# SpMV

## Distributed Sparse Matrix-Vector Multiplication on Hama

### Introduction

In this article we will explore the problem of sparse matrix-vector multiplication, which can be written in form u = Av. Most computational algoritms spends large percent of time for solving large systems of linear equations. In general, system of linear equations can be represented in matrix form Ax = b, where A is matrix with n rows and n columns, b - vector of size n, x - unknown solution vector which we are searching. Some approaches for solving linear systems has iterative nature. Assume, we know the initial approximation of x = x0. After that we represent our system in form xn = Bxn-1 + c, where c - vector of size n. After that we can found next approximations of x and repeat this till the convergence. In real world most of matrices contain relatively small number of non-zero items in comparison to total number of matrix items. Such matrices are called sparse matrices, matrices which filled most with non-zero items are called dense. Sparse matrices arise when each variable from the set is connected with small subset of variables (for example, differential equation of heat conduction). So, this page will describe the problem of sparse matrix vector multiplication (SpMV) with use of Bulk Synchronous Programming (BSP) model implemented in Apache Hama project. As shown above, SpMV can be used in different iterative solvers for system of linear equations. Bulk Synchronous model proposes it's own smart way of parallelization of programs. We can specify input path for problem and number of peers. Framework reads the input and divides it between peers. Peers can be a processors, threads, separate machines, different items of cloud. BSP algorithm is divided in sequence of supersteps. Barrier synchronization of all peers is made after each superstep. The implementation of BSP(Apache Hama) contains primitives for defining peer number, communication with other peers with different communication primitives, optimizations of communication between peers, also it inherits most of features and approaches of Hadoop project.

### Problem description

As a sequential problem SpMV is almost trivial. But in case of parallel version we should think about some additional aspects:

1. Partitioning of matrix and vector components. This means, that we should split the input matrix and vectors by peers, if we want to have benefits from usage of parallel algorithm. Wise partitioning should be made or data locality won't be approached and communication time will rise very much or we will get great load imbalance and algorithm will be inefficient. 2. Load balancing. This means that each peer must perform nearly the same amount of work, and none of them should idle. 3. We must consider Hadoop and Hama approach for parallelization.

### Implementation tips

1. Framework splits the input file to peers automatically. So we don't need to perform mapping of matrix to peers manually. We only must define how matrix can be written to file and how it can be readed from it. If we create matrix, which consists from separate cells, framework will give some subset of cells to each peer. If we create matrix consisting from rows, framework will give subset of rows to each peer. The ways to influence on partitioning: creating different writables for matrices, overriding default partitioner class behavior. 2. We don't need to care about communication in case of row-wise matrix access. First of all, rows of matrix are splitted automatically by the framework. After that we can compute inner product of the vector and concrete matrix row, and the result can be directly printed to output, because it is one of the cells of result vector. In this case we assume, that peer's memory can fit two vectors. Even if we have million x million matrix and vector of size million, some megabytes will be enough to store them. Even if we split input vector the gain in memory will be insignificant.

### Algorithm description

The generic algorithm will contain one superstep, because no communication is needed:

1. Matrix and vector distribution. 2. Custom partitioning. 3. Local computation. 4. Output of result vector. 5. Constructing of dense vector.

In setup stage every peer reads input dense vector from file. After that, framework will partition matrix rows by the algorithm provided in custom partitioner automatically. After that local computation is performed. We gain some cells of result vector in bsp procedure, and they are written to output file. Output file is reread to construct instance of dense vector for further computation.

### Implementation

#### How to get

Implementation can be found in my GitHub repository Apache JIRA] and patch can be found in [https://issues.apache.org/jira/browse/HAMA-524 as soon as JIRA will become available. GitHub repository contains only classes related to SpMV. Before you start with SpMV make sure that you have followed this guide and set up environment variables and so on.

#### Optional additional setup

I considered two possible use cases of SpMV:

1. Usage in pair with `RandomMatrixGenerator`. 2. Usage with arbitrary text files.

In this section you will see how to use SpMV in this two cases. I propose the following directory structure for the following examples

```
/user/hduser/spmv/matrix-seq
/user/hduser/spmv/matrix-txt
/user/hduser/spmv/result-seq
/user/hduser/spmv/result-txt
/user/hduser/spmv/vector-seq
/user/hduser/spmv/vector-txt
```

Suffix `seq` denotes that directory contains sequence files. Suffix `txt` denotes that directory contains human-readable text files.

Also I defined some shell variables in .bashrc file of hadoop user to simplify following code snippets.

```
export HAMA_EXAMPLES=$HAMA_HOME/hama-examples*.jar
export SPMV=/user/hduser/spmv
```

First variable allows fast access to jar with hama examples, which plased in hama home directory, second variable is prefix in HDFS for tests in this tutorial. If you not defined this variables just substitute appropriate values into following scripts.

### Representation of matrices in text format

It was decided to allow users to work with SpMV through text files. So in this section I will describe text format for matrices. I decided to represent all matrices and vectors as follows: each row of the matrix is represented by row index, length of the row, number of non-zero items, pairs of index and value. All values inside rows are separated by whitespace, rows are separated by newline. Vectors are represented as matrix rows with arbitrary row index(not used). So, for example:

```
[1 0 2]    3 2 0 1 2 2
[0 0 0]  = 3 0
[0 5 1]    3 2 1 5 2 1
```

