# An Implementation of Sparse Matrix Into Vector Multiplication Using Vendor Supplied Libraries


Mr. Pawaskar Vinit Vijay Vasundhara
230340141015

Mr. Prathmesh Rajguru
230340141021

Mr. Chaitanya Madame
230340141006

Mr. Tejas Bagul
230340141005

Mrs. Harshada Barve
230340141010


PG - DHPCAP
[March 2023 - August 2023]


**Centre of Development of Advanced Computing**
**ACTS - Pune**

Supervisor : Dr. VCV Rao


# Contents

# 1 Abstract

Sparse matrix-vector multiplication (SpMV) is one of the most important computational kernels in various computational intensive areas. SpMV is the multiplication of sparse matrix $A$ with vector $x$ given by $Ax = b$. Here we have performed a comparison study between (I) naive approach, (II) Intel oneAPI library calls and (III) CUDA enabled NVIDIA libraries. The study was carried out on matrices with single and double prescision random numbers with increasing matrix sizes. Compressed Sparse Row (CSR) format was used for sparse matrix storage. We found that on CPU side Intel MKL's SpMV library calls gives best performance. On GPU side slight difference is observed. CSR kernel and cuSPARSE shows similar performance on both single and double precision.

# 2 Introduction

Sparse matrices are heavily used in areas such as numerical analysis [Hashimoto (1970)], mathematics [Marimont (1960)], graph theory[Harary (2018)], engineering[Van Ness & Griffin (1961)], physics[Pekeris (1946)] etc. Sparsity of a matrix is defined as the percentage of the number of zeros to the total number of entries in the matrix. As the omnipresence of the sparse matrices, methods for efficiently manipulating them are often critical to the performance of many applications. As opposed to dense matrix operations, sparse matrix operations are much less regular in their access patterns and consequently are generally limited by bandwidth.

Sparse matrix-vector multiplication (SpMV) operations are of particular importance in various computational domains. Generally SpMV operations are of the form $Ax = b$, where A is a sparse non-singular matrix of order $N$ and $x$ and $b$ are column vectors of length $N$. They represent the dominant cost in many iterative methods for solving large-scale linear systems and eigenvalue problems that arise in a wide variety of scientific and engineering applications. Many algorithms use SpMV iteratively for their computations. For example, the conjugate gradient method [Saad (2003)] uses SpMV to solve a system of linear equations.

As sparse matrices are mostly zeros, it is very inefficient to operate on them directly.

In general, alterative storage formats of sparse matrix can largely reduce the memory cost by taking advantage of the sparsity nature [Bell & Garland (2008)]. These storage formats include *Diagonal* format (DIA), *ELLPACK* format (ELL), *Coordinate* format (COO), *Compressed Sparse Row* format (CSR), *Hybrid* format (HYB), and *Packet* format (PKT). However, the computational resources are often underutilized in these formats [Vuduc et al. (2010)], some of which could be only applied to a limited range of matrices. CSR is the most common and flexible sparse matrix storage format which is used here.

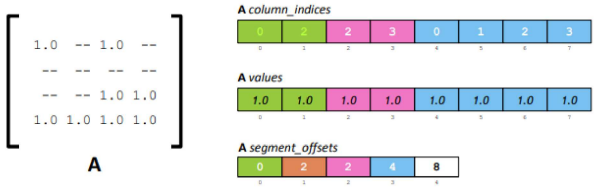


Figure 1: The three-array CSR format [Merrill & Garland (2016)].

Optimizing and tuning of SpMV kernel has been extensively studied in recent years. Bell & Garland (2008) implimented several compressed storage formats and algorithms for SpMV on the CUDA plaform. Hugues & Petiton (2010) presented the *Compressed Sparse Column* (CSC). In opposite of te CSR, it instead compresses each column of a matrix. Cevahir et al. (2009) proposed an efficient SpMV algorithm on multiple GPUs, demonstrating a performance advantage over the scaler kernel of CSR format.

