UNIVERSITY OF SALERNO

DEPARTMENT OF INFORMATION AND ELECTRICAL ENGINEERING
AND APPLIED MATHEMATICS

Parallel High performance computing

Master's Degree Program LM-32

Parallelization of algorithm and Cluster

Parallelization of the Bellman-Ford algorithm with OpenMP, MPI and CUDA and Cluster Parallelization through Apache Hadoop and Apache Spark frameworks

Prof:
Francesco Moscato
Giuseppe D'Aniello

author:
Nunzio Del Gaudio
Mat. 0612705044

YEAR 2023/2024

Contents

Part I

Algorithm parallelization (Moscato)

In this part, we will analyze two different approaches to parallelizing an algorithm. These approaches differ both in the parallelization tools used and the approach taken to parallelization. Both parallelization will be compared in terms of execution time to a sequential version (Running on the same system), varying the amount of data, the level of required optimization and parallelization degree.

Beyond the obvious performance considerations that will be made after analyzing the data, it is immediately apparent that both the technology (hardware and software) and the method used for parallelization can make parallelization more or less advantageous.

Chapter 1

Bellman-Ford algorithm

Algorithm 1 Bellman-Ford Algorithm

```
1: procedure BELLMAN-FORD(G, source)
2:     distance ← initialize array of distances to ∞
3:     distance[source] ← 0
4:     for i ← 1 to |V| − 1 do                          ▷ where V is the set of vertices
5:         for all (u, v) ∈ E do                        ▷ where E is the set of edges
6:             if distance[u] + weight(u, v) < distance[v] then
7:                 distance[v] ← distance[u] + weight(u, v)
8:             end if
9:         end for
10:    end for
11:    return distance
12: end procedure
```

Explaining graphs, shortest path algorithms, and particularly the Bellman-Ford algorithm is not the main focus of this document. That being said, it is appropriate to introduce the algorithm that I aim to parallelize.

In the realm of graph algorithms, the Bellman-Ford algorithm stands as a fundamental tool for finding the shortest paths between nodes in a weighted graph. Briefly, the Bellman-Ford algorithm addresses the "single-source shortest path" problem. In contrast to the more "straightforward" **Dijkstra**'s algorithm, the Bellman-Ford algorithm excels in scenarios where negative edge weights are present, making it a valuable tool for a broader range of applications.

However, the high computational complexity of the Bellman-Ford algorithm: **O(V * E)**, where V is the number of nodes (vertices) in the graph and E is the number of edges, it is natural to attempt parallelizing the algorithm to reduce the perceived execution time.

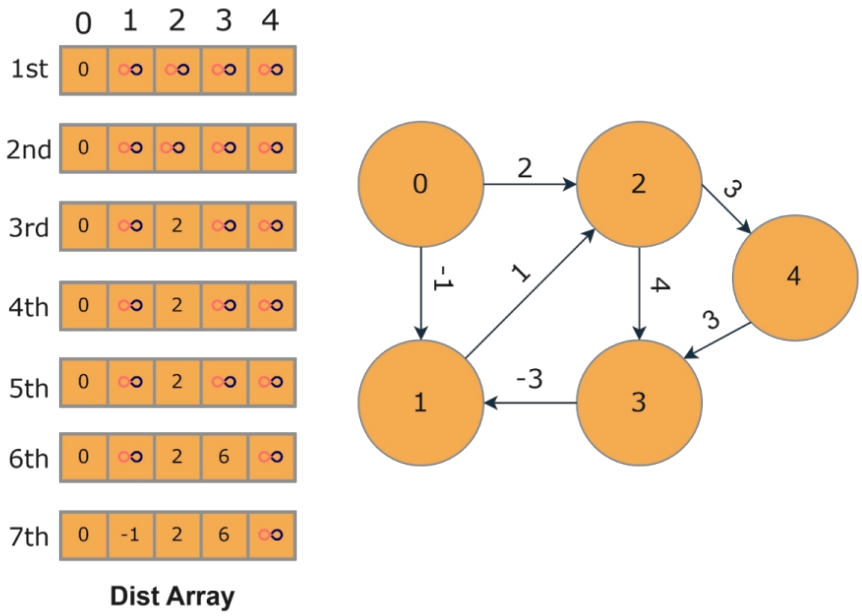


Figure 1.1: This image shows how the distance vector fills up during the execution of the algorithm.

1.1 Note: parent vector

The ultimate goal of the algorithm is to populate the distance vector, which indicates the minimum distance to reach any node from the source node. That being said, it is useful to anticipate a second vector that keeps track of the vertex traversed to reach the destination vertex.

For this reason, in both the sequential version and the parallel version with MPI+OpenMP, the necessary code has been added to keep track of the minimum path. In particular, it is sufficient to uncomment: `//#define PARENTS` in the **main.c** and **bellman-Ford.c** to compile the code necessary for this purpose.

Chapter 2

Bellman-Ford algorithm's code implementation

In this chapter, we will showcase the most interesting code sections for the project. Nevertheless, it is available the file with the complete documentation of all functions, as well as an the on-line help.

2.1 Sequential

In the sequential version, which will be compared to the parallel one.

In addition to the basic steps of the algorithm, improvements have been introduced to enhance the efficiency of the algorithm, which will be maintained in both the sequential and parallel versions:

1. The adjacency matrix is allocated using `malloc` (dynamic allocation) as a vector to allocate the entire graph in contiguous memory partitions.

2. The variable `toUpdate` has been introduced. The Bellman–Ford algorithm can be practically improved by recognizing that if an iteration of the main loop concludes without making any changes, the algorithm can be terminated immediately. Subsequent iterations will not yield further modifications. This early termination condition may result in the main loop using significantly fewer than $|V| - 1$ iterations in some cases, even though the worst-case scenario of the algorithm remains unchanged.

3. Following the same logic, the variable `iterNumber` has been introduced to check whether $V - 1$ iterations have occurred. If not, there is no need to search for negative cycles.

Listing 2.1: C

```
//To be defined if we also want to find the path of parents
//#define PARENTS

/// @brief Original Bellman-Ford algorithm.
///        In this version, in addition to finding the minimum path, it also stores
    the traversed vertices
```

```
/// @param graph The 1D graph to work on
/// @param V Number of vertices in the graph
/// @param src Starting vertex identified by a number
/// @param dist Vector of distances passed from the main
/// @param parent Vector of parents passed from the main
/// @return Presence of negative cycles
int bellmanFord_serial(int* graph, int V, int src, int * dist, int* parent) {

    int i, j, temp, count;
    int toUpdate = 0;
    int iterNumber = 0;

    // SetUp
    for (i = 0; i < V; i++) {
        dist[i] = INF;
    }
    dist[src] = 0; // dist for src to src is 0

    #ifdef PARENTS
        for (i = 0; i < V; i++) {
            parent[i] = ABSENTE_EDGE;
        }
    #endif

    // Relaxation of the strings |V| - 1 times
    for (count = 0; count < V - 1; count++) {

        toUpdate = 0; //Variable indicating whether it is necessary to proceed with
            further iterations
        iterNumber++;

        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (graph[i*V+j] != ABSENTE_EDGE) {
                    if(dist[i]!=INF){ // we must avoid adding a value to the maximum
                        representable value, otherwise it will be interpreted as a
                        negative value after the addition
                        temp = dist[i] + graph[i*V+j];
                        if (dist[j] > temp ) {
                            dist[j] = temp;

                            #ifdef PARENTS
                                parent[j] = i;
                            #endif

                            toUpdate = 1;
                        }
                    }
                }
```

```
            }
        }

        // No updates during this iteration, we're exiting early
        if (toUpdate!=1){
            break;
        }

      }

    // Simply perform another iteration and check if an even smaller path is found
        (which is not possible if there are no negative cycles)
    if(iterNumber == V-1 ){
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (graph[i*V+j] != ABSENTE_EDGE && dist[i]!=INF && dist[j] >
                    dist[i] + graph[i*V+j]) {
                    return 1;
                }
            }
        }
    }
    return 0;
}
```

2.2 OpenMP+MPI

In this chapter, we will analyze the parallelized code with OpenMP and MPI.

2.2.1 Main

The main function in the parallel version is responsible for dividing the work in reading the adjacency matrix from a binary file and writing the output, which includes the distance vector (and the parent vector if output is set to .txt by commenting out `#define binaryWriting` and uncommenting `//#define PARENTS`).

Listing 2.2: C (main reading file)

```
len = (int) V / nProcess;
myStart = len * myRank;
myEnd = len * (myRank + 1);
if (myRank == nProcess - 1) {
    myEnd = V;
}

//Reading file
char*filename_with_extension= filenameInizialzie(fileName, V);
MPI_File filempi;
int openResult = MPI_File_open(comm, filename_with_extension,
    MPI_MODE_RDONLY,MPI_INFO_NULL, &filempi);
if(openResult!=MPI_SUCCESS){
    fprintf(stderr,"opening file error\n");
    MPI_Finalize();
    exit(EXIT_FAILURE);
}

//malloc of my part of the matrix
graph1D=(int*) malloc((myEnd-myStart)*V*sizeof(int));
if(graph1D==NULL){
    fprintf(stderr,"malloc error\n");
    MPI_Finalize();
    exit(EXIT_FAILURE);
}

MPI_File_read_at_all(filempi,(myStart*V)*sizeof(int), graph1D, (myEnd-myStart)*V,
    MPI_INT, MPI_STATUS_IGNORE);

MPI_File_close(&filempi);
```

Listing 2.3: C (main writing file)

```
#ifdef binaryWriting
filename_with_extension= filenameInizialzie(fileName2, V);
openResult = MPI_File_open(comm, filename_with_extension, MPI_MODE_CREATE |
    MPI_MODE_WRONLY,MPI_INFO_NULL, &filempi);
if(openResult!=MPI_SUCCESS){
    fprintf(stderr,"opening output file error\n");
    MPI_Finalize();
    exit(EXIT_FAILURE);
}


//Each process writes a portion of the distances file (same for all processes)
MPI_File_write_ordered(filempi, &dist[myStart], (myEnd-myStart), MPI_INT,
    MPI_STATUS_IGNORE);
if(negativeCicle==1 && (myRank==0)){

    const int flag = 123456789; //"If there is a negative cycle in the binary file,
        process 0 writes a flag number
   MPI_File_write(filempi, &flag, 1, MPI_INT, MPI_STATUS_IGNORE);

}else if(negativeCicle!=0){
    fprintf(stderr,"BellmanFord error\n");
    MPI_Finalize();
    exit(EXIT_FAILURE);
}

MPI_File_close(&filempi);
```

2.2.2 Bellman-Ford parallel

This parallelization approach involves parallelizing the same steps as the sequential algorithm, initially dividing the data among different processes (each process has its own sub-vector read from the file) and subsequently among threads (division managed by OpenMP).

Firstly, OpenMP is utilized to parallelize the initialization of the distance vector (declared as shared for OpenMP). Subsequently, OpenMP is also employed to parallelize the nested for necessary for exploring the sub-array read by each process to both populating the array of distances and searching for negative cycles.

With MPI in this phase, it is crucial to ensure that at each iteration, every process has the updated distance vector through an `Allreduce` operation with the minimum value (as we seek the minimum distances). In the same way is also necessary to check if no process has updated the distance vector.

In the case of populating the `parent vector`, it becomes necessary to determine which process has found the correct parent. Since it is not guaranteed that the minimum path is associated with the vertex with the lowest ID(Each vertex is identified by the row index in the matrix, which is essentially a numerical value.), a simple `Allreduce` cannot be used. For this reason, we will implement a strategy that, in technical terms, is defined as "*pezzot*".

It is, therefore, necessary to identify the process that has found the minimum path for a specific vertex j by comparing a local distance vector with the distance vector obtained after the `Allreduce`. After identifying the process, it is essential to notify all other processes that they need to receive the correct `parent`[j] from process X (hence the `Allreduce`). The correct `parent`[j] is then transmitted via `Bcast` by process X.

Listing 2.4: C

```
//To be defined if we also want to find the path of parents
//#define PARENTS

/// @brief Parallelization of the Bellman-Ford algorithm with OpenMP and MPI.
///        In this version, in addition to finding the minimum path, it also stores
    the traversed vertices
/// @param graph The 1D graph to work on
/// @param V Number of vertices in the graph
/// @param src Starting vertex identified by a number
/// @param dist Vector of distances passed from the main
/// @param parent Vector of parents passed from the main
/// @param comm Communicator to parallelize on
/// @param myStart Starting index of this process.
/// @param myEnd Ending index of this process.
/// @return Presence of negative cycles
int bellmanFord_parallel(int* graph, int V, int src, int * dist, int* parent,
    MPI_Comm comm, int myStart, int myEnd) {

    int temp, count;
```

```
int toUpdate = 0;
int iterNumber = 0;
int hasNegativeCycle = 0;
int i,j;

//MPI
int myRank;
MPI_Comm_rank(comm, &myRank);

#ifdef PARENTS
    int * oldDist
    int x = INF;
#endif
//---

//OPEN_MP!
#pragma omp parallel for default(none) private(i) shared(V, dist)
for (i = 0; i < V; i++) {
    dist[i] = INF;
}
dist[src] = 0; // dist for src to src is 0

#ifdef PARENTS
    //OPEN_MP!
    #pragma omp parallel for default(none) private(i) shared(V, parent)
    for (i = 0; i < V; i++) {
        parent[i] = ABSENTE_EDGE;
    }

    int * oldDist = (int*)malloc(V * sizeof(int));
    if(oldDist == NULL){
        fprintf(stderr,"malloc error\n");
        MPI_Finalize();
        return -1;
    }

#endif

//Evryone Starting at the same (not essential)
MPI_Barrier(comm);

// Relaxation of the strings |V| - 1 times
for (count = 0; count < V - 1; count++) {

    toUpdate = 0; //Variable indicating whether it is necessary to proceed with
        further iterations
    iterNumber ++;

    //OpenMP!
```

```
#ifdef PARENTS
#pragma omp parallel for default(none) private(i,j,temp)
    shared(parent,oldDist,dist,graph,V,myStart,myEnd) reduction (+:toUpdate)
#else
#pragma omp parallel for default(none) private(i,j,temp)
    shared(parent,dist,graph,V,myStart,myEnd) reduction (+:toUpdate)
#endif
for (i = 0; i < myEnd-myStart; i++) { // <-- MPI! (division of matrix)
    for (j = 0; j < V; j++) {
        if (graph[i*V+j] != ABSENTE_EDGE) {
            if(dist[myStart+i]!=INF){
                temp = dist[myStart+i] + graph[i*V+j];
                if (dist[j] > temp) {
                    dist[j] = temp;

                    #ifdef PARENTS
                        oldDist[j] = temp;
                        parent[j] = myStart+i;
                    #endif

                    toUpdate++;

                }
            }
        }
    }
}

//MPI!
MPI_Allreduce(MPI_IN_PLACE, &toUpdate, 1, MPI_INT, MPI_MAX,comm); // we
    check if any process requests to continue

if (toUpdate==0){
    break;
}

//MPI!
MPI_Allreduce(MPI_IN_PLACE, dist, V, MPI_INT, MPI_MIN, comm); //we obtain
    the distance vector by assuming the minimum distances found by the
    processes are correct

#ifdef PARENTS
    //MPI! (Segment that allows us to obtain the parent vector as well)
    for (j=0; j<V; j++){
        if(dist[j]!=INF && dist[j]==oldDist[j]){ //Each process checks if it
            was the one to find the minimum path
            x=myRank;
        }
        else{
```

```
                x=INF;
            }

            MPI_Allreduce(MPI_IN_PLACE, &x, 1, MPI_INT, MPI_MIN, comm);//The
                process that found the minimum path is communicated (if it has
                happened)

            if(x!=INF){ //If at least one process has found a path
                MPI_Bcast(&parent[j],1,MPI_INT,x,comm); //That process will send
                    to all other processes the parent of the node reached with
                    the minimum path
            }

            oldDist[j]=INF;
        }
    #endif
}

// Simply perform another iteration and check if an even smaller path is found
    (which is not possible if there are no negative cycles)
if(iterNumber==V-1){
    hasNegativeCycle = 0;
    //OpenMP!
    #pragma omp parallel for default(none) private(i,j)
        shared(dist,graph,V,myStart,myEnd) reduction(+:hasNegativeCycle)
    for (i = 0; i < myEnd-myStart; i++) {
        for (j = 0; j < V; j++) {
            if (graph[i*V+j] != ABSENTE_EDGE && dist[myStart+i]!=INF && dist[j]
                > dist[myStart+i] + graph[i*V+j]) {
                hasNegativeCycle++;
            }
        }
    }
    //MPI!
    MPI_Allreduce(MPI_IN_PLACE, &hasNegativeCycle, 1, MPI_INT, MPI_MAX, comm);
}
if(hasNegativeCycle>0){
    hasNegativeCycle=1;
}

#ifdef PARENTS
    free(oldDist);
#endif

return hasNegativeCycle;
}
```

2.3 OpenMP+CUDA

In this chapter, we will analyze the parallelized code with OpenMP and CUDA .

2.3.1 Main

Unfortunately, due to linking issues, it was not possible to manage support functions in a separate file as done for the MPI and sequential versions. In the main file, only the CUDA -related functions are fully declared before the main function, while others are defined after the main function.

In the main function, there are no particular operations related to parallelization that need to be highlighted except for the definition of dimension thread block. The reason why this block size was chosen is entirely dependent on the hardware on which this algorithm was tested (we will talk about it later).

Listing 2.5: C (block dim)

```
#define MAX_DIMBLOCK 64

int blocksize = MAX_DIMBLOCK;
```

We use a useful define too:

Listing 2.6: C (define)

```
//In order to avoid having to rewrite an error check for each cuda function
#define CUDA_CHECK(call) {\
    cudaError_t cudaStatus = call;\
    if (cudaStatus != cudaSuccess) {\
        fprintf(stderr, "CUDA_ERROR: %s in file %s at line %d\nerror number:",
            cudaGetErrorString(cudaStatus), __FILE__, __LINE__ );\
        exit(EXIT_FAILURE);\
    }\
}
```

2.3.2 Edge

In contrast to the OpenMP+MPI parallelization, we opted for a completely different approach in the execution of the algorithm. Specifically, we abstracted from the adjacency matrix by envisioning the conversion of the matrix into structures called edges.

Listing 2.7: C (definition of Edge)

```c
typedef struct {
    int start; //starting vertex id
    int end;   // ending vertex id
    int weight; //edge's value
} Edge;
```

Naturally, deciding to work with a vector of edges obtained from an adjacency matrix requires a time complexity of $\mathbf{O(V \times V)}$, where V is the number of nodes (vertices) in the graph. This is because it is necessary to traverse the entire matrix to create new edges.

Listing 2.8: C (definition of Edge)

```c
    //Populating the edges vector
    int x = 0;
    for( i=0; i<V; i++){
        for( j=0; j<V; j++){
            if(host_Graph[i*V+j]!=INF){ //if edge.weight==INF the edge doesn't exist
                host_EdgeArray[x].weight=host_Graph[i*V+j];
                host_EdgeArray[x].start=i;
                host_EdgeArray[x].end=j;
                x++;
            }
        }
    }
```

2.3.3 Bellman-Ford CUDA

This function does not encompass the complete execution of the algorithm, as a significant portion of it is executed by the kernel. Instead, this function is responsible for managing the interaction between the kernel and the host.

Firstly, it clears the variable for errors using GetLastError. Then, it counts the number of edges present in the graph to allocate the correct size for the edge vector. Subsequently, it effectively populates the edge vector by iterating over the graph once more. The population of the edge vector has not been parallelized with OpenMP because it is strictly sequential (only one thread at a time could add at position x, and only one thread at a time could increment x).

It initializes the distance vector (although cudamemset would have sufficed, it was decided to initialize it on the host with subsequent copying using cudamemcopy since the source must be set to 0).

It moves the distance vector and the edge vector to the global memory of the GPU. Depending on the mode requested from the command line, it sets the sharedMemory and L1 size preferences differently and invokes the kernel up to $V - 1$ times with a different implementation. If the iterations of the loop are exactly $V - 1$, it checks for the presence of negative cycles with the same kernel. It releases the memory and terminates.

Additionally, it calculates the number of blocks needed to allocate a sufficient number of threads (each thread works on an edge).

Listing 2.9: C (definition of Edge)

```
/// @brief "Bellman-Ford function that invokes the kernel
/// @param host_Graph Adjacency matrix
/// @param V Number of vertices
/// @param src Starting vertex
/// @param host_Distances Vector of distances
/// @param blocksize Number of threads in a block
/// @param totalThread Total threads executed
/// @param tipe   0 = shared memory and L1 < sh; 1 = no shared memory and L1 > sh;
    2 = shared memory and L1 >sh
/// @return If errors have occurred
int bellmanFord_CUDA (int * host_Graph , int V, int src, int *host_Distances, int
    blocksize, int* totalThread, int tipe) {

    CUDA GetLastError();

    int host_hasNegativeCycle = 0;
    int i,j;
    Edge* device_EdgeArray;
    int *device_Distances;
    Edge* host_EdgeArray;
```

```
//OpenMP!
int edgeNumber = 0;
#pragma omp parallel for default(none) shared(host_Graph,V)
    reduction(+:edgeNumber) private (i,j)
for( i=0; i<V; i++){
    for( j=0; j<V; j++){
        if(host_Graph[i*V+j]!=INF){
            edgeNumber++; //We count the number of edges actually present in the
                adjacency matrix
        }
    }
}

int size_Distances = V * sizeof(int);
int size_Graph = edgeNumber * sizeof(Edge);

//Allocation of the edges vector
host_EdgeArray = (Edge*)malloc(edgeNumber* sizeof(Edge));
if (host_EdgeArray == NULL) {
    fprintf(stderr, "edge malloc error\n");
    return -1;
}

//Populating the edges vector
int x = 0;
for( i=0; i<V; i++){
    for( j=0; j<V; j++){
        if(host_Graph[i*V+j]!=INF){
            host_EdgeArray[x].weight=host_Graph[i*V+j];
            host_EdgeArray[x].start=i;
            host_EdgeArray[x].end=j;
            x++;
        }
    }
}

//OpenMP!
#pragma omp parallel for default(none) shared(host_Distances,V) private(i)
for(i=0; i<V; i++) {
    host_Distances[i] = INF;
}
host_Distances[src] = 0;

dim3 threadForBlocks(blocksize);
dim3 blocksForGrid(((edgeNumber-1) / blocksize)+1); //we allocate a sufficient
    number of blocks so that each thread can work with an edge
*totalThread = (threadForBlocks.x)*(blocksForGrid.x);

//CUDA ! maclloc
```

```
CUDA _CHECK( CUDA Malloc((void**)&device_Distances, size_Distances) );
CUDA _CHECK( CUDA Malloc((void**)&device_EdgeArray, size_Graph) );

//copy
CUDA _CHECK( CUDA Memcpy(device_EdgeArray, host_EdgeArray, size_Graph, CUDA
    MemcpyHostToDevice) );
CUDA _CHECK( CUDA Memcpy(device_Distances, host_Distances, size_Distances, CUDA
    MemcpyHostToDevice) );

int iterNumber=0;
int host_updated=0;

if (tipe==0){ //ShMem > L1; yes shared
    CUDA FuncSetCacheConfig(bellmanFord_kernel, CUDA FuncCachePreferShared);
    for(int i=0; i<V-1; i++){//V-1 kernell calling
        bellmanFord_kernel <<< blocksForGrid, threadForBlocks >>> (edgeNumber,
            device_EdgeArray, device_Distances);

        CUDA _CHECK( CUDA DeviceSynchronize() );
        iterNumber++;
        CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
        if(host_updated==0){
            break;
        }
    }
}else if(tipe==1){ //ShMem < L1; No shared
    CUDA FuncSetCacheConfig(bellmanFord_kernel, CUDA FuncCachePreferL1);
    for(int i=0; i<V-1; i++){//V-1 kernell calling
        bellmanFord_kernel_noShared <<< blocksForGrid, threadForBlocks >>>
            (edgeNumber, device_EdgeArray, device_Distances);

        CUDA _CHECK( CUDA DeviceSynchronize() );
        iterNumber++;
        CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
        if(host_updated==0){
            break;
        }
    }
}else if(tipe==2){ //ShMem < L1; yes shared
    CUDA FuncSetCacheConfig(bellmanFord_kernel, CUDA FuncCachePreferL1);
    for(int i=0; i<V-1; i++){//V-1 kernell calling
        bellmanFord_kernel <<< blocksForGrid, threadForBlocks >>> (edgeNumber,
            device_EdgeArray, device_Distances);

        CUDA _CHECK( CUDA DeviceSynchronize() );
        iterNumber++;
        CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
        if(host_updated==0){
            break;
```

```
        }
    }
}else{
    fprintf(stderr, "CUDA tipe not allowed\n");
    return -1;
}

//Similarly, for the distance search, I opted not to use the define for
    improved code readability.
if(iterNumber==V-1){
    host_hasNegativeCycle = 1;
    if (tipe==0){
            bellmanFord_kernel <<< blocksForGrid, threadForBlocks >>>
                (edgeNumber, device_EdgeArray, device_Distances);

            CUDA _CHECK( CUDA DeviceSynchronize() );
            iterNumber++;
            CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
            if(host_updated==0){
                host_hasNegativeCycle=0;
            }
    }else if(tipe==1){
            bellmanFord_kernel_noShared <<< blocksForGrid, threadForBlocks >>>
                (edgeNumber, device_EdgeArray, device_Distances);

            CUDA _CHECK( CUDA DeviceSynchronize() );
            iterNumber++;
            CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
            if(host_updated==0){
                host_hasNegativeCycle=0;
            }
    }else if(tipe==2){ //ShMem < L1; yes shared
        bellmanFord_kernel <<< blocksForGrid, threadForBlocks >>> (edgeNumber,
            device_EdgeArray, device_Distances);

            CUDA _CHECK( CUDA DeviceSynchronize() );
            iterNumber++;
            CUDA MemcpyFromSymbol(&host_updated, Updated, sizeof(int));
            if(host_updated==0){
                host_hasNegativeCycle=0;
            }
    }
}

// copy-back of the result
if(host_hasNegativeCycle=0)
CUDA _CHECK( CUDA Memcpy(host_Distances, device_Distances, size_Distances, CUDA
    MemcpyDeviceToHost) );
```

```
    // deallocation
    CUDA _CHECK( CUDA Free(device_EdgeArray) );
    CUDA _CHECK( CUDA Free(device_Distances) );

    free(host_EdgeArray);

    return host_hasNegativeCycle;
}
```

2.3.4 Bellman-Ford Kernel

First and foremost, it is necessary to specify what could have been done and what has been done. Summarizing, it would have been desirable to test the kernel with different types of memory, in addition to global: shared, constant, and texture.

The kernel has been implemented using **shared memory**, but it should be noted that shared memory is used solely as a buffer and is not exploited for simultaneous access by different threads to the same data. **Constant memory**, on the other hand, was not usable for hardware reasons (the number of edges tested by the program is too high to allow the use of constant memory), although the algorithm could have benefited from it, given that the array of edges is read-only. Finally, **texture memory** was not used because each thread accesses only one element of the edge array, never accessing contiguous portions of memory beyond its own interest. In practice, no memory type proved to be ideal for the implemented kernel.

In both the shared memory and non-shared memory versions, each thread does nothing but read an element from the edge vector before proceeding with the relaxation of the edge. A 'do-while' loop is present to avoid write conflicts in the distance vector. The purpose of the 'do-while' loop is to anticipate that the distance read initially may have already been updated by another thread.

Listing 2.10: C (no shared)

```
__device__ int Updated;

/// @brief Kernel without the use of shared memory
/// @param edgeNumber Number of edges (excluding infinite weight)
/// @param device_EdgeArray Pointer to the edges
/// @param device_Distances Pointer to the distances
/// @return none
__global__ void bellmanFord_kernel_noShared ( int edgeNumber, Edge *
    device_EdgeArray, int * device_Distances) {

    int oldval, newval;
    unsigned int global_Id = blockIdx.x * blockDim.x + threadIdx.x;

    Updated=0;

    const unsigned int i = device_EdgeArray[global_Id].start;
```

```
    const unsigned int j = device_EdgeArray[global_Id].end;

    if(device_Distances[i]!=INF && global_Id < edgeNumber){
        if( device_Distances[j]> device_Distances[i] +
            device_EdgeArray[global_Id].weight ) {

            Updated=1;

            do { //for device_Distances
                oldval = device_Distances[j];
                newval= device_Distances[i] + device_EdgeArray[global_Id].weight;
                newval=fminf(oldval, newval);

            } while( (atomicCAS((int*)&device_Distances[j], oldval, newval) !=
                newval) );
        }
    }
}
```

The size of shared memory is allocated for each block of the maximum block size (ensuring that we do not encounter issues of excessive memory occupancy). Each thread allocates its element from the edge vector in shared memory (it is necessary for the distance vector to be common to all threads in the kernel).

