkCudaErrors(cudaEventRecord(end));
   checkCudaErrors(cudaEventSynchronize(end));

   float time = 0.0;
   checkCudaErrors(cudaEventElapsedTime(&time,
start, end));

   checkCudaErrors(cudaMemcpy(c, c_dev, size,
cudaMemcpyDeviceToHost));

   // Calculate the error term.

   cout << "CPU sum is =>\n";
   for (int i = 0; i < n; i++) {
      cout << d[i] << ", ";
   }
   cout << "\n\n";

   cout << "GPU sum is =>\n";
   for (int i = 0; i < n; i++) {
      cout << c[i] << ", ";
   }
   cout << "\n\n";
```

```
    int error = 0;
    for (int i = 0; i < n; i++) {
        error += d[i] - c[i];
        if (0 != (d[i] - c[i])) {
            cout << "Error at (" << i << ") => GPU: " <<
c[i] << ", CPU: " << d[i] << "\n";
        }
    }

    cout << "\nError : " << error;
    cout << "\nTime Elapsed: " << time;

    return 0;
}
```

OUTPUT:

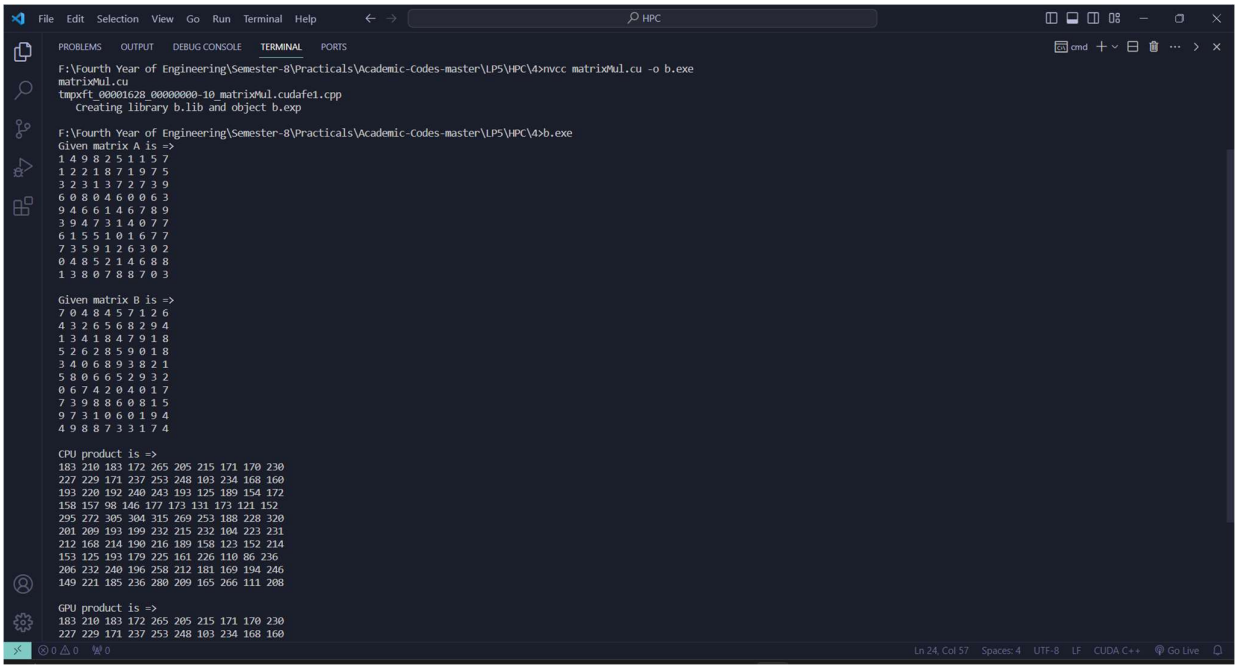


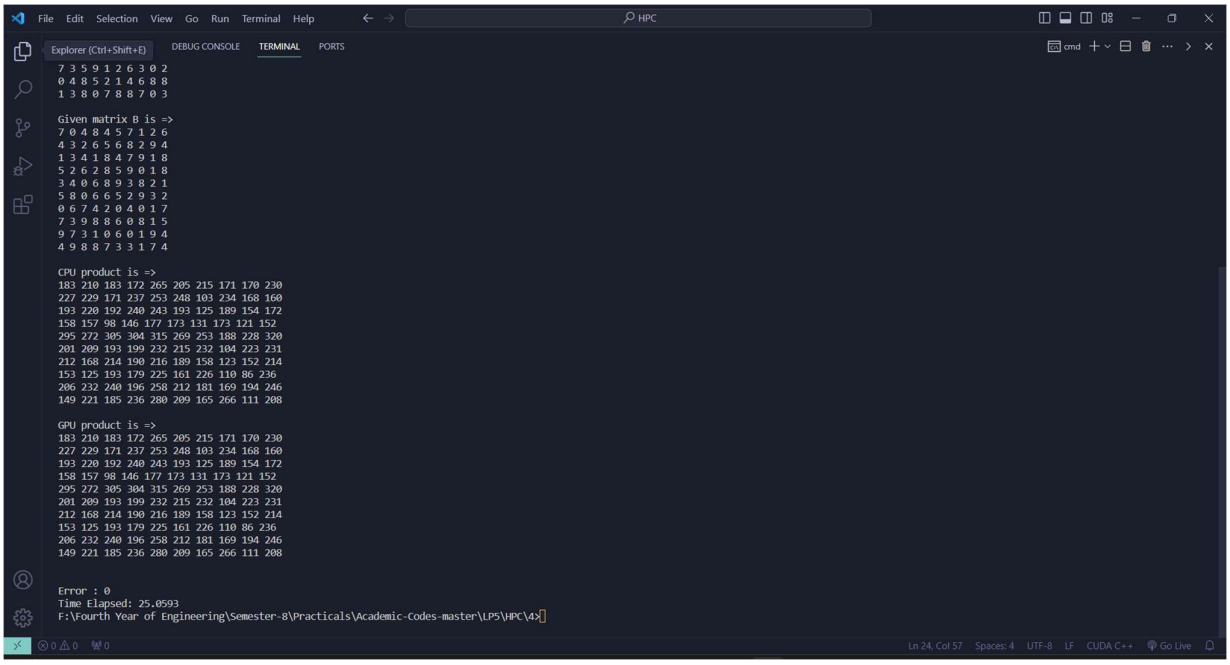
matrixMul.cu

