Parallel Quicksort Implementation

In the program, initially, the declaration of the vector A in which all the elements are stored is given. The size of the vector is 64*1024*1024= 67108864. The function to initialize the matrix is called in the main function. The matrix is initialized using random values as the initialization type, which is given as (init=rand()). The sequence of elements are sorted using qsort_parallel (0, n-1) function, after the matrix is initialised.
This function is subjected to an OpenMP directive: #pragma omp single.
This directive specifies that the enclosed code is executed by only one thread.
The quicksort function has 2 steps: divide and conquer. In order to obtain the parallel version of the quicksort, divide and conquer steps are subjected to parallelization using OpenMP directives.
In qsort_parallel (int l, int r) function, the pivot and low values are initialized. If the values in the last index of the array are equal to or lower than the pivot value, then the values at low and high indexes are swapped. If the values are less than 1000, then the sub-sequences are sorted using qsort(l, low-1) and qsort(low+1, r).
In these functions, if the value in the high index is less than or equal to the pivot value then the values in the low and high indexes are swapped. However, if the values are more than 1000, then qsort_parallel (l, low-1) and q_sort (low+1, r) are called.
These functions are subjected to the OpenMP directive: #pragma omp task.
This directive defines an explicit task, which may be executed by the encouraging threads. Task execution is subject to tasks scheduling.
After this, the directive #pragma omp taskwait is declared.
The taskwait construct is used to specify a wait on the completion of the child tasks generated since the current task has begun.

Execution Times:

Sequential quicksort implementation
Compile: gcc -o qs qsort_seq.c
Run using: /usr/bin/time ./qs
User: 31.42
System: 0.23
Elapsed Time: 0:31.66

Parallel quicksort implementation:
1. 1 CPU/thread
Compile: gcc -fopenmp -o qp1 qsort_par1.c
Run using: /usr/bin/time ./qp1
User: 25.06
System: 0.10
Elapsed Time: 11.98
CPU: 256%

2. 8 CPUs
Compile: gcc -fopenmp -o qp1 qsort_par.c
Run using: /usr/bin/time ./qp
User: 26.01
System: 0.22

Elapsed Time: 0:08.72
CPU: 300%

The execution time of all versions is calculated. The execution time for the sequential version for a vector size of 64*1024*1024 is observed to be 31.66 seconds. When compared to the parallel version that is run on 8CPUs, it is more. The execution time on parallel version for the same vector is 8.72 seconds. The execution time on parallel version for one thread is 11.98 seconds. This is more than both the sequential and parallel version on 8 CPUs.
Hence, it can be concluded that the execution time of parallel version on 8 CPUs is 3 times faster than the sequential version.