```
import random
import sys
from contextlib import contextmanager
from multiprocessing import Manager, Pool
from timeit import default_timer as time


class Timer:
    """
    Record timing information.
    """

    def __init__(self, *steps):
        self._time_per_step = dict.fromkeys(steps)

    def __getitem__(self, item):
        return self.time_per_step[item]

    @property
    def time_per_step(self):
        return {
            step: elapsed_time
            for step, elapsed_time in self._time_per_step.items()
            if elapsed_time is not None and elapsed_time > 0
        }

    def start_for(self, step):
        self._time_per_step[step] = -time()

    def stop_for(self, step):
        self._time_per_step[step] += time()


def merge_sort_multiple(results, array):
    """Async parallel merge sort."""
    results.append(merge_sort(array))


def merge_multiple(results, array_part_left, array_part_right):
    """Merge two sorted lists in parallel."""
    results.append(merge(array_part_left, array_part_right))


def merge_sort(array):
    """Perform merge sort."""
    array_length = len(array)

    if array_length <= 1:
        return array

    middle_index = array_length // 2
    left = array[:middle_index]
    right = array[middle_index:]
    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)

```

```
58
59  def merge(left, right):
60      """Merge two sorted lists."""
61      sorted_list = [0] * (len(left) + len(right))
62      i = j = k = 0
63
64      while i < len(left) and j < len(right):
65          if left[i] < right[j]:
66              sorted_list[k] = left[i]
67              i += 1
68          else:
69              sorted_list[k] = right[j]
70              j += 1
71          k += 1
72
73      while i < len(left):
74          sorted_list[k] = left[i]
75          i += 1
76          k += 1
77
78      while j < len(right):
79          sorted_list[k] = right[j]
80          j += 1
81          k += 1
82
83      return sorted_list
84
85
86  @contextmanager
87  def process_pool(size):
88      """Create a process pool and block until all processes have completed."""
89      pool = Pool(size)
90      yield pool
91      pool.close()
92      pool.join()
93
94
95  def parallel_merge_sort(array, ps_count):
96      """Perform parallel merge sort."""
97      timer = Timer("sort", "merge", "total")
98      timer.start_for("total")
99      timer.start_for("sort")
100
101     # Divide the list in chunks
102     step = int(length / ps_count)
103
104     # Instantiate a multiprocessing.Manager object to store the output of each process.
105     manager = Manager()
106     results = manager.list()
107
108     with process_pool(size=ps_count) as pool:
109         for i in range(ps_count):
110             # We create a new Process object and assign the
111             # merge_sort_multiple function to it,
112             # using as input a sublist
113             if i < ps_count - 1:
114                 chunk = array[i * step : (i + 1) * step]
```

```
            else:
                # Get the remaining elements in the list
                chunk = array[i * step :]
            pool.apply_async(merge_sort_multiple, (results, chunk))

    timer.stop_for("sort")

    print("Performing final merge.")

    timer.start_for("merge")

    # For a core count greater than 2, we can use multiprocessing
    # again to merge sub-lists in parallel.
    while len(results) > 1:
        with process_pool(size=ps_count) as pool:
            pool.apply_async(merge_multiple, (results, results.pop(0), results.pop(0)))

    timer.stop_for("merge")
    timer.stop_for("total")

    final_sorted_list = results[0]

    return timer, final_sorted_list


def get_command_line_parameters():
    """Get the process count from command line parameters."""
    if len(sys.argv) > 1:
        # Check if the desired number of concurrent processes
        # has been given as a command-line parameter.
        total_processes = int(sys.argv[1])
        if total_processes > 1:
            # Restrict process count to even numbers
            if total_processes % 2 != 0:
                print("Process count should be an even number.")
                sys.exit(1)
        print(f"Using {total_processes} cores")
    else:
        total_processes = 1

    return {"process_count": total_processes}


if __name__ == "__main__":
    parameters = get_command_line_parameters()

    process_count = parameters["process_count"]

    main_timer = Timer("single_core", "list_generation")
    main_timer.start_for("list_generation")

    # Randomize the length of our list
    length = random.randint(3 * 10**6, 4 * 10**6)

    # Create an unsorted list with random numbers
    randomized_array = [random.randint(0, i * 100) for i in range(length)]
    main_timer.stop_for("list_generation")
```

```
172
173     print(f"List length: {length}")
174     print(f"Random list generated in {main_timer['list_generation']:.6f}")
175
176     # Start timing the single-core procedure
177     main_timer.start_for("single_core")
178     single = merge_sort(randomized_array)
179     main_timer.stop_for("single_core")
180
181     # Create a copy first due to mutation
182     randomized_array_sorted = randomized_array[:]
183     randomized_array_sorted.sort()
184
185     # Comparison with Python list sort method
186     # serves also as validation of our implementation.
187     print("Verification of sorting algorithm:", randomized_array_sorted == single)
188     print(f"Single Core elapsed time: {main_timer['single_core']:.6f} sec")
189
190     print("Starting parallel sort.")
191
192     parallel_timer, parallel_sorted_list = parallel_merge_sort(
193         randomized_array, process_count
194     )
195
196     print(f"Final merge duration: {parallel_timer['merge']:.6f} sec")
197     print("Sorted arrays equal:", parallel_sorted_list == randomized_array_sorted)
198     print(f"{process_count}-Core elapsed time: {parallel_timer['total']:.6f} sec")
199
200 """
201 OUTPUT:
202
203 Using 16 cores
204 List length: 3252321
205 Random list generated in 1.787607
206 Verification of sorting algorithm: True
207 Single Core elapsed time: 14.204555 sec
208 Starting parallel sort.
209 Performing final merge.
210 Final merge duration: 4.764258 sec
211 Sorted arrays equal: True
212 16-Core elapsed time: 6.984671 sec
213 """
214
```