Nick Carey
ncarey90@gmail.comncarey4@jhu.edu

Assignment #2: Measuring Parallel Performance

1) For reasonable parameters and for however many cores you have on the system, measure the scaleup and speedup of this program.

a) Produce charts and interpret/describe the results. Is the speedup linear?

Note: charts are located at the end.

By increasing the number of threads from 1 to 4, we achieve near linear speedup. Past 4 threads, there is clear sublinear speedup. There is a slight increase in performance at 8 threads, but at 16 and 32 threads we actually see a decrease in performance.

b) Why do you think that your scaleup/speedup are less than linear? What are the causes for the loss of parallel efficiency?

I think that my scaleup and speedup are less than linear for thread counts greater than 4 because of the hardware used to run the experiment. As with the last assignment, the actual hardware being used is four physical cores equipped with hyper-threading, allowing for eight logical cores. Interference between the logical cores and the physical cores is preventing linear speedup. For this particular experiment, hyper-threading allows for only sublinear speedup.

There is a software start-up cost inefficiency that may have helped cause some sublinear speedup from a thread count of 1 to 4. This inefficiency is when the main, serial thread must create n copies of the SealedObject, where n is the number of parallel threads that will test key ranges against their own copy of SealedObject. The reason that each thread must have their own copy of the SealedObject is so that each thread is not contending for the same SealedObject.

c) Extrapolating from your scaleup analysis, how long would it take to brute force a 56 bit DES key on a machine with 64 cores? Explain your answer.

With four physical cores, near linear speedup/scaleup is achieved up to four threads. Therefore, with 64 physical cores, near linear speedup should be achieved up to 64 threads.

A 56 bit key means that there are 2^56 ($7.205759404 \times 10^{16}$) keys to search.

With 1 thread processing a 24 bit key, 233807 keys were examined per second.

With 2 thread processing a 24 bit key, 421717 keys were examined per second.

With 4 thread processing a 24 bit key, 734872 keys were examined per second.

Going from 1 to 2 threads, there is a throughput speedup of 1.8. Going from 2 to 4 threads, there is a throughput speedup of 1.74. I am going to approximate and state that doubling the amount of threads results in multiplying the starting throughput by 1.75.

4 threads * 2^4 = 4 * 16 = 64 threads

734872 kps * 1.75^4 = 734872 * 9.3789 = 6892295 keys examined per second with 64 threads

2^56 / 6892295 = 10454804101 seconds = 174246836 minutes = 2904113 hours = 121004 days = 331.5 years.

Ouch. Let's do this again but assume actual linear speedup. That is, doubling the amount of threads doubles the throughput.

4 threads * 2^4 = 16 threads

734872 kps at 4 threads * 2^4 = 11757952 keys examined per second at 64 threads

2^56 / 11757952 = 6128413692 seconds = 102140228 minutes = 1702337 hours = 70930 days = 194.3 years.

OK, definitely going to need a bigger computer.

2) Design and run an experiment that measures the startup costs of this code.

a) Describe your experiment. Why does it measure startup?

True Throughput = total keys examined / time spent examining keys.

Estimated Throughput = total keys examined / total runtime

Total Runtime = startup time + time running parallel code.

As the time running parallel code dominates startup time more and more, the estimated throughput approaches the true throughput. Time running parallel code dominates startup time as we increase the key space size. Therefore, as we increase the key space size and keep all other variables constant, the estimated throughput approaches the true throughput. By comparing our best estimate for the true throughput to the estimated thoughput at low key space sizes, we can calculate the startup cost for each thread count. Observe the last graph in the appendix.

True Throughput = total keys examined / (total runtime - startup time)

By using our largest key experiment throughput as our best estimate for True Throughput, we can plug in total keys examined and total runtime from smaller key experiments to estimate the startup time.

b) Estimate startup cost. Justify your answer.

1 thread:

Best estimate for True Throughput: 233807 keys per second (kps) while cracking a key length of 24

Key length of 18:

233807 kps = 262144 keys / (1.6 seconds - startup cost)

Startup cost = .478807 seconds

Key length of 19:

233807 kps = 524288 keys / (2.7185 seconds - startup cost)

Startup cost = .47610 seconds

Key length of 20:

233807 kps = 1048576 keys / (4.983 seconds - startup cost)

Startup cost = .4982 seconds

As these equations show, our best estimate for the startup costs of a single threaded experiment seem to be .478 seconds.

