Effective Sparse Matrix Representation for the GPU Architectures

B. Neelima[1] and Prakash S. Raghavendra[2]

[1]National Institute of Technology, Karnataka

reddy_neelima@yahoo.com

[2]National Institute of Technology, Karnataka

srp@nitk.ac.in

ABSTRACT

General purpose computation on graphics processing unit (GPU) is prominent in the high performance computing era of this time. Porting or accelerating the data parallel applications onto GPU gives the default performance improvement because of the increased computational units. Better performances can be seen if application specific fine tuning is done with respect to the architecture under consideration. One such very widely used computation intensive kernel is sparse matrix vector multiplication (SPMV) in sparse matrix based applications. Most of the existing data format representations of sparse matrix are developed with respect to the central processing unit (CPU) or multi cores. This paper gives a new format for sparse matrix representation with respect to graphics processor architecture that can give 2x to 5x performance improvement compared to CSR (compressed row format), 2x to 54x performance improvement with respect to COO (coordinate format) and 3x to 10 x improvement compared to CSR vector format for the class of application that fit for the proposed new format. It also gives 10% to 133% improvements in memory transfer (of only access information of sparse matrix) between CPU and GPU. This paper gives the details of the new format and its requirement with complete experimentation details and results of comparison.

KEYWORDS

GPU, CPU, SPMV, CSR, COO, CSR-vector

1. INTRODUCTION

Graphics processing unit was tricked by the programmer to do general purpose computation than doing only graphics related operations. The motivation behind the development of graphics processor evolution, to general purpose computation processor, is different than that of the CPU evolution, to multi core. Hence data formatting and optimizations designed with respect to CPU and its evolutions have to be tailored to GPU specific architectures. Even though GPU gives better performance of the accelerated applications than CPU and multi core, full utilization of the processor for much better performance is possible by tailor made data formatting and computations with respect to the architecture under consideration. Sparse matrix computations and usage is very large in most of the scientific and engineering applications. In sparse matrix, sparse vector multiplication is of singular importance in wide applications. This paper concentrates on sparse matrix vector multiplication aspect of compute intensive applications and

DOI : 10.5121/ijcsea.2012.2213

through a new format shows the memory transfer and performance improvements than the existing data formats of sparse matrices. The results shown for proposed new data format are applicable to GPU in general but the results are particular to NVIDIA GPU analyzed on GeForce GT 525M.

Optimizing performance on GPU needs creation of thousands of threads, because it uses latency hiding by using thousands of threads and gives high throughput. Few of the existing methods like CSR, use row wise thread creation that cannot use global coalescing feature of GPU and GPU is underutilized if the number of non-zero elements per row is less than 32, the size of a warp. CSR vector is modified version of CSR that benefits from global coalescing by using fragmented reductions. The proposed CSPR (Column only SPaRse format) reduces the sparse matrix vector multiplication to constant time and threads can be launched continuously by parallelizing the outer loop for creating many threads. CSPR can be applied to any sparse matrix in general but better performances are seen for the matrices with large number of rows with minimum number of non-zero values per row and centrally distributed few dense rows as shown in Fig. 3. For such matrices, it can give 2x to 54x performance improvements compared to CSR, COO and CSR vector format. CSPR embeds the row information into column information and uses a single data structure; hence it can also optimize the memory transfer between CPU and GPU. CSPR format uses only one data structure to access the sparse matrix hence it is a good format for the internally bandwidth limited processors like GPU.

The paper is organized as follows. The next section gives the details of GPU architecture in general and CUDA in particular. Section III gives the sparse matrix introduction and its importance in scientific computation along with the introduction to data formats of sparse matrices. Section IV gives related work with respect to data formats and sparse matrices. Section V gives the working set up and introduction to sparse matrices considered for testing the new format. Section VI gives the experimental results and analysis. Section VII gives the conclusions and future work.

2. GPU ARCHITECTURE

GPU is the co-processor on the desktop. It is connected to host environment via peripheral component interconnect (PCI Express 16E) to communicate with the CPU. The GPU used for the experimentation here is NVIDIA Geforce GT 525M, but the format proposed is in general applicable to all types of sparse matrices and all processor architectures including CPU. The proposed format is better suited and gives better performance on latency hiding based throughput oriented processors like GPU for specific class of sparse matrix structure. The third generation NVIDIA GPU has 32 CUDA cores in one SM (Streaming Multiprocessor). It supports double precision floating point operations. NVIDIA GPU has compute unified device architecture that uses the unified pipeline concept and the latest GPU supports up to 30000 co-resident threads at any point of time. GPU uses latency hiding to increase parallelism that is when active threads are running other threads will finish pending loads and become active to execute. It uses single instruction multiple threads concepts (SIMT) and executes the computation in warps that consists of 32 threads [1-3].

GPU has architectural optimizations like hardware multithreading that supports global memory access coalescing for more than half warp(16) access and memory optimizations like using texture cache for read only and reusable data like vector values in sparse computation. Global coalescing is accessing continuous memory locations for continuous threads. In CSR format each thread is assigned to a row. For a 30k row matrix 30k threads are launched in the first iteration

and in the second iteration second element of each row are considered for all 30k rows and the process continues till the largest row finishes. Global coalescing is not used as every iteration accesses one element from each row. In the proposed CSPR format threads are launched per non-zero element and continuous threads access the continuous data and hence the global coalescing (16 threads (half warp) or 32 threads (one warp) access single memory segment) is used [4-5]. GPU texture cache can be used for x vector to be multiplied with the sparse matrix that can be reused from cache and hence the performance improvements. The results shown in this paper are without using the texture cache for x vector.

