Review on Strategies for Shared Memory Sparse Matrix-Vector Multiplication

Komal Deshmukh
Computer Department
MET BKC Adgaon, Nashik,
Savitribai Phule Pune University,
Maharashtra India

M.U. Kharat
Computer Department
MET BKC Adgaon, Nashik,
Savitribai Phule Pune University,
Maharashtra India

ABSTRACT
The sparse matrix is one of the most important data storage format for large amount of data. Sparse matrix-vector multiplication (SpMV) is important operation in many scientific and engineering applications. Many physical systems produce sparse matrices. Sparse matrix-vector multiplication can be implemented sequentially and it includes various methods like one dimensional cache-oblivious method, this method extended to two dimensional methods and Hilbert space filling curve. There is limitation of sequential case as it is not providing efficient result. There are some problems occur related to sequential case of multiplication that are less arithmetic intensity, cache inefficiency and limited memory bandwidth. To avoid such problem parallel techniques were introduced. It mainly focuses on high level strategies for efficient performance of sparse matrix-vector multiplication. High level strategies include no vector distribution, one dimensional distribution and two dimensional distributions. This provides high arithmetic intensity and better performance as compared to sequential technique on NUMA Architecture.

Keywords
Sparse Matrix-Vector Multiplication (SpMV), Cache-Oblivious, NUMA architecture

1. INTRODUCTION
The Matrix-Vector product (MV) is a core operation for a various scientific and engineering applications like image processing, simulation, electrical and control engineering and so on. As many of the application store its data in terms of matrix so it is termed as dense matrix. When the dense matrix contains many zero element then matrix efficiently represented using data structure as Sparse Matrix. A matrix in which number of zero elements is greater than the number of non-zero element is called sparse matrix. The general approach of representing matrices in memory uses two-dimensional arrays. But it may not be suitable for sparse matrices because its storage includes large zero element entries. One may increase storage efficiency by storing only non-zero entries along with its row and column position. Sparse matrix is nothing but the memory efficient representation of data as compared to dense matrix [6].

The sparse matrix vector product (SpMV) is preeminent operation in engineering and scientific application and hence it has been a subject of profound research. The irregular or random data accesses involved in SpMV make its optimization as challenging task. Therefore, tremendous effort has been devoted to cogitate data formats for representation of sparse matrix with the goal of magnifying the performance. There are various iterative solving techniques for sparse linear system such as Conjugate Gradients (CG) [2], Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems (GMRES), BiCGstab [3].

This SpMV is basically done in two ways. It is divided into two parts, where the first part is on optimizing the sequential SpMV multiplication. In a world where parallel machines are increasingly common place, however, a fast method should also be parallelizable. Thus the second part combines the insights of fast sequential SpMV multiplication and traditional distributed SpMV multiplication, to describe parallel implementations both for distributed-memory and shared-memory architectures.

1.1 Sparse Matrix Storage Representation
There is several storage formats used for sparse matrices, but many of them exploit the same basic technique. Representation of these storage formats encompasses all non-zero elements of the dense matrix into a real, complex array and provides subordinate arrays are provided to specify the locations of the non-zero element.

1.1.1 Coordinate Format (COO)
The coordinate format is one of the most flexible, elementary and fast format for efficient representation of sparse matrix. COO format is called as Triplet format which consist of only non-zero elements and the coordinate position of every non-zero elements are provided explicitly. Many a commercial application as well as libraries supports the sparse matrix-vector multiplication in the COO format. This presentation consists of triplet that is (*values, rows, and column*) and a parameter *nz represents* number of non-zero elements from original matrix *Z.* All three arrays have dimension *as nz.* The following triplet (value, row, and column) describes non-zero elements in matrix Z.

values -The array which consist of non-zero elements of *original matrix Z* in random order.

rows - This array consist of index for all non-zero elements available in original matrix Z.

columns - This array has column index for all non-zero elements available in original matrix Z.

Storage requirement for COO format is O(3nz).

1.1.2 Compressed Row Storage (CRS)
This format is basically row oriented format. Compressed sparse row (CRS) is represented by three arrays: the *values, index and row-pointer*. The following array describes values, index, and row-pointer positions of non-zero element. Value array is same as arrays of COO format.

index – index array maintain the column position for each non-zero number.

row-pointer –This provides the pointer according to row for each non-zero element.

Space requirement for CRS format is O(2nz+Θ) where Θ represents space required for row-pointer. This format can also extend to Incremental Compressed Row Storage (ICRS). It contains only one extra array as i*nz+j where i and j are row and column position respectively and nz is number of non-zero element.