2 threads:

Best estimate for True Throughput: 421717 keys per second while cracking a key length of 24

Key length of 18:

421717 kps = 262144 keys / (.9205 seconds - startup cost)

Startup cost = .298 seconds

Key length of 19:

421717 kps = 524288 keys / (1.573 seconds - startup cost)

Startup cost = .329 seconds

Key length of 20:

421717 kps = 1048576 keys / (2.835 seconds - startup cost)

Startup cost = .34 seconds

Startup cost for a double threaded experiment seems to be .34 seconds

4 threads:

Best estimate for True Throughput: 753889

Key length of 18:

753889 kps = 262144 keys / (.6491 seconds - startup cost)

Startup cost = .301 seconds

Key length of 19:

753889 kps = 524288 keys / (1.07 seconds - startup time)

Startup cost = .37 seconds

Key length of 20:

753889 kps = 1048576 keys / (1.73 seconds - startup cost)

Startup cost = .339 seconds

Startup cost for a four threaded experiment seems to be about .33 seconds

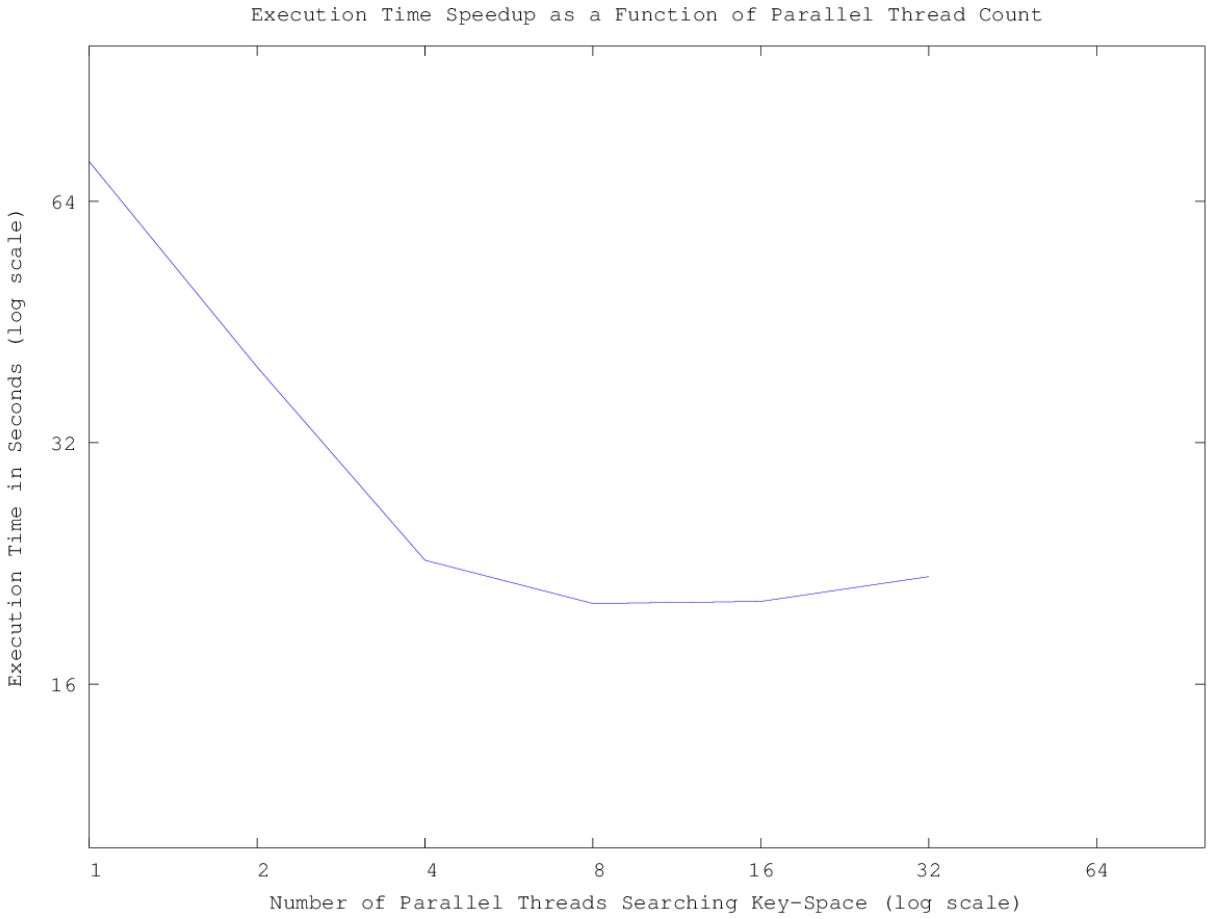
Surprisingly, startup costs seem pretty consistent across experiments with different thread counts. I expected startup costs to be much heavier on experiments with higher thread counts, due to the cost of copying multiple SealedObjects for each thread.

