IIMS Postgraduate Seminar 2009

Parallel sparse matrix algorithms - for numerical computing

Matrix-vector multiplication

Dakuan CUI

Institute of Information & Mathematical Sciences
Massey University at Albany, Auckland, New Zealand

Abstract

Matrix computing has played an important part in numeric computing. Sparse matrix is a type of matrix that used in all kinds of computing, and sparse matrix computing has significant meaning in the different fields. Sparse Matrix computing includes so many different operations, for example, addition, Scalar multiplication, Transpose. This report will discuss the Matrix-vector multiplication that is one of the most important computations in the science computation, and it always involves enormous computation, therefore there is a need to using parallel technology to implement them. This report focuses on how to implement the sparse matrix vector multiplication using parallel in detail; it will base on sparse matrix's low computation and higher communication characteristics under parallel computation and will introduce a simple algorithm with parallel computing, and display the results when using different number of processors.

Key Words: Matrix-Vector Multiplication, Sparse Matrix, Parallel

1. Introduction

Parallel computing is a type of computation, in this kind of computation. All the large problems or problems with more computational steps will divide into many small tasks and all the tasks performed in computers with multiple internal processors simultaneously. Using parallel technology, it will speed up the complex computation. However, the parallel computation speed does not equal sequential speed divide n processors (number of processors used for parallel), it dues to the communication time spends on processors. Along with the improvement of the processors speed, the factor effect the parallel speedup is the network communication.

In this report, it includes several parts. Firstly, introduce the concept of sparse matrix, their storage and Matrix-Vector Multiplication; Second part focuses on the characteristics of parallel computing; Third part discusses how to implement Sparse Matrix-Vector Multiplication using parallel technology. Fourthly analyses the different result base on different number of processors. (For instance, two processors, four processors, eight processors and so on). Finally is the conclusion.

2. Sparse Matrix

2.1 Concept

In the mathematical subfield of numerical analysis, a sparse matrix is a matrix populated primarily with zeros. *(Stoer & Bulirsch 2002, p. 619)*.

For instance, the matrix has m columns n rows, if number of non-zero elements is s, s >> total-s (total elements in the matrix which is m*n), we will say the matrix is a sparse matrix.

0， 0， 0， 2
1， 0， 0， 6　　need space 4*4*4 = 64
0， 1， 0， 0
0， 0， 0， 0

Figure 2.1

This matrix is a typical sparse matrix, the total number of elements is 16. It has 11 zero elements, which is primarily with zeros.

2.2 Sparse Matrix Storage/Save

If it uses a two dimensional array to store a m row n column sparse matrix, if every array element needs L binary to store it, then the total storage need for this sparse matrix is m*n*L. And in this matrix, most of space used to store zero elements, it is a waste that taking huge space to store item zero. For saving the space, the better idea is only store non-zero items.

If it used sparse Matrix storage method to store the matrix in figure 2.1, only need to store non-zero items. It is very easy to find an item in the matrix if know which row and column the item is allocate. Therefore, in a sparse matrix, in order to saving space, we can define an item only via the row and column (coordinate) and the item value.

Define the sparse matrix in Figure 2.2 as listed below:

row column item
0， 3， 2
1， 0， 1　　need space 4*3*4 = 48
1， 3， 6
2， 1， 1

Figure 2.2

The example used is a small sparse matrix; it is easy to notice that it saves some storage space. Imagine a 10000*10000 sparse matrix; it will save a lot of space using this method.

3. Matrix- Matrix-vector multiplication

The vector can be represented by one-dimensional array. The basic algorithm is using the formula provided below, which is very simple to understand.

Matrix Vector multiplication define by $y_i = \sum_j A_{ij} x_j$

Aij is a i*j matrix, Xj is a vector with j number of elements.
Yi is the result Matrix

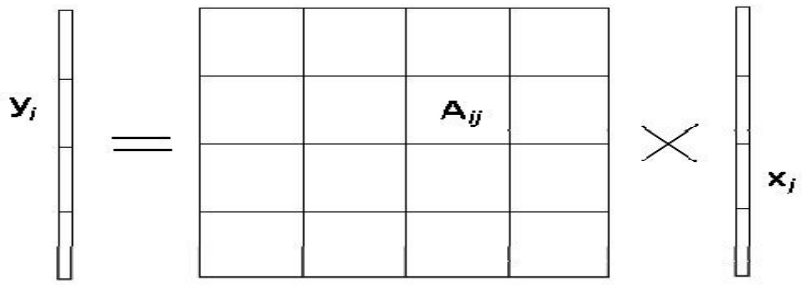

Figure 3.1

Figure 3.1 shows how the formula works.

