```python
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-

import argparse
import random
from contextlib import contextmanager
from multiprocessing import Pool, cpu_count
from timeit import default_timer as time

from tabulate import tabulate

CPU_COUNT = cpu_count()


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

    if array_length <= 1:
        return array

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


def merge(*arrays):
    """Merge two sorted lists."""

    # Support explicit left/right args, as well as a two-item
    # tuple which works more cleanly with multiprocessing.
    left, right = arrays[0] if len(arrays) == 1 else arrays
```

```
    sorted_list = [0] * (len(left) + len(right))
    i = j = k = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_list[k] = left[i]
            i += 1
        else:
            sorted_list[k] = right[j]
            j += 1
        k += 1

    while i < len(left):
        sorted_list[k] = left[i]
        i += 1
        k += 1

    while j < len(right):
        sorted_list[k] = right[j]
        j += 1
        k += 1

    return sorted_list


@contextmanager
def process_pool(size):
    """Create a process pool and block until all processes have completed."""
    pool = Pool(size)
    yield pool
    pool.close()
    pool.join()


def parallel_merge_sort(array, ps_count):
    """Perform parallel merge sort."""
    timer = Timer("sort", "merge", "total")
    timer.start_for("total")
    timer.start_for("sort")

    # Divide the list in chunks
    step = int(len(array) / ps_count)

    # Creates a pool of worker processes, one per CPU core.
    # We then split the initial data into partitions, sized equally per
    # worker, and perform a regular merge sort across each partition.
    with process_pool(size=ps_count) as pool:
        array = [array[i * step : (i + 1) * step] for i in range(ps_count)] + [array[ps_count * step
:]]
        array = pool.map(merge_sort, array)

        timer.stop_for("sort")
        timer.start_for("merge")

        # We can use multiprocessing again to merge sub-lists in parallel.
        while len(array) > 1:
            # If the number of partitions remaining is odd, we pop off the
            # last one and append it back after one iteration of this loop,
            # since we're only interested in pairs of partitions to merge.
            extra = array.pop() if len(array) % 2 == 1 else None
```

```
119             array = [(array[i], array[i + 1]) for i in range(0, len(array), 2)]
120             array = pool.map(merge, array) + ([extra] if extra else [])
121 
122         timer.stop_for("merge")
123         timer.stop_for("total")
124 
125     final_sorted_list = array[0]
126 
127     return timer, final_sorted_list
128 
129 
130 def get_command_line_parameters():
131     """Get the process count, array length from command line parameters."""
132 
133     parser = argparse.ArgumentParser(
134         description="""Implement merge sort and multithreaded merge sort.
135         Compare the time required by both algorithms.
136         Also, analyze the performance of each algorithm for the best case and the worst case."""
137     )
138     parser.add_argument(
139         "-j",
140         "--jobs",
141         help="Number of processes to launch",
142         required=False,
143         default=CPU_COUNT,
144         type=lambda x: int(x)
145         if 0 < int(x) ⩽ CPU_COUNT
146         else parser.error(f"Number of processes must be between 1 and {CPU_COUNT}"),
147     )
148     parser.add_argument(
149         "-l",
150         "--length",
151         help="Length of the array to sort",
152         required=False,
153         default=random.randint(3 * 10**6, 4 * 10**6),  # Randomize the length of our list
154         type=lambda x: int(x) if 0 < int(x) else parser.error("Length of the array must be greater than
    0"),
155     )
156     parser.add_argument(
157         "-a",
158         "--all",
159         help="Test all the variable length",
160         required=False,
161         default=False,
162         action="store_true",
163     )
164     return parser.parse_args()
165 
166 
167 def main(jobs, length, conclusion):
168     """Main function."""
169 
170     main_timer = Timer("single_core", "list_generation")
171     main_timer.start_for("list_generation")
172 
173     # Create an unsorted list with random numbers
174     randomized_array = [random.randint(0, i * 100) for i in range(length)]
175     main_timer.stop_for("list_generation")
176 
177     print(f"List length: {length}")
```

```
    print(f"Random list generated in {main_timer['list_generation']:.6f}s\n")

    # Create a copy first due to mutation
    randomized_array_sorted = randomized_array[:]
    randomized_array_sorted.sort()

    # Start timing the single-core procedure
    print("Starting simple sort.")
    main_timer.start_for("single_core")
    sorted_array = merge_sort(randomized_array)
    main_timer.stop_for("single_core")

    # Comparison with Python list sort method
    # serves also as validation of our implementation.
    assert sorted_array == randomized_array_sorted, "The sorted array is not correct."