Here in this project, we are analysing performance of various ways of SpMV on structured (tri-diagonal) and unstructured matrix. For comparison, we are measuring the time taken to compute the output $b$ vector.

# 3 Software & Hardware Requirements

For this project we were using PARAM Shavak for all the computations. As it is based on Intel processors and Nvidia's GPU cards, we used their respective compilers and supplied libraries. Below is the summary of software and hardware used:

## 3.1 Hardware

**Model name : Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz**

**Core Specification:-**

Total Cores :- 16

Total Threads :- 32

Max Turbo Frequency :- 3.90 GHz

Processor Base Frequency :- 2.90 GHz

Cache :- 22 MB

**Memory Specifications:-**

Max Memory Size (dependent on memory type) :- 64 GB

Memory Types :- DDR4-2933

Maximum Memory Speed :- 2933 MHz

**GPU Specifications:-**

GPU Memory :- 32GB HBM2

NVIDIA CUDA Cores :- 5,120

NVIDIA Tensor Cores :- 640

NVIDIA NVLink Bandwidth :- 200 GB/s

## 3.2 Software

**Operating System:** CentOS Linux 7

**Programming Languages:** C

**Compiler and Build Tools:** Intel C Compiler, nvcc V11.5.50

**Libraries and Packages:**

Intel oneAPI Toolkit: 23.0.0

NVIDIA CUDA Toolkit and Libraries: 11.5.0

# 4 Data and Methodology

As the core theme of this study is to compare performances of SpMV, we will be focusing only on the compute time of SpMV and ignore the rest. Here, we are taking two kinds of matrices i.e. structured and unstructured matrices. In case of structured matrix type we are generating tri-diagonal matrices of varying sizes. Here sparsity of matrix is not constant. In second case, we are generating a matrix where each row is 95% sparse which is stored in *Compressed Sparse Row* (CSR) format.

## 4.1 Sparse Storage Format

Opting for the suitable format hinges on the specific operations intended for the sparse matrix. For sparse matrix-vector multiplication, both CSR and CSC are appropriate choices, contingent upon the traversal pattern of the algorithm. Other formats, such as the Coordinate List (COO) format or the Block Compressed Sparse Row (BCSR) format, may be better suited for specific applications or algorithms.

In our project, the plan involves utilizing the CSR format for generating sparse matrices.This format is particularly well-suited for operations that involve iterating through the matrix row-wise, such as matrix-vector multiplication. Since the row pointers provide quick access to the start of each row, retrieving the non-zero elements of a row is efficient. The space complexity of the CSR format is influenced by the number of non-zero elements in the sparse matrix and the matrix's dimensions. The total space complexity of CSR is approximately $O(nnz + n + n)$, where: nnz is the number of non-zero elements and n is the number of rows.

## 4.2 Intel MKL and Libraries

The Intel Math Kernel Library (Intel MKL) is a collection of highly optimized mathematical functions designed to accelerate scientific and engineering computations on Intel architecture. It provides a set of routines that take advantage of the latest

processor features to improve performance in various domains, including linear algebra, fast Fourier transforms, statistical analysis, and more.

Sparse matrix-vector multiplication (SpMV) is a fundamental operation in scientific and engineering applications, especially in simulations and data analysis. SpMV involves multiplying a sparse matrix (a matrix with mostly zero elements) with a dense vector, resulting in a sparse output vector. MKL provides several SpMV multiplication functions optimized for different scenarios. MKL is better for CSR matrix-vector multiplication due to its optimized algorithms, parallelism, memory efficiency, cache optimization, and hardware-specific enhancements. It leverages modern hardware features to perform highly efficient and fast SpMV operations on sparse matrices, making it an ideal choice for applications that require high-performance numerical computations.