This formula is the core operations when computing the matrix vector multiplication. It not only uses for regular matrix, but also is able to apply it on sparse matrix vector multiplication.

If we make the following assumption, for instance: [1, 3, 2, 1] is a vector, use the sparse matrix Figure 2.2 as the example sparse matrix, multiply it with the vector above

[0 , 3 , 2
1 , 0 , 1
1 , 3 , 6 * [1, 3, 2, 1]
2 , 1 , 1]

The multiplication will be

y0 = A03*X3 = 2*1=2
y1 = A10*X0 = 1*1=1
y1 = A13*X3 = 6*1=6
y2 = A21*X1 = 1*3=3

There are two y1, which is due to there are two non-zero elements in row 1. So it needs to add the value together to get the y1 = 1+6 =7, the result is [2, 7, 3, 0].

4. Parallel Computing

Blaise (2009) presents the parallel computing uses multiple processors to solve a computational problem and the problem is break into several parts, which runs on multiple processors at the same time.

Parallel computing has some advantages comparing with sequential computing.

- Faster. In theory, it saves time when doing the same job using more processors.
- Solve large problems. It is very hard or impossible to solve the huge problem on a single computer, when the complex problem needs much more memory to solve this is due to the

single computer only has limited memory.

- Can do lots things simultaneously. Parallel computing can do many tasks at the same time; however, a single computer cannot do the way like that.

Sparse matrix vector multiplication always involves huge computing. If it is able to implement using parallel computing, it has a big advantage in improving speed.

5. Method

5.1 Sparse Matrix and Vector Data Structure

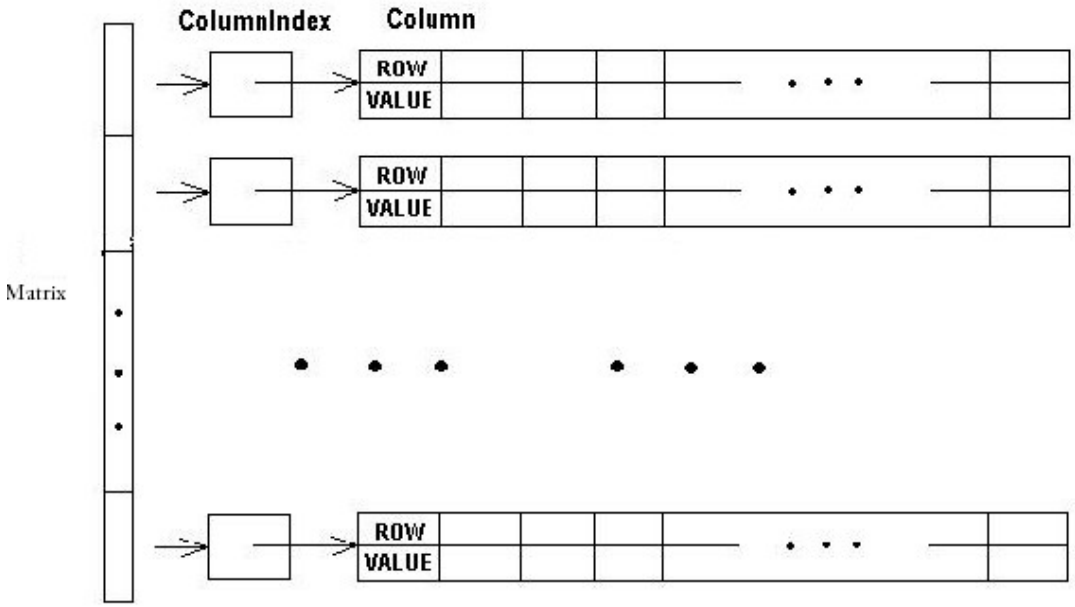

Figure 4.1

Create a matrix with user inputs the rows and columns size which is represented as a two dimensional array. After being allocated the memory need for the two dimensional array, we can produce a matrix whose elements are all random numbers and with lots of zero elements.