    print(f"Single Core elapsed time: {main_timer['single_core']:.6f}s\n")

    print("Starting parallel sort.")
    parallel_timer, parallel_sorted_array = parallel_merge_sort(randomized_array, jobs)
    print(f"Final merge duration: {parallel_timer['merge']:.6f}s")

    assert parallel_sorted_array == randomized_array_sorted, "The sorted array is not correct."
    print(f"{jobs}-Core elapsed time: {parallel_timer['total']:.6f}s\n" + "-" * 40, "\n")

    conclusion.append([length, main_timer["single_core"], parallel_timer["total"]])


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

    jobs = parameters.jobs
    length = parameters.length
    all_cases = parameters.all
    conclusion = []
    print(f"Using {jobs} cores\n")

    if all_cases:
        l = 1
        while l < 10**8:
            main(jobs, l, conclusion)
            l *= 10
    else:
        main(jobs, length, conclusion)

    print(tabulate(conclusion, headers=["Array Length", "Single-Threaded", "Multi-Threaded"],
tablefmt="outline"))

"""
OUTPUT:> python3 merge_sort.py -j 16

Using 16 cores

List length: 3332517
Random list generated in 1.826744s

Starting simple sort.
Single Core elapsed time: 14.580756s

Starting parallel sort.
Final merge duration: 2.910911s
```

```
16-Core elapsed time: 5.131196s
---------------------------------------

+----------------+------------------+------------------+
|   Array Length |   Single-Threaded |   Multi-Threaded |
+================+==================+==================+
|        3332517 |           14.5808 |           5.1312 |
+----------------+------------------+------------------+
"""

"""
OUTPUT:> python3 merge_sort.py -a

Using 16 cores

List length: 1
Random list generated in 0.000005s

Starting simple sort.
Single Core elapsed time: 0.000003s

Starting parallel sort.
Final merge duration: 0.003702s
16-Core elapsed time: 0.028538s
---------------------------------------

List length: 10
Random list generated in 0.000035s

Starting simple sort.
Single Core elapsed time: 0.000023s

Starting parallel sort.
Final merge duration: 0.002112s
16-Core elapsed time: 0.024624s
---------------------------------------

List length: 100
Random list generated in 0.000125s

Starting simple sort.
Single Core elapsed time: 0.000198s

Starting parallel sort.
Final merge duration: 0.001935s
16-Core elapsed time: 0.024828s
---------------------------------------

List length: 1000
Random list generated in 0.000616s

Starting simple sort.
Single Core elapsed time: 0.002207s

Starting parallel sort.
Final merge duration: 0.004501s
16-Core elapsed time: 0.042195s
---------------------------------------

List length: 10000
```

```
Random list generated in 0.006530s

Starting simple sort.
Single Core elapsed time: 0.024342s

Starting parallel sort.
Final merge duration: 0.009236s
16-Core elapsed time: 0.035944s
----------------------------------------

List length: 100000
Random list generated in 0.061283s

Starting simple sort.
Single Core elapsed time: 0.297823s

Starting parallel sort.
Final merge duration: 0.081855s
16-Core elapsed time: 0.166326s
----------------------------------------

List length: 1000000
Random list generated in 0.612833s

Starting simple sort.
Single Core elapsed time: 3.685065s

Starting parallel sort.
Final merge duration: 0.645426s
16-Core elapsed time: 1.235625s
----------------------------------------

List length: 10000000
Random list generated in 5.445338s

Starting simple sort.
Single Core elapsed time: 45.568661s

Starting parallel sort.
Final merge duration: 5.418382s
16-Core elapsed time: 11.996948s
----------------------------------------

+----------------+-------------------+------------------+
|   Array Length |   Single-Threaded |   Multi-Threaded |
+================+===================+==================+
|              1 |       2.934e-06   |        0.0285377 |
|             10 |       2.2908e-05  |        0.0246235 |
|            100 |       0.000198492 |        0.0248281 |
|           1000 |       0.00220744  |        0.0421945 |
|          10000 |       0.0243417   |        0.0359442 |
|         100000 |       0.297823    |        0.166326  |
|        1000000 |       3.68507     |        1.23563   |
|       10000000 |      45.5687      |       11.9969    |
+----------------+-------------------+------------------+
"""

```