Listing 2.11: C (shared)

```
__device__ int Updated;

/// @brief Kernel with the use of shared memory for the edge
/// @param edgeNumber same
/// @param device_EdgeArray same
/// @param device_Distances same
/// @return none
__global__ void bellmanFord_kernel ( int edgeNumber, Edge * device_EdgeArray, int *
    device_Distances) {

    int oldval, newval;
    unsigned int global_Id = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int local_Id = threadIdx.x;

    Updated=0;

    //In this case, shared memory is used as a buffer
    //and not to share data between threads within the same block
    __shared__ Edge buffer[MAX_DIMBLOCK];


    if(global_Id < edgeNumber) {
        buffer[local_Id] = device_EdgeArray[global_Id];
    }
```

```
    __syncthreads();

    const unsigned int i = buffer[local_Id].start;
    const unsigned int j = buffer[local_Id].end;


    if(device_Distances[i]!=INF && global_Id < edgeNumber){
        if( device_Distances[j]> device_Distances[i] + buffer[local_Id].weight ) {

            Updated=1;

            do { //for device_Distances
                oldval = device_Distances[j];
                newval= device_Distances[i] + buffer[local_Id].weight;
                newval=fminf(oldval, newval);
            } while( (atomicCAS((int*)&device_Distances[j], oldval, newval) !=
                newval) );
        }
    }
}
```

Chapter 3

Performance evaluation

In this chapter, we will evaluate the parallelization in terms of execution time. The tests will be conducted on an adjacency matrix randomly generated by a specifically created C program.

The considered graphs include both a 20% and 80% **edge density**. Additionally, graphs with 4000, 8000, and 12000 **vertices** are taken into account.

Both the sequential and parallel versions are compiled with different **GCC optimization levels** (-O0, -O1, -O2, -O3).

For the parallel versions, we also vary the number of **OpenMP threads** (for both CUDA and MPI versions), the number of **MPI processes**, and the type of kernel and memory settings used **(type 0: priority to shared memory with a kernel using shared memory, type 1: priority to L1 with a kernel without shared memory, and type 2: priority to L1 with a kernel using shared memory)**.

3.1 Test hardware

First and foremost, it is, of course, necessary to take into account the hardware on which any performance evaluations will be conducted. Initially, the MPI tests were performed on a platform different from those with CUDA. Since I do not have a computer with an Nvidia graphics card, it was necessary to leverage the Google Colab platform `https://colab.google/`.

It should be noted that the reference sequential version was tested on the same platform as the corresponding parallel version. In practice, the same sequential tests were carried out both on Colab and on the local computer.

3.1.1 OpenMP+MPI hardware

```
CPU
--------------------------------------------------------------------------
Number of cores       6 (max 6)
Number of threads     12 (max 12)
Number of CCDs        0
Manufacturer          AuthenticAMD
Name                  AMD Ryzen 5 1600X
Codename              Summit Ridge
Specification         AMD Ryzen 5 1600X Six-Core Processor
Package               Socket AM4 (1331)
CPUID                 F.1.1
Extended CPUID        17.1
Core Stepping         ZP-B1
Technology            14 nm
TDP Limit             95.0 Watts
Tjmax                 75.0 °C
Core Speed            3530.9 MHz
Multiplier x Bus Speed   36.0 x 98.1 MHz
Base frequency (cores)   98.1 MHz
Base frequency (mem.)    98.1 MHz
Instructions sets     MMX (+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A,
                      x86-64, AES, AVX, AVX2, FMA3, SHA
Microcode Revision    0x8001138
L1 Data cache         6 x 32 KB (8-way, 64-byte line)
L1 Instruction cache  6 x 64 KB (4-way, 64-byte line)
L2 cache              6 x 512 KB (8-way, 64-byte line)
L3 cache              2 x 8 MB (16-way, 64-byte line)

Clock Speed 0         3530.86 MHz (Core #0)
Clock Speed 1         3530.86 MHz (Core #1)
Clock Speed 2         3530.86 MHz (Core #2)
Clock Speed 3         3530.86 MHz (Core #3)
Clock Speed 4         3530.86 MHz (Core #4)
Clock Speed 5         3530.86 MHz (Core #5)
Core 0 max ratio      (effective) 36.00
Core 1 max ratio      (effective) 36.00
Core 2 max ratio      (effective) 36.00
Core 3 max ratio      (effective) 36.00
Core 4 max ratio      (effective) 36.00
Core 5 max ratio      (effective) 36.00
```

```
Memory SPD
------------------------------------------------------------------------

DIMM # 1
    SMBus address 0x52
    Memory type DDR4
    Module format UDIMM
    Module Manufacturer(ID) Corsair (7F7F9E00000000000000000000)
    SDRAM Manufacturer (ID) SK Hynix (AD00000000000000000000000000)
    Size 8192 MBytes
    Max bandwidth DDR4-3200 (1600 MHz)
    Max JEDEC DDR4-2133 (1066 MHz)
    Part number CMU16GX4M2C3200C16
    Nominal Voltage 1.20 Volts
    EPP no
    XMP yes, rev. 2.0
    AMP no
    EXPO no
JEDEC timings table CL-tRCD-tRP-tRAS-tRC @ frequency
    JEDEC #1 9.0-9-10-22-31 @ 666 MHz
    JEDEC #2 10.0-10-11-25-35 @ 733 MHz
    JEDEC #3 11.0-11-12-27-38 @ 800 MHz
    JEDEC #4 12.0-13-13-30-42 @ 900 MHz
    JEDEC #5 13.0-14-14-32-45 @ 966 MHz
    JEDEC #6 14.0-14-15-35-49 @ 1033 MHz
    JEDEC #7 15.0-15-15-36-50 @ 1066 MHz
    JEDEC #8 16.0-15-15-36-50 @ 1066 MHz
XMP profile XMP-3200
    Specification DDR4-3200
    VDD Voltage 1.350 Volts
    Min Cycle time 0.625 ns (1600 MHz)
    Max CL 16.0
    Min tRP 11.24 ns
    Min tRCD 11.24 ns
    Min tRAS 22.36 ns
    Min tRC 33.74 ns
    Min tRRD 3.74 ns
XMP timings table CL-tRCD-tRP-tRAS-tRC-CR @ frequency (voltage)
    XMP #1 9.0-11-11-21-31-n.a @ 900 MHz (1.350 Volts)
    XMP #2 10.0-12-12-23-34-n.a @ 1001 MHz (1.350 Volts)
    XMP #3 11.0-13-13-25-38-n.a @ 1101 MHz (1.350 Volts)
    XMP #4 12.0-14-14-27-41-n.a @ 1201 MHz (1.350 Volts)
    XMP #5 13.0-15-15-30-44-n.a @ 1301 MHz (1.350 Volts)
    XMP #6 14.0-16-16-32-48-n.a @ 1401 MHz (1.350 Volts)
    XMP #7 15.0-17-17-34-51-n.a @ 1501 MHz (1.350 Volts)
    XMP #8 16.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
```

```
    XMP #9 17.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
    XMP #10 18.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
    XMP #11 19.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
    XMP #12 20.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)

DIMM # 2
    SMBus address 0x53
    Memory type DDR4
    Module format UDIMM
    Module Manufacturer(ID) Corsair (7F7F9E00000000000000000000)
    SDRAM Manufacturer (ID) SK Hynix (AD000000000000000000000000)
    Size 8192 MBytes
    Max bandwidth DDR4-3200 (1600 MHz)
    Max JEDEC DDR4-2133 (1066 MHz)
    Part number CMU16GX4M2C3200C16
    Nominal Voltage 1.20 Volts
    EPP no
    XMP yes, rev. 2.0
    AMP no
    EXPO no
JEDEC timings table CL-tRCD-tRP-tRAS-tRC @ frequency
    JEDEC #1 9.0-9-10-22-31 @ 666 MHz
    JEDEC #2 10.0-10-11-25-35 @ 733 MHz
    JEDEC #3 11.0-11-12-27-38 @ 800 MHz
    JEDEC #4 12.0-13-13-30-42 @ 900 MHz
    JEDEC #5 13.0-14-14-32-45 @ 966 MHz
    JEDEC #6 14.0-14-15-35-49 @ 1033 MHz
    JEDEC #7 15.0-15-15-36-50 @ 1066 MHz
    JEDEC #8 16.0-15-15-36-50 @ 1066 MHz
XMP profile XMP-3200
    Specification DDR4-3200
    VDD Voltage 1.350 Volts
    Min Cycle time 0.625 ns (1600 MHz)
    Max CL 16.0
    Min tRP 11.24 ns
    Min tRCD 11.24 ns
    Min tRAS 22.36 ns
    Min tRC 33.74 ns
    Min tRRD 3.74 ns
XMP timings table CL-tRCD-tRP-tRAS-tRC-CR @ frequency (voltage)
    XMP #1 9.0-11-11-21-31-n.a @ 900 MHz (1.350 Volts)
    XMP #2 10.0-12-12-23-34-n.a @ 1001 MHz (1.350 Volts)
    XMP #3 11.0-13-13-25-38-n.a @ 1101 MHz (1.350 Volts)
    XMP #4 12.0-14-14-27-41-n.a @ 1201 MHz (1.350 Volts)
    XMP #5 13.0-15-15-30-44-n.a @ 1301 MHz (1.350 Volts)
    XMP #6 14.0-16-16-32-48-n.a @ 1401 MHz (1.350 Volts)
```

```
        XMP #7 15.0-17-17-34-51-n.a @ 1501 MHz (1.350 Volts)
        XMP #8 16.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
        XMP #9 17.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
        XMP #10 18.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
        XMP #11 19.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
        XMP #12 20.0-18-18-36-54-n.a @ 1600 MHz (1.350 Volts)
```

3.1.2 CUDA hardware

```
    CPU
    ------------------------------------------------------------------------
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         46 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  2
  On-line CPU(s) list:   0,1
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Xeon(R) CPU @ 2.30GHz
    CPU family:          6
    Model:               63
    Thread(s) per core:  2
    Core(s) per socket:  1
    Socket(s):           1
    Stepping:            0
    BogoMIPS:            4599.99
    Flags:               fpu vme de pse tsc msr pae mce cx8
                         apic sep mtrr pge mca cmov pat pse36 clf
                         lush mmx fxsr sse sse2 ss ht syscall nx
                         pdpe1gb rdtscp lm constant_tsc rep_
                         good nopl xtopology nonstop_tsc cpuid
                         tsc_known_freq pni pclmulqdq ssse3 fm
                         a cx16 pcid sse4_1 sse4_2 x2apic movbe
                         popcnt aes xsave avx f16c rdrand hyp
                         ervisor lahf_lm abm invpcid_single ssbd
                         ibrs ibpb stibp fsgsbase tsc_adjust
                         bmi1 avx2 smep bmi2 erms invpcid xsaveopt
                         arat md_clear arch_capabilities
Virtualization features:
  Hypervisor vendor:     KVM
  Virtualization type:   full
Caches (sum of all):
  L1d:                   32 KiB (1 instance)
  L1i:                   32 KiB (1 instance)
  L2:                    256 KiB (1 instance)
  L3:                    45 MiB (1 instance)
```

```
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0,1
Vulnerabilities:
  Gather data sampling:  Not affected
  Itlb multihit:         Not affected
  L1tf:                  Mitigation; PTE Inversion
  Mds:                   Vulnerable; SMT Host state unknown
  Meltdown:              Vulnerable
  Mmio stale data:       Vulnerable
  Retbleed:              Vulnerable
  Spec rstack overflow:  Not affected
  Spec store bypass:     Vulnerable
  Spectre v1:            Vulnerable: __user pointer sanitization and usercopy
                         barriers only; no swap gs barriers
  Spectre v2:            Vulnerable, IBPB: disabled, STIBP: disabled,
                         PBRSB-eIBRS: Not affected
  Srbds:                 Not affected
  Tsx async abort:       Not affected


  GPU
  ------------------------------------------------------------------------------
  +-----------------------------------------------------------------------------+
  | NVIDIA-SMI 535.104.05      Driver Version: 535.104.05   CUDA Version: 12.2     |
  |-----------------------------------------+----------------------+--------------+
  | GPU  Name          Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
  | Fan  Temp   Perf   Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
  |                                         |                      |        MIG M. |
  |=========================================+======================+==============|
  |   0  Tesla T4                       Off | 00000000:00:04.0 Off |             0 |
  | N/A   48C    P8       10W /  70W |      0MiB / 15360MiB |      0%      Default |
  |                                         |                      |                 N/A |
  +-----------------------------------------+----------------------+--------------+

  +-----------------------------------------------------------------------------+
  | Processes:                                                                  |
  |  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
  |        ID   ID                                                   Usage      |
  |=============================================================================|
  |  No running processes found                                                 |
  +-----------------------------------------------------------------------------+
```

Remember that the Tesla T4 card has a computer capability 7.5.

	Compute Capability													
Technical Specifications	5.0	5.2	5.3	6.0	6.1	6.2	7.0	7.2	7.5	8.0	8.6	8.7	8.9	9.0
Maximum number of resident grids per device (Concurrent Kernel Execution)	32		16	128	32	16	128	16	128					
Maximum dimensionality of grid of thread blocks	3													
Maximum x -dimension of a grid of thread blocks	$2^{31}-1$													
Maximum y- or z-dimension of a grid of thread blocks	65535													
Maximum dimensionality of thread block	3													
Maximum x- or y-dimensionality of a block	1024													
Maximum z-dimension of a block	64													
Maximum number of threads per block	1024													
Warp size	32													
Maximum number of resident blocks per SM	32								16	32	16		24	32
Maximum number of resident warps per SM	64								32	64	48			64
Maximum number of resident threads per SM	2048								1024	2048	1536			2048
Number of 32-bit registers per SM	64 K													
Maximum number of 32-bit registers per thread block	64 K		32 K	64 K		32 K	64 K							
Maximum number of 32-bit registers per thread	255													
Maximum amount of shared memory per SM	64 KB	96 KB	64 KB		96 KB	64 KB	96 KB		64 KB	164 KB	100 KB	164 KB	100 KB	228 KB
Maximum amount of shared memory per thread block [32]	48 KB						96 KB	96 KB	64 KB	163 KB	99 KB	163 KB	99 KB	227 KB
Number of shared memory banks	32													
Maximum amount of local memory per thread	512 KB													
Constant memory size	64 KB													
Cache working set per SM for constant memory	8 KB			4 KB	8 KB									
Cache working set per SM for texture memory	Between 12 KB and 48 KB			Between 24 KB and 48 KB			32 ~ 128 KB		32 or 64 KB	28 KB ~ 192 KB	28 KB ~ 128 KB	28 KB ~ 192 KB	28 KB ~ 128 KB	28 KB ~ 256 KB
Maximum width for a 1D texture reference bound to a CUDA array	65536			131072										
Maximum width for a 1D texture reference bound to linear memory	2^{27}			2^{28}	2^{27}		2^{28}	2^{27}	2^{28}					

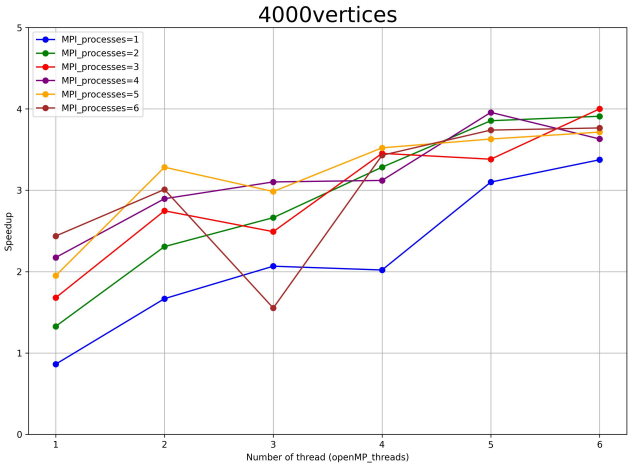
In this phase, it is possible to specify the reason why a block size of 64 was set. As you can see, the maximum number of threads per SM and per block is 1024. This means that we could have allocated 1024 threads per block to occupy 100% of the SM (being certain that the threads would have had enough registers available per block). However, since up to 16 blocks per SM can be allocated, and especially using shared memory more as a buffer than as a means of sharing data between threads, the decision was made to allocate not only the maximum number of threads but also the maximum number of blocks per SM (i.e., 16 blocks x 64 threads = 1024 threads)

3.2 OpenMP+MPI

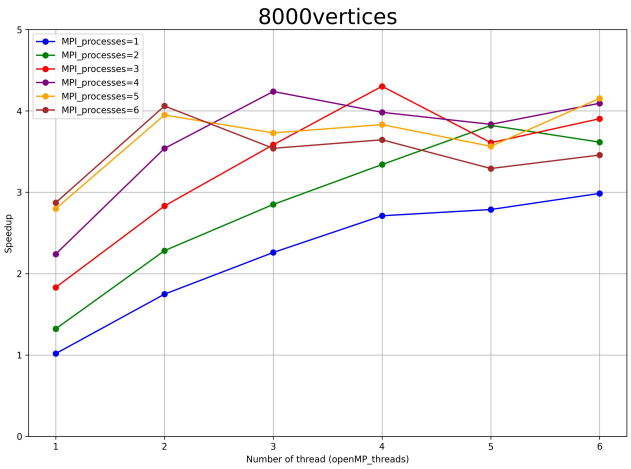
3.2.1 Densely connected graph (80%)

A densely connected graph example was taken, with 80% of edges, meaning a graph whose adjacency matrix is filled with 80%.

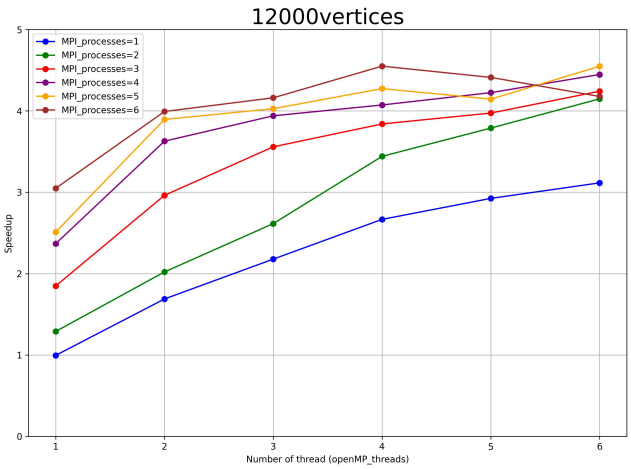
Optimization 0



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.265642	0.225295	0.028157	0.006403	1.000.000	100.000.000
OMP+MPI	1	1	0.308044	0.2638	0.032467	0.003209	0.862352	86.235.223
OMP+MPI	1	2	0.200192	0.172128	0.01814	0.005291	1.326.933	66.346.641
OMP+MPI	1	3	0.15822	0.129842	0.016225	0.008003	1.678.941	55.964.691
OMP+MPI	1	4	0.122227	0.091256	0.015453	0.012562	2.173.341	54.333.517
OMP+MPI	1	5	0.136215	0.093247	0.023106	0.016914	1.950.160	39.003.197
OMP+MPI	1	6	0.108984	0.071188	0.016578	0.016931	2.437.429	40.623.820
OMP+MPI	2	1	0.159304	0.123688	0.027121	0.001762	1.667.511	83.375.548
OMP+MPI	2	2	0.115123	0.087632	0.017371	0.005056	2.307.462	57.686.561
OMP+MPI	2	3	0.096696	0.069412	0.016136	0.007249	2.747.201	45.786.688
OMP+MPI	2	4	0.091727	0.060078	0.017608	0.010817	2.896.007	36.200.083
OMP+MPI	2	5	0.080918	0.048608	0.015612	0.01304	3.282.854	32.828.542
OMP+MPI	2	6	0.088272	0.052101	0.017041	0.014521	3.009.374	25.078.121
OMP+MPI	3	1	0.128549	0.090392	0.028902	0.001893	2.066.465	68.882.164
OMP+MPI	3	2	0.099751	0.070877	0.018217	0.005213	2.663.051	44.384.183
OMP+MPI	3	3	0.10664	0.071084	0.020951	0.008797	2.491.017	27.677.961
OMP+MPI	3	4	0.085629	0.054388	0.017256	0.010913	3.102.225	25.851.877
OMP+MPI	3	5	0.089018	0.055747	0.017611	0.012967	2.984.138	19.894.254
OMP+MPI	3	6	0.171124	0.067517	0.017149	0.075395	1.552.341	8.624.116
OMP+MPI	4	1	0.131512	0.090004	0.031435	0.002093	2.019.907	50.497.673
OMP+MPI	4	2	0.080904	0.054966	0.017098	0.004424	3.283.402	41.042.525
OMP+MPI	4	3	0.07693	0.049666	0.015934	0.00699	3.453.013	28.775.107
OMP+MPI	4	4	0.085123	0.056275	0.01574	0.009862	3.120.684	19.504.276
OMP+MPI	4	5	0.075441	0.045336	0.014709	0.012004	3.521.189	17.605.944
OMP+MPI	4	6	0.077456	0.044489	0.015422	0.014068	3.429.608	14.290.033
OMP+MPI	5	1	0.085676	0.053448	0.023858	0.001717	3.100.560	62.011.193
OMP+MPI	5	2	0.068919	0.043704	0.015912	0.004334	3.854.409	38.544.088
OMP+MPI	5	3	0.078555	0.053043	0.014392	0.007274	3.381.605	22.544.035
OMP+MPI	5	4	0.067136	0.039886	0.014717	0.009101	3.956.774	19.783.872
OMP+MPI	5	5	0.07319	0.043852	0.014798	0.011767	3.629.485	14.517.940
OMP+MPI	5	6	0.071043	0.041317	0.014359	0.012979	3.739.198	12.463.995
OMP+MPI	6	1	0.078721	0.04537	0.025005	0.001687	3.374.474	56.241.240
OMP+MPI	6	2	0.067953	0.043789	0.015566	0.00424	3.909.202	32.576.683
OMP+MPI	6	3	0.066409	0.041805	0.014103	0.006668	4.000.120	22.222.891
OMP+MPI	6	4	0.073174	0.044893	0.015217	0.009723	3.630.279	15.126.160
OMP+MPI	6	5	0.071515	0.041705	0.015127	0.011481	3.714.493	12.381.645
OMP+MPI	6	6	0.070545	0.0402	0.014519	0.012701	3.765.541	10.459.837

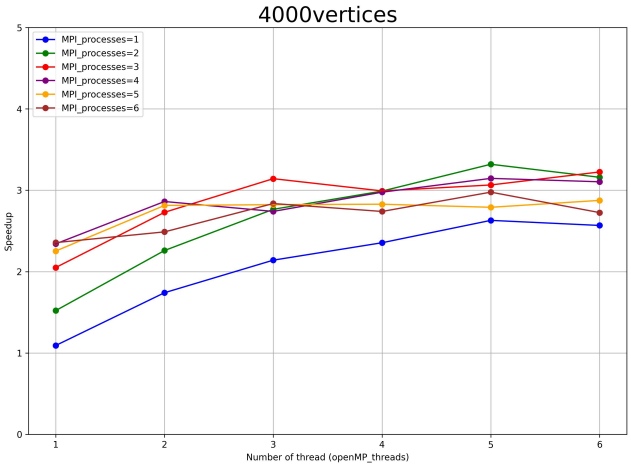


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.74459	0.608018	0.097138	0.012362	1.000.000	100.000.000
OMP+MPI	1	1	0.73233	0.613958	0.090883	0.002204	1.016.741	101.674.108
OMP+MPI	1	2	0.563724	0.479542	0.062975	0.004675	1.320.841	66.042.056
OMP+MPI	1	3	0.40687	0.330686	0.054666	0.007161	1.830.045	61.001.507
OMP+MPI	1	4	0.332464	0.256066	0.0548	0.009446	2.239.616	55.990.394
OMP+MPI	1	5	0.266162	0.197255	0.047134	0.010813	2.797.509	55.950.173
OMP+MPI	1	6	0.259359	0.180477	0.05796	0.011789	2.870.882	47.848.032
OMP+MPI	2	1	0.426146	0.308714	0.091467	0.001628	1.747.268	87.363.412
OMP+MPI	2	2	0.32629	0.24703	0.058568	0.004032	2.281.986	57.049.661
OMP+MPI	2	3	0.262995	0.192296	0.050398	0.006278	2.831.191	47.186.517
OMP+MPI	2	4	0.2104	0.141215	0.048792	0.008545	3.538.928	44.236.603
OMP+MPI	2	5	0.188541	0.116625	0.051641	0.010315	3.949.223	39.492.232
OMP+MPI	2	6	0.1834	0.111541	0.050373	0.012046	4.059.926	33.832.720
OMP+MPI	3	1	0.329511	0.21135	0.092	0.001675	2.259.683	75.322.776
OMP+MPI	3	2	0.261276	0.178029	0.061768	0.004112	2.849.824	47.497.059
OMP+MPI	3	3	0.20778	0.123729	0.063706	0.007146	3.583.561	39.817.344
OMP+MPI	3	4	0.175686	0.108362	0.048167	0.007701	4.238.189	35.318.243
OMP+MPI	3	5	0.199667	0.130905	0.048175	0.010174	3.729.152	24.861.015
OMP+MPI	3	6	0.210337	0.139859	0.050065	0.011688	3.539.988	19.666.601
OMP+MPI	4	1	0.274679	0.156851	0.092119	0.001752	2.710.766	67.769.151
OMP+MPI	4	2	0.222908	0.141252	0.059639	0.004255	3.340.349	41.754.362
OMP+MPI	4	3	0.17307	0.103441	0.050155	0.005979	4.302.251	35.852.088
OMP+MPI	4	4	0.186972	0.119789	0.047275	0.007893	3.982.364	24.889.773
OMP+MPI	4	5	0.19434	0.119385	0.053212	0.011647	3.831.371	19.156.854
OMP+MPI	4	6	0.204305	0.130657	0.0522	0.011614	3.644.505	15.185.436
OMP+MPI	5	1	0.267107	0.140938	0.101251	0.001641	2.787.606	55.752.122
OMP+MPI	5	2	0.194851	0.113957	0.060462	0.004012	3.821.333	38.213.327
OMP+MPI	5	3	0.206319	0.130172	0.055726	0.005763	3.608.937	24.059.581
OMP+MPI	5	4	0.194177	0.118681	0.0556	0.008119	3.834.597	19.172.984
OMP+MPI	5	5	0.208913	0.142869	0.045945	0.010155	3.564.126	14.256.505
OMP+MPI	5	6	0.226208	0.151456	0.053586	0.012382	3.291.626	10.972.087
OMP+MPI	6	1	0.24939	0.125516	0.09634	0.001731	2.985.647	49.760.783
OMP+MPI	6	2	0.20592	0.12464	0.059115	0.003952	3.615.921	30.132.677
OMP+MPI	6	3	0.190761	0.118818	0.05289	0.005884	3.903.264	21.684.799
OMP+MPI	6	4	0.181898	0.115282	0.047377	0.007788	4.093.451	17.056.045
OMP+MPI	6	5	0.179289	0.110253	0.048923	0.009681	4.153.030	13.843.433
OMP+MPI	6	6	0.215392	0.141316	0.052399	0.011699	3.456.901	9.602.502

