Problem Description

The problem involves finding if a given picture contains at least three objects from a set of objects, where each object is a square matrix of integers. The matching between the object and picture is calculated based on the relative difference of overlapping members between the object and picture. The project deals with a sets of Pictures and Objects of different sizes.

The problem is parallelized using MPI+OpenMP. MPI is used to distribute the workload of searching for objects in pictures among multiple processes. The master process reads the input file, sends the input data to all processes, and then searches for objects in pictures. The slave processes receive the input data from the master process, search for objects in pictures, and send the results back to the master process. OpenMP is used within each process to parallelize the search for objects within a single picture.

Parallelization

**MPI**

MPI is used to distribute the workload of searching for objects in pictures among multiple processes. The master process (rank 0) reads the input file and sends the input data to all processes. The picture array is divided among all processes, and each process is responsible for searching for objects in its assigned portion of the picture array. The master process searches for objects in the remaining portion of the picture array. Once the search is completed, the results are received by the master process.

OpenMP

OpenMP is used within each process to parallelize the search for objects within a single picture. The function **mainSearchFunc()** initializes an array of integers to keep track of the objects found in the picture. It then sets the number of threads to 4 using OpenMP. Within the parallel region, it uses a nested loop to search for objects in the picture. If an object is found, it stores the object ID, row, and column position of the object in the result array.

The **isObjInPosition()** function is called to compare the pixel values of the object with the pixels in the picture at the current position. If the difference between the pixel values is less than or equal to the matching threshold, the function returns 1 indicating that the object has been found.

Rational for Choosing MPI+OpenMP

The use of MPI+OpenMP allows the solution to scale efficiently on distributed-memory systems with multiple cores per node. The workload is evenly distributed among all available cores, and each core uses OpenMP to further parallelize the search for objects within a single picture.

Complexity Evaluation

The master process has to read the input file and send the data to all processes, which takes O(MN^2) time. Then, it searches for objects in pictures starting from index perProc to the end of the pictures array, which takes O((M-perProc)N^2) time.

Each slave process receives the input data from the master process, searches for objects in pictures starting from index 0 to sizePerProc, and sends the results back to the master process. This takes O((M/perProc)N^2) time per process.

Within the mainSearchFunc function, a nested loop is used to search for objects in the picture, which takes O(N^2) time. However, the loop is parallelized using OpenMP, and the number of threads is set to 4, which means that the loop is executed by 4 threads concurrently. Therefore, the total time taken by the loop is reduced by a factor of 4. However, there is some overhead associated with creating and synchronizing threads, which depends on the size of the input data and the number of threads. The time complexity of parallelized loop can be expressed as O((N^2)/P), where P is the number of threads used.

The isObjInPosition() function compares the pixel values of the object with the pixels in the picture at the current position, which takes O(N^2) time. However, the function is called within the parallelized loop, and the number of times it is executed depends on the number of objects in the object set and the number of matches found for each object. The time complexity of this function can be expressed as O(MNK), where K is the maximum number of matches found for any object.

Therefore, the total time complexity of the parallelized solution can be expressed as:

O(MN^2 + (M-perProc)N^2 + (M/perProc)N^2 + (NK)/P)