Proposal of Sparse Matrix-vector multiplication on GPU

By Weida Zhu and Feng Mi

1.1Introduction of sparse matrix

In numerical analysis, a sparse matrix is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense. The fraction of non-zero elements over the total number of elements (i.e., that can fit into the matrix, say a matrix of dimension of m x n can accommodate m x n total number of elements) in a matrix is called the sparsity (density).

1.2Propblem in calculating sparse matrix multiplication

When storing and manipulating sparse matrices on a computer, operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices as processing and memory are wasted on the zeroes. Sparse data is by nature more easily compressed and thus require significantly less storage. Some very large sparse matrices are infeasible to manipulate using standard dense-matrix algorithms.

1.3 The meaning of this project

Large sparse matrices multiplication often appear in **scientific** or **engineering** applications when solving partial differential equations. By this means, if we store and calculate every element of the matrix in computer it may waste much space. So we need find a way to calculate the sparse matrix more efficiently in order to make sparse matrix-vector multiplication at high speed.

If we could find a way not only to store but also to calculate the large sparse-vector multiplication very efficiently it would benefit in engineering or scientific domain to save time.

1.4 New opportunity

The massive parallelism of graphics processing units (GPUs) offers tremendous performance in many high-performance computing applications. Modern NVIDIA GPUs are throughput-oriented many core processors that offer very high peak computational throughput. Realizing this potential requires exposing large amounts of fine-grained parallelism and structuring computations to exhibit sufficient regularity of execution paths and memory access patterns.

2.Bottlenecks in using gpu

Because most of cpus only have several cores to do the calculation it is not very good in parallel programming. We use Nvidia GPU and cuda programming as our tool to solve the problem. Though gpu is very good at parallel programming , we want to find a relative more effective way to solve the sparse matrix multiplication utilizing the best use of gpu. So there are three bottlenecks in matrix multiplication in gpu.

1.Memory bandwidth bound: GDDR/L2/TEX/L1/Shared Memory bandwidth etc.

2. Instruction throughput bound: single/double-precision/LDST/SFU throughput etc.

3. Latency bound: No enough warp to switch in while waiting.

3.1Before we do the project, the approach we guess can help us solve the problem

(1.use profiler to help us locate the bottlenecks)

2. find a certain algorithm to compact data(Compressed Sparse Row)

3.Take use of some on chip cache: TEX (make the granularity of the parallel program more *thinner.)*

4. Faster Instructions(Intrinsic Instructions; Shift vs divide; Float vs Integer vs double).

Other: tuning memory access pattern to reduce redundant memory access(coalesced access) or the Heterogeneous method.

3.2Idea to solve the problem:

(We can utilize the profiler(Visual Profiler, NVPROF, Command Line Profiler) to locate the bottlenecks in the program. By this means, we can analyze where to optimize the sparse matrix- vector multiplication program.)

The first step is that we need find a certain algorithm to compact data and we find that "compressed Sparse Row".

The second step is to make the granularity of the parallel program thinner. We want to take advantage of the texture memory. We allocate each row of the matrix to a warp (32 treads). And because the speed of texture memory is much faster than the global memory, It will certainly decrease the time.

The third is that we can utilize the faster instructions. For instance, when we do not calculate the number very precisely, we can consider using the faster instructions such as using integer instead of float or using float instead of double.

There may be some another factors like coalesced access or the heterogeneous method which may influence the outcome.

3.3 Planning process and timeline

We have one month to solve the problem.

First week, we plan to make an appropriate algorithm to run the matrix multiplication in gpu and analyze the advantage which is compared with the basic algorithm in cpu.

Second week, we plan to find a sparse matrix multiplication problem on the internet then apply the algorithm into the calculation and translate it into cuda code.

Third week ,we plan to optimize the code based on that three aspects which is mentioned before.

Fourth week, we will do the evaluation and finish the report.

How to evaluate our outcome

We will show the measurement standard of the optimization of our program. We use the control variate method to calculate each program runtime in different steps. Here is the form we need to finish when we do our project.

Name of Matrix	Time spend on cpu	Time spend on gpu	Time spend on transimission(PCIE BUS)	The ratio of speed up
Matrix1				
Matrix2				
Matrix3				
Matrix4				
Matrix5				

The ratio of speeding up =(time spend on gpu + time spend on transmission on PCIE BUS)/time spend on cpu .

We expect to get almost 30% speeding up ratio at the end.