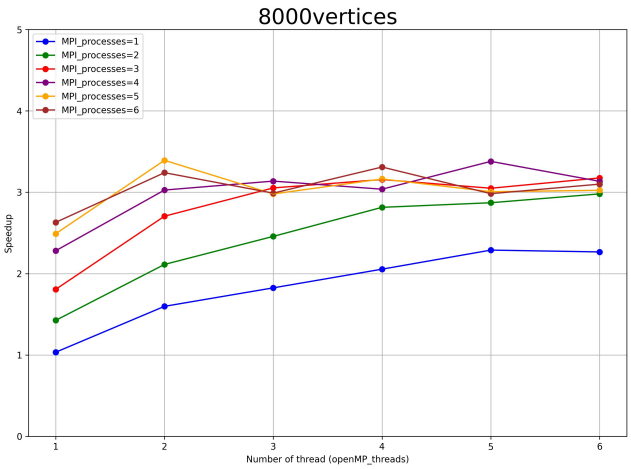


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.666.361	1.368.552	0.219467	0.016838	1.000.000	100.000.000
OMP+MPI	1	1	1.674.162	1.396.410	0.214106	0.002133	0.99534	99.533.976
OMP+MPI	1	2	1.292.755	1.112.570	0.133768	0.004672	1.288.999	64.449.948
OMP+MPI	1	3	0.901706	0.743563	0.118764	0.006952	1.848.009	61.600.289
OMP+MPI	1	4	0.703809	0.560532	0.107517	0.008793	2.367.632	59.190.793
OMP+MPI	1	5	0.663841	0.516854	0.112665	0.011409	2.510.182	50.203.641
OMP+MPI	1	6	0.54669	0.398642	0.114027	0.013905	3.048.088	50.801.459
OMP+MPI	2	1	0.987089	0.701692	0.218478	0.001951	1.688.155	84.407.772
OMP+MPI	2	2	0.824705	0.625552	0.155466	0.004883	2.020.553	50.513.835
OMP+MPI	2	3	0.562591	0.398645	0.125552	0.007608	2.961.939	49.365.658
OMP+MPI	2	4	0.459362	0.310447	0.113452	0.009306	3.627.554	45.344.426
OMP+MPI	2	5	0.427782	0.275238	0.117463	0.012952	3.895.345	38.953.452
OMP+MPI	2	6	0.417406	0.269599	0.11092	0.015306	3.992.181	33.268.179
OMP+MPI	3	1	0.765019	0.471318	0.222508	0.001683	2.178.195	72.606.497
OMP+MPI	3	2	0.63737	0.432532	0.161162	0.004654	2.614.432	43.573.866
OMP+MPI	3	3	0.468316	0.305172	0.126011	0.006216	3.558.197	39.535.520
OMP+MPI	3	4	0.42301	0.268731	0.117845	0.008752	3.939.293	32.827.445
OMP+MPI	3	5	0.413847	0.274358	0.102191	0.01333	4.026.518	26.843.455
OMP+MPI	3	6	0.400523	0.257066	0.108419	0.013279	4.160.456	23.113.646
OMP+MPI	4	1	0.624942	0.357278	0.210151	0.001765	2.666.422	66.660.553
OMP+MPI	4	2	0.484282	0.306893	0.132949	0.00448	3.440.892	43.011.154
OMP+MPI	4	3	0.434038	0.282224	0.11445	0.006857	3.839.204	31.993.368
OMP+MPI	4	4	0.409181	0.264482	0.110635	0.008852	4.072.434	25.452.711
OMP+MPI	4	5	0.389906	0.243216	0.11248	0.011134	4.273.755	21.368.774
OMP+MPI	4	6	0.366271	0.225794	0.107435	0.012386	4.549.529	18.956.370
OMP+MPI	5	1	0.569813	0.300584	0.207999	0.001694	2.924.396	58.487.926
OMP+MPI	5	2	0.439856	0.262362	0.131866	0.004441	3.788.423	37.884.228
OMP+MPI	5	3	0.419423	0.268587	0.111719	0.006749	3.972.988	26.486.586
OMP+MPI	5	4	0.394498	0.253791	0.106131	0.008733	4.224.002	21.120.012
OMP+MPI	5	5	0.402106	0.265661	0.102644	0.011	4.144.078	16.576.310
OMP+MPI	5	6	0.377744	0.239279	0.101957	0.012514	4.411.355	14.704.515
OMP+MPI	6	1	0.534943	0.259985	0.214527	0.001842	3.115.024	51.917.073
OMP+MPI	6	2	0.401755	0.228606	0.129892	0.004675	4.147.708	34.564.236
OMP+MPI	6	3	0.392894	0.248088	0.107644	0.006644	4.241.252	23.562.513
OMP+MPI	6	4	0.374702	0.236003	0.105207	0.008428	4.447.168	18.529.866
OMP+MPI	6	5	0.366382	0.232169	0.101181	0.011047	4.548.151	15.160.502
OMP+MPI	6	6	0.39896	0.259116	0.106553	0.013076	4.176.766	11.602.128

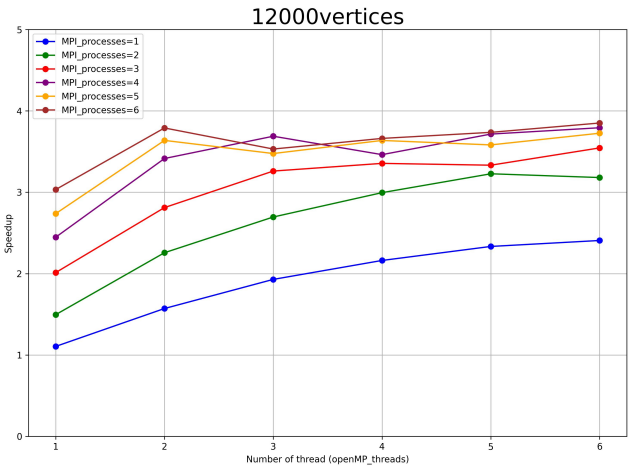
Optimization 1



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.136574	0.097495	0.027272	0.005995	1.000.000	100.000.000
OMP+MPI	1	1	0.125086	0.092519	0.024151	0.001823	1.091.841	109.184.081
OMP+MPI	1	2	0.089817	0.06562	0.015892	0.004293	1.520.581	76.029.037
OMP+MPI	1	3	0.06662	0.042382	0.014572	0.006557	2.050.045	68.334.834
OMP+MPI	1	4	0.058341	0.032669	0.013993	0.00853	2.340.961	58.524.023
OMP+MPI	1	5	0.060625	0.029431	0.014851	0.012786	2.252.767	45.055.340
OMP+MPI	1	6	0.057959	0.026916	0.015524	0.012641	2.356.370	39.272.826
OMP+MPI	2	1	0.078471	0.047282	0.023337	0.001728	1.740.439	87.021.957
OMP+MPI	2	2	0.060454	0.033927	0.015873	0.004432	2.259.139	56.478.480
OMP+MPI	2	3	0.050055	0.025625	0.014615	0.006348	2.728.479	45.474.645
OMP+MPI	2	4	0.047727	0.020803	0.014395	0.009205	2.861.537	35.769.211
OMP+MPI	2	5	0.048551	0.019241	0.014608	0.011381	2.812.972	28.129.718
OMP+MPI	2	6	0.054889	0.023849	0.014611	0.013378	2.488.185	20.734.877
OMP+MPI	3	1	0.0638	0.032044	0.023538	0.001767	2.140.675	71.355.836
OMP+MPI	3	2	0.049383	0.025215	0.015549	0.004214	2.765.608	46.093.460
OMP+MPI	3	3	0.043466	0.018374	0.014417	0.006844	3.142.088	34.912.090
OMP+MPI	3	4	0.04985	0.022505	0.014948	0.009076	2.739.727	22.831.055
OMP+MPI	3	5	0.048384	0.020988	0.013348	0.010708	2.822.681	18.817.872
OMP+MPI	3	6	0.048148	0.01856	0.014041	0.012701	2.836.546	15.758.587
OMP+MPI	4	1	0.057997	0.025473	0.023963	0.001673	2.354.846	58.871.149
OMP+MPI	4	2	0.045687	0.021319	0.015417	0.004293	2.989.341	37.366.756
OMP+MPI	4	3	0.045655	0.021279	0.013677	0.006423	2.991.469	24.928.904
OMP+MPI	4	4	0.045888	0.019909	0.014101	0.008828	2.976.214	18.601.338
OMP+MPI	4	5	0.048288	0.019091	0.014991	0.011467	2.828.292	14.141.462
OMP+MPI	4	6	0.049859	0.019344	0.014695	0.013482	2.739.232	11.413.467
OMP+MPI	5	1	0.051936	0.020376	0.023551	0.001677	2.629.685	52.593.698
OMP+MPI	5	2	0.041147	0.016539	0.015424	0.004228	3.319.213	33.192.131
OMP+MPI	5	3	0.044561	0.021049	0.013843	0.006375	3.064.877	20.432.516
OMP+MPI	5	4	0.043414	0.017626	0.014178	0.008552	3.145.852	15.729.258
OMP+MPI	5	5	0.048944	0.019397	0.01517	0.01161	2.790.442	11.161.768
OMP+MPI	5	6	0.045883	0.016934	0.014281	0.01181	2.976.571	9.921.903
OMP+MPI	6	1	0.053197	0.020004	0.024607	0.001719	2.567.301	42.788.352
OMP+MPI	6	2	0.043231	0.018871	0.015307	0.00425	3.159.132	26.326.097
OMP+MPI	6	3	0.042354	0.018402	0.01408	0.006375	3.224.583	17.914.352
OMP+MPI	6	4	0.043998	0.018516	0.013973	0.008287	3.104.096	12.933.732
OMP+MPI	6	5	0.047498	0.018313	0.015142	0.010917	2.875.333	9.584.443
OMP+MPI	6	6	0.050114	0.018833	0.015657	0.012023	2.725.266	7.570.184

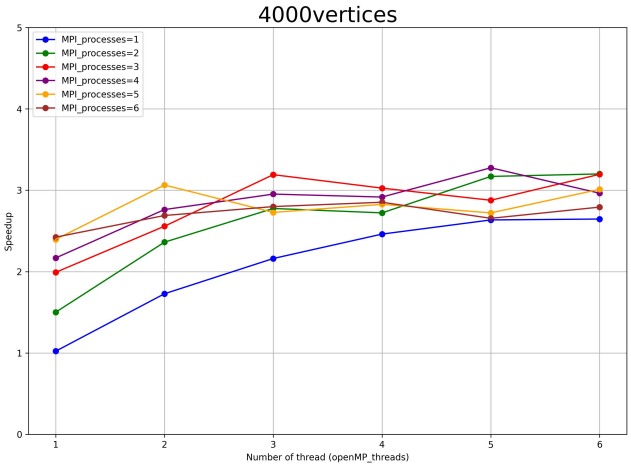


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.387781	0.249795	0.100472	0.01103	1.000.000	100.000.000
OMP+MPI	1	1	0.375043	0.246743	0.10079	0.00187	1.033.963	103.396.277
OMP+MPI	1	2	0.272034	0.187965	0.06154	0.004582	1.425.485	71.274.271
OMP+MPI	1	3	0.214631	0.141425	0.051321	0.006424	1.806.735	60.224.510
OMP+MPI	1	4	0.170004	0.099816	0.047343	0.010572	2.281.008	57.025.202
OMP+MPI	1	5	0.155734	0.08398	0.050174	0.011378	2.490.018	49.800.365
OMP+MPI	1	6	0.147443	0.073416	0.050841	0.013106	2.630.037	43.833.945
OMP+MPI	2	1	0.242706	0.124205	0.091882	0.001734	1.597.734	79.886.715
OMP+MPI	2	2	0.183546	0.104701	0.058013	0.004082	2.112.721	52.818.034
OMP+MPI	2	3	0.143329	0.072489	0.050687	0.006863	2.705.518	45.091.962
OMP+MPI	2	4	0.128113	0.058535	0.04996	0.008313	3.026.851	37.835.640
OMP+MPI	2	5	0.114309	0.047703	0.045855	0.01109	3.392.373	33.923.733
OMP+MPI	2	6	0.119674	0.050146	0.04816	0.012339	3.240.307	27.002.558
OMP+MPI	3	1	0.212495	0.085701	0.098942	0.001763	1.824.897	60.829.888
OMP+MPI	3	2	0.157794	0.072698	0.064962	0.004069	2.457.519	40.958.647
OMP+MPI	3	3	0.127008	0.053675	0.052923	0.006311	3.053.185	33.924.282
OMP+MPI	3	4	0.123622	0.054142	0.04741	0.007991	3.136.812	26.140.097
OMP+MPI	3	5	0.130186	0.060877	0.048861	0.010503	2.978.665	19.857.768
OMP+MPI	3	6	0.129736	0.05309	0.052011	0.014444	2.988.997	16.605.538
OMP+MPI	4	1	0.188736	0.06888	0.092593	0.001823	2.054.619	51.365.466
OMP+MPI	4	2	0.137776	0.058044	0.058754	0.004176	2.814.562	35.182.025
OMP+MPI	4	3	0.122869	0.052583	0.049426	0.006844	3.156.048	26.300.403
OMP+MPI	4	4	0.127673	0.055685	0.050619	0.009019	3.037.283	18.983.016
OMP+MPI	4	5	0.122588	0.055383	0.046295	0.011004	3.163.283	15.816.414
OMP+MPI	4	6	0.117161	0.049518	0.046169	0.012439	3.309.823	13.790.929
OMP+MPI	5	1	0.169422	0.051608	0.091037	0.001686	2.288.851	45.777.012
OMP+MPI	5	2	0.13507	0.050711	0.062759	0.004279	2.870.960	28.709.595
OMP+MPI	5	3	0.127174	0.058834	0.049356	0.006603	3.049.212	20.328.081
OMP+MPI	5	4	0.114754	0.047626	0.046024	0.009352	3.379.248	16.896.238
OMP+MPI	5	5	0.128968	0.061946	0.04666	0.010248	3.006.785	12.027.138
OMP+MPI	5	6	0.130028	0.057331	0.050167	0.013413	2.982.296	9.940.987
OMP+MPI	6	1	0.171087	0.048729	0.096017	0.001802	2.266.576	37.776.261
OMP+MPI	6	2	0.13013	0.050233	0.058097	0.004045	2.979.958	24.832.987
OMP+MPI	6	3	0.122105	0.052301	0.04938	0.00606	3.175.795	17.643.308
OMP+MPI	6	4	0.123727	0.050707	0.052602	0.00832	3.134.150	13.058.957
OMP+MPI	6	5	0.128246	0.056193	0.051526	0.010611	3.023.736	10.079.119
OMP+MPI	6	6	0.125098	0.055103	0.049365	0.011581	3.099.826	8.610.628

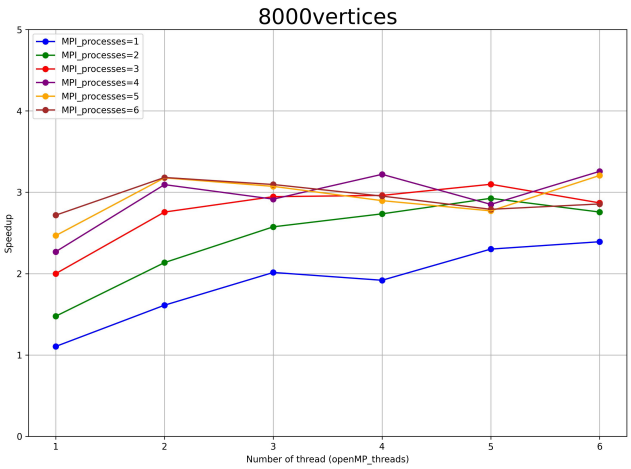


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.898215	0.596572	0.220729	0.019032	1.000.000	100.000.000
OMP+MPI	1	1	0.812663	0.546647	0.206502	0.001666	1.105.273	110.527.328
OMP+MPI	1	2	0.600602	0.422569	0.130557	0.004358	1.495.524	74.776.183
OMP+MPI	1	3	0.446292	0.296025	0.110927	0.007388	2.012.616	67.087.187
OMP+MPI	1	4	0.367042	0.221424	0.109343	0.009766	2.447.176	61.179.407
OMP+MPI	1	5	0.32819	0.185775	0.108648	0.011053	2.736.872	54.737.431
OMP+MPI	1	6	0.296151	0.158372	0.104409	0.012285	3.032.964	50.549.396
OMP+MPI	2	1	0.571735	0.287857	0.221168	0.001783	1.571.033	78.551.642
OMP+MPI	2	2	0.397996	0.223183	0.132686	0.004316	2.256.845	56.421.123
OMP+MPI	2	3	0.319488	0.165581	0.115165	0.006328	2.811.421	46.857.015
OMP+MPI	2	4	0.263054	0.126883	0.105186	0.008457	3.414.560	42.681.994
OMP+MPI	2	5	0.246962	0.104729	0.109371	0.010342	3.637.066	36.370.659
OMP+MPI	2	6	0.236999	0.100033	0.105107	0.012099	3.789.962	31.583.015
OMP+MPI	3	1	0.465658	0.198367	0.20928	0.001716	1.928.916	64.297.206
OMP+MPI	3	2	0.333207	0.159271	0.130976	0.004358	2.695.667	44.927.790
OMP+MPI	3	3	0.275498	0.116996	0.122377	0.006625	3.260.327	36.225.858
OMP+MPI	3	4	0.243532	0.102011	0.105905	0.009381	3.688.292	30.735.766
OMP+MPI	3	5	0.258428	0.121085	0.102532	0.011653	3.475.689	23.171.257
OMP+MPI	3	6	0.254382	0.114359	0.106158	0.01325	3.530.970	19.616.501
OMP+MPI	4	1	0.415513	0.147099	0.208706	0.001717	2.161.702	54.042.548
OMP+MPI	4	2	0.299907	0.128276	0.129236	0.004491	2.994.979	37.437.241
OMP+MPI	4	3	0.267746	0.11801	0.111631	0.007249	3.354.723	27.956.022
OMP+MPI	4	4	0.25944	0.118255	0.106959	0.008585	3.462.124	21.638.277
OMP+MPI	4	5	0.246996	0.106243	0.107255	0.011324	3.636.565	18.182.826
OMP+MPI	4	6	0.245308	0.104191	0.106826	0.013035	3.661.582	15.256.590
OMP+MPI	5	1	0.384863	0.119934	0.206474	0.001725	2.333.857	46.677.142
OMP+MPI	5	2	0.278359	0.1041	0.13038	0.00431	3.226.817	32.268.173
OMP+MPI	5	3	0.269545	0.118382	0.114893	0.006488	3.332.339	22.215.592
OMP+MPI	5	4	0.241795	0.103561	0.104032	0.008697	3.714.788	18.573.939
OMP+MPI	5	5	0.250782	0.114989	0.102834	0.010689	3.581.658	14.326.630
OMP+MPI	5	6	0.240421	0.103022	0.105408	0.012722	3.736.010	12.453.367
OMP+MPI	6	1	0.373166	0.105996	0.20837	0.001679	2.407.009	40.116.822
OMP+MPI	6	2	0.2824	0.108656	0.131186	0.004171	3.180.649	26.505.408
OMP+MPI	6	3	0.253392	0.111267	0.109268	0.006306	3.544.766	19.693.142
OMP+MPI	6	4	0.236838	0.098929	0.103828	0.007926	3.792.530	15.802.209
OMP+MPI	6	5	0.241111	0.10531	0.102623	0.010676	3.725.326	12.417.754
OMP+MPI	6	6	0.233259	0.097762	0.102947	0.012162	3.850.721	10.696.446

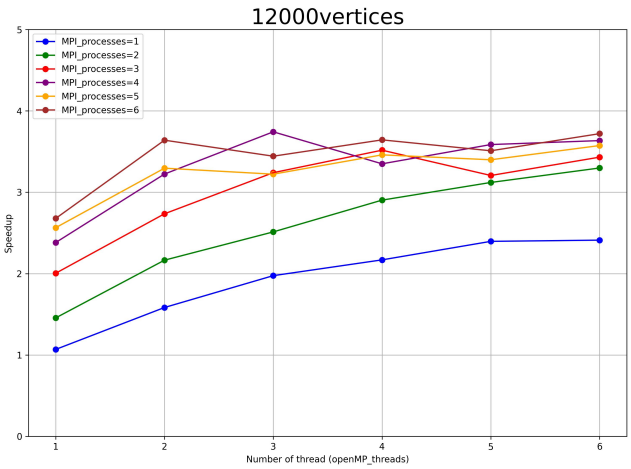
Optimization 2



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.145501	0.107649	0.02549	0.00576	1.000.000	100.000.000
OMP+MPI	1	1	0.142377	0.109381	0.024682	0.00172	1.021.944	102.194.358
OMP+MPI	1	2	0.096933	0.073004	0.015272	0.004365	1.501.052	75.052.614
OMP+MPI	1	3	0.073074	0.049454	0.013305	0.006405	1.991.143	66.371.418
OMP+MPI	1	4	0.067115	0.040965	0.014911	0.008046	2.167.916	54.197.894
OMP+MPI	1	5	0.060732	0.032508	0.014183	0.010212	2.395.784	47.915.679
OMP+MPI	1	6	0.060039	0.030679	0.014458	0.012458	2.423.437	40.390.621
OMP+MPI	2	1	0.084211	0.053089	0.023182	0.001657	1.727.812	86.390.584
OMP+MPI	2	2	0.061602	0.038004	0.014972	0.004206	2.361.948	59.048.712
OMP+MPI	2	3	0.056891	0.031554	0.013962	0.006831	2.557.513	42.625.216
OMP+MPI	2	4	0.052672	0.024845	0.015323	0.009015	2.762.393	34.529.909
OMP+MPI	2	5	0.047485	0.020389	0.014549	0.009736	3.064.141	30.641.413
OMP+MPI	2	6	0.054099	0.023405	0.016126	0.011623	2.689.528	22.412.729
OMP+MPI	3	1	0.067327	0.035567	0.023587	0.001662	2.161.089	72.036.315
OMP+MPI	3	2	0.052415	0.028805	0.015138	0.004138	2.775.964	46.266.062
OMP+MPI	3	3	0.045592	0.022044	0.013628	0.005945	3.191.401	35.460.009
OMP+MPI	3	4	0.049282	0.023386	0.014046	0.008681	2.952.412	24.603.430
OMP+MPI	3	5	0.053345	0.025531	0.014308	0.010587	2.727.542	18.183.616
OMP+MPI	3	6	0.051981	0.022155	0.014913	0.013124	2.799.114	15.550.634
OMP+MPI	4	1	0.059119	0.027402	0.023667	0.001654	2.461.150	61.528.760
OMP+MPI	4	2	0.053455	0.028883	0.015182	0.004267	2.721.955	34.024.439
OMP+MPI	4	3	0.048083	0.023291	0.01495	0.006099	3.026.033	25.216.943
OMP+MPI	4	4	0.049891	0.024245	0.013789	0.008785	2.916.373	18.227.329
OMP+MPI	4	5	0.051474	0.021789	0.014548	0.011945	2.826.684	14.133.422
OMP+MPI	4	6	0.051006	0.022122	0.01466	0.01138	2.852.592	11.885.801
OMP+MPI	5	1	0.05522	0.023925	0.023246	0.001657	2.634.952	52.699.047
OMP+MPI	5	2	0.045904	0.021643	0.015591	0.004055	3.169.675	31.696.748
OMP+MPI	5	3	0.050575	0.026013	0.014115	0.006625	2.876.930	19.179.535
OMP+MPI	5	4	0.044414	0.019957	0.013194	0.008294	3.276.011	16.380.055
OMP+MPI	5	5	0.053462	0.023707	0.015853	0.011274	2.721.548	10.886.191
OMP+MPI	5	6	0.054804	0.02156	0.015377	0.014766	2.654.954	8.849.845
OMP+MPI	6	1	0.054987	0.022886	0.023985	0.001695	2.646.094	44.101.560
OMP+MPI	6	2	0.045463	0.020918	0.015204	0.004207	3.200.421	26.670.177
OMP+MPI	6	3	0.045529	0.021972	0.014077	0.005966	3.195.782	17.754.343
OMP+MPI	6	4	0.049101	0.023566	0.013195	0.008783	2.963.295	12.347.063
OMP+MPI	6	5	0.048324	0.020466	0.014905	0.010137	3.010.973	10.036.576
OMP+MPI	6	6	0.052101	0.022302	0.014447	0.012422	2.792.694	7.757.483

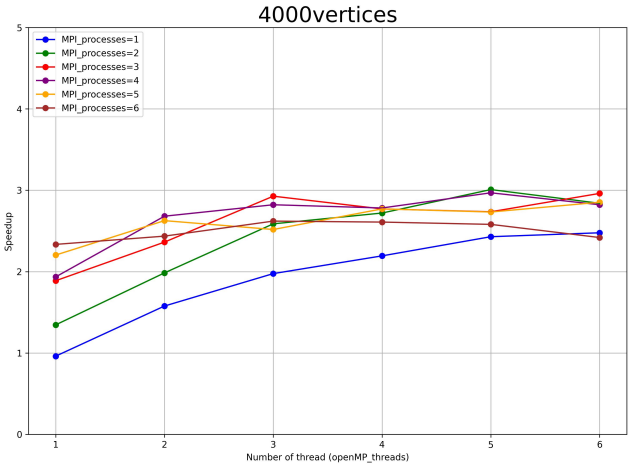


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.433172	0.290412	0.105024	0.011564	1.000.000	100.000.000
OMP+MPI	1	1	0.392105	0.272887	0.090675	0.001854	1.104.733	110.473.265
OMP+MPI	1	2	0.293431	0.212714	0.058409	0.004252	1.476.233	73.811.644
OMP+MPI	1	3	0.216642	0.143868	0.052725	0.006714	1.999.482	66.649.395
OMP+MPI	1	4	0.190906	0.120237	0.050034	0.008457	2.269.032	56.725.790
OMP+MPI	1	5	0.175579	0.10216	0.050891	0.011987	2.467.112	49.342.232
OMP+MPI	1	6	0.159373	0.086586	0.050847	0.012088	2.717.975	45.299.575
OMP+MPI	2	1	0.268767	0.147826	0.09214	0.001695	1.611.703	80.585.145
OMP+MPI	2	2	0.202873	0.120349	0.062299	0.004246	2.135.192	53.379.801
OMP+MPI	2	3	0.157156	0.086743	0.051182	0.005992	2.756.326	45.938.762
OMP+MPI	2	4	0.140029	0.070251	0.050509	0.008214	3.093.454	38.668.177
OMP+MPI	2	5	0.136331	0.066898	0.049493	0.010544	3.177.342	31.773.416
OMP+MPI	2	6	0.136149	0.064009	0.050905	0.011624	3.181.612	26.513.436
OMP+MPI	3	1	0.21506	0.096418	0.093435	0.001819	2.014.195	67.139.830
OMP+MPI	3	2	0.168221	0.089308	0.05776	0.004066	2.575.008	42.916.804
OMP+MPI	3	3	0.147055	0.064985	0.05822	0.008679	2.945.644	32.729.383
OMP+MPI	3	4	0.148586	0.079408	0.048586	0.008799	2.915.293	24.294.110
OMP+MPI	3	5	0.141004	0.070864	0.048749	0.010641	3.072.064	20.480.425
OMP+MPI	3	6	0.13996	0.073091	0.046565	0.012026	3.094.968	17.194.268
OMP+MPI	4	1	0.225815	0.086883	0.10927	0.001959	1.918.264	47.956.592
OMP+MPI	4	2	0.158405	0.075271	0.061032	0.004587	2.734.592	34.182.406
OMP+MPI	4	3	0.146288	0.074575	0.051882	0.006455	2.961.099	24.675.824
OMP+MPI	4	4	0.134521	0.06638	0.047761	0.008821	3.220.105	20.125.656
OMP+MPI	4	5	0.149543	0.07288	0.050557	0.015408	2.896.627	14.483.135
OMP+MPI	4	6	0.146702	0.072709	0.051747	0.013369	2.952.742	12.303.094
OMP+MPI	5	1	0.188215	0.066911	0.093001	0.001736	2.301.473	46.029.461
OMP+MPI	5	2	0.148068	0.060825	0.064488	0.004327	2.925.482	29.254.821
OMP+MPI	5	3	0.139825	0.067226	0.051647	0.006422	3.097.945	20.652.969
OMP+MPI	5	4	0.15195	0.080233	0.049466	0.010053	2.850.761	14.253.806
OMP+MPI	5	5	0.156326	0.08162	0.051148	0.012684	2.770.951	11.083.806
OMP+MPI	5	6	0.155234	0.078647	0.053775	0.013375	2.790.444	9.301.479
OMP+MPI	6	1	0.181117	0.056415	0.097286	0.001785	2.391.675	39.861.245
OMP+MPI	6	2	0.157133	0.072845	0.06202	0.00436	2.756.720	22.972.670
OMP+MPI	6	3	0.150971	0.0694	0.059015	0.007368	2.869.238	15.940.212
OMP+MPI	6	4	0.133019	0.061853	0.050252	0.00933	3.256.465	13.568.605
OMP+MPI	6	5	0.13511	0.067041	0.047126	0.010319	3.206.067	10.686.891
OMP+MPI	6	6	0.151679	0.078664	0.050308	0.013846	2.855.845	7.932.903