## 4.3 CUDA and cuSPARSE

CUDA is a parallel computing platform and API developed by NVIDIA, aimed at harnessing GPU power for general-purpose computing tasks. CUDA programs on the host (CPU) invoke a kernel grid which runs on the device (GPU). CUDA enables developers to write parallel code for GPUs, significantly boosting performance in various applications. The same parallel kernel is executed by many threads.

cuSPARSE is a CUDA library focused on accelerating sparse matrix operations on GPUs. It supports various sparse matrix formats, offers optimized linear algebra operations, and seamlessly integrates with CUDA, making it a critical tool in scientific computing and machine learning. Guarantee external device memory for internal operations.Provide extensive consistency checks across input matrices and vectors. This includes the validation of sizes, data types, layout, allowed operations, etc.Provide constant descriptors for vector and matrix inputs to support const-safe interface and guarantee that the APIs do not modify their inputs.The cuSPARSE library functions are executed asynchronously with respect to the host and may return control to the application on the host before the result is ready. cuSPARSE SpMV (Sparse Matrix-Vector Multiplication) is a core operation in cuSPARSE, efficiently multiplying sparse matrices by dense vectors on GPUs. It excels in parallel execution, is optimized for GPU architectures, and finds applications in solving linear systems, graph analytics, and machine learning.

## 4.4 SpMV Multiplication

SpMV is performed on structured and unstructured matrices. In both methods we have analysed performance for single and double precision non-zero entries. We have varied matrix sizes from [32 X 32] to [32768 X 32768]. Performance on CPU as well as GPU is measured using gettimeofday() routine.

### 4.4.1 Structured Matrix - SpMV multiplication

For this, we have created tri-diagonal matrix with three primary diagonals being non-zero. We have performed SpMV multiplication in four different ways on CPU as well as GPU. I] naive matrix-vector algorithm on sparse matrix, II] vendor supplied library for matrix-vector multiplication (cblas_dgemv, cblas_sgemv, cublasSgemv, cublas-Dgemv), III] converting sparse matrix into CSR format and convenntional SpMV on it, IV] using vendor supplied sparse SpMV routines (mkl_sparse_d_mv, mkl_sparse_s_mv, cusparseSpmv).

We have performed three studies here, I] SpMV multiplication on CPU with single and double precision, II] SpMV multiplication on GPU with single and double precision and III] overall performance of SpMV multiplication algorithms.

### 4.4.2 Unstructured Matrix - SpMV multiplication

Here we have created a CSR format of sparse matrix with random entries. We have maintained sparsity of each row to be 95%. Here we are not creating a complete matrix but directly creating a CSR format.

We have performed two kinds of studies here. I] CSR multiplication on unstructured sparse matrix, II] vendor supplied sparse SpMV library routines (mkl_sparse_d_mv, mkl_sparse_s_mv, cusparseSpmv).

# 5 Observations

We are considering only the compute time of SpMV multiplication. Execution time is measured in seconds using gettimeofday() routine. We have mentioned below different comparisons of SpMV multiplication on CPU and GPU.

## 5.1 Intel MKL Library Routines

In this comparison we observed, in case of Single Precision Structured matrix SpMV Intel MKL's mkl_sparse_s_mv outperforms cblas_sgemv, csr and naive multiplications (Figure2).

| | Execution Time in Seconds | | | |
|---|---|---|---|---|
| N | naive multiplication | csr multiplication | cblas_sgemv | sparse_s_mv |
| 32 | 0.000006 | 0.000001 | 0.000585 | 0.004614 |
| 64 | 0.000016 | 0.000001 | 0.000633 | 0.004516 |
| 128 | 0.000088 | 0.000003 | 0.002447 | 0.002736 |
| 256 | 0.000245 | 0.000004 | 0.004137 | 0.001149 |
| 512 | 0.000916 | 0.000006 | 0.005518 | 0.000143 |
| 1024 | 0.002598 | 0.000011 | 0.002984 | 0.001499 |
| 2048 | 0.010024 | 0.000022 | 0.003626 | 0.000134 |
| 4096 | 0.040287 | 0.000045 | 0.004432 | 0.000071 |
| 8192 | 0.163823 | 0.000098 | 0.008048 | 0.000083 |
| 16384 | 0.660574 | 0.000209 | 0.021281 | 0.000075 |
| 32768 | 2.759746 | 0.000480 | 0.079457 | 0.000071 |