3. SPARSE MATRIX

Sparse matrix is one in which number of non zero elements are less. Hence sparse matrices are represented using different format to avoid zero multiplications. Sparse matrices will have different variety of sparsity (distribution of non-zero elements in the entire matrix) i.e. the distribution of non zero elements make sparsity based data representation to further optimize the performance of sparse based computations. Sparse based computations also consist of sparse matrix to dense matrix computations, sparse matrix to sparse matrix multiplication and sparse matrix to dense vector multiplication. This paper particularly concentrates on sparse matrix vector multiplication which is of high importance in most of the scientific and engineering applications that needs solving large linear systems (Ax=b) and Eigen value problems (Ax=Yx), where, A is a sparse matrix and x is a dense column vector. As sparse matrices are represented in a new format to remove unnecessary zero computations, accessing sparse matrix elements is not direct. Hence the sparse matrices are memory bound and any new format or new optimization that is specific for the architecture is of great importance.

$$A = \begin{pmatrix} 0 & 4 & 5 & 0 \\ 2 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

CSR:

val = [4 5 2 2 1 3]
vol = [1 2 0 3 2 1]
ptr = [0 2 4 5 6]

COO:

val = [4 5 2 2 1 3]
col = [1 2 0 3 2 1]
ptr = [0 0 1 1 2 3]

CSPR:

val = [4 5 2 2 1 3]
ind = [1 2 4 7 10 13]

Fig 1: Sample data format representation of sparse matrix A in CSR, COO and CSPR formats

There are different standard formats like DIA (diagonal), ELL, CSR and COO explained here in brief to give a comparison for the new format proposed. DIA is structure specific data format representation that is suitable for the matrices that have non-zero elements spread across the diagonals of the matrix. DIA format uses two data structures to represent the sparse matrix. One data structure is used to store the data of the size equal to the number of rows multiplied by the number of diagonals that have non-zero elements. Another data structure is to store indices of

diagonals of size equal to number of diagonals. ELL or ELLPACK format that is applicable to the matrices with uniform row lengths. It uses two data structures to store data and indices of the size equal to the number of rows multiplied by max number of elements per row.

CSR (compressed row format) and COO (coordinate format) are applicable to unstructured matrices that have non uniform row lengths in specific but they are more general data format representations of the sparse matrix. CSR uses three data structures, one to represent data, second to represent column index, both of size equal to number of non-zero elements and third data structure is used to store pointer to the row of size equal to number of rows. COO also uses three data structures, same as CSR except the third one is direct row representation of size equal to number of non-zero elements. If DIA and CSR data formats are compared for more structured matrices then DIA will give better performance, because CSR is more general format and DIA is more structure specific and give better performance than the general format. Bell and Garland give a detailed analysis of the data formats with respect to GPU in [6].

Fig. 1 gives the representation of sparse matrix 'A' in CSR, COO and CSPR formats considered in this paper. The proposed data format, CSPR, concentrates on unstructured matrices with non-uniform row lengths. CSPR use two data structures, one to represent the data and the other to represent the column and row indices, both of size equal to number of non-zero elements. CSPR embeds the row information into the column information and hence the reduction in the data structure. This introduces extra computation in extracting the embedded data while performing the sparse matrix vector computation. But in a throughput oriented processor like GPU where 1000s of threads run, this computation will not affect the performance. Hence this format is suitable for computation intensive processor like GPU and hence the performance and memory benefit also. The results shown are considering the formats of unstructured sparse matrix like COO, CSR.

Fig. 2 gives the SPMV implementation algorithm of CSR and CSPR format. As CSPR is using the single data structure that embeds row and column information, it needs extra computation to extract the same in SPMV computation. These extra computations introduced into SPMV are not considered for performance evaluation as they are integer operations. But if these two integer operations are considered for only performance evaluation, then CSPR gives much better performance than that shown in this paper. CSPR needs threads to be synchronized for the row-wise computation values.

```
// Basic SPMV implementation of CSR and CSPR
// Ax = b where A is in sparse, n is size of A
// out to store row wise sum, sout single value multiplication
CSR:   for ( i = 0; i < m; i++)
       {
       for ( k = ptr [ i ]; k < ptr [ i + 1] ; ++k )
       out + = val [ k ] * x [ col [ k ] ] ;
       }
CSPR: for (i = 0; i < m; i ++)
       {
       row  = ind [ i ] / n;
       col  = ind [ i ] % n;
       sout  = val [ row ] * x [ col ]
       atomicadd (out+row, out);
       }
```

Fig. 2: CSR and CSPR SPMV implementations based on the formats given in Fig. 1

The process of synchronization is hidden by the latency hiding mechanism of the GPU and gives higher performance for matrices with large number of rows. It gives comparatively equal or better performance for matrices with very less number of rows that are highly dense; because of the synchronization process takes time for the larger row. Any data format specific to one class of sparse matrix gives best performance for that class and regular performance for the other class of matrices like DIA and ELL format. Bell and Garland [6] have proposed new HYB format that improves performance for the matrices that can take best of both ELL and COO format. HYB gives high performance improvement for those sparse matrices that fit into ELL and COO combination. CSPR is not suited for the structured matrices that can be well represented using ELL and DIA and hence performance comparison cannot be done.

