# TSP Brute CUDA Algorithm

Charles Wade

[Charles.wade@rockets.utoledo.edu](mailto:Charles.wade@rockets.utoledo.edu)

This algorithm processes the exact solution to Traveling Salesman Problem using a CUDA GPU. It is comprised of two major steps:

1. Generate Permutations/ Calculate Distances.

2. Perform a global reduction to find global min or max.

## Generate Permutations/ Calculate Distances

For permutation generation to work each thread needs to calculate a unique permutation given only a thread ID. This allows us to maximize the occupancy of the CUDA cores on a GPU by having each thread only compute the distances for a single permutation. The nature of CUDA allows us to queue up threads so the number of permutations can greatly exceed the max threads in flight allowed on a specific GPU. This means each thread needs to compute its permutation’s distance without the ability to synchronize with any others. The best way to do this is calculate the nth permutation directly using the factorial number system.

1. A reference row is generated that includes every island index except the start. An empty output vector is created to hold our permutation as it’s created.
2. The threadID is converted from the decimal number system into its [factoradic representation](https://en.wikipedia.org/wiki/Factorial_number_system).
3. The thread pops the first digit of the factoradic and uses it as an index to pick a value to remove from the refence row.
4. This value is appended to the output vector.
5. The thread repeats steps 3-4 until there are no digits left in the factoradic.
6. The resulting output vector contains the nth permutation.

After the nth permutations is created, the thread iterates over it to calculate the total Euclidian distance. This value is stored in a global memory array with its threadID acting as its index. All memory allocated by a thread is freed at its destruction.

## Finding Global Min/Max

The host waits for all threads to synchronize before performing a parallel reduction. This implementation uses the built-in Thrust library to simplify this. Finding the global min (or max if flag is set) can be found quickly this way. Once the minimum (or max) value is found, its permutation ID is returned. With this we then use the host to re-calculate that specific permutation and return that as the result. We chose to recalculate as opposed to fetch it from memory because the permutations are not saved when computed by the GPU threads. This greatly reduces the overall memory usage of the algorithm.

## Memory Safety

The core algorithm can run multiple times if the GPU does not have enough memory to complete everything at once. This means that multiple parallel reductions are computed to find the output and the entire processes may be repeated several times. We decide how many permutations to calculate per run by fetching the device’s available memory and dividing it by the amount required to store a distance and the amount a single thread can use. If this number is greater than the total number of permutations than we must run it multiple times.