CS516 : Project 1 : Parallelizing SPMV on GPU

Bhupeshraj Senthilkumar
Rutgers University

1 Introduction

The project goal is to implement three parallel versions of sparse-matrix vector multiplication (spmv) on GPU's. The three versions of SPMV to be implemented in this project are 1. Simple Atomics Based SPMV 2. Segment Scan based SPMV 3. Our Own design.

For each of the parallel versions, the time taken by the kernel is measured by varying the block size and the number of blocks used by the kernel.

2 Simple Atomics Based SPMV

In this implementation, I divide each matrices number of non-zeros equally across all threads. So each thread process almost the same number of non-zero elements. The Matrix is stored in Co-ordinate format (COO), Each non-zero value in the matrix is associated with a row index and a column index.

In Matrix Vector multiplication, the result column vector, $y[i] = A[i, j] * x[j]$ looping through all i,j. So I multiply each non-zero entry with the corresponding entry of the vector. Since different threads can process a single row of the matrix, while adding the multiplication value to the $y[i]$, we might encounter data race conditions. In order to avoid data race conditions, atomic operation is used while adding the multiplicative value to the $y[i]$. This is done for every non-zero element in the matrix. The spmv_atomic code implements the above mentioned above.

3 Segment Scan for SPMV

The segment scan approach used by me was described in this paper "Efficient Parallel Scan Algorithms for GPUs" by Shubhabrata Sengupta, Mark Harris and Michael Garland.

In this approach, we try to find segment scan result for an entire block by trying to co-ordinate the segment scan results from each warp, instead of using atomic add operation to co-ordinate between the warps as discussed in the class. The algorithm described in the paper is as follows:

Intra-Block Segmented Scan Approach:

- For each thread in the block, get the warp id and find if it is the first thread or the last thread in that warp. Since the first warp's first thread is always guaranteed to be thread number 0 in the block in CUDA, our calculation for warp id, first thread and last thread in the warp will be correct.
- Check if the thread is open, i.e. if the element processed by the last thread in the previous warp belongs to the same row as the current element this thread is processing, if yes, then the thread is open, which indicates that we need to add the result of previous warp segmented scan result to this thread.
- Calculate the segment scan result for each thread warp.
- Store the last element of the warp and its associated row in a temporary array after the segmented scan.
- Perform segmented scan on the temporary array.
- Then for each thread, we know if that thread is open, it belongs to the same row the element processed by the last thread in the previous warp. We then add the thread's segmented scan result with the previous thread warp segmented scan result.
- After the previous step, we get the segmented scan result for the whole block. We then use atomic add

to add results of each row the block has calculated to the final results to avoid race conditions between the blocks.

This is the implementation I used for the segmented scan approach. Intuition is that it is supposed to reduce the number of atomic operations performed by the block when compared to the approach mentioned in the class.

4 My Own Design for SPMV:

In my own design for SPMV, I have used the approach mentioned in the project description. The segmented scan approach has thread divergence, since each warp may process different number of non-zero elements. So inorder to reduce the thread divergence, we re-order the non-zero elements as follows:

- First place non-zero elements in sets of 32 from each row.
- Then place non-zero elements in sets of 16 from each row.
- Then place non-zero elements in sets of 8 from each row.
- Then place non-zero elements in sets of 4 from each row.
- Then place non-zero elements in sets of 2 from each row.
- Then place non-zero elements in sets of 1 from each row.

If we reorder the data like this, then the thread divergence in the warp will decrease when compared to the segmented scan approach. In my own design, I needed to use atomic add when synchronizing between the warps within the block as discussed in the class. This is because, the row is not in sorted order anymore since the data has been reordered.

So I use the method of segmented scan approach discussed in the class.

5 Results Obtained:

The Results obtained for the above implementation are as follows given in the table :

Block Size : 1024 Threads , Block Number : 8

Matrix	Atomic Time Taken	Segment Time Taken	Design Time Taken
can't	22	174	111
circuit5M_dc	204	1640	1025
consph	33	261	166
FullChip	289	2296	1446
mac_econ_fwd500	14	110	69
mc2depi	21	175	112
pdb1HYS	24	187	120
pwtk	64	507	322
rail4284	127	970	621
scircuit	10	82	51
shipsec1	43	340	216
turon_m	9	77	48
watson_2	19	156	97
webbase-1M	31	257	163
rma10	26	202	129

Block Size : 1024 Threads, Block Number : 4

A	B	C	D

Matrix	Atomic Time Taken	Segment Time Taken	Design Time Taken
can't	41	307	201
circuit5M_dc	381	2892	1899
consph	60	459	300
FullChip	527	4006	2631
mac_econ_fwd500	25	192	126
mc2depi	42	320	212
pdb1HYS	43	330	215
pwtk	117	892	583
rail4284	227	1707	1115
scircuit	19	144	95
shipsec1	78	599	392
turon_m	18	137	90
watson_2	36	277	97
webbase-1M	60	461	302
rma10	47	357	232

The results are not as expected. The segmented scan approach takes more time and it is counter-intuitive. I expected that the time taken by the segmented scan approach will be less due to the fact that the number of atomic operations performed by the segmented scan approach is less than the atomic approach.

The segmented scan is taking more time than the atomic approach, may be due to the fact that there are more syncthreads(), so it may take a lot of time to run the segmented approach than the atomic approach.

But the design with rearranging the data takes less

time than the segmented scan approach, but it still takes more time than the atomic approach, My own design poor performance may attribute to the fact that I use syncthreads for synchronizing the threads within the block.

My own design results takes less time than the segmented scan approach due to fact that the thread divergence is reduced due to data reordering.