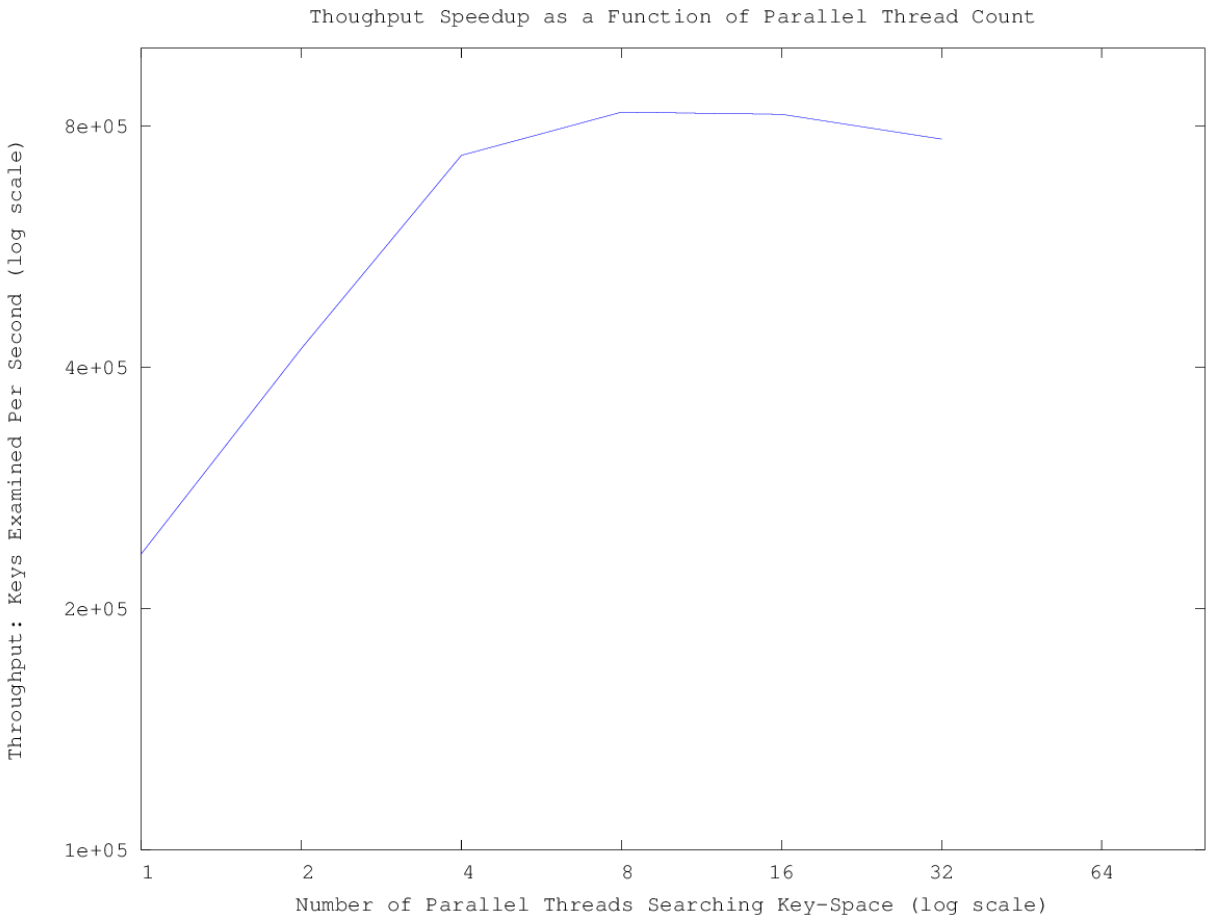
c) Assuming that the startup costs are the serial portion of the code and the remaining time is the parallel portion of the code, what speedup would you expect to realize on 100 threads? 500 threads? 1000 threads? (Use Amdahl's law.)

