PERFORMANCE REPORT: CONCURRENT VS. REGULAR MERGE SORT

Testing has been done using input of 2734 numbers.
Test Machine – AMD A8 6410 (4 cores, 2 Ghz), 3.4 GB usable RAM

Concurrent Merge Sort using Shared Memory vs Regular Merge Sort

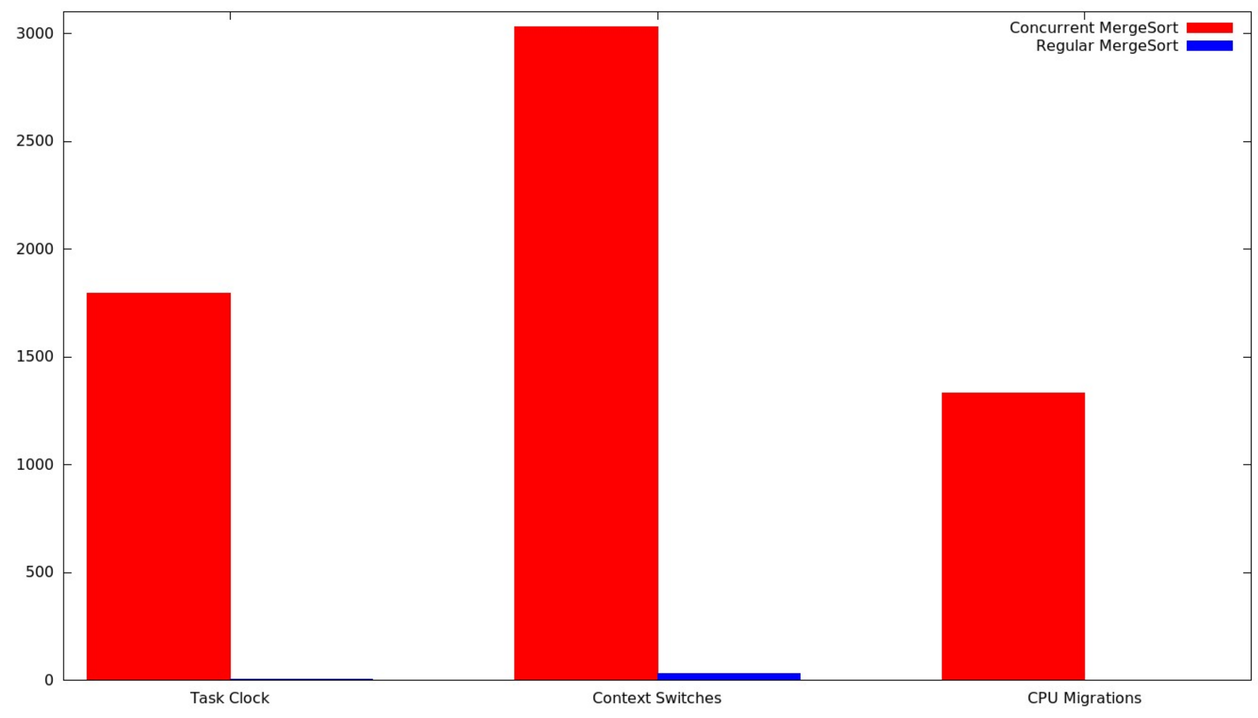
The Concurrent Merge Sort was failing on large amount of numbers (above 5000 on my system). It was implemented as a merge sort which forks and recursively calls merge sort, and if the number of elements are less than 5, it calls selection sort and returns.

I) Concurrent Merge Sort Profiling

```
 Performance counter stats for './mergesortfork':

      1796.436725      task-clock (msec)         #    3.344 CPUs utilized
            3,029      context-switches          #    0.002 M/sec
            1,331      cpu-migrations            #    0.741 K/sec
           50,229      page-faults               #    0.028 M/sec
     93,44,63,328      cycles                    #    0.520 GHz                   [72.75%]
                0      stalled-cycles-frontend   #    0.00% frontend cycles idle    [92.80%]
                0      stalled-cycles-backend    #    0.00% backend  cycles idle    [97.33%]
     53,55,84,337      instructions              #    0.57  insns per cycle         [92.87%]
      8,57,13,450      branches                  #   47.713 M/sec                   [78.30%]
        18,28,297      branch-misses             #    2.13% of all branches         [64.94%]

      0.537142839 seconds time elapsed
```

II) Regular Merge Sort Profiling

```
 Performance counter stats for './normalmergesort':

         4.878725      task-clock (msec)         #    0.760 CPUs utilized
               33      context-switches          #    0.007 M/sec
                2      cpu-migrations            #    0.410 K/sec
               58      page-faults               #    0.012 M/sec
        65,68,439      cycles                    #    1.346 GHz
                0      stalled-cycles-frontend   #    0.00% frontend cycles idle
                0      stalled-cycles-backend    #    0.00% backend  cycles idle
        77,04,028      instructions              #    1.17  insns per cycle         [23.95%]
   <not counted>       branches
   <not counted>       branch-misses

      0.006417530 seconds time elapsed
```



Analysis:

The execution time for the concurrent merge sort is way larger than that of normal merge sort. The number of context switches for concurrent merge sort is the reason for the contrasting result. This is because it is generating 2 processes at every depth of recursion until the base condition, and context switching so many processes adds to the total time. The making of processes to take advantage of multiprogramming and threads backfired as the number of processes were too high.

Thus we see a huge difference (factor of thousands) in Task Clock, Context-Switches, and CPU Migrations (shown in the graph).

We'd have seen better result had the sort been implemented for the number of cores of a system (i.e. number of child processes = number of cores in the system). This would form a parallel sort which would give better result than this implementation of concurrent merge sort.