Lam Hoi Kei e1345735 assignment 2

1. Algorithm

Pre-define sample_batch and sign_batch to be size of sample batch () and sign batch () respectively. Partition samples and signatures into sample_batch and sign_batch with padding. Then, launch a kernel in GPU with grid size = sample_batch and block size = sign_batch with following parameters.

- d_sample_seq : 1d char array of DNA sequence of samples in that batch
- d_sign_seq : 1d char array of DNA sequence of signatures in that batch
- d_match : 1d int array of starting index in sample sequence that match corresponding sign sequence. If no match, then stores -1.
- d_score : 1d double array storing match confidence score if there is a match.
- d_qual : 1d char array of encoded quality string of d_sample_seq
- n,m,i,j : size of samples, signatures, sample batch index, signature batch index respectively
- sample_batch, sign_batch : as defined before

Inside the kernel, each thread block finds matches between one sample sequence with sign_batch number of signature sequences. Each thread corresponds to one distinct pair of matching between sample sequence and signature sequence. The matching algorithm is brute force. After that, in host, we search for non -1 entry in match and push a match with the corresponding sample name, signature name, confidence score computed into matches vector.

Choice of Grid and Block dimensions

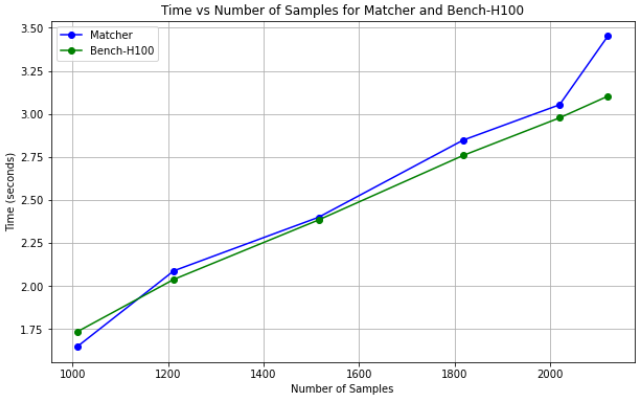
For too small grid and block dimension, there is large overhead created by launching large number of kernels. Meanwhile, for too large grid and block dimension, memory copying before launching a kernel may take too much time and the program may decrease the concurrency. The block dimension is a multiple of 32 with padding if necessary to allow fully utilize 32 threads in a warp.

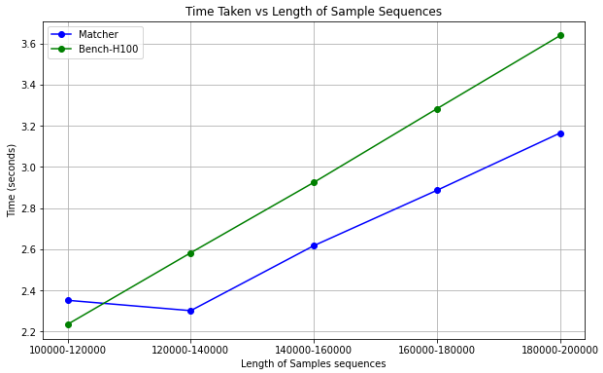
Memory Management

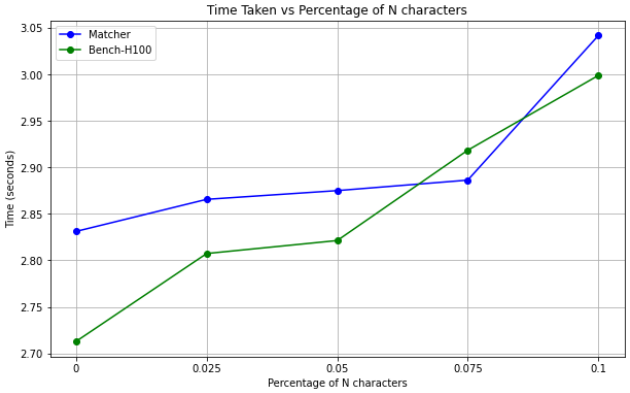
Samples and signatures are partitioned and passed to device as batch to prevent too much memory usage at single time. Sample sequence is stored in global memory as it is too big. Partitioning it into shared memory leads to more tedious implementation, including handling matching across partitions. As signature sequences are distinct for each thread, there is no need to store it in shared memory.