Table 1 represents the data structures required and their sizes for the different formats discussed above. The paper considers square matrices only. If M is the size of the matrix, total number of elements are M*M, including zeros. N is the number of non- zero elements in the given matrix. R is the structure variable representation for the structured matrices like R is number of diagonals with non-zero elements in diagonal format or R is the maximum elements in a row for uniform row lengths in ELL etc. CSPR method reduces the computation time complexity to constant time compared to CSR format, giving abundant data parallelism and memory usage is less and optimizes memory transfer from CPU to GPU than any of the existing methods.

Sparse Data Format	# of Data Structures	Size of Data Structure1	Size of Data Structure2	Size of Data Structure3
DIA	2	M*R	R	--
ELL	2	M*R	M*R	--
CSR	3	N	N	M
COO	3	N	N	N
CSPR	2	N	N	--

Table 1: Data structure requirements of different data formats of SPMV

4. RELATED WORK

The initial work related to improving application performance that have sparse based operations has started with deriving different representations like CSR, ELLPACK, COO, DIA etc., instead of loading the entire matrix on to the memory and do zero computations [7-8]. Later these formats have been optimized with respect to memory systems and different architectures and combination of different formats have been derived to get the maximum performance. Formats like blocked CSR uses memory tiling to improve performance of the applications. Most optimization and parallelization methods are initially derived for the dense matrix and the same is automatically used for the sparse matrix also. For example, blocked or tiled access of dense matrix, when used for sparse matrix, may not be effective as the structure of sparse matrix is different from dense matrix [9-15].

Vuduc, et al. [16-18] has given list of optimizations that can be done with respect to sparse matrices. Then with the advent of multi core era, there is a need to optimize the sparse computations with these new architectures. William, s., et al. [19] show new ways of

optimizations required with respect to new architectures. William, s., et al. has given new optimizations for the multi core architectures and shown huge performance improvements in the applications. Their work has not considered GPUs.

Bell and Garland [20-25] show the implementation of SPMV on GPU and give optimizations to make these computations more effective. Their work has given a new data format from the combination of existing standard formats. Their new format name HYB is a combination of ELL and COO. They did not considered restructuring of the matrix to give another new format. Their work also does not consider the memory transfer between CPU and GPU. They defense their statement by saying that the data structures can be created on the device. If we are using the true data and not creating the data on the device, entire data including the zero and non zero elements has to be transferred on to the device to create the desired data structure. CSPR is designed with respect to GPU architecture, to reduce memory transfer between CPU and GPU and also reduce the memory requirement in the internal GPU architecture. GPU computation power is abundant, so a format that can use less memory and if required with extra computation can give better performance on GPU processor. One such format is CSPR. CSPR work can be considered as the extension of the work by Bell and Garland [6] (shows performance improvement with a new format HYB) and CSPR results show that if GPU specific formats are designed, they can give better improvements in the performance even though applicable to some class of sparse structures. CSPR can be further optimized by considering data layout at the fine grain level and computation mapping at the coarse grain level for the given class of sparse matrix structure and GPU architecture.

5. EXPERIMENTAL SETUP

The data format algorithm implementations are tested on Intel corei7-2630QM CPU with NVIDIA GeForce GT 525M with 1GB-DDR3 memory. The sparse matrices considered here are taken from sparse matrix collection of University of Florida [26]. The matrices selected are same as used by William, s., et al. and Bell and Garland [6, 19]. These set of matrices are taken only because they represent the real data set and results will be more genuine than the synthetic matrices. Table 2 represents the general characteristics of the selected matrices. Fig. 3 to Fig. 5 shows the undirected or bipartite graph representation of the selected matrices. These figures are given to differentiate between the structures of sparse matrices and based on that performance analysis for the new format is explained. The algorithm works with any GPU in general but the implementation is done with respect to CUDA architecture v3.2. The results shown are for single precision and without using the texture cache for the x vector. CSPR can also use the texture cache for index, as every value is accessed twice depending on the optimization possible.

Matrix	Rows	Columns	NNZ	NNZ/Rows
Pdb1HYS	36,417	36,417	4,344,765	199.3
consph	83,334	83,334	6,010,480	72.1
cant	62,451	62,451	4,007,383	64.1
pwtk	217,918	217,918	11,634,424	53.3
rma10	46,835	46,835	2,374,001	50.6
shipsec1	140,874	140,874	7,813,404	55.4
mac_econ	206,500	206,500	1,273,389	6.1
mc2depi	525,825	525,825	2,100,225	3.9
cop20k	121,192	121,192	2,624,331	21.6
scircuit	170,998	170,998	958,936	5.6
webbase	1,000,005	1,000,005	3,105,536	3.1
rail2428	4,284	1,092,610	11,279,748	2632.9

Table 2: Characteristics of the matrices used in the evaluation of CSPR

Fig. 3 represents the graph structure of a sparse structure that has few rows very dense and all other rows are medium dense. CSPR needs synchronization of row values. In these types of matrices that have large row computations, synchronization overhead is overcome by latency handling mechanism and gives the best performance than CSR and COO.

