In conclusion, this work presented multiple parallelization strategies for the quicksort algorithm and demonstrated that a significant difference exists, not only among the different parallel algorithm, but also due to the specific memory paradigm used for the concrete implementation.

The algorithms implemented in distributed memory using MPI showed a clear advantage in terms of scalability and performance when compared to the shared memory implementations. However, the results obtained for the OpenMP implementations showed that valid alternatives also exit for shared memory parallelization, with the **Task Quicksort** being the most promising among the different shared memory implementations. This can be of great importance in solving problems that might demand parallel sorting algorithms in a shared memory environment.

Additionally, as the results highlighted, the **PSRS** algorithm is the most scalable and efficient among all the implementations, both in terms of strong and weak scalability. This confirms the importance of load balancing and demonstrates that prioritizing the choice of pivots in this kind of algorithms greatly impacts its overall performance. Moreover, this algorithm also minimized the communication required between multiple processes which is another reason for its superior performance.

For what regards possible developments of this work, an improvements could be to find a way to further reduce the serial portion of work done in the recursive calls of the **Simple** and **Hyperquicksort** shared memeory algorithms, in order to make them more competitive with the other implementations and increasing their scalability. In fact, it might be interesting to check if solving that problem could yield better results with lower number of threads mirroring the results obtained in the distributed memory versions.