iLab Machine: ilab3.cs.rutgers.edu

Program Testing: “hypertrophy”	spell_t2_singleloop	spell_t2_fastest	spell_t4_singleloop	spell_t4_fastest
Test 1	133.757 ms	133.743 ms	67.167 ms	66.810 ms
Test 2	135.062 ms	133.040 ms	67.238 ms	66.965 ms
Test 3	134.727 ms	134.408 ms	67.114 ms	67.870 ms
Test 4	133.743 ms	134.587 ms	66.439 ms	66.977 ms
Test 5	134.873 ms	134.412 ms	67.009 ms	66.813 ms
Test 6	134.845 ms	133.953 ms	67.297 ms	67.226 ms
Test 7	134.768 ms	133.677 ms	66.338 ms	67.336 ms
Test 8	134.566 ms	133.237 ms	66.544 ms	67.332 ms
Test 9	133.884 ms	133.686 ms	66.237 ms	66.963 ms
Test 10	133.789 ms	133.504 ms	66.278 ms	67.115 ms
Average	134.4014 ms	133.8247 ms	66.7661 ms	67.1407 ms
Median	134.6465 ms	133.7145 ms	66.7765 ms	67.046 ms
Standard Deviation	σ = 0.5113	σ = 0.4893	σ = 0.4126	σ = 0.3034

The data demonstrates that the parallelization of the loop within the code and increasing the number of threads reduces its overall execution time; by doubling the number of threads from two to four, the execution time (as seen by the given average execution times) is reduced by 50%. In all four versions of the code, I chose to parallelize the outer for-loop that was used to create the bit vector, for it utilizes a nested loop structure and thus takes the longest time to complete its computation. The parallelization I chose to implement uses a guided schedule that specifies how the work should be divided among the threads using a guided scheduling algorithm that assigns the work to the threads in chunks of 500 iterations. By doing so, we speed up the execution of the loop by dividing the work among multiple threads that can run simultaneously on different cores or processors. I found this to be the optimal approach. There exist some overheads associated with using #pragma omp parallel for private(j,hash) schedule(guided,500), including overhead due to the need to create and manage smaller chunks and overhead due to the need to balance the workload among threads. However, my tests indicate that employing parallelization with a guided schedule overrides these overheads while producing fast execution times. In addition, I chose to parallelize the for-loop used for the execution of spell-checking in only the spell_t4_fastest file, for it seemed to give me more optimal results. I used #pragma omp parallel for private(hash) schedule(static,4) so that each thread is assigned a fixed number of iterations to process at compile-time, with each chunk consisting of 4 iterations.