```
#include <cmath>
#include <cstdlib>
#include <iostream>

#define checkCudaErrors(call)
\
   do {
\
      cudaError_t err = call;
\
      if (err != cudaSuccess) {
\
         printf("CUDA error at %s %d: %s\n",
__FILE__, __LINE__, cudaGetErrorString(err)); \
         exit(EXIT_FAILURE);
\
      }
\
   } while (0)

using namespace std;

// Matrix multiplication Cuda
__global__ void matrixMultiplication(int *a, int *b,
int *c, int n) {
   int row = threadIdx.y + blockDim.y * blockIdx.y;
   int col = threadIdx.x + blockDim.x * blockIdx.x;
   int sum = 0;

   if (row < n && col < n)
      for (int j = 0; j < n; j++) {
         sum = sum + a[row * n + j] * b[j * n + col];
      }

   c[n * row + col] = sum;
}

int main() {
   int *a, *b, *c;
   int *a_dev, *b_dev, *c_dev;
   int n = 10;

   a = new int[n * n];
   b = new int[n * n];
   c = new int[n * n];
   int *d = new int[n * n];
   int size = n * n * sizeof(int);
   checkCudaErrors(cudaMalloc(&a_dev, size));
   checkCudaErrors(cudaMalloc(&b_dev, size));
   checkCudaErrors(cudaMalloc(&c_dev, size));

   // Array initialization
   for (int i = 0; i < n * n; i++) {
      a[i] = rand() % 10;
      b[i] = rand() % 10;
   }

   cout << "Given matrix A is =>\n";
   for (int row = 0; row < n; row++) {
      for (int col = 0; col < n; col++) {
         cout << a[row * n + col] << " ";
      }
      cout << "\n";
   }
   cout << "\n";

   cout << "Given matrix B is =>\n";
   for (int row = 0; row < n; row++) {
      for (int col = 0; col < n; col++) {
         cout << b[row * n + col] << " ";
      }
      cout << "\n";
   }
   cout << "\n";

   cudaEvent_t start, end;

   checkCudaErrors(cudaEventCreate(&start));
   checkCudaErrors(cudaEventCreate(&end));

   checkCudaErrors(cudaMemcpy(a_dev, a, size,
cudaMemcpyHostToDevice));
   checkCudaErrors(cudaMemcpy(b_dev, b, size,
cudaMemcpyHostToDevice));

   dim3 threadsPerBlock(n, n);
   dim3 blocksPerGrid(1, 1);

   // GPU Multiplication
   checkCudaErrors(cudaEventRecord(start));
   matrixMultiplication<<<blocksPerGrid,
threadsPerBlock>>>(a_dev, b_dev, c_dev, n);

   checkCudaErrors(cudaEventRecord(end));
   checkCudaErrors(cudaEventSynchronize(end));

   float time = 0.0;
   checkCudaErrors(cudaEventElapsedTime(&time,
start, end));

   checkCudaErrors(cudaMemcpy(c, c_dev, size,
cudaMemcpyDeviceToHost));