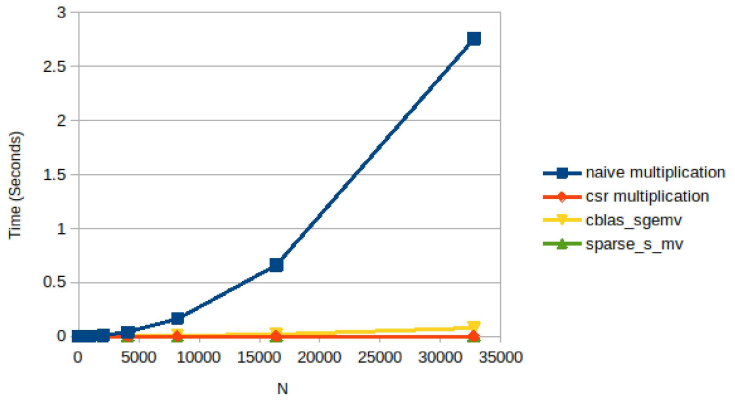


Figure 2: Structured SpMV (Single Precision) multiplication on CPU using Intel MKL.

Same is observed with Double Precision Structured SpMV multiplication (Figure3).

Using CSR format of unstructured matrix, we observed that Intel MKL's sparse SpMV routine gives approximately similar performance as conventional CSR multiplication at this scale.(Figure4).

Overall we observed that the Intel MKL's sparse SpMV routine is best among all other implemented routines here.

| | Execution Time in Seconds | | | |
|---|---|---|---|---|
| N | naive multiplication | csr multiplication | cblas_Dgemv | sparse_d_mv |
| 32 | 0.000004 | 0.000001 | 0.000476 | 0.003102 |
| 64 | 0.000017 | 0.000001 | 0.000511 | 0.003148 |
| 128 | 0.000060 | 0.000002 | 0.002412 | 0.001197 |
| 256 | 0.000247 | 0.000004 | 0.003098 | 0.000604 |
| 512 | 0.000730 | 0.000009 | 0.005216 | 0.000098 |
| 1024 | 0.002542 | 0.000011 | 0.003334 | 0.000091 |
| 2048 | 0.010311 | 0.000023 | 0.003214 | 0.000116 |
| 4096 | 0.041392 | 0.000045 | 0.006469 | 0.000128 |
| 8192 | 0.164940 | 0.000090 | 0.013009 | 0.001011 |
| 16384 | 0.780324 | 0.000225 | 0.041014 | 0.000055 |
| 32768 | 2.952222 | 0.000450 | 0.162442 | 0.000059 |

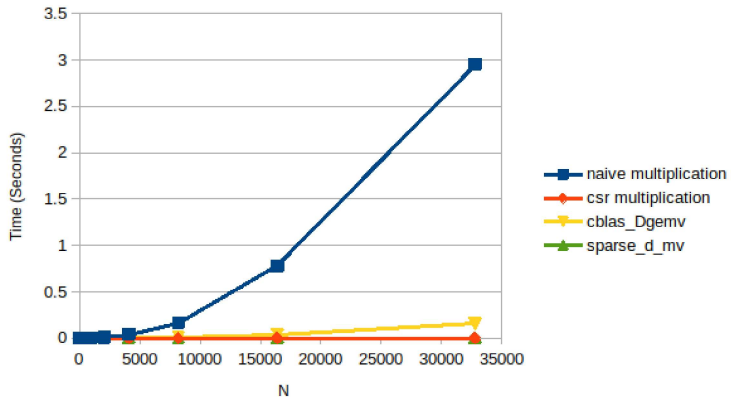


Figure 3: Structured SpMV (Double Precision) multiplication on CPU using Intel MKL.