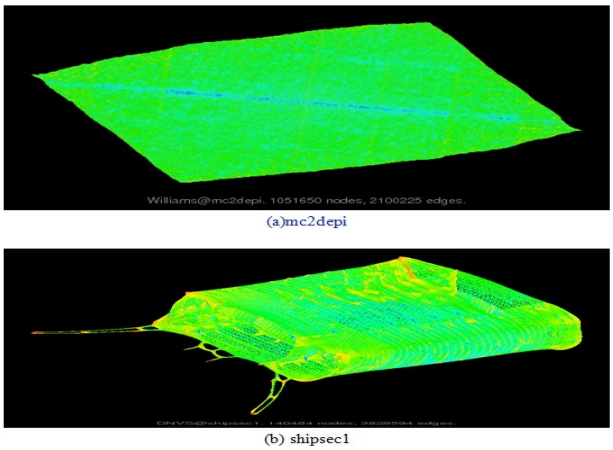


Fig. 3: Graph representation of (a) mc2depi and (b) shipsec1-best suited sparse structure for CSPR

Fig. 4 shows the graph structure of consph and cop20k_A that have dense like matrix structure. CSPR gives good performance than CSR and COO in this case also but percentage of variation in performance is less than the Fig. 3 type graphs. Because of the little synchronization overhead involved for the last few rows when there is no computation.

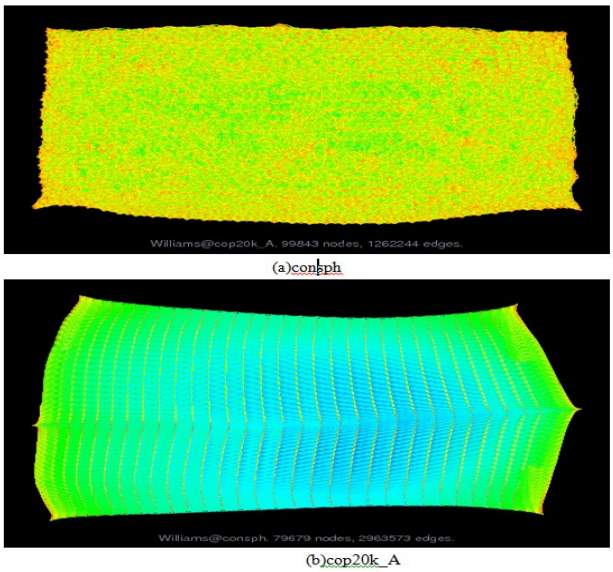
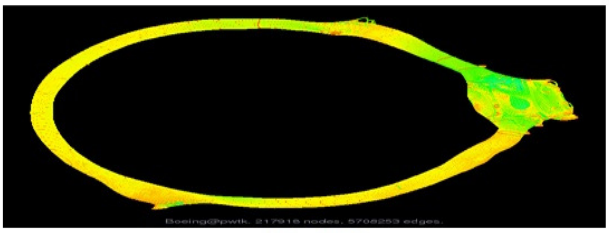


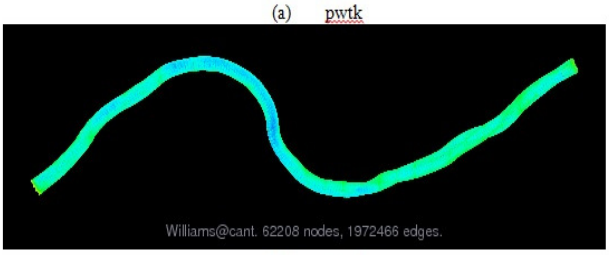
Fig. 4: Graph representation of (a) consph and (b) cop20k_A that have little synchronization overhead for performance lag

Fig. 5 shows the graphs of another sparse matrix structure represented by matrices pwtk and cant. These matrices have many rows with very less dense values. Here CSPR gives performance almost equal to COO or CSR (scalar) because computation time is dominated by synchronization time.

The implementations do not consider any GPU specific optimizations or global matrix transformations like transpose etc. Implementations for COO or CSPR do not use parallel reduction or segmented scan for performance improvements as suggested by Bell and Garland [6]. To create parallelization for multiple threads CSPR implementation uses an explicit parallel loop instead of the CUDA idioms. Hardware managed features like global coalescing and execution divergence are handled accordingly by the hardware. CSPR embeds additional information into the existing column indices because of which memory alignment usage is required for some large matrices. Results with memory alignment implementations are not shown in this paper and relative values are used to show the results and analysis. This paper do not use persistent or block oriented programming as used by Bell and Garland [6].



(a) pwtk



(b) cant

Fig. 5: Graph representation of (a) pwtk and (b) cant that have non dense rows and give high synchronization overhead

6. RESULTS AND ANALYSIS

The results and analysis are done and comparisons are given for the proposed CSPR method to CSR (scalar), CSR (vector) and COO data formats. As CSPR is also more generalized format and gives better performance for more unstructured matrices with small dense rows and large less dense rows. Comparisons or results are not shown with respect to DIA(diagonal) or ELL format that are tailor made for more structured matrices. The results are discussed with respect to three aspects of evaluation for the four data formatting methods of sparse matrix, namely CSR (scalar), COO, CSR (vector) and the proposed method CSPR. The analysis is done with respect to number of floating point operations per second. The performance shown are not the maximum computation capability of the device as GPU specific optimizations are not used to the fullest in the current implementation. All the four algorithms implemented are compared for the performance evaluation and the analysis is valid as they use the same target platform and same programming environment. Hence most of the results are comparative. Some of the results of the

matrices are not shown to avoid much deviation of the graphs and project the other prominent results.

