GPU COMPUTING 2023

Final Project

Clustering algorithms on GPU

Shuxin

Zheng

Academic Year 2022/2023

Contents

1 Abstract

Sparsity matrix has received a lot of attention due to its potential in deep learning research to improve computational efficiency, reduce the model size, and integrate more closely with brain-like computing. However, getting the calculation speedup is not as easy as saving the memory. The idea of 'reordering' is aiming to gather similar non-zero rows together to accelerate the sparse matrix calculation.[1]
This project is aiming to develop a GPU clustering algorithm to accelerate matrix reordering. We start from the algorithm 'IterativeClustering'[1], in order to make the algorithm parallelizable, we do as follow:

1. If there still be rows not grouped, find a list of referring rows that are not grouped and far enough from each other, label them as grouped
2. Each thread checks if the rows in duty are still not grouped, if not, compute the Jaccard distance with the referring rows.
3. If there be rows that are close enough to the referring rows(distance smaller than tau), group them into the nearest referring row(group), otherwise do nothing and go to step 1.

With this algorithm, the distance computation is totally parallelized. To check the correctness, I compared the reordering result with the serial algorithm(checked by density). And for acceleration, I compared the clustering time for each case.

All the code and profiling results can be found in the github.

2 Main design choices

2.1 Data Structures and Functions

2.1.1 Data Structures

No matter the input file format, all data was converted to the CSR(Compressed Sparse Row) format for clustering, the COO format is also used as an intermediary. Besides, the group message was saved in a one-dimension list that has the length as the number of matrix rows. The list is organized as the index of the list represents the row index in the original matrix, and the content of the list is the new group index of the row, -1 means not grouped. For example, a list as '00223' means there be 5 rows in total, while the first and second rows are in the same group, third and fourth also, the fifth row itself is a single group. There is another list that was used to save the referring row index. The COO and CSR data struct were shown below:

```
struct CSR
{
    int rows;
    int cols;
    int nnz;
    std::vector<int> rowPtr;
    std::vector<int> colIdx;
    std::vector<double> values; // not used in our case
    void addRow(struct ROW &row);
    double calculateBlockDensity(int block_rows, int block_cols);
    void print();
}
```

```
struct COO
{
    int rows;
    int cols;
    int nnz;
    std::vector<int> rowIdx;
    std::vector<int> colIdx;
    std::vector<double> values; // not used in our case
};
```

2.1.2 Data Processing Functions

The test datasets have varying sizes and formats, my development offers two different formats: .mtx and .el. The handling functions are:

```
// processing .mtx format, able to process unordered input
COO readMTXFileWeighted(const std::string& filename);
COO readMTXFileUnweighted(const std::string& filename);
// processing .el format, only able to deal with ordered input
COO readELFileWeighted(const std::string& filename);
COO readELFileUnweighted(const std::string& filename);
// convert COO to CSR to compress the size and for calculation
int cooToCsr(COO &coo, CSR &csr);
```

2.1.3 Serial Grouping Function

The grouping function is the function which we time to evaluate the performance

```
void fine_grouping(std::vector<int> &origin_group, CSR &matrix,
            std::vector<std::vector<int>> &result_group, float tau)
```

2.1.4 GPU Grouping Function

```
__global__ void gpu_ref_grouping(int* rowPtr, int* colIdx, float* tau,
                        int* resultList, int* groupSize, int* refRow)
```

2.2 Sychronization Strategy

The most important problem here in my development is the synchronization problem. Here we let each comparison with the referring rows be one iteration, synchronization should be done after

1. A new referring list has been created
2. An iteration finished

In our case, the synchronization may need to be across the block since we will need more blocks when the dataset grows larger. So the function '__syncthreads()' does not meet our requirement. To solve this problem, I included the tech 'cooperative_groups' to synch up the whole grid.

2.3 Distance Function

To check the distance between rows, we introduce the Hamming distance and Jaccard distance. The Hamming distance ($v, w \in 0, 1^N$) is:

$$H(v, w) = |v \backslash w| + |w \backslash v|$$

and the Jaccard distance is

$$J(v, w) = 1 - \frac{|v \bigcap w|}{|v \bigcup w|} = \frac{H(v, w)}{|v \bigcup w|}$$

3 Performance

The correctness is checked by comparing the reordered density, all the results are the same. The raw profiling files and more detailed messages can be found in the github.

