COMPARISON WITH TENSORQTL

This note provides details of the comparison of LiteQTL and tensorQTL as expressed in Table 2.

Both programs were run on the same hardware using the same CUDA library, as detailed in the manuscript. We tested with two sets of conditions.

Full matrix: This test condition computes the whole result matrix, without any filtering. This set of testing condition, is probably not favorable to tensorQTL, since it is not designed to handle full result matrix. In fact, when we used bigger input sizes, CUDA library would report out of GPU memory error, because tensorQTL was chunking the computation based on filtered input and output. But this testing set can be a good measurement to show the advantage of our implementation.

Filtered: This set of test condition follows the example shown in tensorQTL package, for which tensorQTL is optimized for. In this set of testing conditions, we turned on the filtering threshold, p-value threshold, and MAF filtering threshold. These filters reduce the size of data involved in the computation, resulting in much shorter data transfer time, and computation time.

Because of limited GPU memory, we used full phenotype matrix, and 20000 variants for both testing sets. All CPU timing was done with 20 threads. The timings shown are averages of 10 runs measured in seconds.

Elapsed time inlcudes data transfer, core computation, and post processing. For a fair comparison between CPU and GPU, it is important to include data transfer time, since it is a necessary cost when using GPU and could obscure ultimate speedup, if excluded. Core computation is the time spent on the criticle part of algorithm, and generate output for post processing. Post processing includes time spent on processes that generates meaningful output to user, such as calculate p-values, concatenating dataframes, adding additional information, type conversions, etc. Since this part varies depending on user input, and both tensorQTL and LiteQTL are designed with different defaults in mind, we do not think it is fair to use this time to compare performance or speedup between LiteQTL and tensorQTL. We modifed tensorQTL's code to insert time stamps. The repository showing our modification is: https://github.com/senresearch/tensorQTL.

RESULT

Test 1: Full matrix.

tensorQTL: As mentioned earlier, this test is probably not how tensorQTL intends to be used. The data transfer for CPU is theoretically 0. We observed 0.015 for CPU data transfer time, possibly because tensorQTL depends on PyTorch. This could be the cost of wrapping data into a tensor format to prepare for PyTorch.

If we only compare core computation time, GPU provides a speedup of 17 times, from 0.94 seconds down to 0.055 seconds. However, if we count in data transfer time (moving data to and from main memory to GPU memory), which is necessary when using the GPU, it brings GPU timing to 0.616 seconds. The speedup drops from 17 times to 1.5 times. The time spent on post processing for tensorQTL is also significant, most of which is spent calculating p-values.

LiteQTL: In terms of data transfer time and core computation LiteQTL is similar to tensorQTL. There is a small yet still positive speedup. The speedup is swallowed by the data transfer time. Post processing step is 0.785 seconds, which is much longer than core computation (0.054s). Most of post processing for LiteQTL is concatinating results to a data frame.

Test 2: Filtered

tensorQTL: The filtering is based on the MAF filtering threshold (0.05), and p value threshold (10^{-5})). After filtering 20000 variants, there are 11515 variants left. Data transfer time and post processing time benefit from filtering, which are shorter compared to tensorQTL's full matrix test.

LiteQTL: The filtering standards are MAF (minor allele frequency) filtering threshold (0.05), and the maximum LOD score of each transcript. MAF filtering removed about half the genotype, therefore resulting in about half of the computation time. Another filtering step is the find maximum LOD for each transcript, which results in a much smaller output, hence the decreased time in data transfer. This boosts GPU speedup compared to CPU only implementation to 10.7 times.

CONCLUSION:

We find that data transfer and core computation costs are similar for both tensorQTL and LiteQTL. We were limited to testing on a single hardware platform and a singe dataset. We believe this will hold more generally since both packages are using the same core approach. We found that data transfer and post processing can poses burdens, and thus those are areas where there may be room for further improvement in the future.