CS483 Milestone 3 Document

Team members: Ching-Hua Yu, Hanchen Ye, Hanxiao Lu, Mihir Rajpal

Problem review:
As the problem described, the computation involves $\{C \cdot (e @ e^T)) @ H^T\} / (L - 1)$ where C is a symmetric, N×N, block Toeplitz matrix, e is an N×L matrix, and H is an M×N matrix. N is large (10k, 100k, or larger), while M and L are small (10s). From Prof. Butala, the current application only needs C to be a symmetric toeplitz matrix. Also, C is a 90% sparse matrix.

We take the following approaches:

1. Optimize the computation according to the formula $\{C \cdot (e @ e^T)) @ H^T\} / (L - 1)$. The optimization is not limited to the special form of C.
2. Implement Step 1 in cuda code. Consider the condition that N >> L, M in the implementation and choose suitable block dimension, grid dimension accordingly.
3. Applying the parallel programming technique from the lectures

Optimization used:

- Instead of calculating $P = C \cdot (e @ e^T)$ and then $P\_HT = P @ H^T/ (L-1)$ as in Milestone 2, we apply the associative property of the matrix multiplication, and somehow calculate a (C-weighted) $e^T @ H^T$ first and then a (C-weighted) $e @ (e^T @ H^T)$. (See the detail below for a more precise illustration). The way we can avoid big matrix multiplication.
- Because L, M are known to be much smaller than N, for now, we choose the block dimension to be the same (or larger) than L x M x 1.
- We used shared memory to speed up memory access. Particularly, we maintain a shared memory of size L x M (or larger) in a block for now.
- We used the list reduction from Lab 5.1.

Details:

1. Formula Optimization. This part follows the potential approach for a general C described in Milestone 1.

Method:
First, note that if C is removed from $\{ [C \cdot (e @ e^T)) @ H^T\} / (L - 1)$ and temporary leave out L-1 since it's just a scalar, we can apply the associative property of the matrix multiplication on $(e @ e^T) @ H^T$ to get $e @ (e^T @ H^T)$, which can save a lot of computation since L, M<<N. Now consider the computation involves $C = [C_1, \ldots, C_N]$, where Ci is the i-th row of C, and let $e = [e_1, \ldots, e_n]$, where ei is the i-th row of e. In our computation, we can first calculate $Di = ((e^T \odot Ci) @ H^T)$, where $e^T \odot Ci$ here means multiplying each row of e^T by a row vector Ci like $e^T * Ci$ in python, for i from 1 to N, and then calculate fi = ei @ Di for i from 1 to N before concatenate them to get $f = [f1,...,f_N]$. The computation job is equivalent to one NxN matrix dot product, one NxL matrix and LxM matrix multiplications and N times of LxN matrix and NxM matrix multiplications.

2. Choose suitable grid and block dimension.
   Note that each Di = ((e^T ⊙Ci) @ H^T) is a L x M matrix for i = 0 to N-1. Since the next step fi = ei @ Di needs Di as well, it's convenient to assign each block for an index i (which is gridIdx.x in the current implementation).

   That is, we set gridDim (N, 1, 1) and blockDim(L, M, 1)

   And so the way allows us to store Di in the shared memory.

   Note:
   - The improvement is clear. More tests for large cases are needed, but the kernel run time is always faster than the implementation in Milestone 1, sometimes more than 10 times (for the small N, L, M).
   - Some generalization may be needed later. When L, M are super small, one block can deal with several i, and when L, M are large, need several blocks to work on the same i.

3. Applying some parallel techniques
   The calculation fi = ei @ Di (for i from 0 to N-1) involves ei (1xL) and Di (LxM). since each element of fi (M) is handled by a thread, this can be done by a variant version of the list reduction.

   Note:
   - We set the matrix dimension to be 2^ceil(log_2 L) instead of L for ease of reduction.
   - as in the list reduction, there are divergences.
   - The improvement of this step is not obvious (mainly because N, L, M are not large in the test)

Next:
- Further optimization
- Generalization mentioned above
- Techniques from the speeches or generalized the codes

An update of this document
https://docs.google.com/document/d/14yewjo3cukf1sg_OsI7PdB5oY0UAvBmoe-FRG3-3CW4/edit?usp=sharing

Project: Kalman-filter
http://lumetta.web.engr.illinois.edu/408-S20/projects/Kalman-filter.pdf

Public gitlab (sequential codes etc)
https://gitlab.engr.illinois.edu/ece408_sp20/ece408-final-project/

Team gitlab
https://gitlab.engr.illinois.edu/ece408_sp20/group_30