Project1: Parallelizing SPMV on GPU

Sairama Rachakonda
Rutgers University

1 Introduction

This report provides the results of parallelizing SPMV on a GPU, using different approaches.

2 SPMV-Atomic

In SPMV-Atomic approach, different threads process the same matrix row and communicate by atomic read-modify-write instructions. To ensure a three-step operation - read,modify,write the atomicAdd is used to add the result of each thread to the output vector. Hence within each warp,multiple threads may conflict for the resource-atomicAdd.

Observations: The running time of the SPMV-Atomic operation on the input, which is not row wise sorted is less than, when the input is row wise sorted.For instance, the Circuit 5M DC matrix takes 192 milliseconds when the input is not row wise sorted, but takes 204 milliseconds when the input is sorted row wise. The reason for this is that, in the row wise sorted approach, all the rows,can be in a warp and hence, each thread will be accessing the same row and races for the chance to write to the same row of the output matrix. And hence it takes more time.
Pros:

- If the underlying GPU provides a fast hardware support for the atomic operations, then SPMV-Atomic works faster.

Cons:

- There are lot of intra warp atomic operations,ie atomic operations between the threads, compared to the SPMV-Segment Scan and SPMV-Design approaches, which will be detailed in the further sections. Instead of seeing an nProcessor parallel speedup or O(nThreads/nProcessor), SPMV application will only exhibit a sequential runtime of O(nThreads). It may be an anti-thesis of parallel programming.

3 This Section has SubSections

3.1 SPMV-Segment Scan

In the earlier approach, intra warp and inter warp atomic operations are a major problem, if the hardware doesn't provide an efficient lock operation. Hence, in this approach my goal was to eliminate the intra and inter warp atomic operations and restrict it to inter block atomic operations. For this, I used the segment scan tool. After the threads of a row multiply a column value with the row value of the vector, the results can be added in parallel using the segmented scan approach. However, I referred to the paper titled Efficient Parallel Scan Algorithms for GPU by Shubabhrata Sen Gupta,Mark Harris and Michael Garland to get some inspiration. In precise, its a block level segment scan.

Approach: Sort the input data row-wise. Perform a segment scan within each warp.Get the segment scan result of last thread in each warp and store it in the first warp in the respective warp numbered thread. Now once again perform segmented scan on theses values in the first warp.Now for every warp from the second warp on wards,if the last value of the previous warp is same as the values in the current warp, then add the previous warp's cumulative sum to these values. If its different, its not necessary to do a cumulative sum.This is the inter warp segment scan or intra-block segment scan. Now do an atomic add operation between inter block for the same row. The approach discussed in the class is different from this, in that inter warp atomic operations are present.

Observations: The running time was slower

when compared to the SPMV-Atomic which was counter-intuitive to our thought process.

Pros:

- If the underlying GPU doesn't provide a fast hardware support for the atomic operations, then SPMV-Segment Scan works better.

Cons:

- The array needs to be row-wise sorted before performing segmented scan.
- There is a thread divergence.
- Since there is a two level of segment scan, many sync thread operations are there, in the algorithm. This requires, other threads, which are not participating in the cumulative some to wait. This is the main reason for the increase in the running time of SPMV-Segment compared to the SPMV-Atomic.

3.2 SPMV-Design

Intuition: The main intuition behind my design is to minimize the thread divergence within a warp. This can be done, if each thread within a warp go through the same depth of the tree in the segment scan approach.

Approach

- For every rows non-zero elements, divide them into no more than two components,one component has a multiple of 32 nonzeros, the other component has less than 32 elements we name it as remainder component. The component with a multiple of 32 elements for all rows can be placed first in the work list, row by row.
- For all rows that have a remainder component, further divide the remainder into no more than two components: one has a multiple of 16 non-zero elements, the other component has less than 16 non-zero elements. Now place the 16-element non-zero components into the work list (row by row).
- For remainders of Step 2, we further split every reminder into two components,one has a multiple of 8 non-zero elements, the other has less than 8 non-zero elements. Then we place the 8-element components into the work list.
- In the end, we place the remainders into the work list one by one.
- Now, round robin each 32 size chunk to a warp. Similarly round robin 16 size chunks,8 size chunks,2 size chunks.

Observations:

- Works better than earlier scan algorithm
- Checked with various threshold levels. For CANT matrix threshold level 5,time:109milliseconds; threshold level 6 time:109 milliseconds; With the change in the threshold, I didn't find much change in the running time.

Pros: Minimize the thread divergence **Cons:** Need to sort before the input, and pre process the data.

- The array needs to be row-wise sorted before performing segmented scan.
- There is a thread divergence.
- Since there is a two level of segment scan, many sync thread operations are there, in the algorithm. This requires, other threads, which are not participating in the cumulative some to wait. This is the main reason for the increase in the running time of SPMV-Segment compared to the SPMV-Atomic.

3.3 Results:

Block Size:1024 BlockNum: 8

Matrix Name	Number of NZ	Time_Atomic	Time_Segment	Time_Design
Cant	2,034,917	22	173	107
Cirucit_5M_DC	19,194,193	204	1629	997
Consph	3,046,907	33	260	162
Full_Chip	26,621,990	289	2283	1408
mac_econ_fwd500	1,273,389	14	109	67
mc2depi	2,100,225	22	174	109
pdb1HYS	2,190,591	24	186	116
pwtk	5,926,171	65	504	313
rail4284	11284032	127	964	607
rma10	2374001	26	201	126
scircuit	958,936	10	81	50
shipsec_1	3977139	44	338	211
turon_m	912345	9	77	46
watson_2	1,846,391	19	155	95
webbase_1M	3,105,536	31	255	159

Block Size : 1024 Threads, Block Number : 4

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