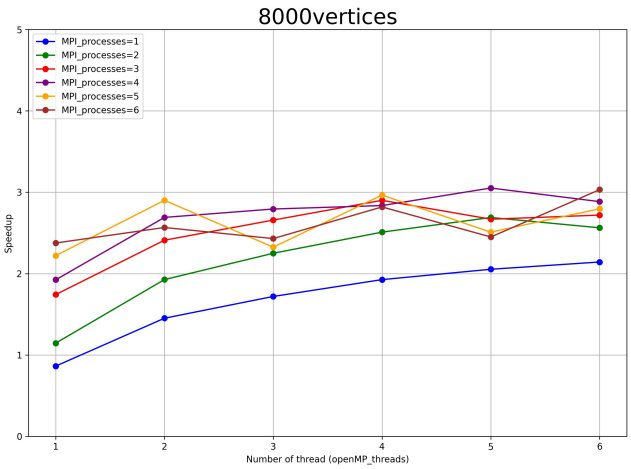


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.945021	0.64753	0.223424	0.015925	1.000.000	100.000.000
OMP+MPI	1	1	0.884593	0.617669	0.206754	0.001772	1.068.312	106.831.225
OMP+MPI	1	2	0.649217	0.470364	0.131849	0.004185	1.455.633	72.781.653
OMP+MPI	1	3	0.471468	0.316084	0.115984	0.006838	2.004.420	66.814.008
OMP+MPI	1	4	0.396757	0.248449	0.113468	0.008443	2.381.866	59.546.661
OMP+MPI	1	5	0.368513	0.218656	0.115868	0.010575	2.564.417	51.288.340
OMP+MPI	1	6	0.352865	0.200982	0.117509	0.012776	2.678.134	44.635.562
OMP+MPI	2	1	0.596811	0.329089	0.206577	0.001826	1.583.451	79.172.552
OMP+MPI	2	2	0.43648	0.253195	0.141684	0.004581	2.165.096	54.127.394
OMP+MPI	2	3	0.345505	0.198746	0.112264	0.007223	2.735.188	45.586.460
OMP+MPI	2	4	0.293172	0.155457	0.104383	0.008813	3.223.441	40.293.011
OMP+MPI	2	5	0.286756	0.149493	0.102341	0.012008	3.295.552	32.955.521
OMP+MPI	2	6	0.259645	0.119212	0.10845	0.012821	3.639.666	30.330.547
OMP+MPI	3	1	0.478516	0.209591	0.209095	0.001711	1.974.899	65.829.983
OMP+MPI	3	2	0.376121	0.196028	0.136471	0.004514	2.512.545	41.875.753
OMP+MPI	3	3	0.291635	0.13784	0.114691	0.007292	3.240.429	36.004.771
OMP+MPI	3	4	0.252508	0.116228	0.103748	0.008543	3.742.539	31.187.824
OMP+MPI	3	5	0.293299	0.155986	0.102375	0.011461	3.222.045	21.480.301
OMP+MPI	3	6	0.274474	0.136311	0.103921	0.01324	3.443.026	19.127.920
OMP+MPI	4	1	0.435725	0.166897	0.210388	0.001704	2.168.845	54.221.121
OMP+MPI	4	2	0.325518	0.15318	0.12826	0.004473	2.903.134	36.289.178
OMP+MPI	4	3	0.268717	0.124089	0.108077	0.006884	3.516.796	29.306.630
OMP+MPI	4	4	0.282153	0.140819	0.1057	0.008689	3.349.327	20.933.294
OMP+MPI	4	5	0.273175	0.1282	0.110858	0.010975	3.459.398	17.296.989
OMP+MPI	4	6	0.25937	0.119582	0.10619	0.012292	3.643.525	15.181.353
OMP+MPI	5	1	0.394344	0.129833	0.206232	0.001672	2.396.438	47.928.763
OMP+MPI	5	2	0.302866	0.129807	0.12851	0.004608	3.120.261	31.202.611
OMP+MPI	5	3	0.294771	0.148405	0.110229	0.00636	3.205.950	21.372.998
OMP+MPI	5	4	0.263529	0.124565	0.103668	0.00828	3.586.016	17.930.080
OMP+MPI	5	5	0.278025	0.137524	0.107099	0.0105	3.399.050	13.596.202
OMP+MPI	5	6	0.269271	0.127227	0.107459	0.013053	3.509.554	11.698.512
OMP+MPI	6	1	0.391927	0.125976	0.20824	0.001631	2.411.214	40.186.897
OMP+MPI	6	2	0.286538	0.114817	0.126876	0.004577	3.298.065	27.483.876
OMP+MPI	6	3	0.275426	0.130601	0.109072	0.006041	3.431.125	19.061.805
OMP+MPI	6	4	0.260007	0.124129	0.102628	0.007872	3.634.605	15.144.189
OMP+MPI	6	5	0.264442	0.126512	0.104293	0.010499	3.573.642	11.912.140
OMP+MPI	6	6	0.254051	0.117421	0.103677	0.012179	3.719.808	10.332.801

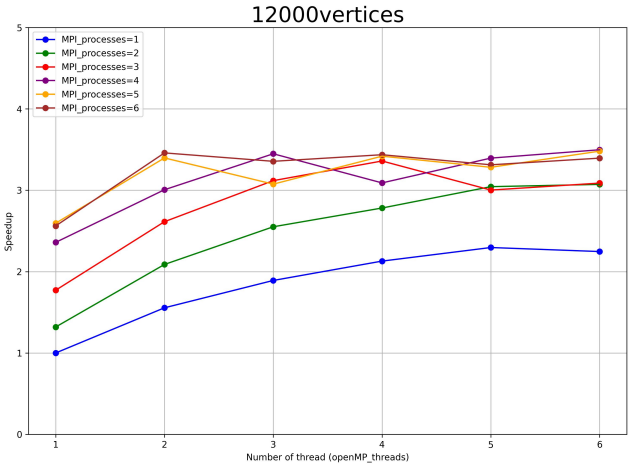
Optimization 3



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.146094	0.108482	0.026566	0.005553	1.000.000	100.000.000
OMP+MPI	1	1	0.152017	0.119881	0.024164	0.001704	0.961039	96.103.890
OMP+MPI	1	2	0.108573	0.084225	0.015296	0.004047	1.345.592	67.279.583
OMP+MPI	1	3	0.077379	0.0543	0.01327	0.006103	1.888.035	62.934.496
OMP+MPI	1	4	0.075494	0.047045	0.01547	0.009547	1.935.190	48.379.745
OMP+MPI	1	5	0.0663	0.039636	0.014134	0.010038	2.203.533	44.070.664
OMP+MPI	1	6	0.062578	0.034098	0.014448	0.011983	2.334.613	38.910.218
OMP+MPI	2	1	0.092641	0.061037	0.023633	0.001661	1.576.993	78.849.672
OMP+MPI	2	2	0.073621	0.048417	0.015706	0.00463	1.984.410	49.610.250
OMP+MPI	2	3	0.06184	0.037715	0.014389	0.005877	2.362.436	39.373.940
OMP+MPI	2	4	0.05449	0.029381	0.013254	0.008324	2.681.096	33.513.697
OMP+MPI	2	5	0.055625	0.024341	0.015019	0.01414	2.626.413	26.264.135
OMP+MPI	2	6	0.059973	0.030459	0.014467	0.012061	2.435.980	20.299.834
OMP+MPI	3	1	0.073954	0.042543	0.023514	0.00169	1.975.475	65.849.154
OMP+MPI	3	2	0.05647	0.033215	0.015215	0.004187	2.587.136	43.118.926
OMP+MPI	3	3	0.049913	0.026409	0.013584	0.006615	2.926.978	32.521.977
OMP+MPI	3	4	0.051778	0.026734	0.013225	0.008621	2.821.578	23.513.149
OMP+MPI	3	5	0.058033	0.029531	0.014731	0.010731	2.517.456	16.783.038
OMP+MPI	3	6	0.055748	0.025829	0.014409	0.012604	2.620.619	14.558.993
OMP+MPI	4	1	0.066603	0.034076	0.024559	0.001677	2.193.492	54.837.302
OMP+MPI	4	2	0.053711	0.029036	0.015144	0.00418	2.719.981	33.999.760
OMP+MPI	4	3	0.052736	0.028011	0.014478	0.006667	2.770.294	23.085.787
OMP+MPI	4	4	0.052515	0.027302	0.013428	0.008681	2.781.953	17.387.205
OMP+MPI	4	5	0.052729	0.024205	0.014321	0.011551	2.770.662	13.853.311
OMP+MPI	4	6	0.056004	0.025163	0.015548	0.012243	2.608.640	10.869.332
OMP+MPI	5	1	0.060149	0.027321	0.023907	0.001717	2.428.872	48.577.449
OMP+MPI	5	2	0.048566	0.023683	0.015687	0.004534	3.008.128	30.081.280
OMP+MPI	5	3	0.053411	0.028723	0.013934	0.006691	2.735.284	18.235.226
OMP+MPI	5	4	0.049215	0.023695	0.013697	0.00848	2.968.490	14.842.451
OMP+MPI	5	5	0.053467	0.025984	0.014117	0.010323	2.732.419	10.929.676
OMP+MPI	5	6	0.056615	0.025353	0.015433	0.012625	2.580.487	8.601.622
OMP+MPI	6	1	0.058976	0.026734	0.024168	0.001672	2.477.202	41.286.707
OMP+MPI	6	2	0.051448	0.027034	0.015305	0.004392	2.839.649	23.663.740
OMP+MPI	6	3	0.049339	0.025938	0.014249	0.006153	2.961.060	16.450.332
OMP+MPI	6	4	0.051771	0.025742	0.014495	0.008506	2.821.932	11.758.051
OMP+MPI	6	5	0.051216	0.023852	0.013826	0.010612	2.852.484	9.508.280
OMP+MPI	6	6	0.060397	0.029164	0.01507	0.013986	2.418.899	6.719.164



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.413356	0.279351	0.097918	0.010544	1.000.000	100.000.000
OMP+MPI	1	1	0.478863	0.343944	0.104087	0.001979	0.863203	86.320.305
OMP+MPI	1	2	0.360879	0.276767	0.062081	0.004868	1.145.414	57.270.720
OMP+MPI	1	3	0.237302	0.166566	0.050836	0.006492	1.741.899	58.063.284
OMP+MPI	1	4	0.214848	0.138759	0.055083	0.009214	1.923.942	48.098.544
OMP+MPI	1	5	0.186169	0.11482	0.049445	0.011494	2.220.333	44.406.653
OMP+MPI	1	6	0.173964	0.099211	0.052599	0.01253	2.376.101	39.601.680
OMP+MPI	2	1	0.28472	0.165907	0.092029	0.001715	1.451.798	72.589.913
OMP+MPI	2	2	0.214547	0.132326	0.06036	0.004938	1.926.650	48.166.248
OMP+MPI	2	3	0.17148	0.101211	0.051327	0.006139	2.410.520	40.175.336
OMP+MPI	2	4	0.153617	0.083552	0.048703	0.009094	2.690.822	33.635.275
OMP+MPI	2	5	0.14251	0.074558	0.047797	0.010652	2.900.540	29.005.403
OMP+MPI	2	6	0.161047	0.084885	0.053178	0.013817	2.566.687	21.389.061
OMP+MPI	3	1	0.240419	0.119024	0.096086	0.001693	1.719.319	57.310.620
OMP+MPI	3	2	0.1837	0.103478	0.059238	0.004167	2.250.163	37.502.710
OMP+MPI	3	3	0.155528	0.078761	0.055161	0.007142	2.657.759	29.530.660
OMP+MPI	3	4	0.147948	0.074034	0.054368	0.008041	2.793.937	23.282.809
OMP+MPI	3	5	0.177793	0.10607	0.048864	0.012856	2.324.928	15.499.523
OMP+MPI	3	6	0.170073	0.094945	0.052825	0.012773	2.430.462	13.502.568
OMP+MPI	4	1	0.214603	0.089075	0.098078	0.001777	1.926.143	48.153.567
OMP+MPI	4	2	0.1647	0.084465	0.058502	0.004039	2.509.743	31.371.793
OMP+MPI	4	3	0.14251	0.07236	0.05193	0.006152	2.900.530	24.171.084
OMP+MPI	4	4	0.145732	0.075726	0.049674	0.008136	2.836.402	17.727.514
OMP+MPI	4	5	0.1394	0.069068	0.049603	0.010729	2.965.262	14.826.309
OMP+MPI	4	6	0.146612	0.074251	0.050756	0.012556	2.819.397	11.747.487
OMP+MPI	5	1	0.2013	0.076161	0.097489	0.001687	2.053.428	41.068.552
OMP+MPI	5	2	0.153682	0.07084	0.061522	0.004701	2.689.675	26.896.751
OMP+MPI	5	3	0.154887	0.083214	0.052499	0.006299	2.668.767	17.791.781
OMP+MPI	5	4	0.135447	0.068373	0.048195	0.008519	3.051.791	15.258.957
OMP+MPI	5	5	0.164728	0.089427	0.054179	0.010771	2.509.324	10.037.298
OMP+MPI	5	6	0.168698	0.087355	0.056085	0.014901	2.450.279	8.167.598
OMP+MPI	6	1	0.192922	0.062714	0.102484	0.001717	2.142.607	35.710.114
OMP+MPI	6	2	0.161278	0.07471	0.062295	0.004074	2.562.995	21.358.292
OMP+MPI	6	3	0.152011	0.077216	0.054478	0.006939	2.719.242	15.106.898
OMP+MPI	6	4	0.143301	0.069236	0.052948	0.009768	2.884.520	12.018.832
OMP+MPI	6	5	0.147788	0.080606	0.047198	0.009859	2.796.952	9.323.175
OMP+MPI	6	6	0.136356	0.068607	0.046642	0.011453	3.031.447	8.420.686

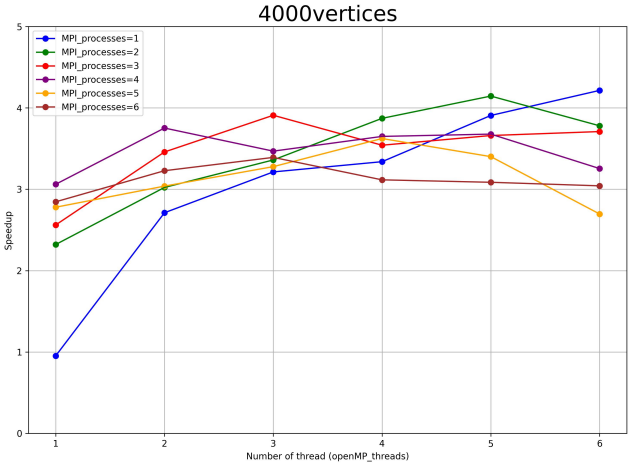


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.984566	0.671236	0.231718	0.0174	1.000.000	100.000.000
OMP+MPI	1	1	0.984382	0.713221	0.208535	0.001698	1.000.188	100.018.777
OMP+MPI	1	2	0.746196	0.56112	0.141782	0.004371	1.319.448	65.972.412
OMP+MPI	1	3	0.555825	0.389093	0.128234	0.007064	1.771.360	59.045.343
OMP+MPI	1	4	0.417233	0.282081	0.104147	0.008056	2.359.749	58.993.725
OMP+MPI	1	5	0.379408	0.241833	0.102512	0.012507	2.595.010	51.900.204
OMP+MPI	1	6	0.384354	0.242536	0.107398	0.013565	2.561.617	42.693.611
OMP+MPI	2	1	0.63304	0.361953	0.210642	0.001682	1.555.299	77.764.938
OMP+MPI	2	2	0.471531	0.290598	0.134369	0.004088	2.088.018	52.200.454
OMP+MPI	2	3	0.376723	0.233674	0.111341	0.006686	2.613.506	43.558.425
OMP+MPI	2	4	0.327508	0.182802	0.110639	0.008785	3.006.236	37.577.950
OMP+MPI	2	5	0.289836	0.153145	0.103153	0.011213	3.396.983	33.969.832
OMP+MPI	2	6	0.284652	0.146124	0.1054	0.013038	3.458.836	28.823.634
OMP+MPI	3	1	0.520589	0.245774	0.215537	0.001665	1.891.256	63.041.880
OMP+MPI	3	2	0.385943	0.213427	0.129366	0.004384	2.551.067	42.517.778
OMP+MPI	3	3	0.315766	0.164854	0.113187	0.006594	3.118.030	34.644.779
OMP+MPI	3	4	0.285539	0.147136	0.103317	0.008675	3.448.104	28.734.197
OMP+MPI	3	5	0.320175	0.179064	0.10698	0.011019	3.075.093	20.500.619
OMP+MPI	3	6	0.293415	0.157353	0.103913	0.012617	3.355.542	18.641.900
OMP+MPI	4	1	0.462418	0.194381	0.207914	0.001783	2.129.170	53.229.239
OMP+MPI	4	2	0.353959	0.183995	0.127577	0.004383	2.781.579	34.769.738
OMP+MPI	4	3	0.293079	0.144956	0.10953	0.006766	3.359.389	27.994.907
OMP+MPI	4	4	0.318613	0.175944	0.108572	0.009034	3.090.164	19.313.523
OMP+MPI	4	5	0.288006	0.150402	0.105382	0.010659	3.418.556	17.092.780
OMP+MPI	4	6	0.286504	0.143454	0.105927	0.01531	3.436.484	14.318.682
OMP+MPI	5	1	0.428816	0.162056	0.207613	0.00165	2.296.011	45.920.224
OMP+MPI	5	2	0.323444	0.14944	0.128999	0.004537	3.044.009	30.440.086
OMP+MPI	5	3	0.327948	0.180364	0.111911	0.00662	3.002.198	20.014.653
OMP+MPI	5	4	0.290011	0.151538	0.105526	0.008702	3.394.933	16.974.667
OMP+MPI	5	5	0.299979	0.161255	0.105683	0.011373	3.282.112	13.128.448
OMP+MPI	5	6	0.297223	0.149631	0.114463	0.013466	3.312.545	11.041.818
OMP+MPI	6	1	0.438248	0.148267	0.234867	0.001631	2.246.599	37.443.314
OMP+MPI	6	2	0.320513	0.14718	0.12958	0.004453	3.071.845	25.598.710
OMP+MPI	6	3	0.318859	0.161685	0.121069	0.006641	3.087.780	17.154.331
OMP+MPI	6	4	0.281525	0.140569	0.107136	0.008453	3.497.255	14.571.894
OMP+MPI	6	5	0.282884	0.150047	0.102755	0.01004	3.480.466	11.601.553
OMP+MPI	6	6	0.290041	0.145292	0.111977	0.013125	3.394.576	9.429.379

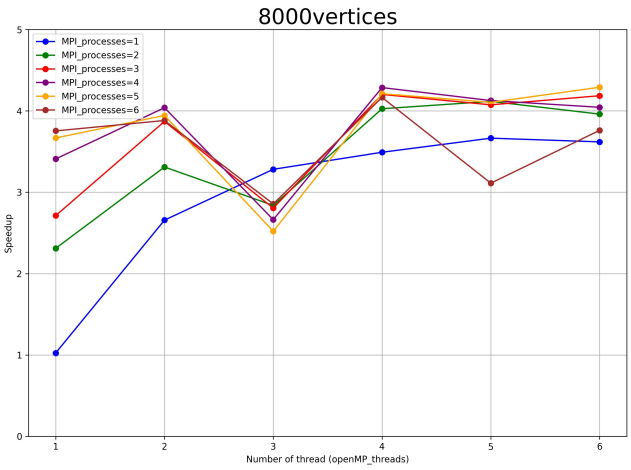
3.2.2 Sparse graph (20%)

For a sparse graph, instead, a percentage of 20% was set.

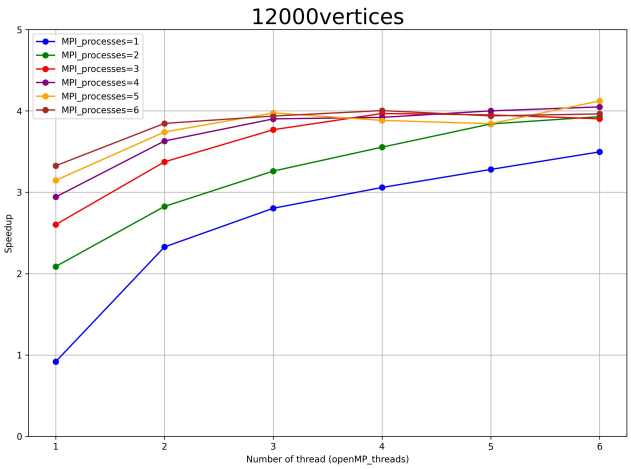
Optimization 0



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.62523	0.220332	0.367294	0.034958	1.000.000	100.000.000
OMP+MPI	1	1	0.656386	0.230238	0.380989	0.042049	0.952533	95.253.330
OMP+MPI	1	2	0.269418	0.158741	0.073184	0.035796	2.320.669	116.033.450
OMP+MPI	1	3	0.244133	0.106898	0.073414	0.062541	2.561.027	85.367.577
OMP+MPI	1	4	0.204251	0.082211	0.068808	0.051921	3.061.087	76.527.165
OMP+MPI	1	5	0.224989	0.071543	0.073539	0.078634	2.778.936	55.578.717
OMP+MPI	1	6	0.219716	0.061638	0.06783	0.087115	2.845.634	47.427.241
OMP+MPI	2	1	0.23061	0.119131	0.072772	0.036011	2.711.207	135.560.330
OMP+MPI	2	2	0.206896	0.089508	0.07378	0.041862	3.021.960	75.549.009
OMP+MPI	2	3	0.18081	0.061862	0.07043	0.047108	3.457.930	57.632.162
OMP+MPI	2	4	0.166633	0.047619	0.067888	0.049971	3.752.127	46.901.583
OMP+MPI	2	5	0.205782	0.046855	0.073441	0.084196	3.038.305	30.383.050
OMP+MPI	2	6	0.193715	0.037628	0.075049	0.079956	3.227.568	26.896.402
OMP+MPI	3	1	0.194603	0.08654	0.074539	0.030305	3.212.840	107.094.682
OMP+MPI	3	2	0.186088	0.060215	0.078372	0.046041	3.359.871	55.997.850
OMP+MPI	3	3	0.159948	0.045351	0.068231	0.044843	3.908.946	43.432.730
OMP+MPI	3	4	0.180269	0.039434	0.07229	0.067188	3.468.317	28.902.640
OMP+MPI	3	5	0.190754	0.051014	0.07065	0.067929	3.277.668	21.851.123
OMP+MPI	3	6	0.184365	0.042656	0.070879	0.069809	3.391.262	18.840.344
OMP+MPI	4	1	0.187275	0.067962	0.078213	0.037854	3.338.557	83.463.934
OMP+MPI	4	2	0.161503	0.048512	0.073966	0.036631	3.871.309	48.391.366
OMP+MPI	4	3	0.176561	0.046385	0.072014	0.056732	3.541.166	29.509.715
OMP+MPI	4	4	0.171334	0.048506	0.070305	0.051342	3.649.188	22.807.426
OMP+MPI	4	5	0.172558	0.039288	0.068869	0.063039	3.623.303	18.116.517
OMP+MPI	4	6	0.200684	0.044687	0.071321	0.083026	3.115.503	12.981.262
OMP+MPI	5	1	0.16004	0.054739	0.073945	0.028067	3.906.723	78.134.461
OMP+MPI	5	2	0.150844	0.042273	0.070997	0.036013	4.144.878	41.448.782
OMP+MPI	5	3	0.17082	0.051976	0.072988	0.044431	3.660.169	24.401.124
OMP+MPI	5	4	0.170019	0.038824	0.070472	0.059576	3.677.413	18.387.063
OMP+MPI	5	5	0.183848	0.042807	0.070576	0.069174	3.400.789	13.603.157
OMP+MPI	5	6	0.202639	0.044057	0.071464	0.085946	3.085.438	10.284.792
OMP+MPI	6	1	0.148342	0.046524	0.073302	0.025876	4.214.773	70.246.221
OMP+MPI	6	2	0.165419	0.046798	0.072332	0.044547	3.779.675	31.497.289
OMP+MPI	6	3	0.168579	0.046315	0.074953	0.045978	3.708.836	20.604.644
OMP+MPI	6	4	0.192138	0.046469	0.075352	0.069038	3.254.076	13.558.649
OMP+MPI	6	5	0.23192	0.058485	0.077441	0.094701	2.695.887	8.986.288
OMP+MPI	6	6	0.205544	0.047091	0.075676	0.081687	3.041.830	8.449.529

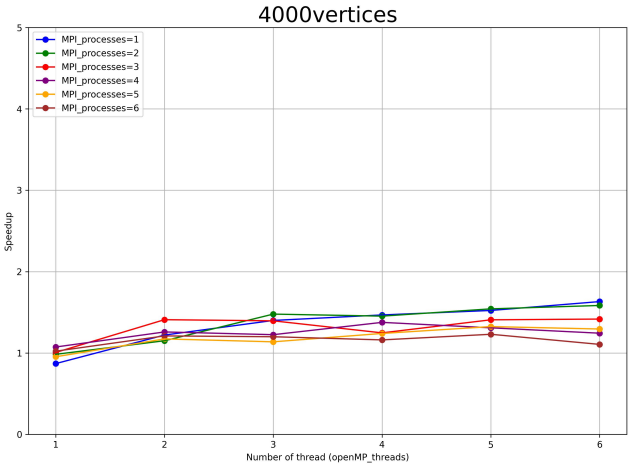


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.864.399	0.583558	1.216.602	0.052775	1.000.000	100.000.000
OMP+MPI	1	1	1.819.844	0.637335	1.132.723	0.039847	1.024.483	102.448.314
OMP+MPI	1	2	0.806875	0.489008	0.270678	0.039983	2.310.642	115.532.115
OMP+MPI	1	3	0.687241	0.387732	0.246308	0.048803	2.712.876	90.429.194
OMP+MPI	1	4	0.546806	0.253825	0.224338	0.064759	3.409.621	85.240.524
OMP+MPI	1	5	0.508435	0.203879	0.231683	0.069306	3.666.941	73.338.827
OMP+MPI	1	6	0.496644	0.181308	0.226572	0.085148	3.754.000	62.566.660
OMP+MPI	2	1	0.701681	0.339854	0.305865	0.043996	2.657.049	132.852.452
OMP+MPI	2	2	0.563271	0.255641	0.254273	0.047253	3.309.951	82.748.779
OMP+MPI	2	3	0.48175	0.181318	0.244013	0.051659	3.870.052	64.500.867
OMP+MPI	2	4	0.461609	0.150149	0.250651	0.056233	4.038.915	50.486.437
OMP+MPI	2	5	0.472739	0.137267	0.243026	0.089156	3.943.820	39.438.200
OMP+MPI	2	6	0.480323	0.142334	0.256685	0.078016	3.881.554	32.346.281
OMP+MPI	3	1	0.568119	0.216412	0.30887	0.031912	3.281.706	109.390.198
OMP+MPI	3	2	0.656969	0.250966	0.325226	0.072592	2.837.880	47.298.008
OMP+MPI	3	3	0.664632	0.188623	0.367436	0.102917	2.805.161	31.168.451
OMP+MPI	3	4	0.700049	0.231839	0.328672	0.128274	2.663.240	22.193.663
OMP+MPI	3	5	0.739811	0.225432	0.349386	0.160715	2.520.101	16.800.672
OMP+MPI	3	6	0.652715	0.221446	0.309108	0.115141	2.856.376	15.868.756
OMP+MPI	4	1	0.53401	0.172405	0.315318	0.035078	3.491.316	87.282.905
OMP+MPI	4	2	0.463078	0.144064	0.270903	0.041777	4.026.103	50.326.281
OMP+MPI	4	3	0.443678	0.143224	0.240165	0.055919	4.202.145	35.017.879
OMP+MPI	4	4	0.434995	0.138714	0.236855	0.055012	4.286.025	26.787.657
OMP+MPI	4	5	0.442749	0.143391	0.226087	0.069665	4.210.963	21.054.813
OMP+MPI	4	6	0.447595	0.132583	0.228002	0.083803	4.165.372	17.355.715
OMP+MPI	5	1	0.508776	0.150469	0.308532	0.03716	3.664.480	73.289.601
OMP+MPI	5	2	0.4529	0.129648	0.272504	0.0445	4.116.585	41.165.855
OMP+MPI	5	3	0.457589	0.138149	0.247954	0.066546	4.074.398	27.162.650
OMP+MPI	5	4	0.451777	0.130302	0.243509	0.073563	4.126.814	20.634.068
OMP+MPI	5	5	0.454221	0.139113	0.233206	0.078338	4.104.609	16.418.435
OMP+MPI	5	6	0.599196	0.216751	0.257918	0.120028	3.111.502	10.371.673
OMP+MPI	6	1	0.515168	0.139761	0.333054	0.031529	3.619.009	60.316.819
OMP+MPI	6	2	0.47077	0.166888	0.253081	0.043441	3.960.315	33.002.625
OMP+MPI	6	3	0.445407	0.132746	0.25318	0.05428	4.185.838	23.254.656
OMP+MPI	6	4	0.461053	0.13025	0.256488	0.070149	4.043.786	16.849.107
OMP+MPI	6	5	0.434571	0.130118	0.226037	0.074549	4.290.202	14.300.673
OMP+MPI	6	6	0.495769	0.165161	0.238531	0.089188	3.760.621	10.446.170

