Q1 Performance Analysis

Single threaded

Invocation:

java q1 **1** "$(LC_ALL=C tr -dc '[:digit:]' < /dev/urandom | head -c [num_digit])" "$(LC_ALL=c tr -dc '[:digit:]' < /dev/urandom | head -c [num_digit] "

Number of digits	Performance (ms)
500	242
600	315
700	327
800	377
900	394
1000	416
2000	785
5000	2720
10000	7690
15000	13620

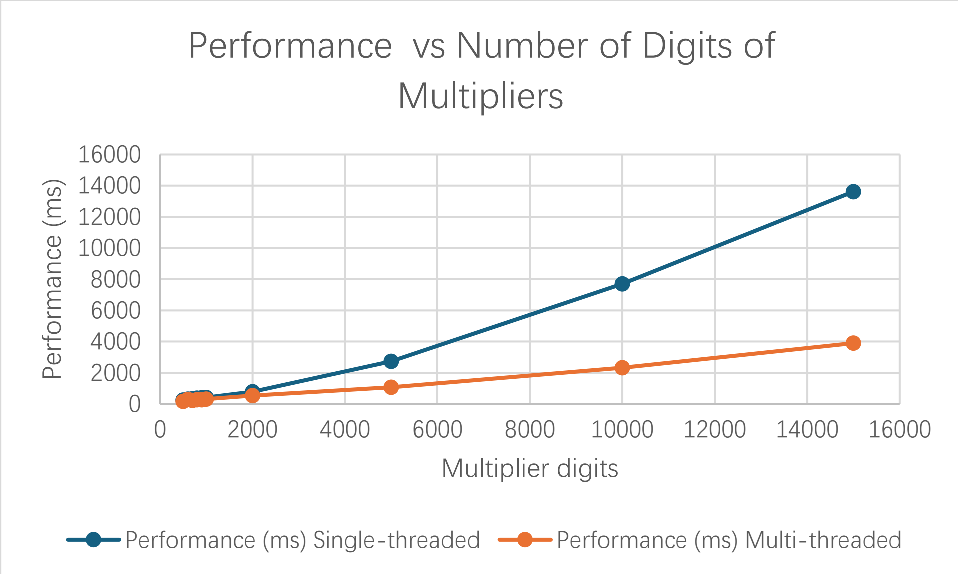
Multi-threaded (number of logical processors: 16)

Invocation:

java q1 **16** "$(LC_ALL=C tr -dc '[:digit:]' < /dev/urandom | head -c [num_digit])" "$(LC_ALL=c tr -dc '[:digit:]' < /dev/urandom | head -c [num_digit]"

Number of digits	Performance (ms)
500	156
600	279
700	225
800	258
900	270
1000	305
2000	529
5000	1058
10000	2322
15000	3897

Comparison Graph



Explanation

Program q1.java shows non-trivial speed up especially when the multiplier's digits have significantly long digits. This behavior is as expected. When testing with multipliers with 15000 digits, the program achieves a speedup of approximately 3.495. This suggests that parallel execution is effectively leveraging the available CPU resources and improving overall performance. the Karatsuba algorithm uses a divide-and-conquer approach, which allows for efficient parallelization of the workload. Each time the multipliers got split, new parallel tasks are generated, enabling multiple threads to process different parts of the computation concurrently. This results in reduced execution time.