Considering the two dimensional array is much more flexibility in partitioning the matrix in parallel, we need to do some work with this two dimensional array first before starting partition.

5.2 Basic Parallel Algorithm

To simplify the multiplication, the idea is to transform the two dimensional array to a one dimensional array, it is much easier to implement parallel. In this one dimensional array, store the coordinates and values sequentially.

The one dimensional array has all the information need for the formula, i, j is the coordinate, Aij is the value in that coordinate, Xj is the vector.

1. To transform a matrix to a sparse matrix, create a new two dimensional array to store the sparse matrix, loop through all the columns, when the element in the two dimensional array is not zero, save the coordinate and respondent value to this sparse matrix.

The sparse matrix will only store the coordinates and values, so it will only has 3 columns and with lots of rows, the total number of rows represent the total number of non-zero elements in the matrix.

2. Partition

The master-slave approach is used in parallelism. One process acts as the master and the others act as slave, the master processor assigns the task to the other slave processes. The master will give each process a task or a list of tasks. After a process completes its work, it will return the value back to master processor.

In the sparse matrix vector multiplication, consider the characteristic of sparse matrix, partition the matrix into sub-matrix, every slave will do the several sub-matrix of multiplication simultaneously and then send all the results back to master processor.

The first column of the sparse matrix represents the row in the original matrix, so we can get the total number (Cn) of elements according to the row. Each row represent a sub-matrix.
Master processor will do Cn/numprocess + Cn%numprocess rows; Slave processor will do Cn/numprocess rows.

Then, the master processor will send those elements in Cn/numprocess row to slave. Each slave will do Cn/numprocess vector multiplication.

3. Communication

They are many technologies that support parallel programming, for instance, MPI (message passing interface), PVM (parallel visual machine), OpenMP, threads. In this report, we will use MPI for communication.
Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is used in computer clusters and supercomputers. MPI was created by William Gropp and Ewing Lusk and others.
(http://en.wikipedia.org/wiki/Message_Passing_Interface)
In the sparse matrix vector multiplication, MPI_Bcast, MPI_Send, MPI_Recv are used follow their grammar.

```
MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root,
MPI_Comm comm)
```

MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
MPI_Status *status)

4. Parallel logical

```
MPI_Status stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);          MPI Regular
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

MPI_Bcast;                              //broadcast vector to slave

if (myid == 0){  //master
    MPI_Send(sendbuffer);               //send each buffer to each slave
•
 }else{  //slave
•   MPI_Recv(recvbuffer);              //receive buffer from master
    SparseMult(recvbuffer,vect);       //multiplication
    MPI_Send(slaveresult);             //send result to master
}

•    MPI_Finalize();
```

6. Results/Conclusion

6.1 Result

The demo result is based on the algorithms mentioned above, it uses the random produced 100*100 matrix multiply with the vector size is 100 and it was running on IIMS cluster in Massey. The running time based on different number of processors applied.

Number of processors used	Running time
2	0.009006s
4	0.007533s
6	0.004769s
8	0.001441s

Figure 6.1

6.2 Complexity

Assume the saved sparse matrix is a one dimensional array (_dsm), the vector is a one dimensional array (_vi), the pseudo_code is showing as follows,

```
double * SparseMult(){
      double temp = 0;                          (a)
       int a = 0;                                      (b)
       if (_dsmsize%3==0) {
            for (int adsm = 0; adsm < _dsmsize; adsm=adsm+3) {      (c)
                  temp = _dsm[adsm+2]*_vi[_dsm[adsm+1]]+temp;  (d)
                   if (_dsm[adsm] != _dsm[adsm+3]) {
                        multsm[a] = temp;                  (e)
                        a++;                               (f)
```

```
                    temp=0;                    (g)
                }
            }
        }
    return multsm;
}
```

Time complexity of (a): 1
Time complexity of (b): 1
Time complexity of (c): n
Time complexity of (d): n-1
Time complexity of (e): n/3
Time complexity of (f): n/3
Time complexity of (g): n/3
The sequential time complexity is T (n) = 1+1+n+ (n-1) +n/3+n/3+n/3 = O (n).