   // CPU matrix multiplication
   int sum = 0;
```

```
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            sum = 0;
            for (int k = 0; k < n; k++) sum = sum + a[row *
n + k] * b[k * n + col];
            d[row * n + col] = sum;
        }
    }

    cout << "CPU product is =>\n";
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            cout << d[row * n + col] << " ";
        }
        cout << "\n";
    }
    cout << "\n";

    cout << "GPU product is =>\n";
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            cout << c[row * n + col] << " ";
        }
        cout << "\n";
    }
    cout << "\n";

    int error = 0;
    int _c, _d;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            _c = c[row * n + col];
            _d = d[row * n + col];
            error += _c - _d;
            if (0 != (_c - _d)) {
                cout << "Error at (" << row << ", " << col <<
") => GPU: " << _c << ", CPU: " << _d
                     << "\n";
            }
        }
    }
    cout << "\n";

    cout << "Error : " << error;
    cout << "\nTime Elapsed: " << time;

    return 0;
}
```

OUTPUT:


File Edit Selection View Go Run Terminal Help
HPC
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
cmd
F:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\4>nvcc matrixMul.cu -o b.exe
matrixMul.cu
tmpxft_00001628_00000000-10_matrixMul.cudafe1.cpp
Creating library b.lib and object b.exp
F:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\4>b.exe
Given matrix A is =>
1 4 9 8 2 5 1 1 5 7
1 2 2 1 8 7 1 9 7 5
3 2 3 1 3 7 2 7 3 9
6 0 8 0 4 6 0 0 6 3
9 4 6 6 1 4 6 7 8 9
3 9 4 7 3 1 4 0 7 7
6 1 5 5 1 0 1 6 7 7
7 3 5 9 1 2 6 3 0 2
0 4 8 5 2 1 4 6 8 8
1 3 8 0 7 8 8 7 0 3
Given matrix B is =>
7 0 4 8 4 5 7 1 2 6
4 3 2 6 5 6 8 2 9 4
1 3 4 1 8 4 7 9 1 8
5 2 6 2 8 5 9 0 1 8
3 4 0 6 8 9 3 8 2 1
5 8 0 6 6 5 2 9 3 2
0 6 7 4 2 0 4 0 1 7
7 3 9 8 8 6 0 8 1 5
9 7 3 1 0 6 0 1 9 4
4 9 8 8 7 3 3 1 7 4
CPU product is =>
183 210 183 172 265 205 215 171 170 230
227 229 171 237 253 248 103 234 168 160
193 220 192 240 243 193 125 189 154 172
158 157 98 146 177 173 131 173 121 152
295 272 305 304 315 269 253 188 228 320
201 209 193 199 232 215 232 104 223 231
212 168 214 190 216 189 158 123 152 214
153 125 193 179 225 161 226 110 86 236
206 232 240 196 258 212 181 169 194 246
149 221 185 236 280 209 165 266 111 208
GPU product is =>
183 210 183 172 265 205 215 171 170 230
227 229 171 237 253 248 103 234 168 160
Ln 24, Col 57 Spaces: 4 UTF-8 LF CUDA C++ Go Live


File Edit Selection View Go Run Terminal Help
HPC
Explorer (Ctrl+Shift+E) DEBUG CONSOLE TERMINAL PORTS
cmd
7 3 5 9 1 2 6 3 0 2
0 4 8 5 2 1 4 6 8 8
1 3 8 0 7 8 8 7 0 3
Given matrix B is =>
7 0 4 8 4 5 7 1 2 6
4 3 2 6 5 6 8 2 9 4
1 3 4 1 8 4 7 9 1 8
5 2 6 2 8 5 9 0 1 8
3 4 0 6 8 9 3 8 2 1
5 8 0 6 6 5 2 9 3 2
0 6 7 4 2 0 4 0 1 7
7 3 9 8 8 6 0 8 1 5
9 7 3 1 0 6 0 1 9 4
4 9 8 8 7 3 3 1 7 4
CPU product is =>
183 210 183 172 265 205 215 171 170 230
227 229 171 237 253 248 103 234 168 160
193 220 192 240 243 193 125 189 154 172
158 157 98 146 177 173 131 173 121 152
295 272 305 304 315 269 253 188 228 320
201 209 193 199 232 215 232 104 223 231
212 168 214 190 216 189 158 123 152 214
153 125 193 179 225 161 226 110 86 236
206 232 240 196 258 212 181 169 194 246
149 221 185 236 280 209 165 266 111 208
GPU product is =>
183 210 183 172 265 205 215 171 170 230
227 229 171 237 253 248 103 234 168 160
193 220 192 240 243 193 125 189 154 172
158 157 98 146 177 173 131 173 121 152
295 272 305 304 315 269 253 188 228 320
201 209 193 199 232 215 232 104 223 231
212 168 214 190 216 189 158 123 152 214
153 125 193 179 225 161 226 110 86 236
206 232 240 196 258 212 181 169 194 246
149 221 185 236 280 209 165 266 111 208
Error : 0
Time Elapsed: 25.0593
F:\Fourth Year of Engineering\Semester-8\Practicals\Academic-Codes-master\LP5\HPC\4>
Ln 24, Col 57 Spaces: 4 UTF-8 LF CUDA C++ Go Live