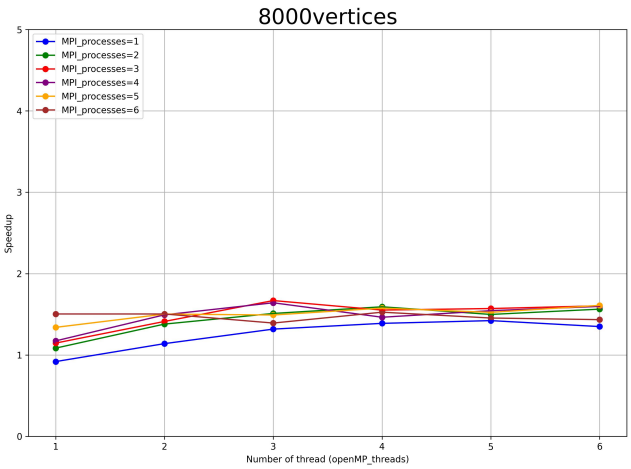


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	3.498.461	1.318.365	2.108.745	0.049442	1.000.000	100.000.000
OMP+MPI	1	1	3.811.513	1.421.887	2.323.104	0.039265	0.917867	91.786.674
OMP+MPI	1	2	1.676.125	1.061.084	0.558641	0.043175	2.087.232	104.361.578
OMP+MPI	1	3	1.344.534	0.734447	0.550116	0.048866	2.601.987	86.732.893
OMP+MPI	1	4	1.189.167	0.57421	0.524015	0.081352	2.941.943	73.548.564
OMP+MPI	1	5	1.112.071	0.49376	0.529658	0.080081	3.145.896	62.917.915
OMP+MPI	1	6	1.051.989	0.411639	0.538023	0.094417	3.325.568	55.426.134
OMP+MPI	2	1	1.503.009	0.742347	0.684571	0.05193	2.327.638	116.381.905
OMP+MPI	2	2	1.238.223	0.581744	0.599018	0.044329	2.825.388	70.634.712
OMP+MPI	2	3	1.037.046	0.417313	0.556683	0.052358	3.373.488	56.224.807
OMP+MPI	2	4	0.963843	0.313802	0.559073	0.08181	3.629.700	45.371.251
OMP+MPI	2	5	0.935379	0.301019	0.55347	0.073258	3.740.155	37.401.555
OMP+MPI	2	6	0.909635	0.281157	0.533039	0.088563	3.846.007	32.050.061
OMP+MPI	3	1	1.247.979	0.514142	0.660064	0.048071	2.803.301	93.443.373
OMP+MPI	3	2	1.073.116	0.417088	0.596651	0.044915	3.260.094	54.334.905
OMP+MPI	3	3	0.928347	0.314814	0.55036	0.051954	3.768.484	41.872.047
OMP+MPI	3	4	0.896691	0.271273	0.544397	0.071357	3.901.521	32.512.678
OMP+MPI	3	5	0.880203	0.27158	0.512773	0.087383	3.974.605	26.497.365
OMP+MPI	3	6	0.888713	0.275946	0.520528	0.08382	3.936.550	21.869.721
OMP+MPI	4	1	1.143.857	0.395212	0.675566	0.047913	3.058.476	76.461.906
OMP+MPI	4	2	0.98439	0.330099	0.58566	0.054214	3.553.940	44.424.247
OMP+MPI	4	3	0.881706	0.280312	0.540555	0.050504	3.967.832	33.065.264
OMP+MPI	4	4	0.892401	0.273798	0.552799	0.057308	3.920.279	24.501.744
OMP+MPI	4	5	0.900956	0.263895	0.544695	0.084515	3.883.054	19.415.271
OMP+MPI	4	6	0.87385	0.267357	0.524523	0.075121	4.003.503	16.681.262
OMP+MPI	5	1	1.066.451	0.322696	0.681002	0.040309	3.280.472	65.609.440
OMP+MPI	5	2	0.911132	0.283519	0.557009	0.055898	3.839.686	38.396.862
OMP+MPI	5	3	0.885628	0.277056	0.53626	0.062518	3.950.262	26.335.083
OMP+MPI	5	4	0.874595	0.269029	0.534996	0.061533	4.000.090	20.000.452
OMP+MPI	5	5	0.910505	0.281732	0.526833	0.093392	3.842.332	15.369.330
OMP+MPI	5	6	0.888346	0.261918	0.520614	0.098597	3.938.176	13.127.254
OMP+MPI	6	1	1.000.660	0.28471	0.658038	0.036233	3.496.152	58.269.197
OMP+MPI	6	2	0.89051	0.265175	0.564995	0.048872	3.928.602	32.738.347
OMP+MPI	6	3	0.896436	0.266543	0.559946	0.060117	3.902.635	21.681.308
OMP+MPI	6	4	0.864048	0.263339	0.525678	0.065932	4.048.920	16.870.499
OMP+MPI	6	5	0.848714	0.25757	0.510829	0.072747	4.122.073	13.740.243
OMP+MPI	6	6	0.882781	0.253967	0.517262	0.103536	3.962.998	11.008.327

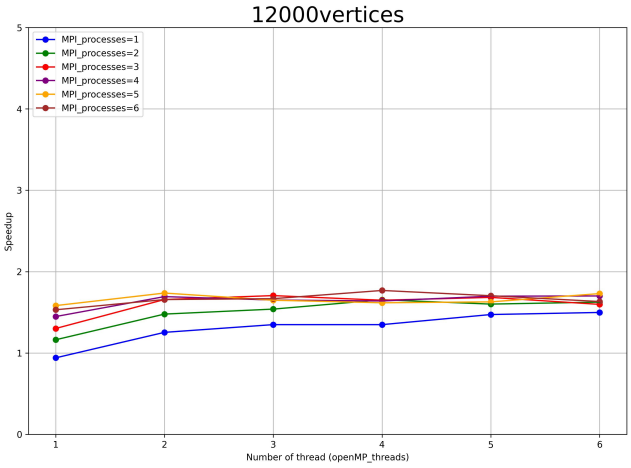
Optimization 1



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.198036	0.089572	0.070373	0.035552	1.000.000	100.000.000
OMP+MPI	1	1	0.227772	0.0997	0.082875	0.042598	0.869448	86.944.811
OMP+MPI	1	2	0.201383	0.074823	0.079827	0.044604	0.983375	49.168.750
OMP+MPI	1	3	0.196762	0.048784	0.07801	0.067303	1.006.472	33.549.076
OMP+MPI	1	4	0.184515	0.044701	0.078127	0.060014	1.073.273	26.831.824
OMP+MPI	1	5	0.207676	0.041246	0.085949	0.079213	0.953579	19.071.583
OMP+MPI	1	6	0.193784	0.026664	0.074388	0.087312	1.021.939	17.032.323
OMP+MPI	2	1	0.162293	0.051028	0.076123	0.031712	1.220.234	61.011.720
OMP+MPI	2	2	0.172148	0.051449	0.078491	0.040295	1.150.383	28.759.567
OMP+MPI	2	3	0.140532	0.026271	0.070017	0.042721	1.409.189	23.486.490
OMP+MPI	2	4	0.157353	0.020927	0.069544	0.065479	1.258.539	15.731.736
OMP+MPI	2	5	0.168825	0.020642	0.071029	0.075825	1.173.019	11.730.189
OMP+MPI	2	6	0.163691	0.020837	0.071571	0.070158	1.209.809	10.081.744
OMP+MPI	3	1	0.141465	0.033808	0.076997	0.027566	1.399.890	46.663.014
OMP+MPI	3	2	0.13402	0.026197	0.070674	0.035452	1.477.656	24.627.605
OMP+MPI	3	3	0.141947	0.02073	0.071749	0.047729	1.395.137	15.501.521
OMP+MPI	3	4	0.161675	0.024459	0.0743	0.06154	1.224.899	10.207.489
OMP+MPI	3	5	0.174159	0.021573	0.071882	0.07944	1.137.096	7.580.640
OMP+MPI	3	6	0.165088	0.020234	0.071893	0.071818	1.199.575	6.664.308
OMP+MPI	4	1	0.134946	0.026351	0.076677	0.029116	1.467.511	36.687.780
OMP+MPI	4	2	0.136294	0.021181	0.076245	0.036932	1.453.008	18.162.596
OMP+MPI	4	3	0.158785	0.024809	0.075161	0.057246	1.247.193	10.393.273
OMP+MPI	4	4	0.143902	0.020299	0.070337	0.051622	1.376.183	8.601.144
OMP+MPI	4	5	0.159697	0.016728	0.067754	0.074052	1.240.070	6.200.351
OMP+MPI	4	6	0.170604	0.019943	0.07226	0.077376	1.160.790	4.836.627
OMP+MPI	5	1	0.130069	0.023411	0.074941	0.028996	1.522.548	30.450.955
OMP+MPI	5	2	0.128405	0.017248	0.075462	0.033756	1.542.272	15.422.725
OMP+MPI	5	3	0.140683	0.023191	0.072466	0.04367	1.407.672	9.384.479
OMP+MPI	5	4	0.151271	0.018015	0.068296	0.063469	1.309.144	6.545.719
OMP+MPI	5	5	0.149545	0.020068	0.068613	0.059799	1.324.254	5.297.014
OMP+MPI	5	6	0.161013	0.017349	0.067587	0.074918	1.229.935	4.099.783
OMP+MPI	6	1	0.121382	0.019733	0.072777	0.026284	1.631.513	27.191.884
OMP+MPI	6	2	0.125027	0.018965	0.070681	0.033906	1.583.942	13.199.516
OMP+MPI	6	3	0.139786	0.018497	0.067733	0.052097	1.416.700	7.870.554
OMP+MPI	6	4	0.159288	0.019748	0.069187	0.069014	1.243.254	5.180.226
OMP+MPI	6	5	0.152965	0.018181	0.07215	0.061408	1.294.646	4.315.486
OMP+MPI	6	6	0.179091	0.021156	0.068901	0.087973	1.105.781	3.071.615

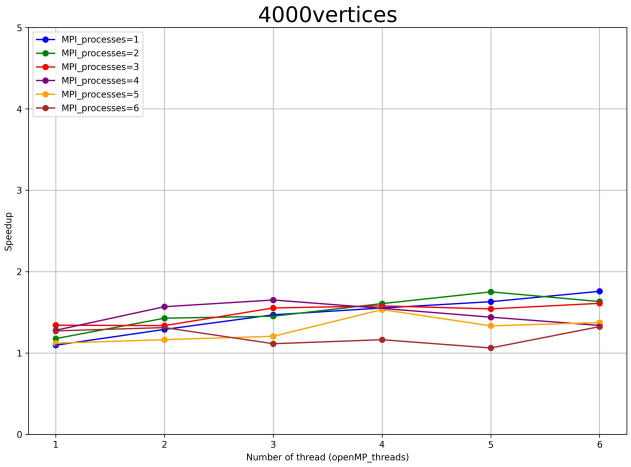


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.57685	0.237836	0.283845	0.045513	1.000.000	100.000.000
OMP+MPI	1	1	0.627695	0.256798	0.326611	0.032458	0.918996	91.899.576
OMP+MPI	1	2	0.531957	0.196683	0.268623	0.060609	1.084.390	54.219.510
OMP+MPI	1	3	0.503251	0.145348	0.263149	0.08948	1.146.247	38.208.242
OMP+MPI	1	4	0.491651	0.123223	0.269678	0.085487	1.173.291	29.332.265
OMP+MPI	1	5	0.430613	0.0957	0.249312	0.082086	1.339.601	26.792.015
OMP+MPI	1	6	0.383528	0.078874	0.229504	0.071978	1.504.059	25.067.650
OMP+MPI	2	1	0.50626	0.13245	0.318701	0.04522	1.139.433	56.971.665
OMP+MPI	2	2	0.41811	0.111748	0.256437	0.043254	1.379.658	34.491.450
OMP+MPI	2	3	0.40848	0.082146	0.251215	0.069751	1.412.185	23.536.424
OMP+MPI	2	4	0.386232	0.069844	0.249418	0.062872	1.493.531	18.669.139
OMP+MPI	2	5	0.383841	0.059867	0.239992	0.079548	1.502.835	15.028.345
OMP+MPI	2	6	0.383574	0.060774	0.228889	0.090914	1.503.879	12.532.322
OMP+MPI	3	1	0.437664	0.092417	0.298479	0.036313	1.318.019	43.933.969
OMP+MPI	3	2	0.381887	0.077531	0.246688	0.051865	1.510.522	25.175.368
OMP+MPI	3	3	0.345674	0.056596	0.238181	0.045775	1.668.767	18.541.860
OMP+MPI	3	4	0.351407	0.052146	0.233674	0.06151	1.641.542	13.679.520
OMP+MPI	3	5	0.386793	0.064072	0.240388	0.078646	1.491.365	9.942.433
OMP+MPI	3	6	0.414276	0.07561	0.230909	0.104078	1.392.426	7.735.702
OMP+MPI	4	1	0.415417	0.072859	0.300805	0.031796	1.388.603	34.715.087
OMP+MPI	4	2	0.36231	0.061776	0.252885	0.041126	1.592.143	19.901.793
OMP+MPI	4	3	0.371717	0.060162	0.246926	0.059743	1.551.851	12.932.094
OMP+MPI	4	4	0.394058	0.075961	0.241766	0.071747	1.463.871	9.149.196
OMP+MPI	4	5	0.366071	0.067057	0.232797	0.062914	1.575.788	7.878.940
OMP+MPI	4	6	0.378038	0.064394	0.234193	0.076435	1.525.903	6.357.931
OMP+MPI	5	1	0.405573	0.06213	0.301873	0.032108	1.422.309	28.446.184
OMP+MPI	5	2	0.384997	0.060296	0.270937	0.046187	1.498.322	14.983.221
OMP+MPI	5	3	0.367217	0.06612	0.244019	0.051428	1.570.870	10.472.469
OMP+MPI	5	4	0.373492	0.057911	0.241414	0.069771	1.544.476	7.722.381
OMP+MPI	5	5	0.378608	0.059684	0.236914	0.078248	1.523.606	6.094.425
OMP+MPI	5	6	0.396682	0.062072	0.238129	0.093165	1.454.186	4.847.287
OMP+MPI	6	1	0.427554	0.055288	0.316391	0.046017	1.349.187	22.486.445
OMP+MPI	6	2	0.368987	0.055641	0.255933	0.050744	1.563.333	13.027.774
OMP+MPI	6	3	0.359633	0.058637	0.249141	0.047686	1.603.995	8.911.083
OMP+MPI	6	4	0.36121	0.055592	0.234113	0.067075	1.596.990	6.654.124
OMP+MPI	6	5	0.358332	0.050453	0.232642	0.071415	1.609.821	5.366.069
OMP+MPI	6	6	0.402172	0.075373	0.230331	0.093072	1.434.334	3.984.260

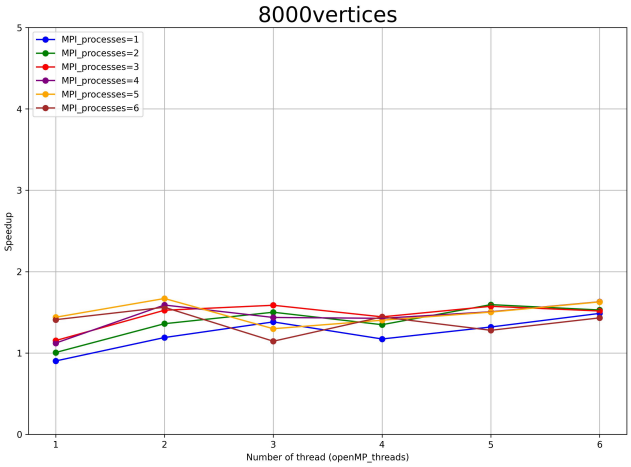


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.243.291	0.536783	0.636079	0.048592	1.000.000	100.000.000
OMP+MPI	1	1	1.322.240	0.565377	0.699633	0.034404	0.940292	94.029.221
OMP+MPI	1	2	1.069.619	0.442565	0.569086	0.044376	1.162.369	58.118.428
OMP+MPI	1	3	0.955682	0.316657	0.569709	0.057447	1.300.946	43.364.873
OMP+MPI	1	4	0.858745	0.236572	0.55008	0.063924	1.447.800	36.194.993
OMP+MPI	1	5	0.785738	0.191737	0.52032	0.066348	1.582.323	31.646.465
OMP+MPI	1	6	0.812146	0.167044	0.535276	0.103175	1.530.872	25.514.532
OMP+MPI	2	1	0.991872	0.284937	0.651669	0.034059	1.253.480	62.673.989
OMP+MPI	2	2	0.841174	0.239274	0.548212	0.040967	1.478.043	36.951.080
OMP+MPI	2	3	0.750307	0.167969	0.520935	0.050759	1.657.044	27.617.395
OMP+MPI	2	4	0.73447	0.133205	0.503875	0.088036	1.692.774	21.159.671
OMP+MPI	2	5	0.71599	0.120589	0.514613	0.071824	1.736.464	17.364.637
OMP+MPI	2	6	0.75057	0.135202	0.528478	0.078687	1.656.464	13.803.868
OMP+MPI	3	1	0.92203	0.204586	0.659547	0.036539	1.348.428	44.947.591
OMP+MPI	3	2	0.807737	0.173581	0.573932	0.044686	1.539.228	25.653.802
OMP+MPI	3	3	0.729119	0.125318	0.540541	0.053205	1.705.198	18.946.646
OMP+MPI	3	4	0.753229	0.135093	0.52761	0.080938	1.650.616	13.755.130
OMP+MPI	3	5	0.754182	0.136693	0.52237	0.074237	1.648.531	10.990.206
OMP+MPI	3	6	0.745626	0.120336	0.536938	0.081382	1.667.445	9.263.586
OMP+MPI	4	1	0.922069	0.160786	0.680627	0.058549	1.348.371	33.709.286
OMP+MPI	4	2	0.751597	0.135561	0.559294	0.04346	1.654.201	20.677.509
OMP+MPI	4	3	0.754301	0.137758	0.545062	0.059271	1.648.269	13.735.572
OMP+MPI	4	4	0.75848	0.138447	0.547759	0.063254	1.639.188	10.244.927
OMP+MPI	4	5	0.768935	0.133984	0.527481	0.097355	1.616.902	8.084.508
OMP+MPI	4	6	0.703084	0.121272	0.498022	0.076508	1.768.340	7.368.083
OMP+MPI	5	1	0.84433	0.131557	0.652561	0.039281	1.472.518	29.450.369
OMP+MPI	5	2	0.776657	0.120332	0.575773	0.067925	1.600.824	16.008.244
OMP+MPI	5	3	0.738642	0.130478	0.544153	0.052612	1.683.213	11.221.417
OMP+MPI	5	4	0.732166	0.123603	0.539817	0.059266	1.698.101	8.490.503
OMP+MPI	5	5	0.763613	0.128499	0.535981	0.090996	1.628.170	6.512.675
OMP+MPI	5	6	0.729795	0.11482	0.522992	0.085031	1.703.617	5.678.725
OMP+MPI	6	1	0.829935	0.114979	0.654411	0.039505	1.498.059	24.967.648
OMP+MPI	6	2	0.765051	0.133489	0.571576	0.047767	1.625.108	13.542.569
OMP+MPI	6	3	0.778405	0.127381	0.556138	0.085168	1.597.230	8.873.498
OMP+MPI	6	4	0.731013	0.122841	0.539362	0.060681	1.700.778	7.086.574
OMP+MPI	6	5	0.718734	0.122113	0.517269	0.071166	1.729.835	5.766.118
OMP+MPI	6	6	0.761698	0.119373	0.533822	0.101746	1.632.264	4.534.067

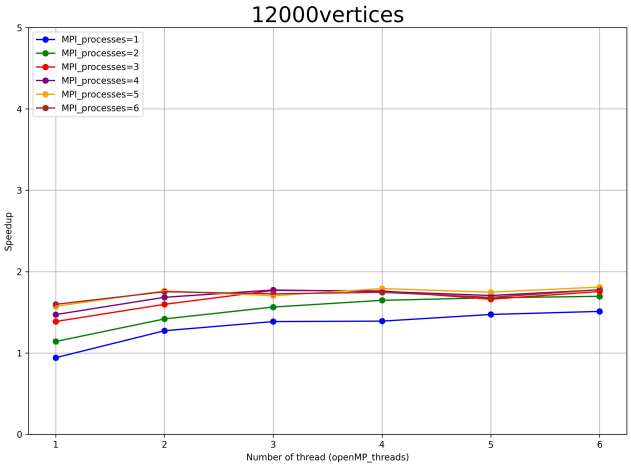
Optimization 2



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.230544	0.105744	0.070256	0.050368	1.000.000	100.000.000
OMP+MPI	1	1	0.210397	0.103276	0.075758	0.028554	1.095.759	109.575.945
OMP+MPI	1	2	0.195935	0.075561	0.078029	0.040329	1.176.635	58.831.733
OMP+MPI	1	3	0.171855	0.050733	0.071871	0.047317	1.341.502	44.716.734
OMP+MPI	1	4	0.180274	0.041694	0.067541	0.069655	1.278.856	31.971.402
OMP+MPI	1	5	0.205806	0.047147	0.071765	0.085833	1.120.206	22.404.115
OMP+MPI	1	6	0.181341	0.031558	0.06939	0.079313	1.271.328	21.188.798
OMP+MPI	2	1	0.178806	0.055736	0.083863	0.036489	1.289.359	64.467.955
OMP+MPI	2	2	0.161536	0.044557	0.075482	0.039191	1.427.202	35.680.050
OMP+MPI	2	3	0.172526	0.037617	0.078917	0.054169	1.336.285	22.271.410
OMP+MPI	2	4	0.146967	0.024637	0.070235	0.050623	1.568.677	19.608.459
OMP+MPI	2	5	0.198147	0.024919	0.072093	0.099922	1.163.502	11.635.023
OMP+MPI	2	6	0.175702	0.028563	0.07441	0.071511	1.312.137	10.934.478
OMP+MPI	3	1	0.156912	0.040786	0.080794	0.032246	1.469.265	48.975.484
OMP+MPI	3	2	0.158787	0.030971	0.076304	0.049577	1.451.906	24.198.431
OMP+MPI	3	3	0.14845	0.024255	0.07475	0.047854	1.553.011	17.255.679
OMP+MPI	3	4	0.139634	0.01968	0.068569	0.050062	1.651.063	13.758.856
OMP+MPI	3	5	0.191302	0.026286	0.070024	0.093488	1.205.134	8.034.225
OMP+MPI	3	6	0.206981	0.033694	0.081155	0.088618	1.113.846	6.188.036
OMP+MPI	4	1	0.148477	0.032643	0.079987	0.033175	1.552.734	38.818.348
OMP+MPI	4	2	0.143498	0.025423	0.072606	0.043688	1.606.604	20.082.553
OMP+MPI	4	3	0.146049	0.021485	0.071796	0.05146	1.578.542	13.154.518
OMP+MPI	4	4	0.14889	0.023999	0.07413	0.049499	1.548.427	9.677.668
OMP+MPI	4	5	0.150347	0.020728	0.070721	0.057743	1.533.411	7.667.055
OMP+MPI	4	6	0.198222	0.030934	0.079106	0.087118	1.163.059	4.846.080
OMP+MPI	5	1	0.141425	0.027917	0.075874	0.034978	1.630.160	32.603.191
OMP+MPI	5	2	0.131671	0.022799	0.071801	0.035732	1.750.920	17.509.199
OMP+MPI	5	3	0.149538	0.026384	0.067978	0.053581	1.541.712	10.278.079
OMP+MPI	5	4	0.160099	0.020978	0.073592	0.064089	1.440.012	7.200.061
OMP+MPI	5	5	0.172874	0.027067	0.069135	0.075163	1.333.595	5.334.378
OMP+MPI	5	6	0.217166	0.024237	0.088536	0.103227	1.061.603	3.538.675
OMP+MPI	6	1	0.131165	0.0237	0.073696	0.03038	1.757.661	29.294.352
OMP+MPI	6	2	0.141275	0.028004	0.074545	0.036878	1.631.885	13.599.039
OMP+MPI	6	3	0.143162	0.023238	0.072199	0.046279	1.610.369	8.946.496
OMP+MPI	6	4	0.17221	0.025708	0.077049	0.068344	1.338.740	5.578.085
OMP+MPI	6	5	0.167843	0.024969	0.076926	0.064986	1.373.576	4.578.588
OMP+MPI	6	6	0.174173	0.021728	0.073246	0.078241	1.323.649	3.676.802

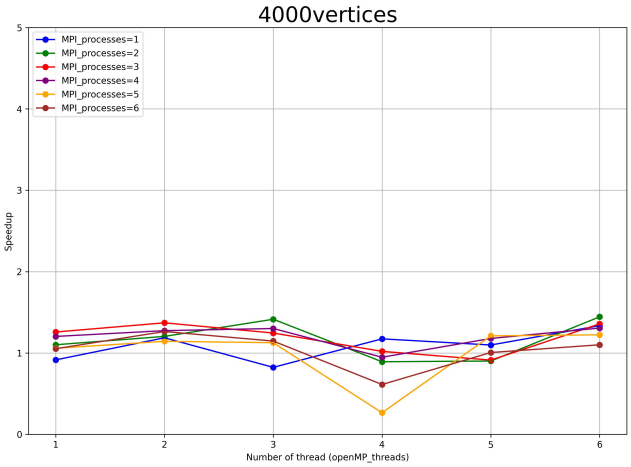


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.61725	0.272612	0.280357	0.053986	1.000.000	100.000.000
OMP+MPI	1	1	0.684303	0.315568	0.321391	0.034463	0.902014	90.201.409
OMP+MPI	1	2	0.613555	0.261393	0.295127	0.049708	1.006.023	50.301.155
OMP+MPI	1	3	0.535568	0.180106	0.28378	0.063829	1.152.516	38.417.188
OMP+MPI	1	4	0.550379	0.18658	0.271049	0.088553	1.121.501	28.037.521
OMP+MPI	1	5	0.428879	0.106806	0.245901	0.07243	1.439.218	28.784.366
OMP+MPI	1	6	0.437936	0.096	0.237406	0.101108	1.409.454	23.490.894
OMP+MPI	2	1	0.518757	0.14548	0.324584	0.034682	1.189.863	59.493.164
OMP+MPI	2	2	0.453906	0.128595	0.266682	0.050272	1.359.864	33.996.604
OMP+MPI	2	3	0.404705	0.093646	0.25222	0.053463	1.525.188	25.419.802
OMP+MPI	2	4	0.388371	0.076705	0.24662	0.06076	1.589.330	19.866.626
OMP+MPI	2	5	0.36972	0.066172	0.230042	0.069098	1.669.508	16.695.080
OMP+MPI	2	6	0.395184	0.074124	0.226951	0.087234	1.561.934	13.016.116
OMP+MPI	3	1	0.446657	0.099494	0.30682	0.030312	1.381.936	46.064.519
OMP+MPI	3	2	0.411268	0.091701	0.271589	0.041752	1.500.849	25.014.153
OMP+MPI	3	3	0.389072	0.077091	0.247038	0.058219	1.586.471	17.627.451
OMP+MPI	3	4	0.429659	0.082167	0.266732	0.075779	1.436.606	11.971.713
OMP+MPI	3	5	0.475493	0.076509	0.249391	0.144768	1.298.127	8.654.183
OMP+MPI	3	6	0.539261	0.118515	0.264524	0.152753	1.144.623	6.359.016
OMP+MPI	4	1	0.526649	0.094482	0.360622	0.059444	1.172.035	29.300.876
OMP+MPI	4	2	0.458037	0.085284	0.313018	0.052256	1.347.598	16.844.977
OMP+MPI	4	3	0.427438	0.084005	0.281211	0.057238	1.444.069	12.033.905
OMP+MPI	4	4	0.432859	0.078341	0.277083	0.073715	1.425.985	8.912.407
OMP+MPI	4	5	0.440041	0.072738	0.26894	0.094098	1.402.711	7.013.557
OMP+MPI	4	6	0.427735	0.091115	0.252242	0.080278	1.443.068	6.012.781
OMP+MPI	5	1	0.467851	0.078828	0.329259	0.045159	1.319.331	26.386.627
OMP+MPI	5	2	0.387215	0.064829	0.266973	0.049809	1.594.079	15.940.790
OMP+MPI	5	3	0.392255	0.073285	0.249815	0.064024	1.573.597	10.490.647
OMP+MPI	5	4	0.409735	0.066402	0.250748	0.087521	1.506.463	7.532.314
OMP+MPI	5	5	0.410776	0.072347	0.258663	0.075898	1.502.647	6.010.587
OMP+MPI	5	6	0.482654	0.101846	0.255131	0.122232	1.278.867	4.262.892
OMP+MPI	6	1	0.415268	0.061015	0.304618	0.035967	1.486.389	24.773.149
OMP+MPI	6	2	0.403876	0.079512	0.269508	0.047566	1.528.317	12.735.974
OMP+MPI	6	3	0.407131	0.07072	0.276579	0.055006	1.516.096	8.422.756
OMP+MPI	6	4	0.378829	0.065944	0.254827	0.053693	1.629.362	6.789.010
OMP+MPI	6	5	0.37912	0.061576	0.246571	0.067478	1.628.112	5.427.039
OMP+MPI	6	6	0.431184	0.089626	0.233281	0.104631	1.431.523	3.976.453