3.1 Complexity Analysis

3.1.1 Computational Complexity Analysis

The complexity for each distance computation is $2 \times min(rank1, rank2)$ where the rank means the number of non-zero values in the row. The complexity is brought from two parts(n is the reference list size):

1. Finding the referring row(s) - O(1) for searching and O(1) for comparison among referring list(at most $C_n^2 \times 2 \times min(rank1, rank2)$, but n is usually small, so we think it's a constant).
2. Comparing between each row, at worst case the complexity of $nrows \times (nrows - 1) \times nnz$

So the computation time should grow as the number of rows and the number of non-zero values grow in theory. But there be some other influencing factors: tau and n(reference list size) will affect the execution time - the smaller tau/ larger n, the longer the time spent. The influence from tau is due to: the higher tau, the easier the row to find a group, so less comparison should be handled. While for the case of n, the larger n, the more time a row need to compare with others. (However, decreasing tau and increasing n does not really means the new density will increase since we do not consider the block boundary message. In the case considered, it will for sure give benefits)

3.1.2 Memory Complexity Analysis

Memory boundary is for the size of CSR matrix. The boundary (global memory size of CPU/GPU(A30)) is combined of several part:

1. CSR message: $((nrows + 1) + ncols) \times sizeof(int)$
2. Referring list and result list: $(nrows + n) \times sizeof(int)$
3. tau: $sizeof(float)$

3.1.3 Real Execution Time Analysis

To evaluate the performance of our development, we collect several datasets with different sizes and formats. The execution time is listed in the table3.1 below(only clustering time counted). From this table, we can see that when the matrix size is small, the overhead of parallelization(mostly due to synchronization) can not be hidden.

Dataset Size $(nrows, ncols, nnz)$	**CPU time** ms	**GPU time** ms	**speedup** $Sp = T_s/T_p$
(28, 29, 250)	0.04	0.54784	0.073
(60, 61, 353)	0.259	2.50163	0.104
(289, 289, 1089)	4.379	26.3342	0.166
(494, 494, 1080)	6.43	50.6829	0.127
(1138, 1138, 2596)	40.063	221.846	0.181
(2358, 2358, 9815)	211.375	533.595	0.396
(3296, 3296, 6432)	1161.43	1678.89	0.692
(3918, 3918, 16697)	1094.61	1574.49	0.695
(6144, 6143, 613871)	12745.5	5195.88	2.453
(8141, 8141, 1012521)	41882	2207.37	18.974
(8275, 8298, 103689)	19985.7	9087.07	2.199
(9507, 9507, 591626)	15785.2	1316.35	11.992

Dataset Size $(nrows, ncols, nnz)$	**CPU time** ms	**GPU time** ms	**speedup** $Sp = T_s/T_p$
(9507, 9507, 684169)	14990.2	1156.89	12.957
(11949, 11949, 80519)	168349	18287.8	9.206
(12009, 12009, 118522)	51114.3	17404.9	2.937
(21608, 94757, 238714)	TLE	164399	-
(58226, 58228, 214078)	TLE	160861	-
(65536, 65536, 2456398)	TLE	TLE	-

Table 3.1: The clustering time for each dataset(n=1, tau=0.6, block size=8, time-limited=5 minutes)

From this table, we can find that the speedup for the large matrix is somehow very different. By checking the result, we find that in the case we obtained the low speedup, the final group number is much larger than the high speedup cases. This means the clustering needs more iterations and therefore, more synchronization overhead.

3.1.4 Nsight System Analysis

The most important part of the Nsight System profiling is shown in the table3.2, 3.3, and 3.4. Also, a screenshot of the result is shown in Fig3.1. It's unsurprisingly that the 'cudaEventSynchronize' is the domain issue. Since our development is for a single GPU, no extra communication was needed. The memory occupation is fixed since the matrix message size is fixed - no more inter-message was produced to consume the memory.

Time (%)	**Total Time (ns)**	**Name**
92.0	2,279,391,680	cudaEventSynchronize
7.8	194,175,594	cudaMalloc
0.1	1,357,186	cudaGetDeviceProperties_v2_v12000
0.0	802,489	cudaMemcpy
0.0	342,175	cudaLaunchCooperativeKernel

