Techniques in High Performance Computing

Course work 3

Sparse matrix formats on GPUs

Author: George William Bowden

Task 1

0.1 Introduction

Sparse matrices are important to a variety of mathematical problems involving linear systems. Therefore, it is desirable to evaluate them effectively without making countless useless calculations corresponding to the large proportion of zero elements. The assumptions for a sparse matrix is very high dimensionality and a low proportion of zeros (single-digit percentage). The major problem with SParse Matrix Vector products (SPMVs) on GPUs is the none negligible cost of memory operations which is a consequence of ineffective data structures [4]. Because the proportion of non-zero entries is low, the memory cost of useless computations is much higher than those of useful operations. The relative cost of useless operations is large. This means that storing and multiplying data efficiently is important. The speed of sparse matrix-vector products depends heavily on the choice of data structure and the algorithm used to compute the SPVM. Note, the composition of the matrix also plays a role in the size of the speed up. Indeed different multiplication schemes work better on different types of matrices [5].

Here I discuss several data storage schemes and give an example of an algorithm that is favourable to execute using GPUs, such as the Compressed Row Storage (CRS) and then a more effective method called ELLPACK and its variants ELLPACK-R and ELLPACK-T as well as a hybrid technique (HYB).

0.2 CSR

Consider the problem

$$Ax = b$$

where A is a sparse $M \times N$ matrix with N_z non-zero elements and $u, v \in R^N$.

A standard form of sparse matrix data storage for CPU SPMV is Compressed Row Storage (CRS) which uses three arrays to store sparse matrix data for A. First a N_z dimensional array A_d which contains all of the none-zero values of the matrix, an integer array J contains the row indices of each data point which also has length N_z and finally an integer array C array containing the pointer of the start of each row for each data point which has length $N + 1$.

The benefit of the CSR format is the fact that zero padding is needed, meaning that there are no dummy elements in the new data structure. When memory locality is not an issue then CSR are an efficient format for SPMV, such as when one is using a CPU [2]. Unfortunately this is not the case on GPUs since the format is unable to make good use of the massive potential for parallelisation due to lack of memory locality.

We want to use a single GPU thread to compute each row of the vector b, with each block containing all threads that share common information for more sophisticated schemes. The problem with this approach to SPMV with CSR is memory discontinuity, non-coalesced access to the column index and the value array which is very expensive to deal with on a GPU. This issue can be overcome by using a warp to process each row which can prevent uncoalesced access by keeping all relevant elements in the thread block's local memory, but can result is significant thread divergence [8]. Additionally, it is hard to take advantage of the parallelisation available on GPUs because the size of the inner loop which performs the row vector dot product varies from thread to thread which could cause one thread to compute a dense row and negate the speed up. Also consider that for multiplication in CSR there can be many calls to each element the vector x but only one to each of the matrix elements which means that each thread needs to make a significant number of memory calls.

There is a modification to the CSR format called CSR 5, which introduces two additional arrays, the first being a pointer to a tile which contains a subset of the data array, and the second information about the tile's shape. This format can be helpful for blocking schemes and is slightly more efficient on a GPU since each thread needs to make less memory calls. [8]

0.3 ELLPACK

ELLPACK is an alternative storage structure that is designed to allow SPMV operations to be parallelised on a GPU. This format uses two arrays to store the data, the array A_d is the data array with dimension $N \times maxN_z$ where N is the number of rows, and $maxN_z$ is the largest number of non-zero elements across all the rows. The second array J which has the same dimension and is responsible for storing all of the column indices of the data. Since these arrays will not naturally be the same shape, they are padded with zeros, which can cause useless computations for a naive implementation (an issue that CSR avoids). On a GPU, one thread is mapped to one row, and array values for the vector x and the column value J are stored in global memory, so the only expensive memory operations are those to retrieve the column index, the column indexes are stored in row order, so each thread only needs to access the row of the array that stores its column indices [6]. This structural

similarity to a dense matrix makes the ELLPACK format perfect for parallelisation on GPUs [10]. Note that the ELLPACK format is at its best when the number of zeros in each row is similar. This is because the number of useless operations can be reduced or truncated for all threads [6]. An extreme example of a matrix that would be a problem for ELLPACK is one where all rows have one non-zero element except the final row, which is full. Then the matrices A, J will both be full size and the speed up will be completely negated since the same number of rows and columns (and hence elements) need to be iterated over. There are modern approaches that consider the statistics of the matrix to use a hybrid storage format for such eventualities. There are also various small modifications on the ELLPACK format that provide contextual improvements, such as ELLPACK-R and ELLPACK-T.