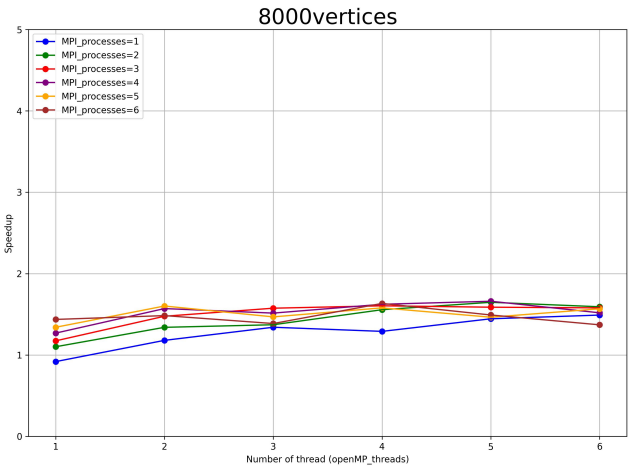


Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.306.534	0.610373	0.624932	0.049427	1.000.000	100.000.000
OMP+MPI	1	1	1.386.874	0.638316	0.688206	0.036559	0.942072	94.207.186
OMP+MPI	1	2	1.144.739	0.486711	0.594731	0.05086	1.141.338	57.066.917
OMP+MPI	1	3	0.942201	0.346469	0.538003	0.048658	1.386.683	46.222.756
OMP+MPI	1	4	0.887703	0.281711	0.527023	0.070205	1.471.816	36.795.393
OMP+MPI	1	5	0.830622	0.223809	0.525502	0.073704	1.572.960	31.459.203
OMP+MPI	1	6	0.817514	0.202086	0.512747	0.095245	1.598.180	26.636.333
OMP+MPI	2	1	1.025.975	0.324732	0.643867	0.035112	1.273.456	63.672.823
OMP+MPI	2	2	0.92108	0.268552	0.593664	0.042571	1.418.480	35.462.006
OMP+MPI	2	3	0.817702	0.201689	0.550277	0.05174	1.597.813	26.630.209
OMP+MPI	2	4	0.775683	0.162306	0.534093	0.070088	1.684.367	21.054.582
OMP+MPI	2	5	0.740932	0.134118	0.524833	0.074611	1.763.367	17.633.675
OMP+MPI	2	6	0.745429	0.136343	0.516566	0.085256	1.752.727	14.606.059
OMP+MPI	3	1	0.942694	0.231664	0.651374	0.035784	1.385.958	46.198.607
OMP+MPI	3	2	0.835089	0.20866	0.55849	0.05563	1.564.545	26.075.754
OMP+MPI	3	3	0.738712	0.145507	0.531651	0.051039	1.768.664	19.651.827
OMP+MPI	3	4	0.736546	0.146401	0.522453	0.058548	1.773.868	14.782.233
OMP+MPI	3	5	0.767217	0.163873	0.514314	0.081045	1.702.952	11.353.013
OMP+MPI	3	6	0.756476	0.146023	0.515927	0.087382	1.727.134	9.595.188
OMP+MPI	4	1	0.939018	0.188011	0.676349	0.04833	1.391.383	34.784.578
OMP+MPI	4	2	0.793254	0.161502	0.564728	0.053507	1.647.058	20.588.225
OMP+MPI	4	3	0.741214	0.149124	0.528602	0.05304	1.762.695	14.689.128
OMP+MPI	4	4	0.744202	0.149531	0.524167	0.061354	1.755.618	10.972.613
OMP+MPI	4	5	0.729255	0.132784	0.513693	0.074814	1.791.602	8.958.009
OMP+MPI	4	6	0.748563	0.139124	0.523748	0.078596	1.745.389	7.272.454
OMP+MPI	5	1	0.886759	0.158004	0.670575	0.035604	1.473.382	29.467.634
OMP+MPI	5	2	0.778482	0.143509	0.57162	0.049119	1.678.310	16.783.105
OMP+MPI	5	3	0.786572	0.16508	0.547578	0.063134	1.661.049	11.073.659
OMP+MPI	5	4	0.765688	0.143921	0.546628	0.06664	1.706.353	8.531.763
OMP+MPI	5	5	0.747899	0.145821	0.515795	0.078512	1.746.941	6.987.764
OMP+MPI	5	6	0.776381	0.154697	0.518413	0.096215	1.682.853	5.609.511
OMP+MPI	6	1	0.864663	0.135408	0.66923	0.037129	1.511.034	25.183.901
OMP+MPI	6	2	0.770123	0.147519	0.562473	0.047896	1.696.526	14.137.716
OMP+MPI	6	3	0.74417	0.147097	0.530826	0.055133	1.755.695	9.753.860
OMP+MPI	6	4	0.73558	0.136921	0.516808	0.071682	1.776.198	7.400.823
OMP+MPI	6	5	0.721804	0.13553	0.511431	0.066761	1.810.096	6.033.653
OMP+MPI	6	6	0.734499	0.129952	0.511698	0.085898	1.778.810	4.941.140

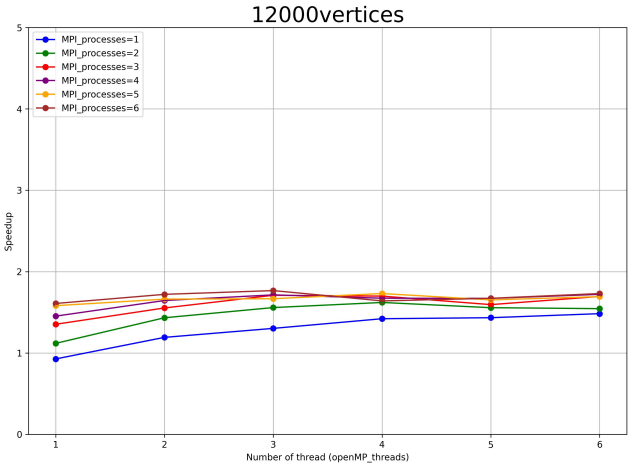
Optimization 3



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.214931	0.102987	0.072271	0.03686	1.000.000	100.000.000
OMP+MPI	1	1	0.234879	0.120202	0.073937	0.037907	0.915071	91.507.116
OMP+MPI	1	2	0.195215	0.083168	0.075246	0.035304	1.100.996	55.049.817
OMP+MPI	1	3	0.170915	0.056283	0.070955	0.042346	1.257.532	41.917.717
OMP+MPI	1	4	0.178592	0.047568	0.074546	0.055295	1.203.475	30.086.874
OMP+MPI	1	5	0.203207	0.046108	0.074932	0.081043	1.057.695	21.153.897
OMP+MPI	1	6	0.204231	0.038611	0.08422	0.080313	1.052.392	17.539.861
OMP+MPI	2	1	0.181104	0.067096	0.078667	0.032478	1.186.782	59.339.109
OMP+MPI	2	2	0.178567	0.04701	0.076675	0.052845	1.203.640	30.091.002
OMP+MPI	2	3	0.156797	0.036992	0.072839	0.045579	1.370.755	22.845.921
OMP+MPI	2	4	0.168652	0.02914	0.070331	0.06777	1.274.402	15.930.019
OMP+MPI	2	5	0.187955	0.030909	0.077091	0.07808	1.143.521	11.435.207
OMP+MPI	2	6	0.170099	0.03101	0.069306	0.068736	1.263.564	10.529.701
OMP+MPI	3	1	0.261022	0.035878	0.075023	0.147616	0.823421	27.447.367
OMP+MPI	3	2	0.151931	0.038477	0.070827	0.040387	1.414.662	23.577.699
OMP+MPI	3	3	0.172633	0.034686	0.078027	0.058336	1.245.017	13.833.521
OMP+MPI	3	4	0.165207	0.029829	0.072669	0.061537	1.300.984	10.841.533
OMP+MPI	3	5	0.190798	0.031102	0.07704	0.081379	1.126.485	7.509.897
OMP+MPI	3	6	0.187492	0.027729	0.071772	0.086864	1.146.348	6.368.598
OMP+MPI	4	1	0.183137	0.044678	0.077925	0.049324	1.173.608	29.340.193
OMP+MPI	4	2	0.240843	0.048219	0.087112	0.101935	0.892413	11.155.164
OMP+MPI	4	3	0.21076	0.041827	0.092185	0.074749	1.019.788	8.498.232
OMP+MPI	4	4	0.226747	0.052477	0.09071	0.07988	0.947889	5.924.307
OMP+MPI	4	5	0.808627	0.034711	0.094183	0.638526	0.265798	1.328.988
OMP+MPI	4	6	0.350747	0.044465	0.089904	0.210729	0.612781	2.553.253
OMP+MPI	5	1	0.195746	0.034984	0.111244	0.045998	1.098.010	21.960.193
OMP+MPI	5	2	0.238503	0.049701	0.124155	0.062295	0.901169	9.011.688
OMP+MPI	5	3	0.235503	0.040614	0.088367	0.103816	0.912648	6.084.323
OMP+MPI	5	4	0.182393	0.032021	0.081397	0.067453	1.178.398	5.891.991
OMP+MPI	5	5	0.177558	0.029399	0.07761	0.069301	1.210.483	4.841.933
OMP+MPI	5	6	0.213704	0.032938	0.086318	0.092927	1.005.739	3.352.464
OMP+MPI	6	1	0.160938	0.028729	0.088546	0.040782	1.335.489	22.258.157
OMP+MPI	6	2	0.148803	0.030717	0.076297	0.039815	1.444.404	12.036.704
OMP+MPI	6	3	0.158323	0.027827	0.083597	0.045074	1.357.548	7.541.931
OMP+MPI	6	4	0.164553	0.027658	0.07356	0.061991	1.306.147	5.442.278
OMP+MPI	6	5	0.175684	0.027231	0.070949	0.076222	1.223.392	4.077.973
OMP+MPI	6	6	0.195284	0.02908	0.073511	0.091168	1.100.607	3.057.243



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.606467	0.274023	0.278743	0.043825	1.000.000	100.000.000
OMP+MPI	1	1	0.660136	0.319809	0.296907	0.030425	0.918699	91.869.863
OMP+MPI	1	2	0.550712	0.253598	0.253961	0.037151	1.101.240	55.061.988
OMP+MPI	1	3	0.516688	0.191164	0.241104	0.079384	1.173.758	39.125.255
OMP+MPI	1	4	0.477946	0.160339	0.245237	0.068115	1.268.902	31.722.543
OMP+MPI	1	5	0.452376	0.12224	0.250088	0.076432	1.340.626	26.812.526
OMP+MPI	1	6	0.422067	0.10757	0.225285	0.085775	1.436.896	23.948.271
OMP+MPI	2	1	0.514084	0.173681	0.299237	0.030739	1.179.703	58.985.156
OMP+MPI	2	2	0.452654	0.14163	0.266725	0.037647	1.339.800	33.495.000
OMP+MPI	2	3	0.410891	0.111017	0.24526	0.04921	1.475.979	24.599.651
OMP+MPI	2	4	0.386221	0.089153	0.235588	0.056944	1.570.256	19.628.196
OMP+MPI	2	5	0.378642	0.073354	0.239573	0.062054	1.601.688	16.016.884
OMP+MPI	2	6	0.408719	0.091491	0.231923	0.081818	1.483.823	12.365.189
OMP+MPI	3	1	0.452307	0.114207	0.298399	0.029045	1.340.828	44.694.262
OMP+MPI	3	2	0.44218	0.109632	0.27029	0.055901	1.371.538	22.858.960
OMP+MPI	3	3	0.38508	0.080791	0.245653	0.053717	1.574.908	17.498.982
OMP+MPI	3	4	0.400405	0.077885	0.246818	0.07138	1.514.631	12.621.923
OMP+MPI	3	5	0.413044	0.086447	0.244995	0.078011	1.468.285	9.788.570
OMP+MPI	3	6	0.43742	0.108999	0.247648	0.077	1.386.463	7.702.570
OMP+MPI	4	1	0.470066	0.101878	0.315659	0.041119	1.290.174	32.254.361
OMP+MPI	4	2	0.389654	0.084847	0.25559	0.042394	1.556.423	19.455.289
OMP+MPI	4	3	0.377998	0.082619	0.241879	0.048313	1.604.417	13.370.143
OMP+MPI	4	4	0.37361	0.082512	0.23009	0.056856	1.623.263	10.145.394
OMP+MPI	4	5	0.384071	0.073269	0.226621	0.080548	1.579.048	7.895.239
OMP+MPI	4	6	0.371248	0.072433	0.222653	0.073002	1.633.589	6.806.619
OMP+MPI	5	1	0.419765	0.083362	0.293029	0.031995	1.444.776	28.895.525
OMP+MPI	5	2	0.368109	0.072774	0.248984	0.03992	1.647.519	16.475.188
OMP+MPI	5	3	0.382127	0.085557	0.23503	0.057101	1.587.081	10.580.540
OMP+MPI	5	4	0.36528	0.07126	0.233681	0.056361	1.660.278	8.301.392
OMP+MPI	5	5	0.414194	0.101289	0.225886	0.083366	1.464.209	5.856.835
OMP+MPI	5	6	0.40666	0.092856	0.227303	0.083052	1.491.334	4.971.112
OMP+MPI	6	1	0.40686	0.069639	0.293322	0.034193	1.490.602	24.843.374
OMP+MPI	6	2	0.38083	0.083291	0.25153	0.040755	1.592.486	13.270.718
OMP+MPI	6	3	0.384039	0.080451	0.240453	0.058147	1.579.179	8.773.219
OMP+MPI	6	4	0.399424	0.087582	0.2357	0.072053	1.518.351	6.326.462
OMP+MPI	6	5	0.386848	0.073935	0.242252	0.067365	1.567.713	5.225.709
OMP+MPI	6	6	0.442475	0.105361	0.231719	0.102211	1.370.623	3.807.287



Type	OMP_threads	MPI_process	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.330.078	0.619395	0.638809	0.049634	1.000.000	100.000.000
OMP+MPI	1	1	1.434.991	0.7214	0.656802	0.032955	0.926889	92.688.877
OMP+MPI	1	2	1.189.526	0.573999	0.556407	0.044965	1.118.157	55.907.855
OMP+MPI	1	3	0.983106	0.379997	0.531757	0.061663	1.352.934	45.097.799
OMP+MPI	1	4	0.915525	0.302832	0.531019	0.074104	1.452.802	36.320.056
OMP+MPI	1	5	0.840947	0.254151	0.511549	0.067444	1.581.642	31.632.850
OMP+MPI	1	6	0.826883	0.227431	0.506632	0.084695	1.608.543	26.809.047
OMP+MPI	2	1	1.116.000	0.377004	0.670117	0.045552	1.191.826	59.591.313
OMP+MPI	2	2	0.928365	0.315676	0.558057	0.04191	1.432.710	35.817.741
OMP+MPI	2	3	0.85633	0.236478	0.554385	0.0534	1.553.230	25.887.168
OMP+MPI	2	4	0.808648	0.192306	0.541079	0.065612	1.644.815	20.560.192
OMP+MPI	2	5	0.800388	0.168874	0.53821	0.085304	1.661.792	16.617.919
OMP+MPI	2	6	0.773062	0.160745	0.518189	0.087656	1.720.533	14.337.772
OMP+MPI	3	1	1.021.265	0.279251	0.67931	0.03923	1.302.383	43.412.766
OMP+MPI	3	2	0.853638	0.230136	0.559234	0.049462	1.558.128	25.968.804
OMP+MPI	3	3	0.778202	0.178371	0.528259	0.05982	1.709.167	18.990.749
OMP+MPI	3	4	0.775961	0.17124	0.52279	0.072534	1.714.105	14.284.205
OMP+MPI	3	5	0.797926	0.187975	0.520353	0.081271	1.666.919	11.112.796
OMP+MPI	3	6	0.752835	0.163815	0.502541	0.079712	1.766.758	9.815.324
OMP+MPI	4	1	0.936043	0.213103	0.663984	0.036214	1.420.958	35.523.942
OMP+MPI	4	2	0.820579	0.18971	0.573581	0.043706	1.620.901	20.261.265
OMP+MPI	4	3	0.782892	0.186245	0.531991	0.053955	1.698.927	14.157.728
OMP+MPI	4	4	0.794961	0.197	0.530021	0.059967	1.673.136	10.457.097
OMP+MPI	4	5	0.768601	0.159002	0.531574	0.07052	1.730.516	8.652.582
OMP+MPI	4	6	0.811043	0.173562	0.526725	0.099219	1.639.959	6.833.164
OMP+MPI	5	1	0.927826	0.184802	0.673244	0.044027	1.433.541	28.670.824
OMP+MPI	5	2	0.854428	0.181241	0.613181	0.047186	1.556.689	15.566.885
OMP+MPI	5	3	0.83425	0.200059	0.566982	0.054879	1.594.339	10.628.928
OMP+MPI	5	4	0.79585	0.172411	0.550594	0.064014	1.671.267	8.356.333
OMP+MPI	5	5	0.804836	0.169382	0.534744	0.092432	1.652.607	6.610.427
OMP+MPI	5	6	0.795489	0.171576	0.540861	0.076772	1.672.025	5.573.417
OMP+MPI	6	1	0.896733	0.158093	0.681202	0.035681	1.483.248	24.720.801
OMP+MPI	6	2	0.860965	0.17818	0.60583	0.063978	1.544.868	12.873.902
OMP+MPI	6	3	0.784843	0.171517	0.545032	0.056365	1.694.705	9.415.029
OMP+MPI	6	4	0.773102	0.167712	0.535434	0.0609	1.720.442	7.168.510
OMP+MPI	6	5	0.785953	0.158468	0.530088	0.090214	1.692.312	5.641.039
OMP+MPI	6	6	0.768712	0.154736	0.527582	0.079227	1.730.268	4.806.299

3.2.3 Considerations

Taking into account that the tests were conducted on a system not exclusively dedicated to testing, it is predictable that other concurrently running processes could have influenced the results. By increasing the iterations, these fluctuations would likely decrease, although they may never completely vanish. Consequently, it is observed that the performance of certain speedups is irregular for a specific number of processes. Fortunately, the ideal trend is easily discernible. Indeed, it is apparent that the overall program (all speedups refer to the entire program execution, from graph reading to output writing) has benefited from parallelization with both multiple threads and processes. Simultaneously, it is clear that the growth of speedup is not linear with the increase in processes and threads but rather logarithmic.

Considering the efficiency, it is evident that it is maximized when only 2 MPI processes or only 2 OpenMP threads are present. This implies that the algorithm, as implemented, consistently reduces the execution time (at least until hardware resources can meet the resource demands) but maintains higher efficiency for a lower number of processes and threads. It is worth noting that, in the calculation of efficiency, the number of processes was considered as the product of the number of processes and the number of threads (since each process, in turn, generated a certain number of threads).

Furthermore, it is observed that increasing the optimization level from 0 to 1 disproportionately benefits the sequential algorithm rather than the parallel algorithm. In fact, the parallel speedup is high because, in reality, the sequential version compiled with optimization level 0 is much slower than when compiled with optimization level 1.

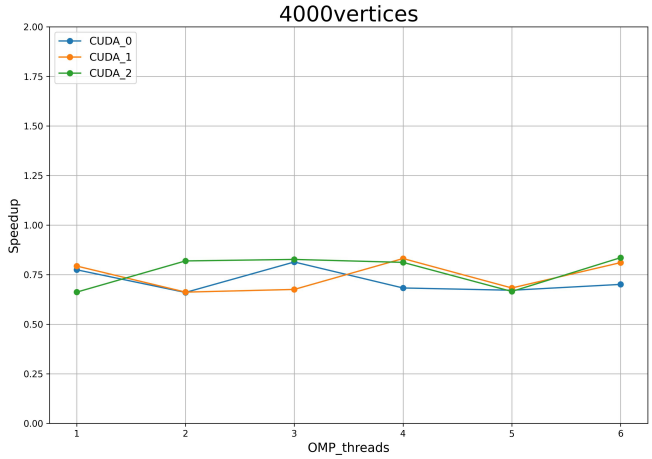
Hence, it is evident that the first-level optimization is crucial for the sequential algorithm to expedite all the necessary 'for' loops in the algorithm's execution.

3.3 OpenMP+CUDA

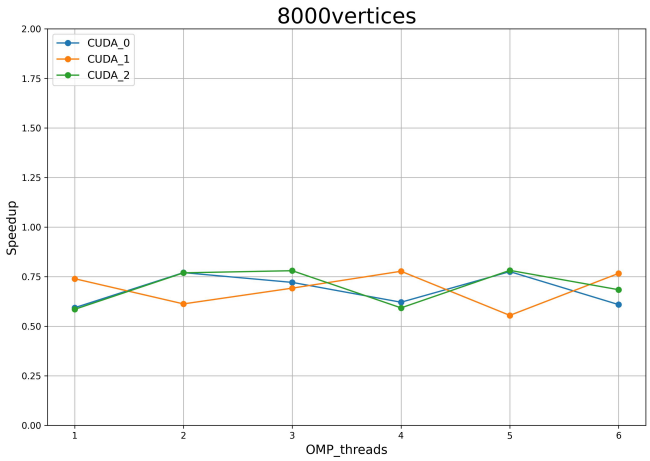
3.3.1 Densely connected graph (80%)

A densely connected graph example was taken, with 80% of edges, meaning a graph whose adjacency matrix is filled with 80%.

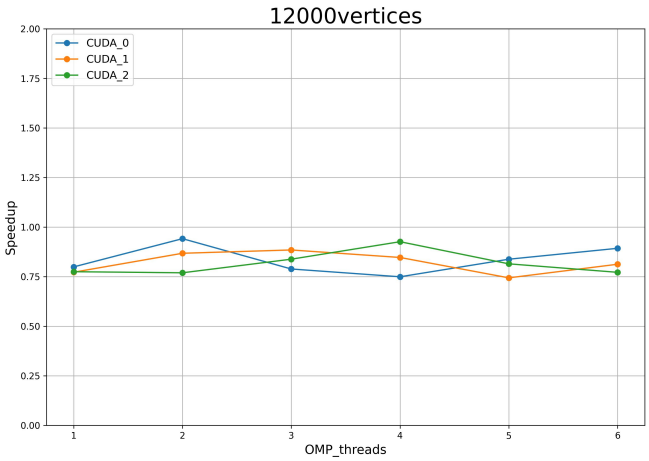
Optimization 0



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.473965	0.305385	0.154984	0.006292	1.000.000
OMP+CUD	1	12801792	0.611953	0.505664	0.093943	0.007435	0.774511
OMP+CUD	2	12801792	0.717896	0.580968	0.123372	0.006986	0.660213
OMP+CUD	3	12801792	0.582413	0.473139	0.086513	0.017352	0.813794
OMP+CUD	4	12801792	0.69419	0.558303	0.122018	0.006709	0.682759
OMP+CUD	5	12801792	0.706027	0.558225	0.1305	0.009959	0.671312
OMP+CUD	6	12801792	0.676268	0.541308	0.121535	0.006927	0.700854
OMP+CUD	1	12801792	0.597409	0.491916	0.091109	0.007883	0.793367
OMP+CUD	2	12801792	0.716118	0.580843	0.120846	0.007229	0.661853
OMP+CUD	3	12801792	0.70173	0.557073	0.130764	0.006719	0.675423
OMP+CUD	4	12801792	0.570184	0.47155	0.086243	0.006879	0.831248
OMP+CUD	5	12801792	0.694186	0.556763	0.122594	0.006802	0.682763
OMP+CUD	6	12801792	0.584624	0.484113	0.087939	0.007067	0.810717
OMP+CUD	1	12801792	0.716107	0.581703	0.12056	0.00707	0.661863
OMP+CUD	2	12801792	0.578567	0.472805	0.092326	0.007615	0.819204
OMP+CUD	3	12801792	0.573276	0.470135	0.090469	0.00706	0.826765
OMP+CUD	4	12801792	0.583767	0.482422	0.089284	0.006577	0.811907
OMP+CUD	5	12801792	0.712546	0.56855	0.130553	0.006699	0.66517
OMP+CUD	6	12801792	0.567206	0.468989	0.086079	0.006774	0.835612

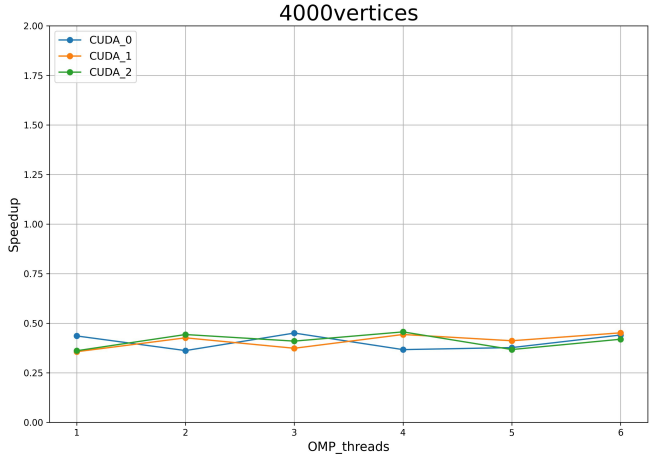


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	1.350.478	0.926641	0.392096	0.011516	1.000.000
OMP+CUD	1	51203520	2.275.983	1.753.406	0.488688	0.009489	0.59336
OMP+CUD	2	51203520	1.753.312	1.393.985	0.330291	0.009675	0.770244
OMP+CUD	3	51203520	1.873.109	1.352.014	0.485258	0.015264	0.720982
OMP+CUD	4	51203520	2.174.682	1.661.771	0.47837	0.009598	0.621
OMP+CUD	5	51203520	1.741.610	1.366.986	0.345323	0.009806	0.77542
OMP+CUD	6	51203520	2.217.599	1.701.246	0.480707	0.009649	0.608982
OMP+CUD	1	51203520	1.825.912	1.447.354	0.349613	0.009762	0.739618
OMP+CUD	2	51203520	2.204.702	1.685.249	0.470383	0.022086	0.612545
OMP+CUD	3	51203520	1.951.449	1.571.261	0.341687	0.012135	0.692038
OMP+CUD	4	51203520	1.737.656	1.362.679	0.34594	0.009676	0.777184
OMP+CUD	5	51203520	2.434.875	1.781.648	0.618205	0.009348	0.55464
OMP+CUD	6	51203520	1.762.148	1.383.555	0.349681	0.00938	0.766381
OMP+CUD	1	51203520	2.307.757	1.737.706	0.535106	0.008977	0.585191
OMP+CUD	2	51203520	1.755.673	1.382.505	0.34392	0.009767	0.769208
OMP+CUD	3	51203520	1.731.457	1.367.979	0.334971	0.00935	0.779966
OMP+CUD	4	51203520	2.279.267	1.763.645	0.475062	0.008856	0.592506
OMP+CUD	5	51203520	1.727.749	1.356.051	0.339985	0.0096	0.78164
OMP+CUD	6	51203520	1.974.079	1.467.206	0.477208	0.009853	0.684105

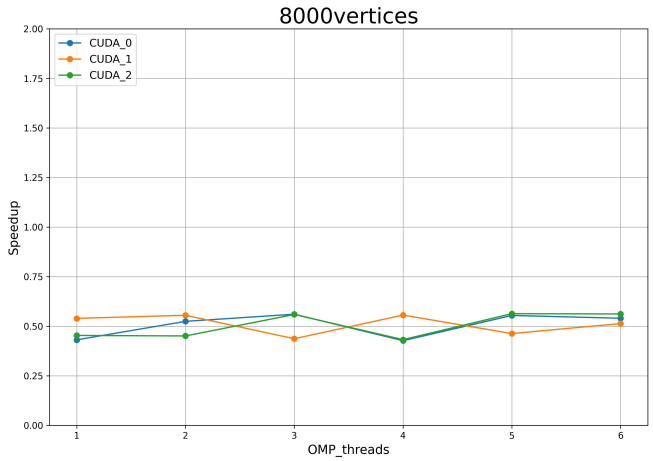


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	3.410.169	2.318.122	1.026.601	0.011541	1.000.000
OMP+CUD	1	115204096	4.266.938	3.131.091	1.070.790	0.011418	0.799208
OMP+CUD	2	115204096	3.622.000	2.806.028	0.758111	0.012455	0.941516
OMP+CUD	3	115204096	4.324.414	3.208.261	1.065.313	0.011976	0.788585
OMP+CUD	4	115204096	4.551.012	3.445.365	1.053.501	0.012502	0.749321
OMP+CUD	5	115204096	4.069.783	3.226.799	0.772822	0.012641	0.837924
OMP+CUD	6	115204096	3.819.127	2.988.567	0.764981	0.012316	0.892919
OMP+CUD	1	115204096	4.415.569	3.573.226	0.767567	0.014228	0.772306
OMP+CUD	2	115204096	3.929.187	3.117.706	0.744765	0.011541	0.867907
OMP+CUD	3	115204096	3.855.252	2.906.586	0.875341	0.012623	0.884552
OMP+CUD	4	115204096	4.027.559	2.935.058	1.039.626	0.012581	0.846709
OMP+CUD	5	115204096	4.583.797	3.488.974	1.037.532	0.012615	0.743962
OMP+CUD	6	115204096	4.196.929	3.377.902	0.750257	0.012822	0.812539
OMP+CUD	1	115204096	4.400.776	3.298.683	1.049.837	0.012622	0.774902
OMP+CUD	2	115204096	4.431.893	3.548.204	0.81983	0.011759	0.769461
OMP+CUD	3	115204096	4.069.263	3.241.668	0.760979	0.012418	0.838031
OMP+CUD	4	115204096	3.681.955	2.867.652	0.742243	0.012082	0.926184
OMP+CUD	5	115204096	4.186.544	3.047.086	1.083.498	0.012296	0.814555
OMP+CUD	6	115204096	4.418.399	3.313.610	1.052.839	0.01298	0.771811