## 5.2 CUDA Library Routines

We have done same analysis on GPU using CUDA and its various libraries. In case of Structured Single precision matrix, we have observed that CSR implementaion is better than CUDA libraries and naive implementaion of matrix-vector multiplication performs worst (Figure5).

Same is observed with double precision also, that is CSR implementation outperforms all other used implementations (Figure6).

With using CSR format as unstructured matrix for single and double precision, CSR multiplication as well as CUDA sparse library are showing similar performance (Figure7).

Overall we observed that the CSR multiplication on CUDA gives better performance both on structured and unstructured matrices (Single & Double Precision).

| | Single Precision | | Double Precision | |
|---|---|---|---|---|
| N | csr multiplication_1 | sparse_s_mv | csr multiplication_2 | sparse_d_mv |
| 32 | 0.000000 | 0.003048 | 0.000000 | 0.003705 |
| 64 | 0.000001 | 0.005200 | 0.000001 | 0.004677 |
| 128 | 0.000004 | 0.002975 | 0.000005 | 0.004585 |
| 256 | 0.000013 | 0.003078 | 0.000013 | 0.004579 |
| 512 | 0.000051 | 0.004682 | 0.000074 | 0.004037 |
| 1024 | 0.000198 | 0.003981 | 0.000198 | 0.003375 |
| 2048 | 0.000560 | 0.003819 | 0.000609 | 0.004452 |
| 4096 | 0.002185 | 0.006578 | 0.002300 | 0.005184 |
| 8192 | 0.008908 | 0.011747 | 0.009576 | 0.012729 |
| 16384 | 0.037600 | 0.043340 | 0.039669 | 0.045951 |
| 32768 | 0.158238 | 0.165072 | 0.157598 | 0.169270 |

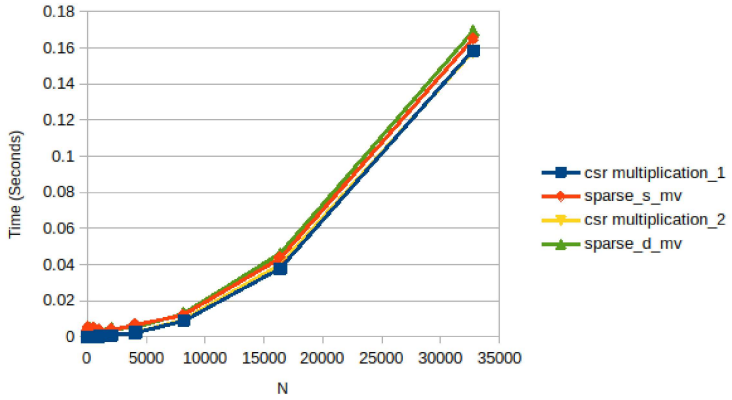


Figure 4: Unstructured SpMV (Single & Double Precision) multiplication on CPU using Intel MKL.

# 6 Discussion

In this project, we analysed the SpMV multiplication using vendor supplied libraries. We ran our codes on CPU as well as GPU. On CPU using Intel MKL's libraries we found that the sparse implementation outperformed all other kinds that we used. In both the cases i.e. structured as well as unstructured matrices this performance is observed. While on GPU we found little deviation from this trend. We found CSR kernel performed best with respect to others on Structured matrix. In case of unstructured SpMV multiplication, both CSR and sparse library performed well. Use of CSR format turned out excellent on GPU. Above points indicates that most of the times supplied libraries perform really well on their specific architectures.