0.4 ELLPACK-R

The ELLPACK-R format is largely the same as ELLPACK but is slightly optimised. Like the ELLPACK structure matrix data is stored in two arrays, A_d and J defined in exactly the same way, but there is the addition of an additional array denoted rl which contains the number of non-zero elements in each row and has length N [11]. There are a number of advantages outlined in [11], including coalesced global memory access due to the column major ordering used by ELLPACK formats, the lack of need for thread synchronisation and finally a reduction of waiting times due to unbalanced threads between threads of one warp. This final advantage is due to the knowledge of the number of non-zero elements in each row which allows us to assign more threads within a warp to the longer rows.

When this format is used in SPMV with T threads computing each row of the new vector then the new algorithm is called ELLR-T [10].

COO The Coordinate format is a very simple one. Matrix elements are described by three $N \times 1$ arrays. The first, A, contains the data. The second, X, contains the row index and the third, Y, contains the column index. [10] While this method would reduce the effort of multiplying sparse matrices, it is usually less effective that CSR and ELLPACK. The exception is when it is used in conjunction with one of the others in a hybrid method. This approach can be helpful in the situation where there are some (but not too many) rows with many non-zero values. [5]

0.5 Hybrid

The Hybrid HYB format stores the matrix rows in either COO or ELLPACK formats depending on the number of non-zero entries. COO stands for the COOrdinate format, where the data is split into three arrays, similarly to CSR and but instead of a column pointer array we have both the x and y This way rows with a relatively similar number of non-zero entries can be efficiently computed using the ELLPACK method discussed above, and the longer denser rows can be split up across several threads and recombined at the end. One of the issues with this can be thread coherence and additional memory overhead for the threads that are responsible for the COO data. A threshold parameter is used to decide which rows will be stored in each format, if the number of non-zero values exceeds this number then the COO format is used, otherwise ELLPACK is used [8]. In 2017 an in depth analysis of many leading SPMV friendly formats were tested and the HYB format was found to perform the best on average [3].

References

Bell, N., and Garland, M. Efficient sparse matrix-vector multiplication on cuda. Tech. rep., Citeseer, 2008.

Dufrechou, E., Ezzatti, P., and Quintana-Ortí, E. S. Selecting optimal spmv realizations for gpus via machine learning. *The International Journal of High Performance Computing Applications 35*, 3 (2021), 254–267.

Fukaya, T., Ishida, K., Miura, A., Iwashita, T., and Nakashima, H. Accelerating the spmv kernel on standard cpus by exploiting the partially diagonal structures. *CoRR abs/2105.04937* (2021).

Heller, M., and Oberhuber, T. Adaptive row-grouped csr format for storing of sparse matrices on gpu, 2012.

Isupov, K., Babeshko, I., and Krutikov, A. Implementation of multiple precision sparse matrix-vector multiplication on cuda using ellpack format. In *Journal of Physics: Conference Series* (2021), vol. 1828, IOP Publishing, p. 012013.

Mahmoud, M., Hoffmann, M., and Reza, H. Developing a new storage format and a warp-based spmv kernel for configuration interaction sparse matrices on the gpu. *Computation 6*, 3 (2018), 45.

Nisa, I., Siegel, C., Rajam, A. S., Vishnu, A., and Sadayappan, P. Effective machine learning based format selection and performance modeling for spmv on gpus. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2018), pp. 1056–1065.

Niu, Y., Lu, Z., Dong, M., Jin, Z., Liu, W., and Tan, G. Tilespmv: A tiled algorithm for sparse matrix-vector multiplication on gpus. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2021), pp. 68–78.

Vázquez, F., Ortega, G., Fernández, J., and Garzón, E. Improving the performance of the sparse matrix vector product with gpus. In *2010 10th IEEE International Conference on Computer and Information Technology* (2010), pp. 1146–1151.

Wang, Z., and Gu, T. Pellr: A permutated ellpack-r format for spmv on gpus. *Journal of Computer and Communications 8*, 4 (2020), 44–58.