The performance evaluation is done by taking number of non-zero elements of the matrix multiplied by two, for the two floating point operations, divided by the total average time of execution taken for 500 trials. CSPR includes extra computation in terms of extracting the access information which is embedded into single data structure. If we consider all these integer operations as single floating point operation, then the performance improvement is much higher. But as these are introduced computations but not the actual sparse computation, they are not considered in the results shown here.

In general, the performance of the COO matrix is almost constant irrespective of the matrix structure. CSR scalar gives better performance when the number of elements per row is high, i.e for matrices with highly dense rows with less number of rows. CSR (vector) gives better performance for very large number of non-zero values per row.

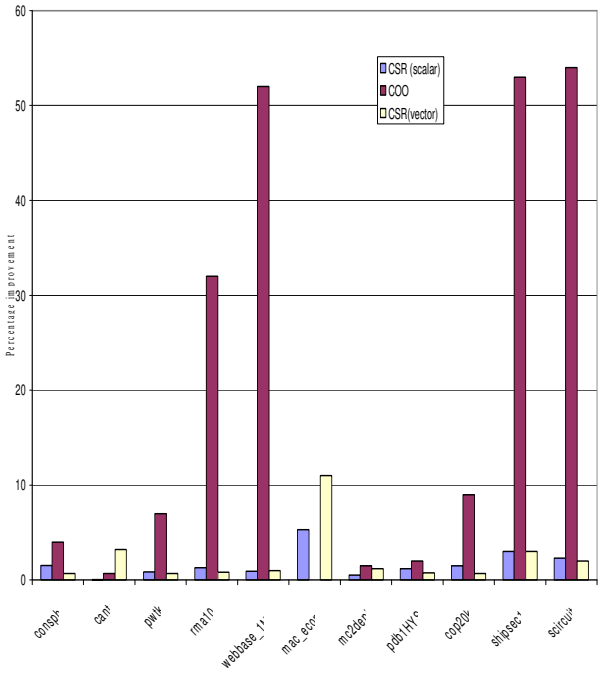


Fig. 6: Percentage improvement of performance of CSPR compared with the three formats under consideration

CSPR gives very high performance when matrices have very large number of rows with less non zero values per row and very few dense rows dominated in the center part of the matrix. When middle rows are dense, by the time dense row computation finishes the previous rows synchronization will also be done. The performance of CSPR is very high than all the other methods considered here for such matrices. If the number of non-zero values per row is medium then the performance is still good. But for highly dense large rows, the performance decreases than the best suited because of synchronization effect of last few rows. If the number of non-zero elements per row is very less, irrespective of whether number of rows is large or small, the performance decreases because computation time is dominated by synchronization time. Hence

CSPR can be considered for high performance gains for matrices that are unstructured, have very large number of rows with very minimum zero values per row and very few dense rows. The details of performance variation are shown in Fig. 7.

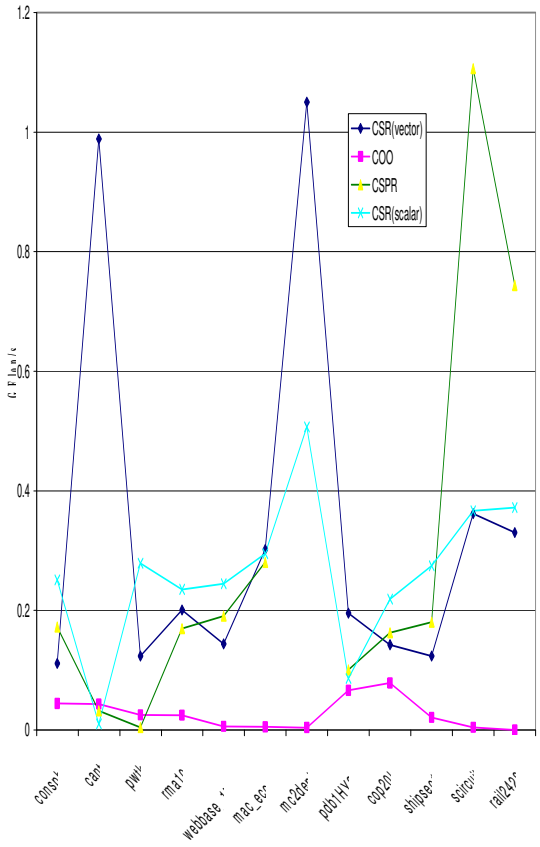


Fig. 7: Throughput comparison for the SPMV computation

12 matrices considered here are analyzed with respect to the new data format and performance analysis. The matrix mc2depi (shown in Fig. 3) which has 530K rows with an average of 3.9elements/row gives the highest performance (not shown) than any other format and any other matrix considered here. The performance improvements seen are 54x than the other methods. The overhead of synchronization is overcome by the maximum number of rows and its distribution of non-zero elements in rows. It also gives high performance than other three formats, for matrices scircuit, shipsec1 and rail2428 that have large rows with minimum distribution of elements and few rows with dense distribution. The matrices mac_econ, rma10, webbase_1m are more suited for the matrices with structure that fall in Fig. 3 and hence the performance improvement because of the same computation and synchronization behavior.

Matrices consph and cop20k_A have large number of dense rows and introduces synchronization overhead especially for the last row computations. Hence decrease in performance than CSR (vector). For the matrices cant and pwtk the performance is better than COO and CSR (scalar) but less than the CSR (vector), because there are large numbers of rows that are sparse and computation time is very less that is dominated by the synchronization time. Hence there is a

decrease in performance improvement. These results are given in Fig 7. CSPR format gives 2x to 5x performance improvement compared to CSR (compressed row format), 2x to 54x performance improvement with respect to COO (coordinate format) and 3x to 10x improvement compared to CSR vector format for the class of application that fit for the proposed new format. The results of comparison are shown in Fig. 6.