With parallel code with p processors, the parallel time complexity is (Cost) optimal parallel time complexity = sequential time complexity/number of processors = O (n)/p which is better than the sequential algorithms.

6.3 Limitation

The algorithm introduced above has some limitations. Every slave processors only allocated fixed number of rows of matrix, the elements number in each row is not fix. Therefore, it may cause that the buffer size sent to each slave processors is different.

If the sparse matrix elements do not allocate evenly in the matrix, it means some row contains lots of element, the others may only have one element or no element, if master processor sends the task to each slave processors by certain number of rows. It will result in unloading balancing for each slave processors. To solve the unload balancing problem, may apply dynamic load balancing technology, put all the tasks into a queue in the work-pool, if slave processor finishes its task, it will send the result back to master process and will ask master process for new task. About when the computation terminate, when the task queue is empty, we cannot consider all the tasks finished, if one or more processors are still running. If the task queue is empty, all the processors have already made the new request, and there are no more tasks to assign, then the computation is terminated.

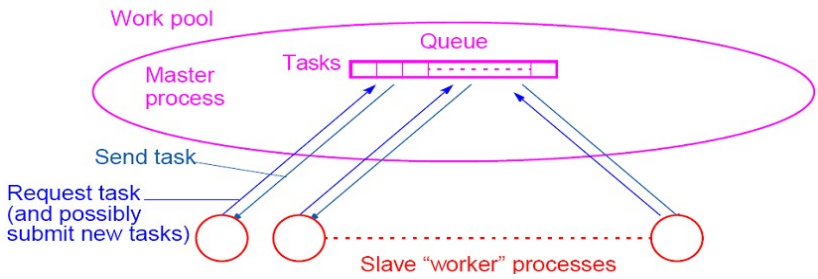

Figure 6.3

6.4 Bottleneck

When the spars matrix have lots of rows with one element in each row, when parallel the multiplication, the master processor has to send each row to each slave processor. In this situation, the computation for each process is very easy and simple, however the communication between master and slave processors is huge. It will slow down the speed.

7. Bibliography

[1]**Blaise,B.** (2009). *Lawrence Livermore National Laboratory.*Retrieved May 2009 from the World Wide Web: https://computing.llnl.gov/tutorials/parallel_comp/

[2] **Bruce Hendrickson**, **Robert Leland** and **Steve Plimpton**, *An Efficient Parallel Algorithm for Matrix – Vector Multiplication*, Sandia National Laboratories, Albuquerque, NM87185

[3] **Stoer, Josef; Bulirsch, Roland** (2002), *Introduction to Numerical Analysis* (3rd ed.), Berlin, New York: Springer-Verlag, ISBN 978-0-387-95452-3 .

[4] **L.M. Romero** and **E.L. Zapata**, *Data Distributions for Sparse Matrix Vector Multiplication*, Univ ersity of Malaga, J.Parallel Computing, vol. 21, no.4, April 1999

[5] **Martin Johnson** *Numerical Algorithm*, Lecture notes in Parallel Computing, IIMS Massey University at Albany, Auckland, New Zealand. 2009

[6] **Message Passing Interface** http://en.wikipedia.org/wiki/Message_Passing_Interface

[7] **R.Raz.** *On the complexity of matrix product*. SIAM Journal on Computing, 32:1356-1369, 2003

[8] **SPARSE MATRIX** http://en.wikipedia.org/wiki/Sparse_matrix

[9] **SPARSE MATRIX** http://baike.baidu.com/view/891721.htm

[10] **V. Pan**. *How to multiply matrices faster*. Lecture notes in computer science, volume 179. Springer-Verlag, 1985

[11] **Wang Shun** and **Wang Xiao Ge** *Parallel Algorithm for Matrix – Vector Multiplication*, Tsinghua University Library, CHINA