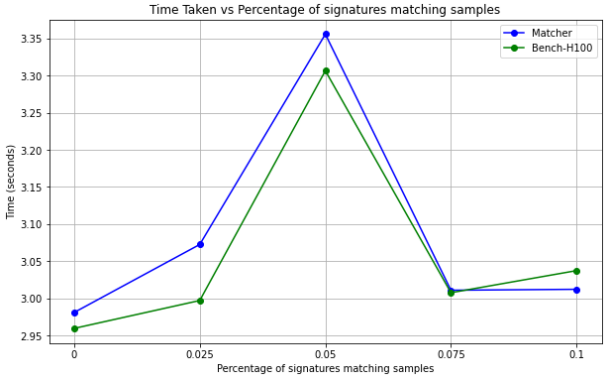
2. Question

1. By default, samples sequence length from 100000 to 200000, signatures sequence length = 3000 to 10000, number of signatures = 1000 and percentage of N = 0.1, percentage of signatures matching samples = 0.1*number of samples, number of viruses = 1 to 2, average Phred score from 10 to 30. Otherwise specified:









2. Time increases for increasing number of samples, length of samples and percentage of N characters in general. However, the relationship between percentage of signatures matching samples does not have an obvious trend and may only affect the time minimally. Increasing samples and sample length both lead to increasing amount of data to process by the program so time increases. The increase in time when sequence length increases is not that significant for matcher program because the memory passed to device is for sequence length of 200000 so increase in sequence length does not increase the memory copying and allocation time. Increase in N characters lead to more matchings in substrings but not full matching so lead to extra time of processing these cases.

3. Performance Optimizations

1. Optimize grid and block size:

   The optimized grid and block size are 2048 and 1024 respectively.

2. Do end checking to exit when sample end is reached rather than checking whole memory which is 200000 in length of sequence:

   This allows more efficient matching for each thread and lead to performance boost.

Bonus

- File : Bonus.cu
- Reproduce result : Modify Makefile and follow appendix
- Algorithm : KMP algorithm is used in each thread for each pair of matching Firstly, find the longest prefix that is also a suffix for every length of signature sequence. This is stored in lps array. Lps of length 0 is 0. Start with length 1, keeping track of previous lps value. Loop over the length of signature sequence, there are 3 cases, if i-th char match with prevlps than prevlps and i increases by 1 repectively and lps[i] = prevlps as there is one more match in length i. Else if

prevlps = 0, and we also know there is no match so lps[i] = 0 and increments i. Else, there is no match with i-th char and prevlps-th char and prev lps not equals to 0, so we can check if the previous lps chacater match with i-th char or not as it is the longest possible match given the information we have so prevlps=lps[prevlps-1]. Runtime is linear in length of signature sequence(let it be m). As i can increase at most m times so first 2 cases is O(m). For the last case, notice that prevlps only increases in first case and increases at most m times. Prevlps only decreases in the third case but it cannot decrease more than it increase as it is non-negative. Thus, third case is also O(m).
Then, we can find the matching using the lps array and the runtime is linear in length of sample sequence(let it be n). Init i and j to be 0, 0. Loop i over n, first, check if sample sequence is exhausted while signature sequence is not then we can break as there is no match found. We should also check if signature sequence is exhausted then a match is found and can calulate the starting index of this match. Then, we can do the checking, if i-th char in sample matches j-th char in signature, increments both I and j. Else if, j = 0, so there is no match then we can increment i. Else, j != 0 and i-th char in sample does not match j-th char in signature, meaning that there is some part precisely before this mismatch that is a match. Then, we can find the lps of the previous part and we know that that this part is the longest part precisely before mismatch in sample is same as prefix of signature so we can match following that prefix of signature so j = lps[j-1].i and j can increase at most n times due to first 2 cases in checking. So third case is so linear as j can only decrease at most n times as it is non-negative.

- Time measurement is inside bonus.csv

Appendix

- Reproduce result :
    - In MakeFile : make sure -arch=sm_80
    - Run make
    - Generate samples and signatures similar to following examples

        - ./gen sig 1000 3000 10000 0.1 > sig.fasta
        - ./gen sample sig.fasta 2000 20 1 2 100000 200000 10 30 0.1 > samp.fastq
    - srun --ntasks 1 --cpus-per-task 1 –cpu-bind core --mem 20G --gpus h100-96 --constraint xgpi matcher samp.fastq sig.fasta
- The program is mainly tested using h100-96 by the above command