5549: Assignment 2

Deadline: 23rd Nov 2018 (11:59 PM EST)

SpMM multiplies a sparse-matrix with a dense-matrix(also known as sparse matrix multi-vector multiplication). It is of the form O=S*D. where S is an m*n (double) sparse matrix and D is n*k (double) dense matrix and O is an m*k (double) dense matrix. The below algorithm shows the SpMM algorithm.

```
SpMM: Sparse Matrix Multi-Vector Multiplication

input  :  CSR S[M][N], float D[N][K]
output :  float O[M][K]
1 for i = 0 to S.rows-1 do
2     for j = S.rowptr[i] to S.rowptr[i+1]-1 do
3         for k = 0 to K-1 do
4             O[i][k] += S.values[j] * D[S.colidx[j]][k]
```

1. Write cuda code for SpMM where S is represented in CSR format (see eval section)
2. Write cuda code for SpMM where S is represented in CSC format (see eval section)
3. Compare CSR VS CSC implementation. What are the advantages and disadvantages of each format for this problem? Wherever applicable, use hardware counters to back your claim. (see eval section)
4. Develop optimized SpMM code (see eval section):
    1. Design and implement an optimized GPU SpMM. You can use any data structure (CSR, CSC, ELL,COO). You can even use your own custom data-structures including hybrid data-structures.
    2. Describe the possible optimizations for SpMM (implemented + unimplemented)

Note: The pre-processing time to create the data-structures will be ignored for this assignment. In practice pre-processing time cannot be ignored; but that's not our focus

Evaluation

The makefile and reference implementation for both CSR and CSC format is provided. Do not change the name of the files. The name of the kernels should be dev_csc_spmm, dev_csr_spmm and dev_opt_spmm. "small.mtx" can be used for testing your implementation (use others also to ensure correctness).

Q1,Q2

- [must for Q1; optional for Q2]: Use pinned memory; add code to measure the kernel time using cuda-events; Use cuda streams to overlap computation and memcpy (no need to stream D for Q1);
- Report the results (GFLOPS) in a table in PDF/DOC/ODT format (don't put screenshots).
- The evaluation should be performed on all the datasets provided for K=31,32,33,64,128 (your code should accept any value of K)
- Submit the code in a tarball /zip
- Measuring GFLOPS: update host code to compute flops (or compute theoretically as 2*K*nnz). The time should include mem-copy and compute time. You can use omp_get_wtime() to compute this. (no need to include allocation time)
- Events timing: Use cuda events to measure the time taken by each stream for computation . Report the min, max and average time.

Q4 [1]

- Report the results (GFLOPS) in a table in PDF/DOC/ODT format (don't put screenshots).
- The evaluation should be performed on all the datasets provided for K=64 (for this question, your code can be hardwired for K=64. Other values of K will not be tested)
- Submit the code in a tarball /zip

Q3,Q4[2]

- Describe your solution in text in PDF/DOC/ODT format.