### Usage with RandomMatrixGenerator

`RandomMatrixGenerator` as a SpMV works with sequence file format. So, to multiply random matrix with random vector we will do the following: generate matrix and vector; convert matrix, vector and result to text file; view matrix, vector and result. This sequence is described by the following code snippet:

```
1:    hadoop dfs -rmr $SPMV/*/*
2:    hama jar $HAMA_EXAMPLES rmgenerator $SPMV/matrix-seq 6 6 0.4 4
3:    hama jar $HAMA_EXAMPLES rmgenerator $SPMV/vector-seq 1 6 0.9 4
4:    hama jar $HAMA_EXAMPLES spmv $SPMV/matrix-seq $SPMV/vector-seq $SPMV/result-seq 4
5:    hadoop dfs -rmr $SPMV/result-seq/part
6:    hama jar $HAMA_EXAMPLES matrixtotext $SPMV/matrix-seq $SPMV/matrix-txt
7:    hama jar $HAMA_EXAMPLES matrixtotext $SPMV/vector-seq $SPMV/vector-txt
8:    hama jar $HAMA_EXAMPLES matrixtotext $SPMV/result-seq $SPMV/result-txt
9:    hadoop dfs -cat /user/hduser/spmv/matrix-txt/*
   0         6 3 5 0.24316243288531214 2 0.638622414091597 3 0.5480468710898891
   3         6 2 5 0.5054043538570098 2 0.03911646523753309
   1         6 3 4 0.5077528966368161 5 0.5780340816354201 3 0.4626752204959449
   4         6 2 1 0.6512355661856207 4 0.08804976645891671
   2         6 2 4 0.7200271909735554 1 0.3510851368183805
   5         6 2 2 0.5848717104309032 3 0.0889791409798859

10:   hadoop dfs -cat /user/hduser/spmv/vector-txt/*
   0         6 6 0 0.3365077672167889 1 0.17498609722570935 2 0.32806410950648845 3 0.6016567879100464 4
0.786158850847722 5 0.6856872945972037
11:   hadoop dfs -cat /user/hduser/spmv/result-txt/*
   0         6 6 0 0.7059786044267415 1 1.0738967463653346 2 0.6274907669206862 3 0.35938205240905363 4
0.18317827331814918 5 0.24541032101100438
```

We got the expected result. So, now we will explain the meaning of each line in code snippet above.

Line 1: Clean up of directories related to SpMV tests.

Line 2-3: Generation of input matrix and vector. In this example we test 6x6 matrix and 1x6 vector multiplication

Line 4: SpMV algorithm.

Line 5: Deletion of part files from output directory at line 4. NOTE: `matrixtotext` will fail if this step will not be performed, because `result-seq` will containg part folder and `matrixtotext` don't know how to deal with it yet.

Line 6-8: Convertion of input matrix, input vector and result to text format.

Line 9-11: Showing the result.

### Usage with arbitrary text files

SpMV works with `SequenceFile`, so we need to provide tools to convert input and output of SpMV between sequence file format and text format. These tools are `matrixtoseq` and `matrixtotext`. This programs are included in example driver, so they can be launched like any other example. `matrixtoseq` converts matrix, represented in text file to sequence file format. Also this program gives choice to choose target writable: `DenseVectorWritable` and `SparseVectorWritable`.

```
Usage: matrixtoseq <input matrix dir> <output matrix dir> <dense|sparse> [number of tasks (default max)]
```

`matrixtotext` converts matrix from sequence file format to text file.

```
Usage: matrixtotext <input matrix dir> <output matrix dir> [number of tasks (default max)]
```

Now let's show some example. To use SpMV in this mode you should provide text files in appropriate format, as described above. Imagine that you need to multiply

```
[1 0 6 0]   [2]   [38]
[0 4 0 0] * [3] = [12]
[0 2 3 0]   [6]   [24]
[3 0 0 5]   [0]   [6]
```

First of all, you should create appropriate text files for input matrix and input vector. For input matrix file should look like

```
0 4 2 0 1 2 6
1 4 1 1 4
2 4 2 1 2 2 3
3 4 2 0 3 3 5
```

For vector file should be look like

```
0 4 3 0 2 1 3 2 6
```

After that you should copy these files to HDFS. If you don't feel comfortable with HDFS please see this tutorial. After you have copied input matrix into `matrix-txt` and input vector into `vector-txt`, we are ready to start. The following code snippet shows, how you can multiply matrices in this mode. Explanations will be given below.

```
1: hama jar $HAMA_EXAMPLES matrixtoseq $SPMV/matrix-txt $SPMV/matrix-seq sparse 4
2: hama jar $HAMA_EXAMPLES matrixtoseq $SPMV/vector-txt $SPMV/vector-seq dense 4
3: hama jar $HAMA_EXAMPLES spmv $SPMV/matrix-seq $SPMV/vector-seq $SPMV/result-seq 4
4: hadoop dfs -rmr $SPMV/result-seq/part
5: hama jar $HAMA_EXAMPLES matrixtotext $SPMV/result-seq $SPMV/result-txt
6: hadoop dfs -cat $SPMV/result-txt/*
    0         4 4 0 38.0 1 12.0 2 24.0 3 6.0
```

Line 1: Converting input matrix to sequence file format, internally consisting of `SparseVectorWritable`.

Line 2: Converting input vector to sequence file format, internally consisting of `DenseVectorWritable`.

Line 3: SpMV algorithm.

Line 4: We delete part files from output directory. NOTE: `matrixtotext` will fail if this step will not be performed, because result-seq will containg part folder and matrixtotext don't know how to deal with it yet.

Line 5: Convertion of result vector to text format.

Line 6: Output of result vector. You can see that we gained an expected vector.

## Possible improvements

1. Significant improvement in total time of algorithm can be achieved by creating custom partitioner class. It will give us load balancing and therefore better efficiency. This is the main possibility for optimization, because we decided, that using of row-wise matrix access i acceptable. Maybe it can be achieved by reordering of input or by customizing partitioning algorithm of framework.

## Literature

1. Rob H. Bisseling - Parallel Scientific computation. (chapter 4). 2. Steve Rennich - Block SpMV on GPU.