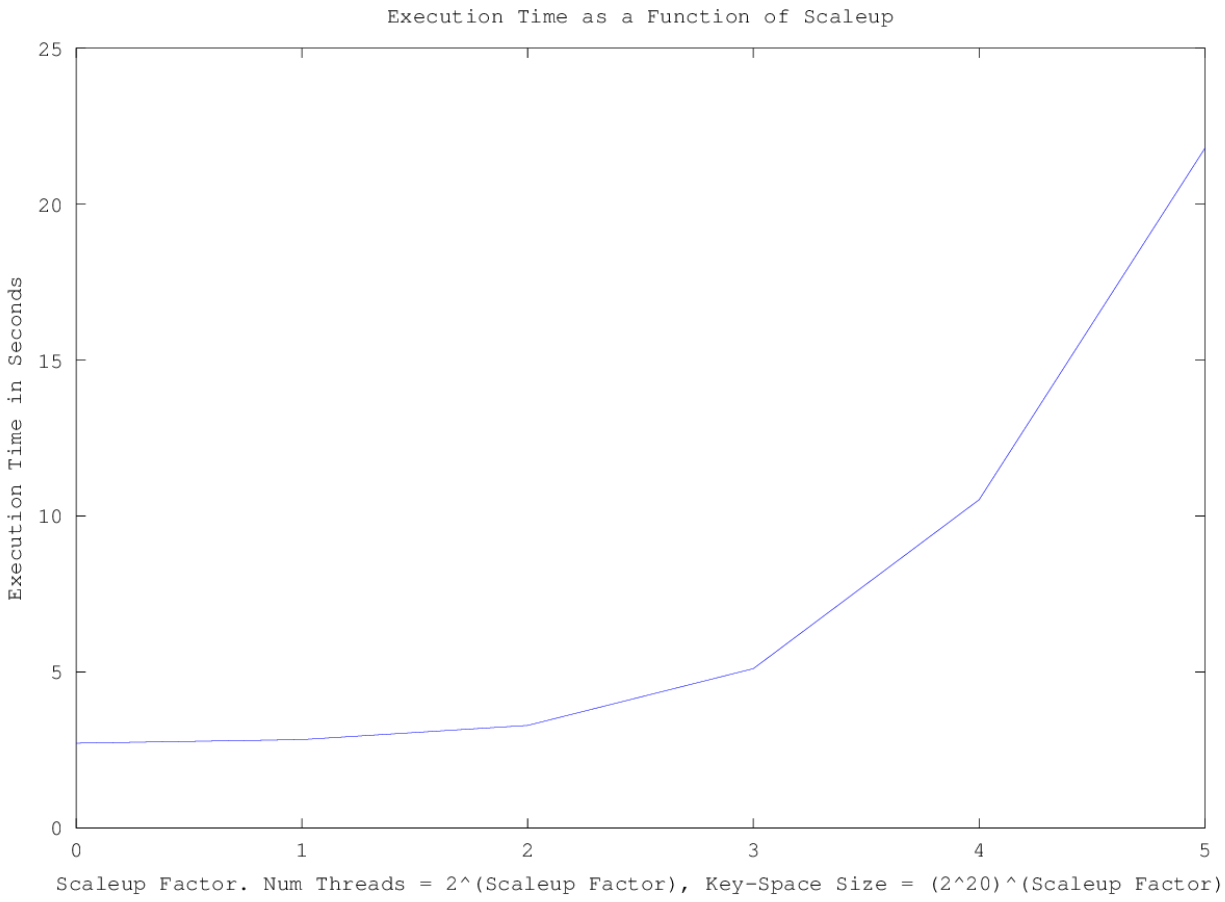
For this question, I will assume that for 100, 500, and 1000 threads there will be appropriate hardware resources allowing the possibility of linear speedup. I will also assume a constant key length of 20, and a startup cost of .34 seconds which is 7% of a serial execution time. I must assume a constant key length because it appears startup cost is uniform across key lengths and that the speedup realized will be different for different key lengths.

