M. Kayanan(1) and S. Kanaganathan(2)

Sparse Matrix-Vector multiplication using parallel algorithm with block striped partitioning

(1) Department of Physical Science, Faculty of Applied Science, Vavuniya Campus of the University of Jaffna, Sri Lanka, email : mgayanan@gmail.com

(2) Department of Physical Science, Faculty of Applied Science, Vavuniya Campus of the University of Jaffna, Sri Lanka

Abstract: Parallel algorithm may be executed as a piece at a time on many processing devices, and then return results at the end. A sparse matrix is huge and consists many zero elements. Its execution process is extremely time-consuming. This paper describes, how to do Sparse Matrix-Vector multiplication using parallel algorithm with block striped partitioning. Speed-up of Sparse Matrix-Vector multiplication measured with increasing number of processors. Experiment results shows that this method increase the speed-up of Sparse Matrix-Vector multiplication.

Keywords: Parallel algorithm, Sparse matrix, Matrix-Vector multiplication, Speedup

Introduction

Traditionally, computer software has been written for serial computation. To solve a problem, an algorithm is constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit in one computer. Only one instruction may be executed at a time-after that instruction is finished, the next is executed. On the other hand, parallel computing uses multiple processing elements simultaneously to solve a problem (Grama et al., 2003) This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of algorithm simultaneously with others (Parhami, 2002). The processing elements can be diverse and includes resources such as a single computer with multiple processors, several of networked computers, specialized hardware, or any combination of the above.

Information of many pieces of human activity is frequently represented in the form of matrices. Matrix computing has played an important part in numeric computing. The matrix is sparse when many of the elements are equal to zero. Huge sparse matrices often appear in science or engineering when solving partial differential equations. Sparse Matrix computing includes so many different operations; most popular sparse matrix techniques are Matrix-Vector multiplication, Matrix-Matrix multiplication and solving System of Linear Equations. Sparse Matrix-Vector multiplication is an important operation because it involved in many scientific computations. Some examples are solution to Linear System with iterative methods, network traffic applications and linear programming. Sparse matrix solution is extremely time consuming for large problems.So itsparallelization is desirable to improve its execution time. A parallel formulation for sparse matrix factorization can be easily obtained by simply distributing rows to different processors.

In this work, parallel sparse Matrix-Vector multiplication algorithm with block strip partitioning based upon the standard Message Passing Interface (MPI).

Methodology

In this piece of work sparse Matrix-Vector multiplication using block striped partitioning was implemented. The main goal with the parallelization is to perform Matrix-Vector multiplication faster than what is possible on a single processor.

A sparse matrix is one whose entries are mostly zero. There are many ways of storing a sparse matrix. Whichever method is chosen, some form of compact data structure is required that avoids storing the numerically zero entries in the matrix. It needs to be simple and flexible so that it can be used in a wide range of matrix operations. In this experiment Compressed Row Storage (CRS) scheme was used.

The CRS is a widely used scheme for storing sparse matrices. In the CRS format, a sparse matrix with rows having non-zero entries is stored using three arrays: two integer arrays ro and cc, and one array of real entries v. The array rc is of size , and the other two arrays are each of size . The array cc stores the column indices of the non-zero entries in , and the array values stores the corresponding non-zero entries. In particular, the array cc stores the column indices of the first row followed by the column indices of the second row followed by the column indices of the third row, and so on. The array ro is used to determine where the storage of the different rows starts and ends in the array cc and values. In particular, the column-indices of row are stored starting at col_ind [row_pand ending at (but not including) col_ind [row_ptr[i . Similarly, the values of the non-zero entries of row are stored at values [row_p and ending at (but not including) values [row_ptr[i . And also note that the number of non-zero entries of row is simply row_ptr[i + 1] − row_ CITATION Don08 \l 1033 (Dongarra, 2008) .

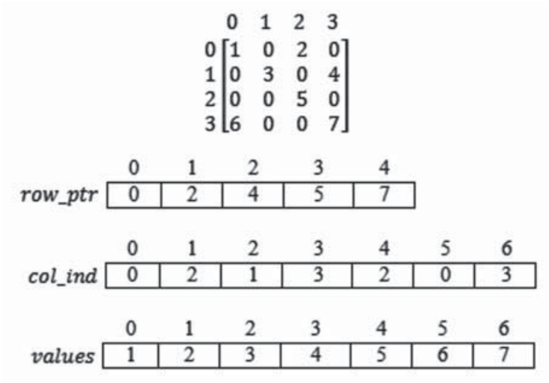

Figure 1: Representation of Sparse Matrix in Compresses Row Storage (CRS) format

We consider the problem of computing the sparse matrix-vector product [y] = [, where is a sparse matrix of size , and is a dense vector using block striped partitioning. In the block striped partitioning of a matrix, the matrix is divided into groups of complete rows or columns, and each process is assigned one such group.