| | Execution Time in Seconds | | | |
|---|---|---|---|---|
| **N** | **naïve Kernel** | **csr kernel** | **cublassSgemv** | **cusparseSpmv** |
| **32** | 0.000010 | 0.000005 | 0.000069 | 0.000012 |
| **64** | 0.000008 | 0.000002 | 0.000075 | 0.000012 |
| **128** | 0.000010 | 0.000003 | 0.000073 | 0.000012 |
| **256** | 0.000009 | 0.000003 | 0.000056 | 0.000010 |
| **512** | 0.000090 | 0.000003 | 0.000075 | 0.000012 |
| **1024** | 0.000011 | 0.000004 | 0.000073 | 0.000012 |
| **2048** | 0.000012 | 0.000004 | 0.000089 | 0.000013 |
| **4096** | 0.000008 | 0.000003 | 0.000087 | 0.000012 |
| **8192** | 0.000009 | 0.000005 | 0.000083 | 0.000013 |
| **16384** | 0.000011 | 0.000005 | 0.000090 | 0.000011 |
| **32768** | 0.000010 | 0.000006 | 0.000080 | 0.000011 |

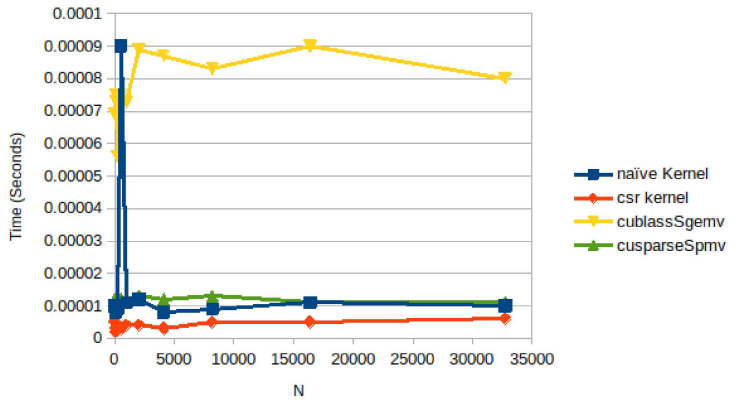


Figure 5: Structured SpMV (Single Precision) multiplication on GPU using CUDA.

# Bibliography

Bell N., Garland M., 2008, Technical report, Efficient sparse matrix-vector multiplication on CUDA. Nvidia Technical Report NVR-2008-004, Nvidia Corporation

Cevahir A., Nukada A., Matsuoka S., 2009, in Computational Science–ICCS 2009: 9th International Conference Baton Rouge, LA, USA, May 25-27, 2009 Proceedings, Part I 9. pp 893–903

Harary F., 2018, Graph Theory (on Demand Printing Of 02787). CRC Press

Hashimoto M., 1970, Journal of the ACM (JACM), 17, 629

Hugues M. R., Petiton S. G., 2010, in 2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC). pp 122–129

Marimont R. B., 1960, Siam Review, 2, 259

Merrill D., Garland M., 2016, Acm Sigplan Notices, 51, 1

Pekeris C., 1946, Journal of Applied Physics, 17, 678

| | Execution Time in Seconds | | | |
|---|---|---|---|---|
| **N** | **naïve kernel** | **csr kernel** | **cublassDgemv** | **cusparseSpmv** |
| **32** | 0.000009 | 0.000002 | 0.000049 | 0.000011 |
| **64** | 0.000011 | 0.000004 | 0.000047 | 0.000012 |
| **128** | 0.000009 | 0.000002 | 0.000045 | 0.000011 |
| **256** | 0.000014 | 0.000006 | 0.000050 | 0.000011 |
| **512** | 0.000011 | 0.000003 | 0.000046 | 0.000010 |
| **1024** | 0.000008 | 0.000004 | 0.000048 | 0.000011 |
| **2048** | 0.000010 | 0.000003 | 0.000059 | 0.000013 |
| **4096** | 0.000009 | 0.000003 | 0.000064 | 0.000012 |
| **8192** | 0.000009 | 0.000005 | 0.000073 | 0.000015 |
| **16384** | 0.000009 | 0.000008 | 0.000066 | 0.000011 |
| **32768** | 0.000011 | 0.000006 | 0.000072 | 0.000011 |