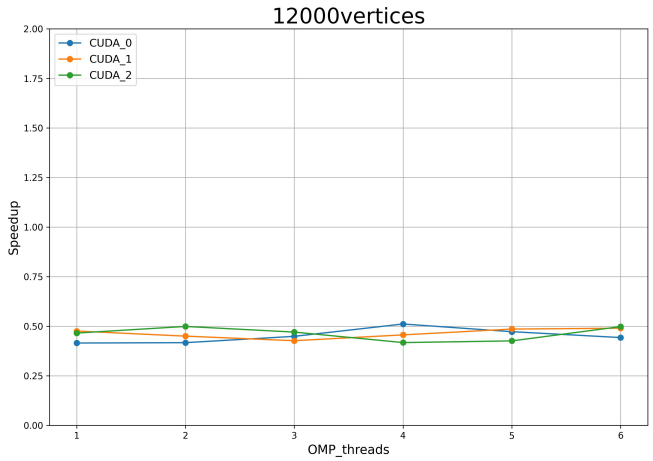
Optimization 1



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.259271	0.124995	0.12136	0.007396	1.000.000	1.000.000
OMP+CUD	1	12801792	0.594969	0.494739	0.087523	0.007321	0.435772	0
OMP+CUD	2	12801792	0.716217	0.579819	0.122287	0.006989	0.362001	0
OMP+CUD	3	12801792	0.575714	0.472021	0.091143	0.006809	0.450346	0
OMP+CUD	4	12801792	0.706825	0.57107	0.12162	0.006646	0.366811	0
OMP+CUD	5	12801792	0.687553	0.552983	0.121165	0.006502	0.377093	0
OMP+CUD	6	12801792	0.588616	0.482181	0.094471	0.006827	0.440475	0
OMP+CUD	1	12801792	0.727716	0.582433	0.131796	0.006566	0.35628	0
OMP+CUD	2	12801792	0.608231	0.484148	0.110389	0.007053	0.426271	0
OMP+CUD	3	12801792	0.693884	0.553812	0.125717	0.006889	0.373652	0
OMP+CUD	4	12801792	0.585073	0.482309	0.09082	0.006731	0.443143	0
OMP+CUD	5	12801792	0.629957	0.489416	0.128512	0.006881	0.411569	0
OMP+CUD	6	12801792	0.574302	0.474371	0.087578	0.006927	0.451454	0
OMP+CUD	1	12801792	0.717853	0.584253	0.119913	0.006721	0.361176	0
OMP+CUD	2	12801792	0.585391	0.486206	0.086848	0.00722	0.442902	0
OMP+CUD	3	12801792	0.632802	0.516676	0.100287	0.006964	0.409719	0
OMP+CUD	4	12801792	0.568274	0.47058	0.085854	0.006761	0.456243	0
OMP+CUD	5	12801792	0.705969	0.575786	0.116662	0.006787	0.367256	0
OMP+CUD	6	12801792	0.618513	0.503178	0.087699	0.021357	0.419184	0

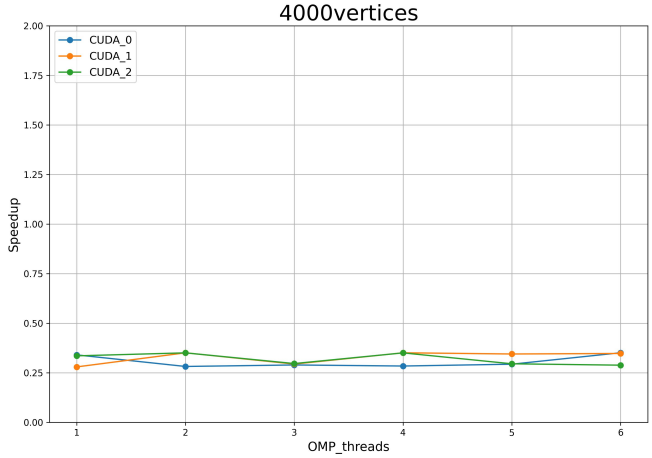


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.973455	0.538335	0.400146	0.012117	1.000.000	1.000.000
OMP+CUD	1	51203520	2.257.540	1.744.737	0.475216	0.009821	0.431202	0
OMP+CUD	2	51203520	1.855.966	1.485.183	0.333702	0.011062	0.5245	0
OMP+CUD	3	51203520	1.736.708	1.358.893	0.348542	0.009847	0.560518	0
OMP+CUD	4	51203520	2.279.346	1.771.546	0.47153	0.009368	0.427076	0
OMP+CUD	5	51203520	1.756.298	1.371.637	0.347511	0.017456	0.554265	0
OMP+CUD	6	51203520	1.802.087	1.364.604	0.40571	0.009872	0.540182	0
OMP+CUD	1	51203520	1.804.373	1.428.052	0.345001	0.010329	0.539498	0
OMP+CUD	2	51203520	1.753.163	1.368.008	0.354973	0.010113	0.555256	0
OMP+CUD	3	51203520	2.228.373	1.708.652	0.476545	0.009758	0.436846	0
OMP+CUD	4	51203520	1.750.425	1.379.255	0.334932	0.017129	0.556125	0
OMP+CUD	5	51203520	2.102.836	1.595.257	0.477002	0.010485	0.462925	0
OMP+CUD	6	51203520	1.896.074	1.517.796	0.343457	0.009248	0.513406	0
OMP+CUD	1	51203520	2.146.654	1.642.000	0.475527	0.010027	0.453475	0
OMP+CUD	2	51203520	2.158.936	1.688.147	0.434735	0.010179	0.450896	0
OMP+CUD	3	51203520	1.741.014	1.366.421	0.343733	0.010071	0.559131	0
OMP+CUD	4	51203520	2.251.857	1.700.744	0.511964	0.013921	0.43229	0
OMP+CUD	5	51203520	1.728.196	1.360.767	0.332398	0.015905	0.563278	0
OMP+CUD	6	51203520	1.732.336	1.351.256	0.347125	0.009788	0.561932	0

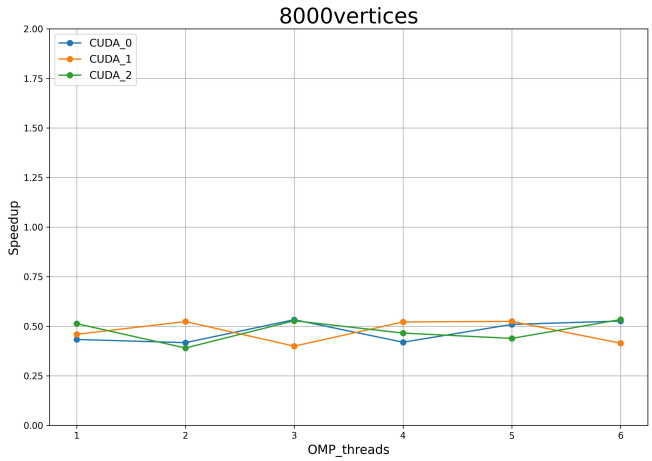


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	1.895.650	1.071.462	0.768879	0.018368	1.000.000	1.000.000
OMP+CUD	1	115204096	4.565.416	3.463.222	1.050.746	0.012221	0.41522	0
OMP+CUD	2	115204096	4.545.549	3.500.180	0.990703	0.015197	0.417034	0
OMP+CUD	3	115204096	4.222.440	3.391.309	0.76558	0.013697	0.448947	0
OMP+CUD	4	115204096	3.708.460	2.876.000	0.765373	0.011947	0.511169	0
OMP+CUD	5	115204096	4.012.907	2.915.888	1.043.889	0.012597	0.472388	0
OMP+CUD	6	115204096	4.280.926	3.183.816	1.041.399	0.012262	0.442813	0
OMP+CUD	1	115204096	3.985.488	2.991.124	0.940046	0.012836	0.475638	0
OMP+CUD	2	115204096	4.212.892	3.111.343	1.041.874	0.019046	0.449964	0
OMP+CUD	3	115204096	4.440.476	3.324.120	1.064.467	0.012291	0.426902	0
OMP+CUD	4	115204096	4.153.776	3.326.151	0.760337	0.014029	0.456368	0
OMP+CUD	5	115204096	3.904.008	3.073.562	0.761539	0.012597	0.485565	0
OMP+CUD	6	115204096	3.868.226	2.836.172	0.979855	0.01271	0.490057	0
OMP+CUD	1	115204096	4.072.242	3.243.702	0.756102	0.017946	0.465505	0
OMP+CUD	2	115204096	3.800.092	2.949.469	0.777658	0.013126	0.498843	0
OMP+CUD	3	115204096	4.030.422	2.929.540	1.048.952	0.012431	0.470335	0
OMP+CUD	4	115204096	4.540.883	3.432.824	1.055.957	0.012717	0.417463	0
OMP+CUD	5	115204096	4.450.916	3.537.988	0.83821	0.013031	0.425901	0
OMP+CUD	6	115204096	3.806.001	2.982.908	0.748009	0.018312	0.498069	0

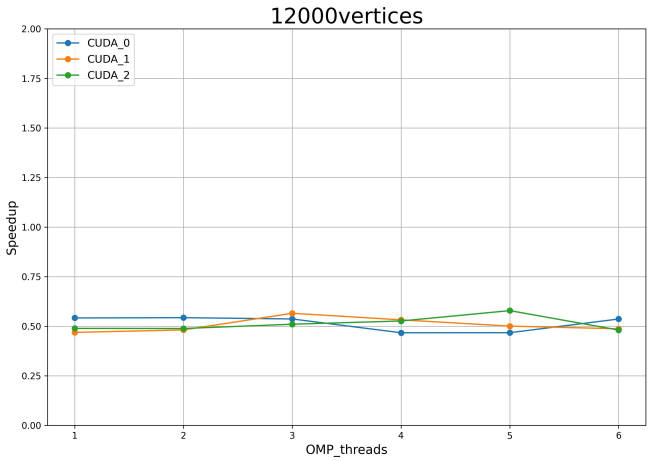
Optimization 2



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.203105	0.103072	0.087862	0.006792	1.000.000
OMP+CUD	1	12801792	0.596966	0.49764	0.087613	0.006776	0.340228
OMP+CUD	2	12801792	0.720558	0.582615	0.123338	0.007417	0.281871
OMP+CUD	3	12801792	0.701663	0.583128	0.105442	0.006492	0.289462
OMP+CUD	4	12801792	0.714963	0.571979	0.127343	0.009109	0.284077
OMP+CUD	5	12801792	0.692448	0.557181	0.121615	0.006802	0.293314
OMP+CUD	6	12801792	0.578591	0.479787	0.086072	0.007173	0.351033
OMP+CUD	1	12801792	0.726331	0.58373	0.119723	0.014674	0.279631
OMP+CUD	2	12801792	0.579071	0.47814	0.08871	0.006947	0.350742
OMP+CUD	3	12801792	0.690963	0.556225	0.120559	0.006979	0.293944
OMP+CUD	4	12801792	0.578958	0.472114	0.086494	0.014465	0.350811
OMP+CUD	5	12801792	0.589101	0.483815	0.092519	0.007415	0.34477
OMP+CUD	6	12801792	0.584612	0.47961	0.092226	0.007089	0.347418
OMP+CUD	1	12801792	0.605472	0.496708	0.096358	0.007148	0.335448
OMP+CUD	2	12801792	0.579479	0.480111	0.086452	0.007342	0.350495
OMP+CUD	3	12801792	0.684956	0.550662	0.120042	0.007494	0.296522
OMP+CUD	4	12801792	0.579678	0.481314	0.086503	0.006752	0.350375
OMP+CUD	5	12801792	0.687661	0.553035	0.120017	0.007776	0.295356
OMP+CUD	6	12801792	0.704315	0.569314	0.119223	0.008788	0.288372

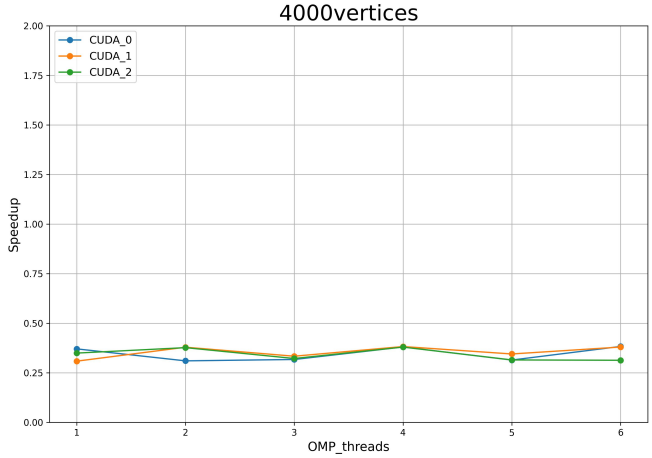


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.917509	0.481164	0.399763	0.012532	1.000.000
OMP+CUD	1	51203520	2.118.162	1.611.754	0.477905	0.009723	0.433163
OMP+CUD	2	51203520	2.199.885	1.688.641	0.475491	0.009293	0.417071
OMP+CUD	3	51203520	1.721.903	1.347.005	0.34638	0.009483	0.532846
OMP+CUD	4	51203520	2.187.881	1.657.235	0.48866	0.016675	0.41936
OMP+CUD	5	51203520	1.802.488	1.422.137	0.345471	0.00905	0.509023
OMP+CUD	6	51203520	1.743.555	1.374.273	0.338467	0.009926	0.526229
OMP+CUD	1	51203520	1.997.567	1.629.168	0.331996	0.011319	0.459313
OMP+CUD	2	51203520	1.752.426	1.382.739	0.34012	0.010126	0.523565
OMP+CUD	3	51203520	2.297.300	1.681.515	0.573845	0.016098	0.399386
OMP+CUD	4	51203520	1.759.280	1.389.806	0.340394	0.009668	0.521525
OMP+CUD	5	51203520	1.749.737	1.373.150	0.346847	0.009598	0.52437
OMP+CUD	6	51203520	2.215.188	1.700.475	0.473127	0.008688	0.41419
OMP+CUD	1	51203520	1.788.965	1.412.627	0.345453	0.01036	0.512872
OMP+CUD	2	51203520	2.352.207	1.831.173	0.475954	0.018924	0.390063
OMP+CUD	3	51203520	1.739.075	1.368.692	0.342031	0.009428	0.527584
OMP+CUD	4	51203520	1.971.906	1.453.289	0.488583	0.009668	0.46529
OMP+CUD	5	51203520	2.092.163	1.643.375	0.414249	0.009105	0.438546
OMP+CUD	6	51203520	1.719.274	1.357.006	0.332894	0.00995	0.533661

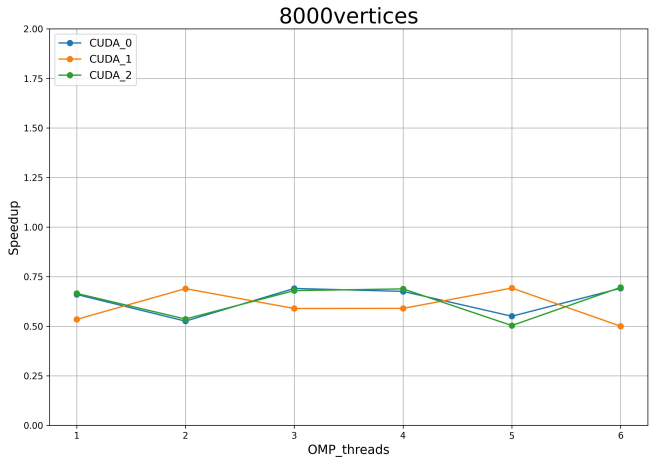


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	2.114.126	1.012.653	1.037.262	0.011339	1.000.000
OMP+CUD	1	115204096	3.902.923	3.075.079	0.763996	0.011312	0.541678
OMP+CUD	2	115204096	3.892.169	3.063.352	0.761307	0.011869	0.543174
OMP+CUD	3	115204096	3.940.142	2.899.119	0.989756	0.012311	0.536561
OMP+CUD	4	115204096	4.527.667	3.412.591	1.055.198	0.019397	0.466935
OMP+CUD	5	115204096	4.523.321	3.468.500	0.998042	0.013272	0.467384
OMP+CUD	6	115204096	3.941.644	3.099.758	0.768166	0.018763	0.536356
OMP+CUD	1	115204096	4.508.570	3.646.013	0.79333	0.012427	0.468913
OMP+CUD	2	115204096	4.391.821	3.519.827	0.802019	0.01263	0.481378
OMP+CUD	3	115204096	3.740.683	2.903.048	0.770672	0.012398	0.565171
OMP+CUD	4	115204096	3.972.935	2.869.222	1.051.206	0.012051	0.532132
OMP+CUD	5	115204096	4.223.846	3.112.354	1.052.875	0.018482	0.500522
OMP+CUD	6	115204096	4.340.638	3.497.079	0.777873	0.012352	0.487054
OMP+CUD	1	115204096	4.323.702	3.222.681	1.048.573	0.012662	0.488962
OMP+CUD	2	115204096	4.327.815	3.216.676	1.058.901	0.013237	0.488497
OMP+CUD	3	115204096	4.145.359	3.330.444	0.749783	0.011792	0.509998
OMP+CUD	4	115204096	4.016.526	3.185.040	0.758265	0.017653	0.526357
OMP+CUD	5	115204096	3.653.608	2.825.469	0.777088	0.01199	0.578641
OMP+CUD	6	115204096	4.398.820	3.260.362	1.080.875	0.018539	0.480612

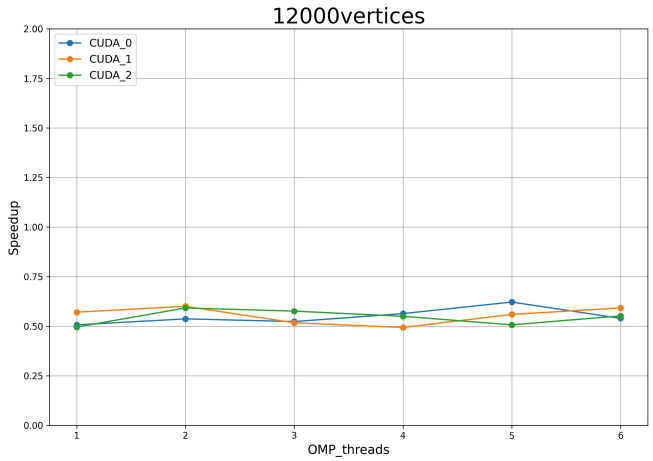
Optimization 3



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup	efficency
Sequentia	0	0	0.221516	0.120188	0.086602	0.007484	1.000.000	1.000.000
OMP+CUD	1	12801792	0.597762	0.493669	0.091873	0.006819	0.370576	0
OMP+CUD	2	12801792	0.714162	0.578453	0.122399	0.006464	0.310176	0
OMP+CUD	3	12801792	0.69864	0.56444	0.12084	0.006855	0.317067	0
OMP+CUD	4	12801792	0.582726	0.481174	0.08958	0.006775	0.380137	0
OMP+CUD	5	12801792	0.704965	0.572023	0.120335	0.006472	0.314223	0
OMP+CUD	6	12801792	0.578522	0.479849	0.086355	0.006893	0.3829	0
OMP+CUD	1	12801792	0.717694	0.579308	0.124233	0.007035	0.30865	0
OMP+CUD	2	12801792	0.584607	0.478691	0.093671	0.007229	0.378914	0
OMP+CUD	3	12801792	0.664091	0.531581	0.120283	0.00673	0.333563	0
OMP+CUD	4	12801792	0.579489	0.469874	0.091168	0.012958	0.382261	0
OMP+CUD	5	12801792	0.642286	0.53639	0.092269	0.006538	0.344887	0
OMP+CUD	6	12801792	0.583195	0.482148	0.088373	0.007007	0.379832	0
OMP+CUD	1	12801792	0.634211	0.530921	0.089439	0.006639	0.349278	0
OMP+CUD	2	12801792	0.587952	0.484232	0.091715	0.006905	0.376759	0
OMP+CUD	3	12801792	0.687881	0.554113	0.120106	0.006841	0.322027	0
OMP+CUD	4	12801792	0.583593	0.480821	0.089162	0.006659	0.379572	0
OMP+CUD	5	12801792	0.703645	0.570693	0.119593	0.006538	0.314812	0
OMP+CUD	6	12801792	0.70758	0.560966	0.132551	0.006713	0.313062	0



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	1.198.342	0.69882	0.450139	0.013208	1.000.000
OMP+CUD	1	51203520	1.815.330	1.439.609	0.343587	0.010302	0.660124
OMP+CUD	2	51203520	2.279.993	1.754.281	0.490915	0.00906	0.52559
OMP+CUD	3	51203520	1.735.444	1.374.594	0.331741	0.009829	0.690511
OMP+CUD	4	51203520	1.773.624	1.354.223	0.391122	0.009493	0.675646
OMP+CUD	5	51203520	2.177.437	1.659.687	0.47294	0.009334	0.550345
OMP+CUD	6	51203520	1.733.834	1.354.324	0.342125	0.017639	0.691152
OMP+CUD	1	51203520	2.241.862	1.741.146	0.465486	0.011026	0.53453
OMP+CUD	2	51203520	1.738.174	1.363.036	0.345332	0.009966	0.689426
OMP+CUD	3	51203520	2.033.153	1.533.440	0.470735	0.009577	0.589401
OMP+CUD	4	51203520	2.031.436	1.643.162	0.352565	0.009429	0.589899
OMP+CUD	5	51203520	1.730.870	1.368.326	0.333286	0.00949	0.692335
OMP+CUD	6	51203520	2.395.170	1.755.468	0.605124	0.009269	0.500316
OMP+CUD	1	51203520	1.801.367	1.419.825	0.352894	0.00956	0.665241
OMP+CUD	2	51203520	2.234.835	1.702.433	0.49471	0.009889	0.536211
OMP+CUD	3	51203520	1.763.289	1.387.651	0.340268	0.009719	0.679606
OMP+CUD	4	51203520	1.741.004	1.371.433	0.340479	0.009612	0.688305
OMP+CUD	5	51203520	2.380.591	1.872.799	0.469495	0.012813	0.50338
OMP+CUD	6	51203520	1.722.700	1.357.849	0.33396	0.010365	0.695619

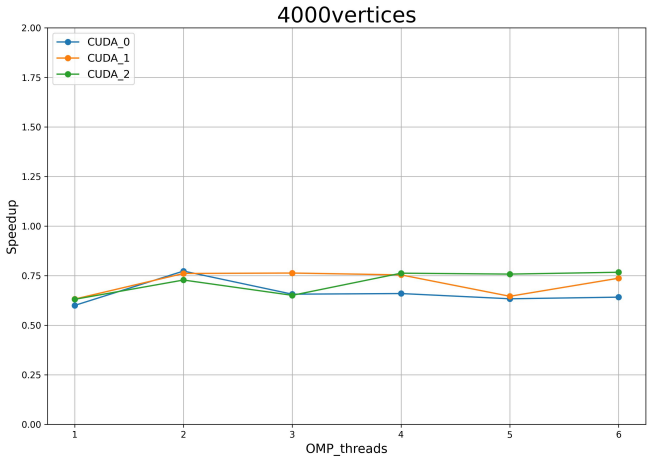


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	2.265.754	1.179.598	1.036.474	0.012089	1.000.000
OMP+CUD	1	115204096	4.470.251	3.371.703	1.047.721	0.012083	0.506852
OMP+CUD	2	115204096	4.218.766	3.139.602	1.027.581	0.012262	0.537066
OMP+CUD	3	115204096	4.327.889	3.513.429	0.748604	0.012116	0.523524
OMP+CUD	4	115204096	4.019.725	3.196.134	0.75361	0.011706	0.563659
OMP+CUD	5	115204096	3.643.866	2.847.402	0.744641	0.012363	0.6218
OMP+CUD	6	115204096	4.199.076	3.117.301	1.029.613	0.012379	0.539584
OMP+CUD	1	115204096	3.969.417	3.123.054	0.773215	0.01198	0.570803
OMP+CUD	2	115204096	3.773.768	2.856.607	0.860238	0.014337	0.600396
OMP+CUD	3	115204096	4.379.181	3.267.935	1.057.767	0.012301	0.517392
OMP+CUD	4	115204096	4.590.645	3.496.862	1.041.250	0.012303	0.493559
OMP+CUD	5	115204096	4.047.454	3.207.320	0.759394	0.017174	0.559797
OMP+CUD	6	115204096	3.826.942	2.905.894	0.852821	0.01446	0.592054
OMP+CUD	1	115204096	4.570.355	3.630.242	0.888629	0.01253	0.49575
OMP+CUD	2	115204096	3.827.751	3.001.877	0.760779	0.011731	0.591928
OMP+CUD	3	115204096	3.932.103	2.841.993	1.028.048	0.012502	0.57622
OMP+CUD	4	115204096	4.120.372	3.020.816	1.046.580	0.01266	0.549891
OMP+CUD	5	115204096	4.469.185	3.461.441	0.955442	0.012855	0.506973
OMP+CUD	6	115204096	4.103.371	3.252.913	0.785116	0.013531	0.552169

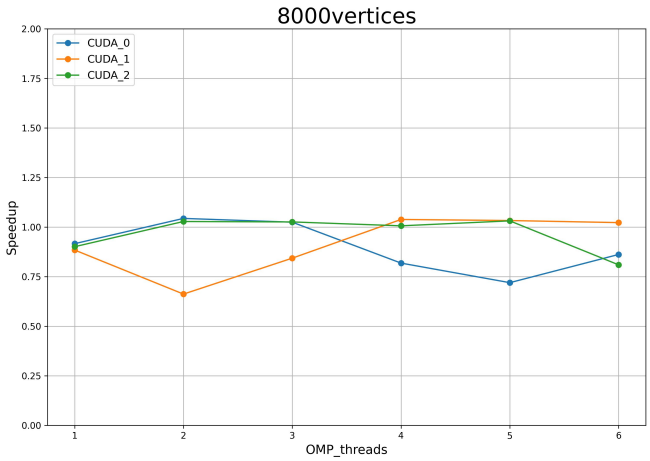
3.3.2 Sparse graph (20%)

For a sparse graph, instead, a percentage of 20% was set.