1.1.3 Compressed Column Storage (CCS)

The compressed column Storage (CCS) is analogous to the CSR format, but the column pointer are used instead the row-pointer. In other words, both format as CCS and CRS are same for transposed matrix. The CCS format includes three arrays: the *values*, *index and column-pointer*. The following array describes values, index, and column-pointer. value array is same as arrays of COO format.

index – index array maintain the row position for each non-zero number.

Column-pointer –This provides the pointer according to column for each non-zero element.

Space requirement for CRS format is O(2nz+Θ) where Θ represents space required for column-pointer. This format can also extend to Incremental Compressed Column Storage (ICCS). It contains only one extra array as i*nz+j where i and j are row and column position respectively and nz is non-zero element in matrix.

1.2 Cache Behavior with SpMV Multiplication

During SpMV multiplication data access is random and it reduces efficiency. When the requested data is available in cache then Cache hit, otherwise Cache miss occurs. For efficient SPMV multiplication it is important to increase cache hit rate as well as increase bandwidth between cache and main memory. In the today era, many architecture supports multiple caches placed between main memory and processor. Cache which close to processor are lower level cache and are faster as compared to higher level cache [1].

There are many processor or core are available in parallel shared memory architecture. These cores may share higher level cache or used their own individual cache.

1.3 Parallelization for SpMV Multiplication

For parallelization of SpMV multiplication, the available non-zero elements of any sparse matrix are evenly split into number of core which are available, consider p are number of available core and nz are total non-zero element then splitting is done by nz/p. Then each core has its own set of non-zero element, on which they performed sequential SpMV. Finally it combined the result of all core after multiplication.

While dealing with parallelization, when optimization is depend upon the computer hardware then it is called as cache-aware technique. But when focus is on attending good performance rather than computer architecture, it is nothing but the cache-oblivious technique.

2. RELATED WORK

There are various techniques are available for sparse matrix vector multiplication from Sequential technique to Parallel high level strategies.

2.1 Sequential Methods

One of the sequential methods is nothing but Hypergraph partitioning. As the aim of SpMV multiplication is to increase cache hit rate. So it requires permutation of rows and column of matrix. This permutation done by using row-net and column-nets. In the hypergraph input matrix is represented using (V, N) where V is all non-zero elements in matrix and N is set of all nets. As Shown in Fig 1, in partitioning V is divided into Vleft and Vright. Then all row net ni are placed in Nrow (minus) if ni ∩ Vright=NULL, if ni ∩ Vleft = NULL then placed it is in Nrow(plus), otherwise in Nrow(const). Same procedure repeated for column. In this way it create Separated Block diagonal (SBD). Finally apply Sequential SpMV multiplication on each block [6].

Another technique is nothing but the reordering strategy which mainly focuses on ordering of non- zero element. There is freedom of ordering of non-zero element so any element can be access from input and output vector.

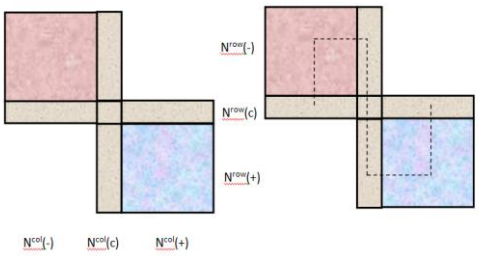

Fig 1: Separated Block Diagonal form of Sparse Matrix

2.2 Parallel Methods

Basically the efficiency of SpMV computation kernel is not only depends on adaptation of matrix for SpMV but also depends upon the storage scheme of non-zero element. One of the parallel techniques is nothing but the parallel shared memory SpMV multiplication which is derived by Yzelman and Bisseling. Parallel shared memory architecture consists of number of cores and each processor has an active part in SpMV multiplication. This techniques is divided into three stages that are-

1. Firstly each processor read elements from input vector consider as x and copies these elements into local buffer of core xs.
2. Above step is followed by ICRS based multiplication on each row i. In this case if i is local to the core s then it directly writes its contribution to local output vector ys otherwise it has to send its contribution to remote process.
3. The last step is Global Synchronization in which it ensures all the processes are done with sending their contribution. And finally it gathers all the contribution to main output vector y.

Another parallel technique of shared memory SpMV multiplication is Morton Curve. It uses quad tree to store sparse matrix. In quad tree root node is representing as full matrix, it is splitted into four internal nodes which represents rectangular sub matrices. Each sub matrix is corresponds to child of internal node. Basically on the quad tree two concurrent threads are run, in which one thread traverse node corresponds to top rows of sub matrices while other thread traverse the bottom rows of sub matrices.