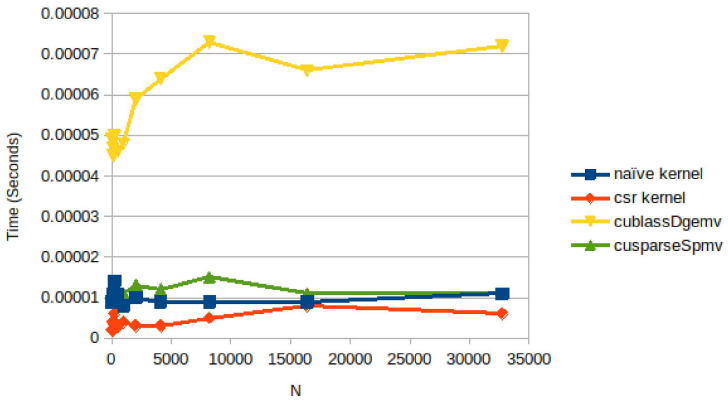


Figure 6: Structured SpMV (Double Precision) multiplication on GPU using CUDA.

Saad Y., 2003, Iterative methods for sparse linear systems. SIAM

Van Ness J. E., Griffin J. H., 1961, Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems, 80, 299

Vuduc R., Chandramowlishwaran A., Choi J., Guney M., Shringarpure A., 2010, in Proceedings of the 2nd USENIX conference on Hot topics in parallelism.

| | Single Precision | | Double Precision | |
|---|---|---|---|---|
| **N** | **csr_kernel_single** | **cusparseSpmv_single** | **csr_mult _double** | **cusparseSpmv_double** |
| **32** | 0.000010 | 0.000013 | 0.000017 | 0.000011 |
| **64** | 0.000009 | 0.000013 | 0.000012 | 0.000014 |
| **128** | 0.000009 | 0.000013 | 0.000009 | 0.000012 |
| **256** | 0.000009 | 0.000011 | 0.000010 | 0.000010 |
| **512** | 0.000009 | 0.000013 | 0.000010 | 0.000011 |
| **1024** | 0.000010 | 0.000013 | 0.000009 | 0.000013 |
| **2048** | 0.000012 | 0.000011 | 0.000011 | 0.000012 |
| **4096** | 0.000010 | 0.000013 | 0.000009 | 0.000010 |
| **8192** | 0.000009 | 0.000011 | 0.000009 | 0.000010 |
| **16384** | 0.000010 | 0.000010 | 0.000009 | 0.000012 |
| **32768** | 0.000010 | 0.000011 | 0.000017 | 0.000012 |

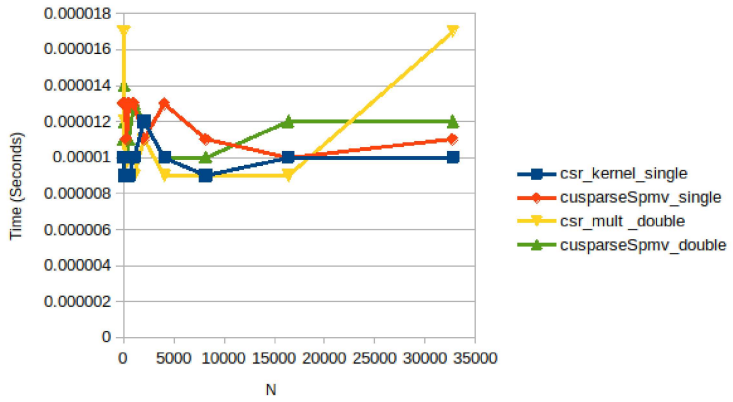


Figure 7: Unstructured SpMV (Single & Double Precision) multiplication on GPU using CUDA.