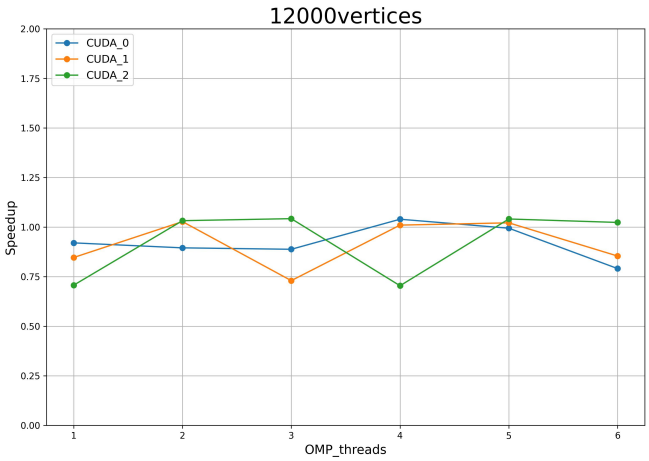
Optimization 0



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.349565	0.19443	0.137667	0.011234	1.000.000
OMP+CUD	1	3197568	0.582766	0.48649	0.081333	0.009123	0.599836
OMP+CUD	2	3197568	0.452155	0.349199	0.088386	0.009136	0.773107
OMP+CUD	3	3197568	0.532454	0.401102	0.115226	0.009395	0.656516
OMP+CUD	4	3197568	0.529856	0.39894	0.11426	0.009683	0.659734
OMP+CUD	5	3197568	0.551952	0.41699	0.118627	0.009123	0.633324
OMP+CUD	6	3197568	0.544947	0.413339	0.116028	0.009194	0.641466
OMP+CUD	1	3197568	0.553337	0.422644	0.113987	0.009307	0.631739
OMP+CUD	2	3197568	0.459444	0.362494	0.082255	0.009557	0.760842
OMP+CUD	3	3197568	0.458088	0.353414	0.089437	0.009865	0.763095
OMP+CUD	4	3197568	0.4637	0.362532	0.086069	0.00965	0.753858
OMP+CUD	5	3197568	0.541186	0.408588	0.115831	0.009459	0.645923
OMP+CUD	6	3197568	0.474268	0.358739	0.099997	0.009892	0.73706
OMP+CUD	1	3197568	0.554347	0.417945	0.119357	0.00959	0.630588
OMP+CUD	2	3197568	0.480222	0.383995	0.079832	0.00938	0.727923
OMP+CUD	3	3197568	0.537188	0.407928	0.113034	0.009944	0.65073
OMP+CUD	4	3197568	0.458493	0.361482	0.080183	0.009747	0.762421
OMP+CUD	5	3197568	0.461271	0.36088	0.085082	0.009739	0.757829
OMP+CUD	6	3197568	0.455703	0.358621	0.082348	0.009631	0.767089

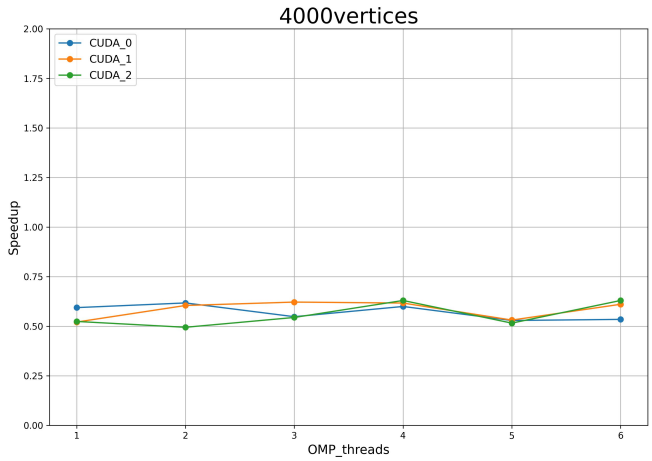


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	1.260.295	0.773891	0.45297	0.010909	1.000.000
OMP+CUD	1	12797312	1.375.296	0.989742	0.355344	0.012389	0.916381
OMP+CUD	2	12797312	1.207.548	0.851482	0.326433	0.011357	1.043.681
OMP+CUD	3	12797312	1.229.973	0.863826	0.336132	0.011882	1.024.653
OMP+CUD	4	12797312	1.540.099	1.024.115	0.479193	0.012073	0.818321
OMP+CUD	5	12797312	1.751.230	1.120.929	0.585185	0.013492	0.719663
OMP+CUD	6	12797312	1.462.344	0.979545	0.451338	0.012642	0.861832
OMP+CUD	1	12797312	1.425.563	1.056.898	0.33212	0.011381	0.884068
OMP+CUD	2	12797312	1.903.599	1.252.211	0.60725	0.011883	0.662059
OMP+CUD	3	12797312	1.494.637	0.997051	0.461813	0.012102	0.843211
OMP+CUD	4	12797312	1.213.678	0.862202	0.32029	0.012559	1.038.409
OMP+CUD	5	12797312	1.220.059	0.866456	0.322035	0.012405	1.032.979
OMP+CUD	6	12797312	1.232.535	0.879422	0.320728	0.012945	1.022.523
OMP+CUD	1	12797312	1.398.505	0.979401	0.388501	0.012363	0.901173
OMP+CUD	2	12797312	1.225.836	0.857633	0.33667	0.01343	1.028.110
OMP+CUD	3	12797312	1.228.450	0.86323	0.334508	0.012424	1.025.923
OMP+CUD	4	12797312	1.252.294	0.888458	0.323636	0.022116	1.006.389
OMP+CUD	5	12797312	1.222.027	0.864439	0.325096	0.012848	1.031.315
OMP+CUD	6	12797312	1.556.679	1.052.281	0.462007	0.011738	0.809605

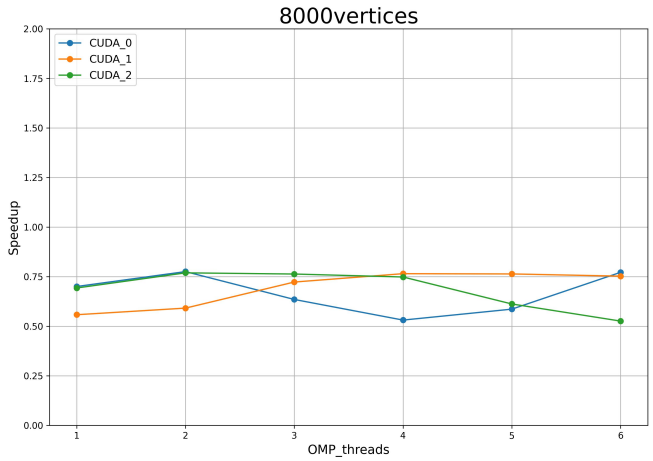


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	2.615.058	1.521.127	1.035.376	0.023518	1.000.000
OMP+CUD	1	28801152	2.841.565	2.017.339	0.770536	0.015338	0.920288
OMP+CUD	2	28801152	2.921.092	2.122.045	0.719928	0.014909	0.895233
OMP+CUD	3	28801152	2.944.901	1.737.422	1.152.344	0.015936	0.887995
OMP+CUD	4	28801152	2.515.525	1.716.919	0.739718	0.015598	1.039.567
OMP+CUD	5	28801152	2.630.092	1.806.542	0.74106	0.022189	0.994284
OMP+CUD	6	28801152	3.307.297	1.931.991	1.319.761	0.015505	0.790693
OMP+CUD	1	28801152	3.088.977	1.993.785	1.040.504	0.015377	0.846577
OMP+CUD	2	28801152	2.544.850	1.753.730	0.733999	0.015867	1.027.588
OMP+CUD	3	28801152	3.582.626	2.322.190	1.162.312	0.022741	0.729928
OMP+CUD	4	28801152	2.589.299	1.773.931	0.760274	0.016456	1.009.948
OMP+CUD	5	28801152	2.560.337	1.772.308	0.729497	0.017177	1.021.373
OMP+CUD	6	28801152	3.062.432	2.250.113	0.717013	0.025903	0.853915
OMP+CUD	1	28801152	3.698.805	2.305.187	1.321.344	0.021992	0.707001
OMP+CUD	2	28801152	2.533.345	1.722.781	0.756315	0.015609	1.032.255
OMP+CUD	3	28801152	2.508.523	1.738.565	0.716223	0.015671	1.042.469
OMP+CUD	4	28801152	3.713.743	2.326.992	1.317.254	0.015283	0.704157
OMP+CUD	5	28801152	2.512.633	1.708.244	0.75158	0.015163	1.040.764
OMP+CUD	6	28801152	2.555.262	1.760.533	0.739474	0.017198	1.023.401

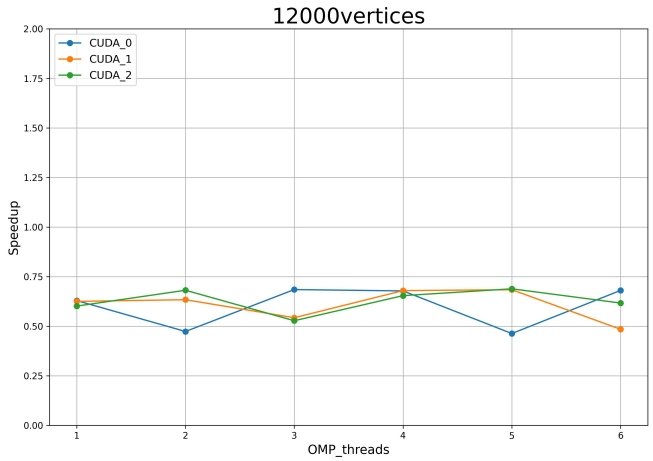
Optimization 1



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.289695	0.126235	0.147917	0.00845	1.000.000
OMP+CUD	1	3197568	0.487862	0.390443	0.08164	0.010464	0.593806
OMP+CUD	2	3197568	0.469252	0.367926	0.086577	0.00925	0.617354
OMP+CUD	3	3197568	0.528673	0.395811	0.117048	0.008882	0.547966
OMP+CUD	4	3197568	0.483159	0.361935	0.105162	0.009583	0.599585
OMP+CUD	5	3197568	0.54774	0.412717	0.118815	0.009171	0.528891
OMP+CUD	6	3197568	0.542053	0.403477	0.121908	0.009534	0.53444
OMP+CUD	1	3197568	0.556048	0.413094	0.117598	0.018406	0.520989
OMP+CUD	2	3197568	0.479169	0.360306	0.082627	0.030079	0.604578
OMP+CUD	3	3197568	0.465987	0.363159	0.082666	0.012334	0.62168
OMP+CUD	4	3197568	0.469638	0.363915	0.081198	0.018067	0.616847
OMP+CUD	5	3197568	0.545621	0.414613	0.11514	0.009747	0.530945
OMP+CUD	6	3197568	0.474428	0.361357	0.089501	0.017995	0.61062
OMP+CUD	1	3197568	0.552824	0.418671	0.118695	0.009622	0.524028
OMP+CUD	2	3197568	0.585755	0.451751	0.115405	0.011673	0.494567
OMP+CUD	3	3197568	0.532405	0.398438	0.117881	0.00922	0.544125
OMP+CUD	4	3197568	0.459987	0.362677	0.082285	0.009527	0.629789
OMP+CUD	5	3197568	0.561855	0.417034	0.12737	0.00931	0.515605
OMP+CUD	6	3197568	0.460028	0.361969	0.082957	0.009656	0.629734

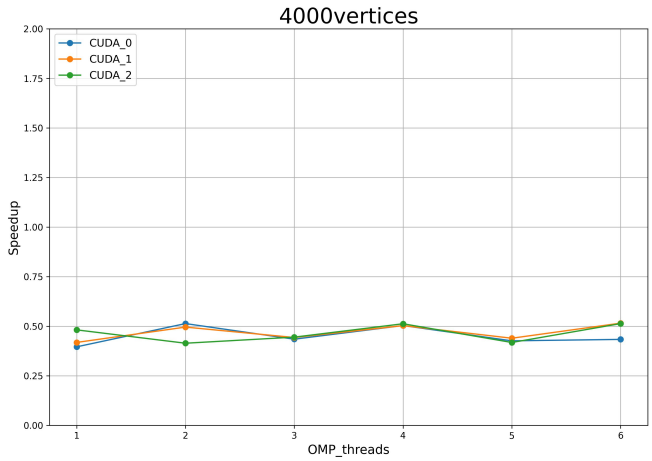


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.934616	0.460338	0.446298	0.011696	1.000.000
OMP+CUD	1	12797312	1.334.610	0.971057	0.332603	0.013064	0.700291
OMP+CUD	2	12797312	1.205.274	0.847887	0.325379	0.013157	0.775438
OMP+CUD	3	12797312	1.472.010	0.996583	0.437788	0.013005	0.634925
OMP+CUD	4	12797312	1.760.379	1.135.526	0.583041	0.011845	0.530917
OMP+CUD	5	12797312	1.594.673	1.058.401	0.499085	0.013489	0.586086
OMP+CUD	6	12797312	1.210.946	0.850728	0.32912	0.012921	0.771806
OMP+CUD	1	12797312	1.675.037	1.162.428	0.46911	0.012744	0.557967
OMP+CUD	2	12797312	1.580.666	1.014.891	0.528515	0.012323	0.59128
OMP+CUD	3	12797312	1.293.137	0.867193	0.390165	0.016377	0.722751
OMP+CUD	4	12797312	1.221.817	0.863236	0.326914	0.013172	0.764939
OMP+CUD	5	12797312	1.223.963	0.874301	0.319144	0.012676	0.763598
OMP+CUD	6	12797312	1.241.896	0.879255	0.330435	0.013581	0.752572
OMP+CUD	1	12797312	1.349.152	0.960325	0.352797	0.012364	0.692743
OMP+CUD	2	12797312	1.215.197	0.866398	0.317929	0.01275	0.769107
OMP+CUD	3	12797312	1.224.641	0.854542	0.3291	0.022507	0.763175
OMP+CUD	4	12797312	1.249.058	0.87605	0.333334	0.020844	0.748257
OMP+CUD	5	12797312	1.526.725	1.027.000	0.451774	0.021949	0.61217
OMP+CUD	6	12797312	1.777.270	1.138.435	0.594871	0.012754	0.525872

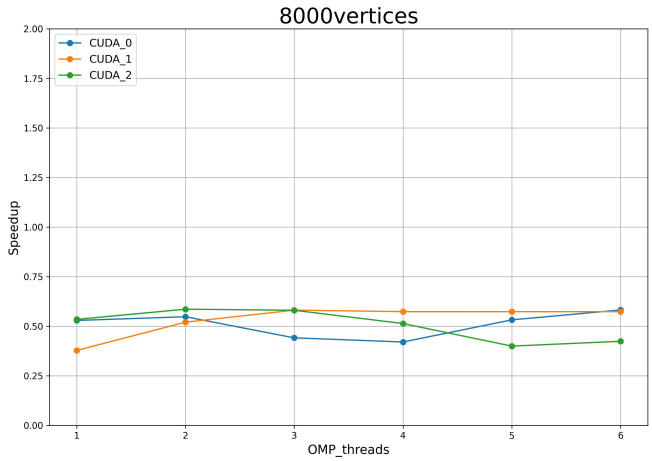


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	1.731.163	0.967383	0.71341	0.014894	1.000.000
OMP+CUD	1	28801152	2.751.013	1.961.631	0.734515	0.015014	0.629282
OMP+CUD	2	28801152	3.656.553	2.260.260	1.329.401	0.015376	0.473441
OMP+CUD	3	28801152	2.527.890	1.749.245	0.723987	0.016019	0.684825
OMP+CUD	4	28801152	2.553.288	1.749.846	0.740464	0.025211	0.678013
OMP+CUD	5	28801152	3.737.931	2.343.096	1.316.183	0.014072	0.463134
OMP+CUD	6	28801152	2.543.750	1.758.166	0.731436	0.015528	0.680556
OMP+CUD	1	28801152	2.768.690	1.978.484	0.736367	0.016285	0.625264
OMP+CUD	2	28801152	2.731.833	1.898.886	0.752239	0.016168	0.6337
OMP+CUD	3	28801152	3.190.168	1.826.178	1.310.953	0.015026	0.542656
OMP+CUD	4	28801152	2.547.441	1.736.331	0.757305	0.015582	0.679569
OMP+CUD	5	28801152	2.530.774	1.716.035	0.747868	0.016576	0.684045
OMP+CUD	6	28801152	3.573.253	2.172.053	1.348.040	0.015703	0.484478
OMP+CUD	1	28801152	2.878.219	1.952.451	0.869749	0.016911	0.60147
OMP+CUD	2	28801152	2.540.642	1.731.092	0.747468	0.024561	0.681388
OMP+CUD	3	28801152	3.280.769	2.312.371	0.883682	0.015814	0.52767
OMP+CUD	4	28801152	2.646.262	1.735.267	0.856512	0.015676	0.654192
OMP+CUD	5	28801152	2.513.825	1.722.910	0.728363	0.024373	0.688657
OMP+CUD	6	28801152	2.806.507	2.001.237	0.722511	0.015249	0.616839

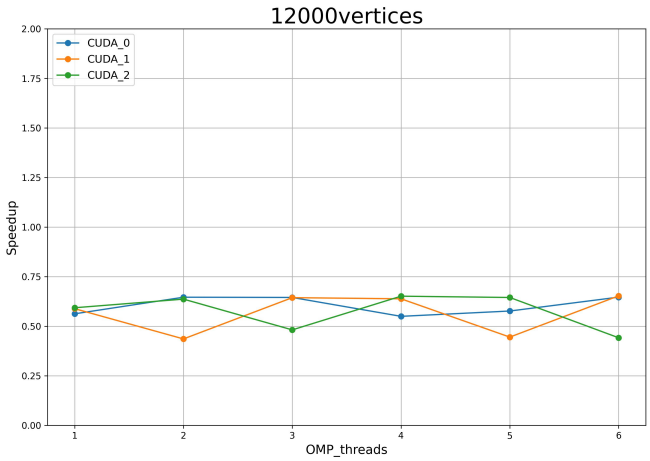
Optimization 2



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.234455	0.097958	0.119054	0.008726	1.000.000
OMP+CUD	1	3197568	0.591923	0.445117	0.12619	0.011581	0.39609
OMP+CUD	2	3197568	0.456811	0.353457	0.088715	0.009368	0.513241
OMP+CUD	3	3197568	0.5395	0.404878	0.117863	0.009094	0.434578
OMP+CUD	4	3197568	0.465807	0.364217	0.086372	0.009406	0.50333
OMP+CUD	5	3197568	0.549995	0.410637	0.121556	0.010427	0.426285
OMP+CUD	6	3197568	0.540811	0.40734	0.115961	0.011159	0.433524
OMP+CUD	1	3197568	0.560573	0.424108	0.11999	0.009766	0.418241
OMP+CUD	2	3197568	0.473135	0.372723	0.085089	0.009466	0.495535
OMP+CUD	3	3197568	0.529208	0.397532	0.116196	0.00911	0.443029
OMP+CUD	4	3197568	0.466149	0.362931	0.087759	0.009711	0.50296
OMP+CUD	5	3197568	0.533542	0.396027	0.119994	0.010332	0.439431
OMP+CUD	6	3197568	0.454931	0.359527	0.08064	0.00942	0.515363
OMP+CUD	1	3197568	0.486896	0.380695	0.09084	0.009336	0.481529
OMP+CUD	2	3197568	0.566022	0.431001	0.119015	0.008927	0.414214
OMP+CUD	3	3197568	0.527058	0.394098	0.116128	0.009101	0.444836
OMP+CUD	4	3197568	0.457612	0.361985	0.079681	0.010311	0.512343
OMP+CUD	5	3197568	0.560951	0.419521	0.124071	0.00956	0.417959
OMP+CUD	6	3197568	0.456879	0.357632	0.08445	0.009698	0.513165

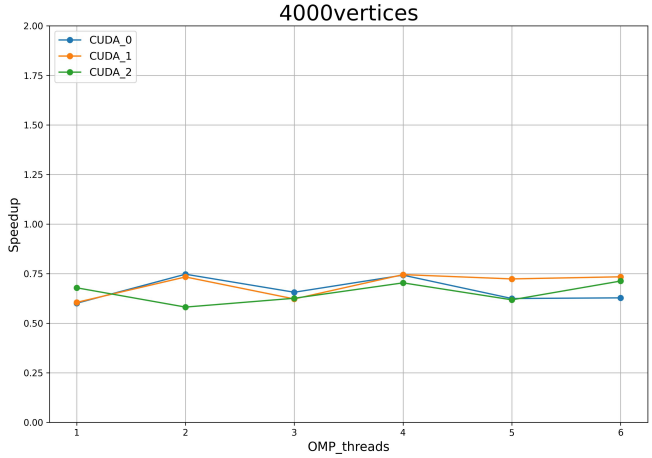


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.707774	0.361367	0.317557	0.012181	1.000.000
OMP+CUD	1	12797312	1.337.811	0.983074	0.323643	0.012808	0.529054
OMP+CUD	2	12797312	1.292.091	0.929338	0.324405	0.011637	0.547774
OMP+CUD	3	12797312	1.603.910	1.102.287	0.453585	0.011892	0.44128
OMP+CUD	4	12797312	1.683.588	1.049.971	0.596021	0.012994	0.420396
OMP+CUD	5	12797312	1.330.237	0.88263	0.412945	0.012387	0.532066
OMP+CUD	6	12797312	1.216.322	0.854376	0.331287	0.012417	0.581897
OMP+CUD	1	12797312	1.873.472	1.226.905	0.601144	0.012795	0.377787
OMP+CUD	2	12797312	1.360.264	0.879587	0.449152	0.012839	0.520321
OMP+CUD	3	12797312	1.218.227	0.865708	0.3217	0.012674	0.580987
OMP+CUD	4	12797312	1.233.958	0.869757	0.331693	0.013075	0.573581
OMP+CUD	5	12797312	1.234.506	0.860333	0.341102	0.012958	0.573326
OMP+CUD	6	12797312	1.234.010	0.860878	0.342282	0.012519	0.573556
OMP+CUD	1	12797312	1.324.197	0.958341	0.33351	0.013415	0.534493
OMP+CUD	2	12797312	1.208.071	0.851056	0.322924	0.013535	0.585871
OMP+CUD	3	12797312	1.219.534	0.865077	0.323618	0.012756	0.580364
OMP+CUD	4	12797312	1.378.578	1.003.286	0.337265	0.012432	0.513409
OMP+CUD	5	12797312	1.771.009	1.171.151	0.556579	0.012078	0.399644
OMP+CUD	6	12797312	1.669.613	1.070.359	0.562538	0.01271	0.423915

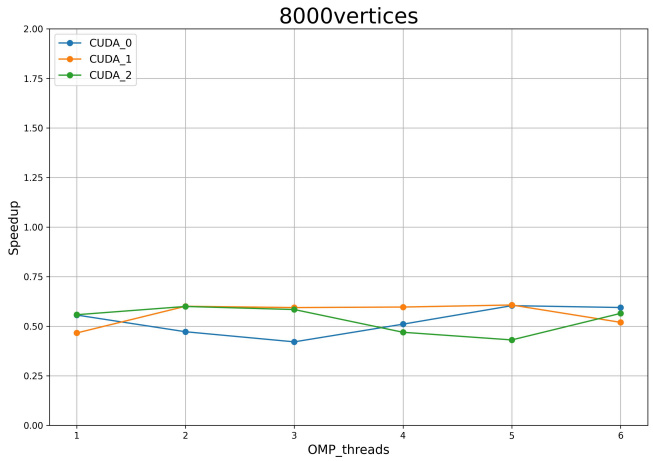


Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	1.637.320	0.853663	0.719597	0.014299	1.000.000
OMP+CUD	1	28801152	2.910.462	2.117.756	0.724247	0.014906	0.562563
OMP+CUD	2	28801152	2.533.670	1.715.291	0.763807	0.015785	0.646225
OMP+CUD	3	28801152	2.538.367	1.731.313	0.746165	0.022294	0.645029
OMP+CUD	4	28801152	2.978.009	2.132.775	0.757725	0.01552	0.549804
OMP+CUD	5	28801152	2.839.536	1.741.162	1.042.848	0.015947	0.576615
OMP+CUD	6	28801152	2.532.262	1.741.665	0.737624	0.015364	0.646584
OMP+CUD	1	28801152	2.783.079	1.994.929	0.73504	0.015295	0.588312
OMP+CUD	2	28801152	3.756.267	2.364.188	1.323.134	0.015314	0.43589
OMP+CUD	3	28801152	2.542.311	1.760.918	0.720595	0.021261	0.644028
OMP+CUD	4	28801152	2.565.505	1.761.109	0.748771	0.016203	0.638206
OMP+CUD	5	28801152	3.678.371	2.350.863	1.242.265	0.016432	0.445121
OMP+CUD	6	28801152	2.507.411	1.722.714	0.731779	0.014748	0.652992
OMP+CUD	1	28801152	2.760.635	1.979.071	0.726392	0.01603	0.593095
OMP+CUD	2	28801152	2.572.017	1.773.742	0.722287	0.015404	0.63659
OMP+CUD	3	28801152	3.400.422	2.027.841	1.319.031	0.015294	0.481505
OMP+CUD	4	28801152	2.513.438	1.726.808	0.722471	0.025746	0.651426
OMP+CUD	5	28801152	2.538.919	1.709.523	0.774971	0.015938	0.644888
OMP+CUD	6	28801152	3.706.510	2.323.495	1.323.407	0.016045	0.441742

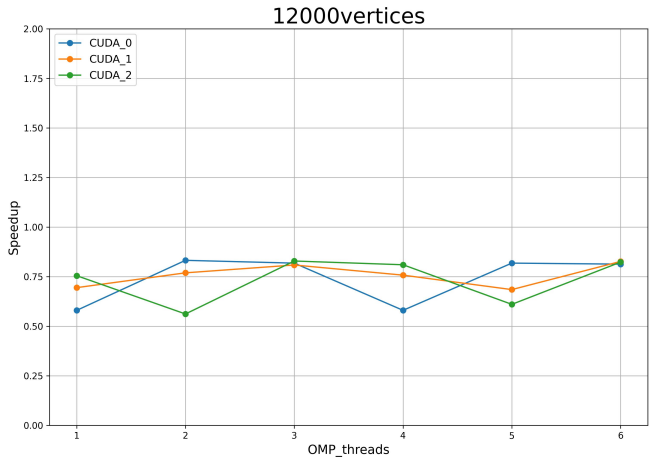
Optimization 3



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.337163	0.14694	0.159556	0.021787	1.000.000
OMP+CUD	1	3197568	0.561235	0.42865	0.117208	0.009171	0.600751
OMP+CUD	2	3197568	0.451286	0.354316	0.081365	0.010081	0.747116
OMP+CUD	3	3197568	0.513812	0.38115	0.117258	0.009417	0.656198
OMP+CUD	4	3197568	0.454163	0.358592	0.080344	0.009459	0.742382
OMP+CUD	5	3197568	0.539836	0.402716	0.120468	0.009323	0.624564
OMP+CUD	6	3197568	0.536976	0.404566	0.115459	0.009665	0.627891
OMP+CUD	1	3197568	0.557203	0.419856	0.116305	0.011244	0.605098
OMP+CUD	2	3197568	0.459758	0.359	0.08268	0.012629	0.733347
OMP+CUD	3	3197568	0.541642	0.408125	0.117	0.009206	0.622482
OMP+CUD	4	3197568	0.45251	0.355352	0.08167	0.009576	0.745093
OMP+CUD	5	3197568	0.466055	0.358068	0.091879	0.010034	0.72344
OMP+CUD	6	3197568	0.459292	0.3627	0.081466	0.009657	0.734091
OMP+CUD	1	3197568	0.496841	0.389786	0.082379	0.018904	0.678612
OMP+CUD	2	3197568	0.579975	0.431697	0.114351	0.026574	0.581339
OMP+CUD	3	3197568	0.53874	0.399241	0.116164	0.016165	0.625835
OMP+CUD	4	3197568	0.479243	0.359955	0.091467	0.020578	0.703532
OMP+CUD	5	3197568	0.545571	0.400015	0.129528	0.009033	0.617999
OMP+CUD	6	3197568	0.473074	0.355609	0.084667	0.027149	0.712706



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	0.736717	0.392917	0.316327	0.011643	1.000.000
OMP+CUD	1	12797312	1.324.301	0.970314	0.320861	0.01372	0.556306
OMP+CUD	2	12797312	1.560.499	1.047.511	0.4699	0.011654	0.472103
OMP+CUD	3	12797312	1.749.392	1.119.969	0.587658	0.012876	0.421127
OMP+CUD	4	12797312	1.443.442	0.940996	0.471745	0.012294	0.510389
OMP+CUD	5	12797312	1.220.419	0.85969	0.321501	0.021194	0.603659
OMP+CUD	6	12797312	1.239.609	0.87108	0.328704	0.021678	0.594314
OMP+CUD	1	12797312	1.581.221	1.090.870	0.452662	0.012456	0.465917
OMP+CUD	2	12797312	1.226.824	0.864806	0.331401	0.012365	0.600507
OMP+CUD	3	12797312	1.240.339	0.864334	0.344805	0.012768	0.593964
OMP+CUD	4	12797312	1.234.612	0.859246	0.343862	0.012845	0.596719
OMP+CUD	5	12797312	1.213.741	0.855939	0.327302	0.012455	0.60698
OMP+CUD	6	12797312	1.418.085	1.031.211	0.351074	0.012055	0.519515
OMP+CUD	1	12797312	1.320.427	0.961364	0.328508	0.012266	0.557938
OMP+CUD	2	12797312	1.228.646	0.869602	0.327535	0.012523	0.599617
OMP+CUD	3	12797312	1.261.202	0.897204	0.325385	0.012884	0.584139
OMP+CUD	4	12797312	1.569.491	1.065.284	0.462062	0.012016	0.469399
OMP+CUD	5	12797312	1.709.925	1.066.547	0.603131	0.015017	0.430848
OMP+CUD	6	12797312	1.304.492	0.863085	0.409789	0.012706	0.564754



Type	OMP_threads	Cuda_threads	total_time	algorithm	input_time	output_time	speedup
Sequentia	0	0	2.083.119	0.990287	1.029.127	0.013996	1.000.000
OMP+CUD	1	28801152	3.591.828	2.386.144	1.129.609	0.024511	0.579961
OMP+CUD	2	28801152	2.502.924	1.713.330	0.731132	0.015696	0.832274
OMP+CUD	3	28801152	2.546.133	1.757.264	0.735496	0.015446	0.81815
OMP+CUD	4	28801152	3.590.891	2.207.183	1.315.605	0.015484	0.580112
OMP+CUD	5	28801152	2.545.969	1.756.325	0.732234	0.016391	0.818203
OMP+CUD	6	28801152	2.561.910	1.756.125	0.751989	0.015672	0.813112
OMP+CUD	1	28801152	2.999.312	2.189.360	0.720279	0.018413	0.694532
OMP+CUD	2	28801152	2.708.133	1.719.253	0.928513	0.022193	0.769209
OMP+CUD	3	28801152	2.575.797	1.775.965	0.744121	0.016514	0.808728
OMP+CUD	4	28801152	2.749.790	1.934.194	0.731302	0.015651	0.757556
OMP+CUD	5	28801152	3.042.504	1.735.659	1.249.124	0.017159	0.684673
OMP+CUD	6	28801152	2.519.844	1.741.044	0.724321	0.016035	0.826686
OMP+CUD	1	28801152	2.759.645	1.959.719	0.745462	0.015654	0.75485
OMP+CUD	2	28801152	3.708.528	2.392.565	1.233.178	0.016827	0.56171
OMP+CUD	3	28801152	2.513.448	1.728.760	0.726432	0.018367	0.828789
OMP+CUD	4	28801152	2.571.880	1.761.980	0.755243	0.015929	0.80996
OMP+CUD	5	28801152	3.413.222	2.376.434	0.958624	0.014593	0.610309
OMP+CUD	6	28801152	2.530.481	1.746.375	0.730156	0.015525	0.823211

3.3.3 Considerations

"Let's recall that the CUDA kernel was executed in various implementations identified by:

- **Tipo 0:** Priority to shared memory with use of shared memory.
- **Tipo 1:** Priority to L1 and non-use of shared memory.
- **Tipo 2:** Priority to L1 and use of shared memory.

Clearly, for each test case, parallelization with CUDA does not provide any advantage to the program's execution. The increase in OpenMP threads does not significantly impact the algorithm's speedup (which may also be attributed to the CPU used for testing).

It is also observed that the algorithm's execution time is independent of the memory and kernel settings. This occurs for a main reasons: **data passing from CPU to GPU is extremely slow (GPU actually performs operations faster than CPU). There could be other 2 reasons (less important):** here is not an intensive use of shared mem-ory (making it almost irrelevant), and the execution principle among kernels is the same. In fact, the simultaneous attempt of multiple threads to modify a global vector significantly slows down the execution. Perhaps, implementing a different execution approach, more similar to the sequential algorithm's execution, with data division among threads and the use of texture memory, could yield better results.

In addition, as in the OpenMP+MPI case, the significant improvement achieved by the sequential version with first-level optimization becomes evident. In fact, the speedup of the CUDA version without optimization is almost equal to one.