3. HIGH-LEVEL STRATEGIES

As there are various SpMV multiplication strategies starting from sequential to parallel. And then it is followed by High-Level strategies. These are starts with simple distribution of matrix but not take input and output vector into account for distribution. It is followed by One-Dimensional (1D) strategy and the Two-Dimensional strategy (2D).

3.1 No Vector Distribution

In this scheme there is simple distribution of matrix and it input and output vector does not taken into account for distribution. Input and Output vectors are stored in an interleaved fashion so that any core can request any element from input vector at random for multiplication. After multiplication each core writes its output of multiplication into local buffer of output vector. And finally postprocessing step combines all local result. In this scheme Cache-Oblivious Hilbert Curve (CO-H+) is used which totally depend upon the non-zero structure of input matrix [1].

3.2 One-Dimensional Distribution

This high level strategy includes the distribution of matrix and input vector x, but output vector does not taken into for distribution. Input vector stored in an interleaved fashion so that any core can request any element from input vector at random for multiplication. In this strategy each core has its own local input element for SpMV multiplication [1].

3.3 Two-Dimensional Distribution

One-Dimensional Distribution is followed by Two-Dimensional Distribution in which matrix and both input and output vector gets distributed. So that each processor has own set of input vector element and output vector to store the result of multiplication [1].

4. USE OF GPU FOR SpMV MULTIPLICATION

Different techniques of SpMV multiplication are mainly based on shared memory parallel architecture. Shared memory architecture has large number of core for parallelization. Graphical Processing Unit (GPU) is one in which many cores are available for execution. So in case of SpMV multiplication, requirement of number of core is fulfill by GPU.

Using CPU with GPU together accelerate various scientific, engineering and enterprise application. Basically CPU consists of very few cores as compared to GPU. GPU consist of thousands of small, efficient core design for handling the multiple tasks simultaneously. Numbers of available core are varying in GPU according to capacity and need. There are different types of memory available on GPU that are Shared memory, Texture memory, Global memory, Register memory and Constant memory. Depends upon the need different memories are available.

GPU is important for execution of shared memory parallel architecture SpMV multiplication.

5. CONCLUSION

Many scientific and engineering applications store their data in matrix format and dense matrices are efficiently represented using sparse matrix. Sparse Matrix-Vector multiplication is important operation in many applications like web crawling, robot control application, electrical engineering application, and term to index documenting etc. SpMV multiplication uses various storage schemes like COO, CRS, CCS, ICRS etc. The main aim of SpMV multiplication is to increase the cache hit rate and increase main memory to processor bandwidth.

A higher-level strategy includes No vector distribution, One-Dimensional Distribution and Two-Dimensional Distribution. Higher-level strategies provide high arithmetic intensity, increase cache hit rate and increase bandwidth ratio. This provides efficient performance as compared to sequential techniques and it is decided by calculating speed up for SpMV that is, ratio of time required by sequential SpMV multiplication to time required by Parallel SpMV multiplication.

6. ACKNOWLEDGMENTS

The author wish to thank MET's Institute of Engineering Bhujbal Knowledge City Nasik, HOD computer department, guide and parents for supporting and motivating for this work because without their blessing this was not possible.

7. REFERENCES

[1] Albert-Jan Nicholas Yzelman and Dirk Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix-Vector Multiplication," IEEE Transaction on parallel and distributed systems, vol. 25, no. 1, Jan 2014.

[2] M.R. Hestenes and E. Stiefel, "Methods of ConjugateGradients for Solving Linear Systems," J. Research Nat'lBureau of Standards, vol. 49, pp. 409-436, 1952.

[3] H. van der Vorst, "BiCGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," SIAM J. Scientific and Statistical Computation, vol. 13, pp. 631-644, 1992.

[4] Y. Saad and M. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," SIAM J. Scientific and Statistical Computation, vol. 7, pp. 856-869,1986.

[5] P. Sonneveld and M.B. van Gijzen, "IDR: A Family of Simple and Fast Algorithms for Solving Large Non symmetric Linear Systems," SIAM J. Scientific Computing, vol. 31, no. 2, pp. 1035- 1062, 2008.

[6] A.N. Yzelman and R.H. Bisseling, "Cache-Oblivious Sparse Matrix-Vector Multiplication by Using Sparse Matrix Partitioning Methods," SIAM J. Scientific Computing, vol. 31, no. 4, pp. 3128- 3154, 2009.

[7] A.N. Yzelman, "Fast Sparse Matrix-Vector Multiplication by Partitioning and Reordering," PhD dissertation, Utrecht Univ., 2011.

[8] A. Buluc¸, S. Williams, L. Oliker, and J. Demmel, "Reduced- Bandwidth Multithreaded Algorithms for Sparse Matrix-Vector Multiplication," Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '11), pp. 721- 733, 2011.