Chen Zeng (cz39)
Yuanqing Zhu (yz120)

1. Use a couple of English paragraphs describing your multi-threaded implementation. How did you implement the multi-threading? Was it difficult? Did you follow my suggestions? Did you do something else?

All of the multi-threaded implementation are done using the library of #include <thread> in C++ 11 and this library provide easily used API. We just need to put all the function excused by different into threads and then push them to a vector. Using the join function of library <thread> and these functions will be run under multi-thread.

As for RegularSelection and ScanJoin, we divide the input table into multiple sub-table which are done by different threads. And for Aggregate, we divide the hash-table to be handled by multiple threads.

As for the implementations of RegularSelection and ScanJoin, they are easy. However, as for Aggregate, the implementation is pretty difficult. I didn't completely implement the Aggregate successfully, although I've tried lots of times and maybe my operation on keeping the sharing of hash-map is something unreasonable. Some other students tell me that maybe I can try to keep a global vector to store the hash-map in a vector and push them into it in every threads. However, it still doesn't on my case and I've really spent a lot of time on it.

I follow some of your suggestions and my main methods of how to make these three operations to be multi-threads are the same as yours. However, I don't use the pthread library but use the thread and I think they are similar.

2. Two plots or tables where you run the computations on lines 545 and 576 of RelOpQUnit.cc multiple times, using one, two, three, and four threads

The unit of measurement is milliseconds. We can directly run the updated A8 to get the result, please help to change the variable THREAD_NUM to control the number of threads. The test are done on Chen's Macbook Pro.

	Normal	1-thread	2-thread	3-thread	4-thread
RegularSelection	72	59	37	30	27
ScanJoin	23661	22936	30919	24605	23885
~~Aggregate~~	~~121660~~	~~119380~~	~~127559~~	~~116370~~	~~114215~~

3. Run each of the the ten different A7 queries that you were able to complete, and report both the regular (non-threaded) running times and multi-threaded running times (using four threads) for those ten queries.

The unit of measurement is milliseconds. Please add the file of ScanJoinMultiThread, RegularSelectionMultiThread, and AggregateMultiThread into submission A7 under the directory of RelOps. The test are done on Chen's Macbook Pro. Due to the completion amount of A7, we can only test on Query1 – Query7.

	Regular	Multi-threads (4-threads)
Query1	109304	150292
Query2	106228	168611
Query3	14446	14341
Query4	15511	15478
Query5	107638	100032
Query6	101790	104077
Query7	16673	16754

4. Conclude your report with a paragraph or two discussion of whether or not your multi-threaded implementation was effective. If not, why was it not effective? What might you have done differently to make it better?

The results show that the multi-threaded implementation is not must to be effective. The result of RegularSelection works and the result of ScanJoin doesn't improve a lot on the performance. Sometimes it's even slower. The possible reason is that as for multi-threads, it need to create, delete, and manage the mutex and these operations may take some time. Also, maybe some other running program on my laptop may influence on the work of our tested program.

If I have time, maybe I'll try to use pthread to test the performance, although I guess the performance may be similar. I think the best way to make it more effective is to make more functions and operations to be multi-thread.