Table 3.2: Executing 'cuda_api_sum' stats report

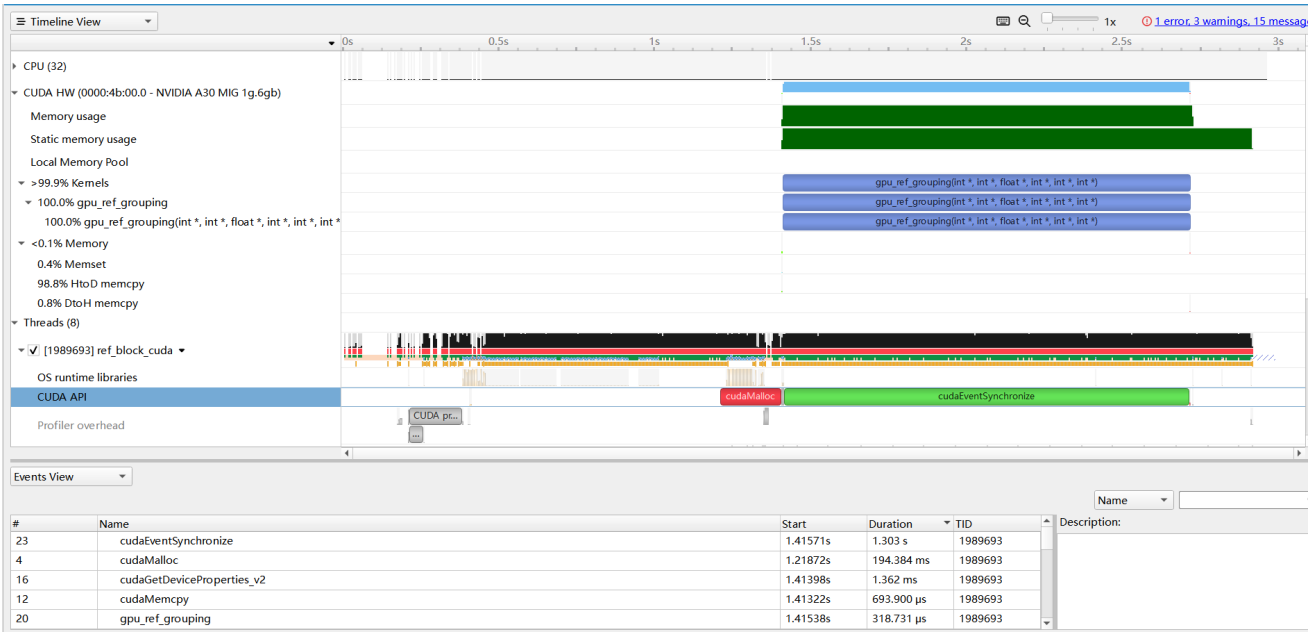

Figure 3.1: profiling screenshot

Time (%)	Total Time (ns)	Operation
98.8	403,647	[CUDA memcpy HtoD]
0.8	3,136	[CUDA memcpy DtoH]
0.5	1,856	[CUDA memset]

Table 3.3: Executing 'cuda_gpu_mem_time_sum' stats report

Total (MB)	Count	Operation
2.443	5	[CUDA memcpy HtoD]
0.038	1	[CUDA memcpy DtoH]
0.000	1	[CUDA memset]

Table 3.4: Executing 'cuda_gpu_mem_size_sum' stats report

4 Conclusion

4.1 Conclusion

This project implements a GPU version for clustering similar rows in a sparse matrix. The most time-consuming part of this project is due to the synchronization issue, while the speedup compared to the serial one is ready case-depending - it's generally influenced by the number of rows and non-zeros values of the matrix, but also strongly affected by the data distributions. The bad data distributions(in the worst case, each row itself as a group) will lead to more synchronization overhead, and therefore hardly any improvement from parallelization can be obtained.

4.2 Further consideration

This project only considers the execution time but lets out the reordering result. In order to gain better results for reordering, several strategies can be taken:

1. Make use of the block column size message. The reordering may even make the structure worse if we do not consider the block boundary. Example: we have three lines

   1 0 0 0 1 0 0 0
   0 0 0 0 1 0 0 0
   0 0 0 0 0 1 1 1

   When considering the block size = (4,4), according to Jaccard distance, the second row will be grouped with the first one. But obviously, it will perform worse, we should group it with the third row. The idea is: instead of using a whole line message to calculate the Jaccard distance, use a 'mask' which can present the block boundary message.

2. Make use of the block row size message. If a group does not have a number of rows that can be divided by block row size, it means there will be some blocks that are combined by two or more different groups after reordering. It's not good for our results. The idea is to fill the group with empty rows so no block will be combined by different groups.

References

[1] Paolo Sylos Labini, Massimo Bernaschi, Werner Nutt, Francesco Silvestri, and Flavio Vella. Blocking sparse matrices to leverage dense-specific multiplication. In *2022 IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 19–24, 2022.