Parallel and Distributed Computing

2022/2023

OpenMP Implementation

Group 45

João Gonçalves 85211 Luis Maçorano 96265 Miguel Albuquerque 105828

Approach used for parallelization:

The parallelization focus was solely on the *procedure* function as parallelization for other parts would have a low impact. Since the *weight matrix* and the *min1* and *min2* calculations amount to an insignificant impact on the computational times there was no point in parallelizing them.

Our first approach was to have the root (rank 0) calculate all the neighbours of node 0 and then divide them between itself and the other ranks. This proved to be very inefficient since while the root was doing all the work all the other ranks were stall and then there would be an unnecessary overhead as the root would do the distribution of the nodes. We decided to change our approach and all the nodes will compute the neighbours of zero (instead of being stall and waiting for the node zero) and then push to their queue the nodes assigned to them using the same logic from the previous approach. This way we completely eliminate the communication overhead.

After the nodes were divided equally between processors, when the synchronization was reached, we inserted nodes from the queue of the processor with the lowest bound on the queues of the other ones. This way all the processors will be working on the optimal branches of the tree.

Decomposition used:

We start by discovering which processor had the node with the lowest bound. Then, this processor will be responsible for inserting a node from its queue on each of the other processors queues. This way all the processors will be performing computations on the region of the tree where the lowest bound was reached, and will allow our implementation to reach a good solution sooner and avoid performing unnecessary computations, eliminating branches with bigger lower bounds.

Synchronization concerns:

Our root will be responsible for finding which processor has the lowest bound. When the processors reach a certain number of iterations they will send their lowest bound and best tour cost to the root. The root will then update all nodes with the lowest best tour cost and find which processor has the lowest bound. It will then send this information to all processors and the one with the lowest bound will send its nodes to the others.

When a processor leaves the cycle the synchronization will stop. Instead of sending its lowest bound it will send a flag to inform the root it has ended its cycle. When the next synchronization happens the root will inform the other processors the synchronization has stopped. If the root is the first one to stop it will immediately send a flag to the other processors.

We introduced a barrier for the final synchronization. When a node reaches the end of the cycle it will wait for all the other nodes to finish before sending its information to rank 0 (master rank). Rank 0 will then find which processor has the best tour cost and inform it to the other processors so that only the best one will write the final result.

We reached the conclusion that performing this synchronization multiple times during our computation was worth it until 4 processors. Above that number of processors, due to a computer only having 4, the objective is to do fewer number of synchronizations to avoid communication overhead.

Load balancing:

As stated before, to obtain a good balance between the processors we start by dividing them equally. Later, for each synchronization we would insert the optimal nodes in all threads to guarantee that all of them are focused on the right area of the tree and we perform this synchronization multiple times to ensure we reach the end of the queues as fast as we can.

To improve our load balancing, when a thread finished its queue, instead of stopping the synchronization, we could assign it nodes from the queues that are still active to avoid it being stall. This didn’t have a major impact on our results, since when a thread finished its queue we were already close to finding the optimal solution.

Another aspect that helped us balance the work between threads was the shared flag to ensure that all threads started the synchronization at the same time, this way avoiding threads being stall waiting for the other to reach the synchronization stage.

What are the performance results:

The results obtained in the lab computers are presented in table 1.

	Serial	Serial updated	OMP (2 thread)	OMP (4 threads)	MPI (2 threads)	MPI (4 threads)
Gen24	68.4	24.9	16.7	8.8	-	41.5
Gen26	55.4	21.7	9.3	4.1	15.7	8.8
Gen30	199.3	59.3	33.1	18.5	39.6	33.2

Table 1 – Performance results in seconds