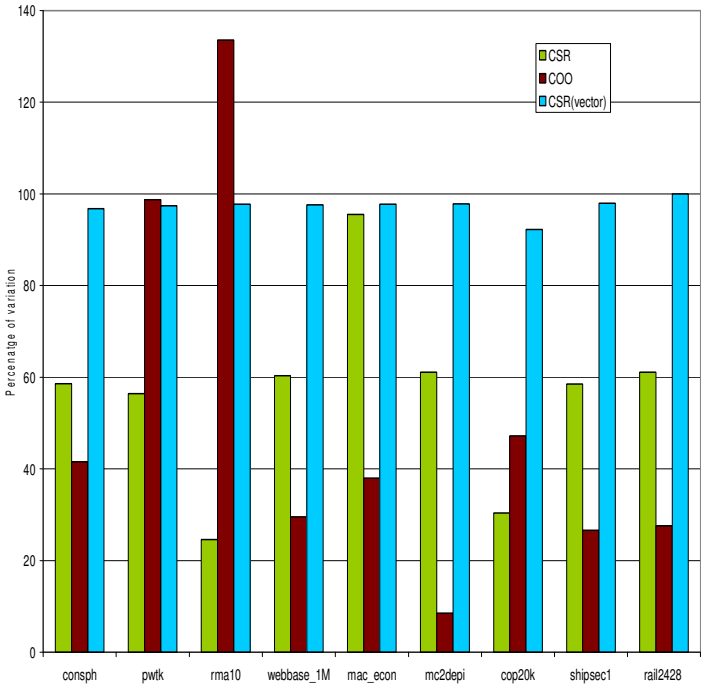


Fig. 8: Percentage variation of CPU-GPU memory transfer w.r.t. CSPR

Next evaluation criteria considered is effect of memory transfer from CPU to GPU. As the proposed method reduces the number of data structures required form two to one for the accessing information of the sparse matrix. Percentage variation of these memory transfers with respect to CSR, COO and CSR vector are given in Fig. 8. As it has reduced the data structures required it gives 10% to 133% improvement in only memory transfer time. This may be negligible if the data structure created on the device but if number of data structures required is reduced, the GPU architecture internal memory access optimization can also be made effective (not considered in this paper). This memory transfer is considered only for the access information transfer only and data value transfer time is not considered. The comparison is given in terms of percentage of variation. The memory transfer time is calculated as number of non-zero elements divided by the time taken for respective data structure transfer time and the computation time. Then these times are compared for the CSPR format against all the other three formats considered here. These results are encouraging and show that other new formats with respect to GPU can be created to improve sparse matrix computation.

Next evaluation considered is effective bandwidth utilization in terms of GBytes/s. It is computed as total number of reads and writes in bytes divided by average execution time. Number of reads and writes are taken as number of nonzero elements and the corresponding accesses from the data structure. The results are shown in Fig. 9. In most of the case CSPR is better than CSR-scalar and COO data formats.

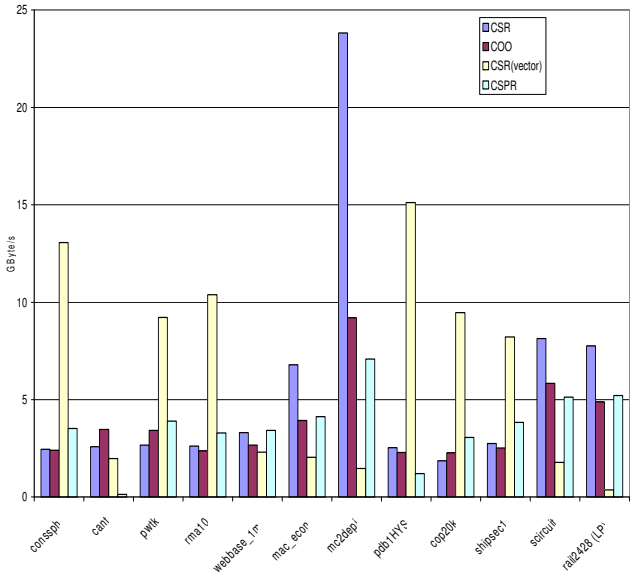


Fig 9: Effective bandwidth comparison w.r.t CSPR

The above method was not scalable to very large matrices. The scalable implementation of CSPR has lead to the following results: The memory copy (memCopy for short) time between CPU and GPU is compared for all the matrices given in the workload. They are compared by considering the time of memCopy in milliseconds. Fig. [10], shows that CSPR takes less time for memCopy in all the cases. CSPR is up to 107% better than CSR, 204% better than COO and 217% better than HYB when compared for memCopy time of sparse matrix data from CPU to GPU. The percentage of variations in memory transfer time of COO, CSR and HYB with respect to BLSI is shown in Fig [11].