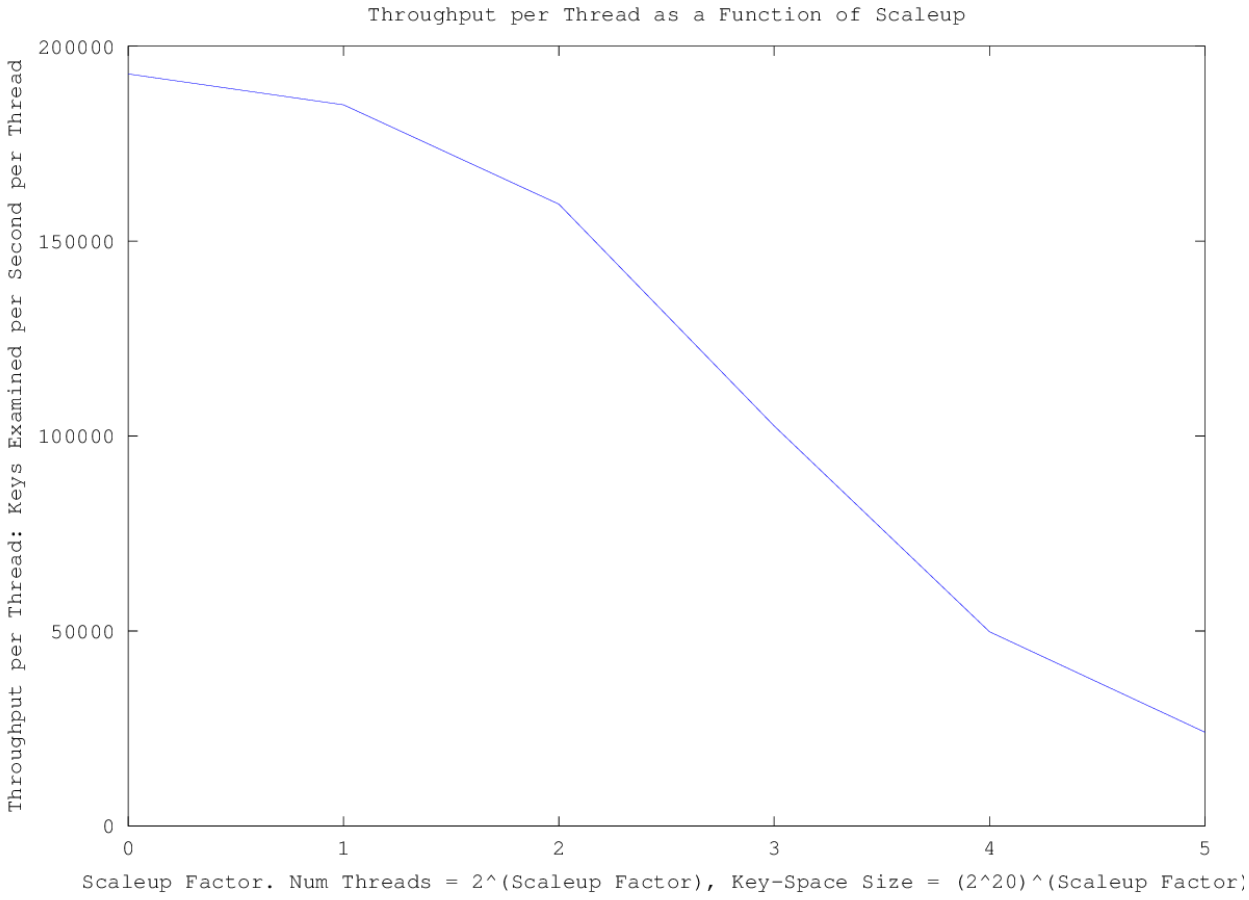
S(100) = 1 / (.07 + 1/100 (.93)) = 12.61
S(500) = 1 / (.07 + 1/500 (.93)) = 13.91
S(1000) = 1 / (.07 + 1/1000(.93)) = 14.09