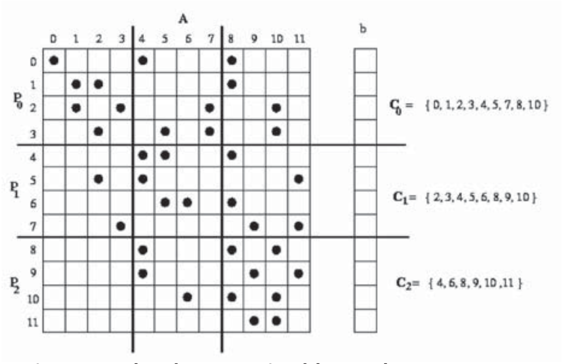
A
b
0 1 2 3 4 5 6 7 8 9 10 11
P0
P1
P2
C0 = { 0, 1, 2, 3, 4, 5, 7, 8, 10 }
C1 = { 2, 3, 4, 5, 6, 8, 9, 10 }
C2 = { 4, 6, 8, 9, 10, 11 }

Figure 2 :The data required by each processor to compute sparse matrix-vector product

The algorithm partitions the rows of matrix using block-striped partitioning and the corresponding entries of vector among the processes, so that each of the processes gets rows of the matrix and elements of the vector. The portion of the matrix obtained by block-striped partitioning, is assigned to each process and the non-zero entries of the sparse matrix is stored using the CSR format in the arrays ro, cc and v. To obtain the entire vector on all processes, MPI_Allgather collective communication is performed.

Each process is responsible for computing the elements of the vector that correspond to the rows of the matrix that it stores locally. This can be performed as soon as each process receives the elements of the vector that required to compute these serial sparse dot-products.

The set of elements depends on the position of the non-zeros in the rows of assigned to each process. In particular, for each process , let be the set of column-indices that contain non-zero entries overall the rows assigned to this process. Then process needs to receive all the entries of the form for all in.

The parallel sparse Matrix-Vector multiplication has been implemented using C language in fedora core 6.0 (Linux platform) and using MPICH2 with eight connected computer. In this setup, a computational test using increasing number of processors was experimented. While implementing, it is considered one processor for a sequential sparse Matrix-Vector multiplication. And, execution and speedup time werecalculated for increasing number of processors. Where, execution was calculated using MPI routine MPI_Wtime() and speedup is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solving the same problem on a parallel computer with identical processing elements.

Results and Discussion

The results were obtained from 16×16 matrix and 16×1 vector implementation. It is observed, execution time decreased with increasing number of processors and after 4th processor the execution time was not changed [Figure 3]. It shows that there is an optimum number of processor for a particular problem.

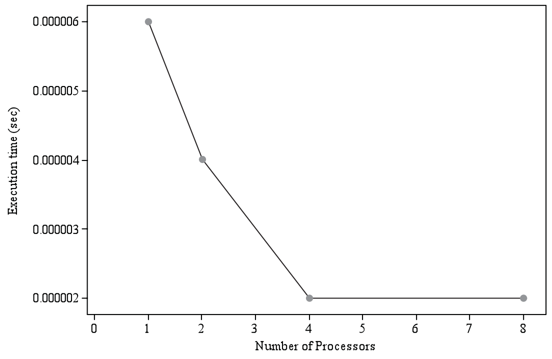

Figure 3 : Observed execution time of Sparse Matrix-vector multiplication with increasing number of processors

Speedup was increased as increasing number of processors and after 4th processor the execution time was not changed [Figure 4] because calculation of the speed up depends on execution time. We can observe that the speedup was suddenly increased as increasing number of processors.

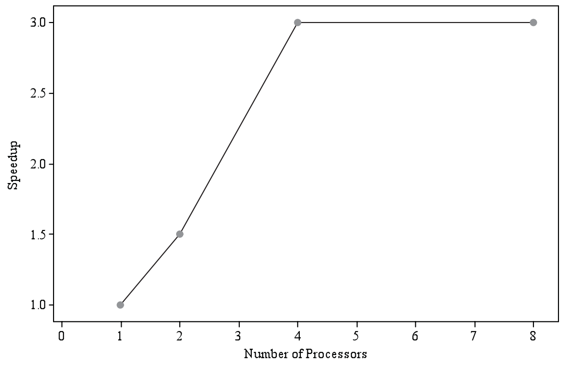

Figure 4: Observed speedup of Sparse Matrix-Vector multiplication with increasing number of processors

Conclusion

In this piece of work we have described a Sparse Matrix-Vector multiplication using parallel algorithms with block striped partitioning was experimented. In this instance to avoid zero multiplication we used the compressed row storage matrix format was used.

By implementing it is possible to conclude that the parallel algorithms reduce the execution time as opposed to the sequential algorithms for computation of Sparse Matrix-Vector multiplication. The results emphasize that the parallel algorithm is an effective when sparse matrix computation.

Increasing number of processors continuously leads to communication overhead. Study on communication overhead in parallel sparse matrix computation may be the window for future work.

References

Buluc, A., Fineman, J., , Frigo, M., Gilbert, J.R. and Leiserson, C.E. (2009) 'Parallel Sparse Matrix Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks', twenty-first annual symposium on Parallelism in algorithms and architectures, USA.

CDAC (2008) *Programming Using MPI 1.X Implementing Dense and Spase Matrix Computations Algorithms*, [Online], Available: "http://www.cdac.in/HTmL/events/beta-test/archives/promcore-2008/mpi-1x-promcore-2008/matrix-vector-matrix-comp-mpi.html" http://www.cdac.in/HTmL/events/beta-test/archives/promcore-2008/mpi-1x-promcore-2008/matrix-vector-matrix-comp-mpi.html [6 December 2012].

Dongarra, J. (2008) *Survey of Sparse Matrix Storage Formats*, [Online], Available "http://www.netlib.org/linalg/html_templates/node90.html" http://www.netlib.org/linalg/html_templates/node90.html [6 December 2012].

Grama, A., Karypis, G., Kumar, V. and Gupta, A. (2003) *Introduction to Parallel Computing*, 2nd edition, Addison Wesley.

Parhami, B. (2002) *Introduction to Parallel Processing: Algorithms and Architectures*, PLENUM SERIES IN COMPUTER SCIENCE edition, New York: Plenum Press.