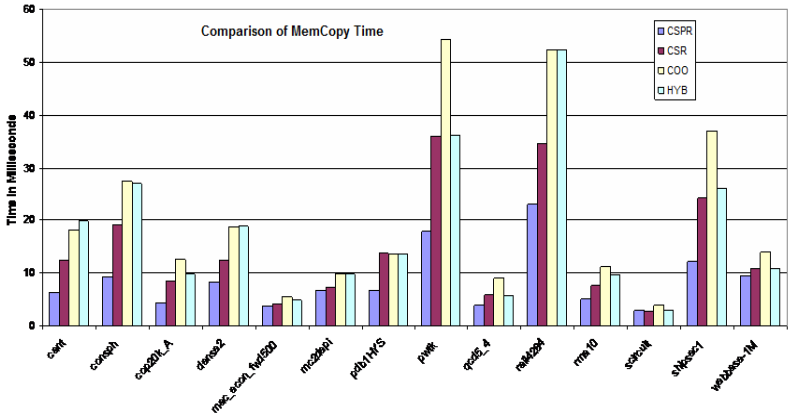


Fig. 10: Comparison of Memory copy time from CPU to GPU

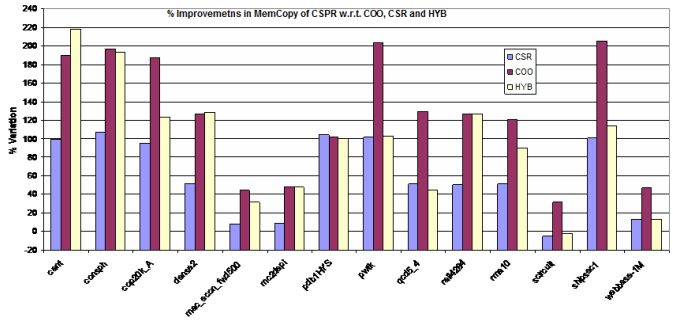


Fig. 11: Percentage variation of CSPR with respect to COO, CSR, HYB for memory copy time between CPU and GPU

By taking the memCopy time and kernel execution time as total time, for all these formats, CSPR is still better than other formats considered. As explained earlier, this format was proposed to reduce the CPU to GPU communication, this optimization has resulted in overall better performance also. CSPR is 80% better than CSR, 164% better than COO and 161% better than HYB (as shown in Fig [12]) when both the memory transfer time and kernel time is taken

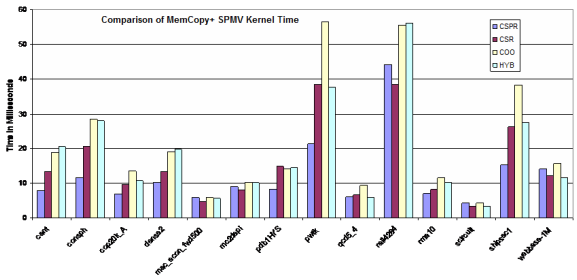


Fig. 12: Comparison of memory copy and kernel time of CSPR, COO, CSR and HYB data structures of sparse matrices

7. CONCLUSIONS AND FUTURE WORK

GPU processor evolution as general purpose computation processor has given challenges and opportunities to the scientific community. It is challenging in effective and efficient way of utilizing the processor. It gives high performance opportunities to increase the application performance using massive data parallelism. The implementation of CSPR is to tackle the challenges and increase the opportunities of GPU in high performance computing. SPMV computation optimization is of utmost importance for the scientific community in any form i.e. by new data formats or new optimization techniques. Results and analysis of CSPR shows that it can give 2x to 54x performance improvement for various matrices compared to CSR, COO and CSR vector formats. It gives 10% to 133% improvement in CPU to GPU memory transfer time. Effective memory bandwidth utilization is also on par with the other methods.

CSPR results are encouraging to work towards any other possibilities of new formats specific to GPU that can give better data parallelism and also optimizes for the internal memory architecture of GPUs. CSPR needs large data type to represent the new data structure. This can be overcome by memory align. This also can be optimized by using multi kernel merge launch that can reduce

this large data type requirement. Other formats like embedding the information into bits and extracting from bits can also be looked-in with respect to GPU. This work will be extended by considering optimizations for data layout optimization in internal architecture of GPU at the fine grain level and thread assignment mapping tailored to requirement of the application to give much desired performance benefits from the GPU.

REFERENCES

[1] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[2] www.drdobbs.com/supercomputingforthemasses (Jul. 28, 2010)

[3] http://developer.nvidia.com/ (Dec., 2010)

[4] Ryoo, S., et al. (2008). "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA." Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programing (PPoPP'08). ACM New York, NY, USA

[5] Ryoo, S., et al. (2008). "Program Optimization Space Pruning for a Multithreaded GPU." Proceedings of the 2008 International Symposium on Code Generation and Optimization.195–204. ACM New York, NY, USA

[6] Nathan Bell and Michael Garland (2009). "Efficient sparse matrix-vector multiplication on CUDA." Proceedings of ACM/IEEE Conf. Supercomputing (SC-09). ACM New York, NY, USA

[7] Eduardo, F. D., Mark, R. F. and Richard, T. M. (2005). "Vectorized sparse matrix multiply for compressed row storage." In Proc. Int'l. Conf. Computational Science (ICCS), LNCS: 3514/2005 : 99–106. Springer Verlag.

[8] Richard, W., Vuduc, R. and Hyun-Jin, M. (2005). "Fast sparse matrix-vector multiplication by exploiting variable block structure." In Proc. High- Performance Computing and Communications Conf., LNCS 3726/2005: 807–816. Springer Verlag.

[9] Blelloch, G. E., Heroux, M. A. and Zagha, M. (1993). "Segmented operations for sparse matrix computations on vector multiprocessors." Technical Report, CMU-CS-93-173, Department of Computer Science, Carnegie Mellon University (CMU), Pittsburgh, PA, USA.

