SPMV Research on GPU Report CS 415

Tien-Lun Li
Rutgers University

Abstract

GPU has great computational power in parallel programming since it has highly parallel structure, and I take advantage of that to do SPMV.

In project 1, We have seen the atomic version of SPMV runs faster than the other two, which are the segmented scan and my design versions. In this research, I aim to exploit more of GPU architecture and try to make my design version actually runs faster by avoiding three types of performance hazard. The result comes out promising. Most of the matrices give shorter run time to my design version.

1 How

I avoid three types of performing hazard by transforming my data in such a way that grouping them according to the non-zero entries' rows and sort them by the numbers of rows. By doing that, thread divergence can be avoided by grouping similar numbers of rows together to avoid threads diverge at each if statement of segment scan. Memory coalescing can be solved by having the transformed data stored in a consecutive order, which is by creating a new array and copy over instead of have a function transforming the data and scatter read all over the place in the original arrays. There is no memory bank conflicts since each thread reads the shared memory according to its own thread id and those reads do not conflict with each other.

2 Data Transformation

I transform my data of rows, columns and values. First, I read through all non-zero entries; I have a for loop that count the occurrence of each row number index and store each non-zero entries in a array of linked lists that has the size of number of rows of the whole matrix. Then, I sort the number of rows according to the number of occurrence of each row in the matrix. Then, I read the array of linked lists to get the index of each values of the original data and put them into a new rIndex, cIndex and value arrays. All data are now sorted in rows and all entries with the same row are now put together. I do it in O(nlogn), which n stands for non-zero entries.

3 Result

Cant
Atomic: 5980 mirco-seconds, 3890 mirco-seconds, 6201 mirco-seconds
Design: 5010 mirco-seconds, 4995 mirco-seconds, 5001 mircoseconds
Circuit5M
Atomic: 40027 mirco-seconds, 49081 mirco-seconds, 38074 mirco-seconds
Design: 47334 mirco-seconds, 46807 mirco-seconds, 47026 mirco-seconds