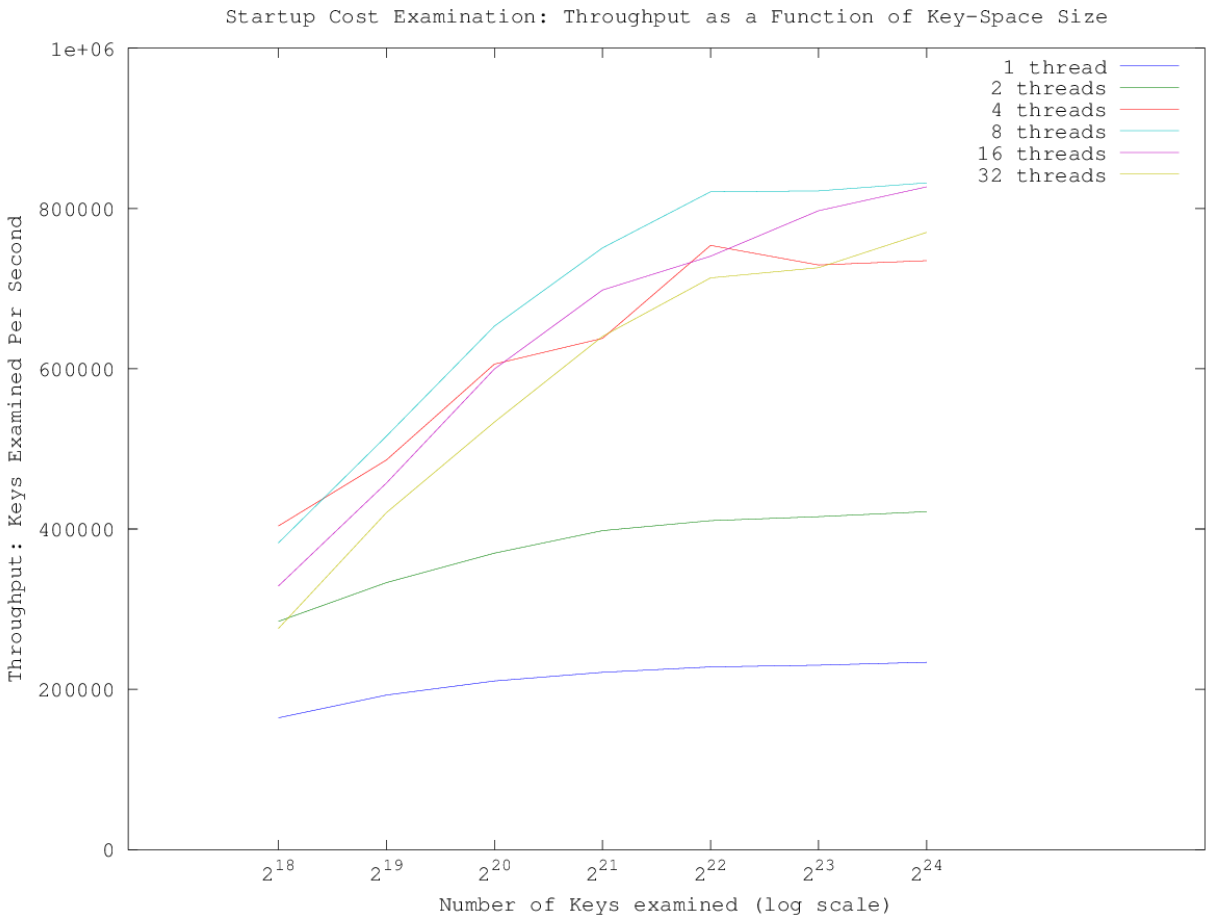
Appendix: Graphs




Thoughput Speedup as a Function of Parallel Thread Count
Throughput: Keys Examined Per Second (log scale)
8e+05
4e+05
2e+05
1e+05
1
2
4
8
16
32
64
Number of Parallel Threads Searching Key-Space (log scale)


Execution Time as a Function of Scaleup
Execution Time in Seconds
25
20
15
10
5
0
0
1
2
3
4
5
Scaleup Factor. Num Threads = 2^(Scaleup Factor), Key-Space Size = (2^20)^(Scaleup Factor)


Throughput per Thread as a Function of Scaleup
Throughput per Thread: Keys Examined per Second per Thread
200000
150000
100000
50000
0
0
1
2
3
4
5
Scaleup Factor. Num Threads = 2^(Scaleup Factor), Key-Space Size = (2^20)^(Scaleup Factor)


Startup Cost Examination: Throughput as a Function of Key-Space Size
Throughput: Keys Examined Per Second
1e+06
800000
600000
400000
200000
0
1 thread
2 threads
4 threads
8 threads
16 threads
32 threads
2^18
2^19
2^20
2^21
2^22
2^23
2^24
Number of Keys examined (log scale)