[10] Geus, R. and Röllin, S. (2001). "Towards a fast parallel sparse matrix-vector multiplication." Proceedings of the International Conference on Parallel Computing (ParCo).

[11] Mellor-Crummey, J. and Garvin, J. (2002). "Optimizing sparse matrix vector multiply using unroll-and-jam." International Journal of High Perfromance Computing Applications, 18 (2). Sage Publications, CA, USA.

[12] Nishtala, R., Vuduc, R., Demmel, J. W. and Yelick, K. A. (2007). "When cache blocking sparse matrix vector multiply works and why." Journal of Applicable Algebra in Engineering, Communication, and Computing, 18 (3).

[13] Temam, O. and Jalby, W. (1992). "Characterizing the behavior of sparse algorithms on caches." Proceedings of the 1992 ACM/ IEEE Conference on Supercomputing (SC-92). IEEE Computer Society Press, Los Alamitos, CA, USA.

[14] Toledo, S. (1997). "Improving memory-system performance of sparse matrix-vector multiplication." Peoceeding of Eighth SIAM Conference on Parallel Processing for Scientific Computing.

[15] Vastenhouw, B. and Bisseling, R. H.(2005). "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication." Journal of SIAM Review, 47 (1): 67–95, 2005. Philadelphia, PA, USA.

[16] Im, E. J., Yelick, K. and Vuduc, R. (2004). "Sparsity: Optimization framework for sparse matrix kernels." International Journal of High Performance Computing Applications, 18(1):135–158, 2004. Sage Publications, CA, USA.

[17] Vuduc, R. (20030. "Automatic performance tuning of sparse matrix kernels." Doctoral Dissertation, University of California, Berkeley, Berkeley, CA, USA.

[18] Vuduc, R., James, W. D. and Katherine, A. Y. (2005). "OSKI: A library of automatically tuned sparse matrix kernels." Proceedings of SciDAC, J. Phys.: Conf. Series, IOP Science. 16: 521–530, 2005.

[19] Williams, s., et al. (2007). "Scientific computing kernels on the Cell processor." Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07), International Journal of Parallel Programming, 35(3):263–298, 2007. Kluwer Academic Publishers Norwell, MA, USA.
[20] Lee, B. C., Vuduc, R., Demmel, J. and Yelick, K. (2004). "Performance models for evaluation and automatic tuning of symmetric sparse matrix-vector multiply." Proceedings of the International Conference on Parallel Processing (ICPP'04). IEEE Computer Scoiety, Washington, DC, USA.
[21] Muthu Manikandan Baskaran and Rajesh Bordawekar (2008). "Optimizing sparse matrix-vector multiplication on GPUs using compile-time and run-time strategies." Technical Report RC24704 (W0812-047), IBM T.J.Watson Research Center, Yorktown Heights, NY, USA, December 2008.
[22] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schr¨oder (2003). "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid." Proceedings of Special Interest Group on Graphics Conf. (SIGGRAPH), San Diego, CA, USA, July 2003.
[23] Matthias Christen and Olaf Schenk (2007). "General-purpose sparse matrix building blocks using the NVIDIA CUDA technology platform." Proceedings of Workshop on General-Purpose Processing on Graphics Processing Units (GPGPU).
[24] Michael Garland (2008). "Sparse matrix computations on manycore GPUs. " Proceeding of ACM/IEEE Design Automation Conf. (DAC),2–6. Anaheim, CA, USA.
[25] Roman Geus and Stefan R¨ollin. (2001). towards a fast sparse symmetric matrix-vector multiplication. Journal of Parallel Computing, 27 (7):883–896.
[26] http://www.cise.ufl.edu/research/sparse/matrices/Williams/index.html

Authors:

B. Neelima is working as Asst. Professor in Dept. of Computer Science and Engineering at NMAMIT, Nitte, Karnataka. She is also pursuing her research at NITK surathkal under the guidance of Dr. Prakash S. Raghavendra, in the area of High Performance Computing. She has around 11 years of teaching and research experience. She is instrumental in getting a Department of Science and Technology (DST) R&D, Govt. of India project with 16lakhs funding.

She also brought in various industry associations to the college like IBM center of excellence, NVIDIA CUDA Teaching Center, Intel and AMD University programs etc to name a few. She has guided 6 PG projects and around 40 UG projects till today. She has around 20 publications in various International and National Journals and conferences. She is also instrumental in bringing in various new electives into the curricula of NMAMIT, Nitte, an autonomous college.

Dr. Prakash Raghavendra is a Faculty at Depatment of information Technology, NITK, Surathkal from Feb 2009. He received his Doctorate from Computer Science and Automation Department (IISc, Bangalore) in 1998, after graduating from IIT Madras in 1994.

Earlier, Dr. Prakash was working in Kernel, Java and Compilers Lab in Hewlett-Packard ISO in Bangalore from 1998 to 2007. He was exposed to some of the area in Computer systems, during this period. Dr. Prakash also was working for Adobe Systems, Bangalore from 2007 to 2009 before joining NITK. At Adobe, Prakash was working in the area of Flex Profilers.

Dr. Prakash's current research interests include Programming for Heterogeneous Computing, Web Usage Mining and Rich Internet Apps. Dr. Prakash has been honoured with 'Intel Parallelism Content Award' in 2011 and 'IBM Faculty award' for the year 2010.