EXTENDED MATRIX-MARKET FILE FORMAT – AN EXTENSION TO THE STANDARD SCHEME

A.N. YZELMAN AND R.H. BISSELING

Contents

In this white paper, the Matrix–Market format is extended so that it can store distributed sparse matrices, and other related information which may be of interest during (for example but not limited to):

- parallel sparse matrix–vector (SpMV) multiplication,
- matrix reordering into (doubly) bordered block diagonal (BBD) or separated block diagonal (SBD) form,
- solving (linear) systems in parallel,
- matrix nonzero clustering.

This Extended-Matrix-Market (EMM) format is currently applied in the setting of a sparse matrix partitioner (Mondriaan), which reads in a (Matrix-Market) matrix file, partitions and/or reorders the corresponding matrix, and stores this information in EMM format. This file can then be used by e.g. parallel solvers, which previously would need several input files as input. There is no reason this scheme should be limited to a partioner-related setting, however; see Section 2.1

1. Basic format

The basics are derived from the basic Matrix Market file definition [2, 3]. Additions follow per file section.

Date: First version, November 2010.

1.1. **Header.** The original header is extended from
`%%MatrixMarket object format field symmetry`
to
`%%Extended-MatrixMarket object format field symmetry view`.

Originally, `object` could be one of `matrix` or `vector`. A new addition is `distributed-matrix` to indicate the file contains a matrix as well as an indication which nonzero should go to which processor. Also newly defined is the similar `distributed-vector`, and the `vector-collection`. This latter format stores a collection of vectors, which can be of varying lengths (as opposed to a matrix); this will be discussed in more detail in the following subsections. For brevity, `Extended-MatrixMarket` may be abbreviated to `EMM`.

The `format`, `field`, and `symmetry` fields remain unchanged. Their possible values are included here for completeness:

- `format`: `coordinate` or `array`;
- `field`: `real`, `double`, `complex`, `integer`, or `pattern`;
- `symmetry`: `general`, `symmetric`, `skew-symmetric`, or `hermitian`.

A new field has been introduced: `view`. Its possible values are `original`, `global` or `local`, and they refer to the indices of the matrices stored and how to relate these to a parallel distribution. In `original` mode, the indices of nonzeroes are exactly equal to those in the original, non-distributed, matrix. The `global` mode adds after each data element (nonzero) the number of the processor that the nonzero is distributed to; see Section 1.4. In `local` mode, the indices are adapted so that they correspond to the matrix originating from looking only at locally distributed nonzeroes, with any empty rows and columns removed from it.

For example, take the following 4×3 sparse matrix:

$$\begin{pmatrix} \mathrm{x} & & \mathrm{x} \\ & \mathrm{x} & \\ & \mathrm{x} & \mathrm{x} \\ \mathrm{x} & \mathrm{x} & \end{pmatrix}.$$

The nonzeroes, denoted by a single mark "x", can be distributed over two processors "0" and "1":

$$\begin{pmatrix} 1 & & 1 \\ & 0 & \\ & 1 & 0 \\ 1 & 0 & \end{pmatrix}.$$

In original mode, the matrix would be stored as is; in local mode, the following two matrices would be stored *instead*:

$$\begin{pmatrix} 0 & \\ & 0 \\ 0 & \end{pmatrix} \text{ and } \begin{pmatrix} 1 & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}.$$

After the header line follow the size line, the distributed information lines, and the values lines; as described in the following three subsections. After the last value, a new header line may signal an additionally stored object. This is described in detail in Section 2. In between the header line and the following size line, commentary lines may be inserted, where each such comment line is preceded by a `%`. These lines may be used to describe where the stored matrix (or other object) originates from, along with author and contact information. It is also recommended (but not required) that each additional object is mentioned in these comments, along with a suitable description.

1.2. **Size line.** If the format is `array`, stored is:
`m n P`,
or if the format is `coordinate`:
`m n nz P`.
The parameter `P` is added only when the object is distributed. The integers m, n refer to the matrix size, whereas P refers to the number of processors the matrix is distributed over. The dimension n is not stored when the object type is a (distributed) vector. It is also not given in the case of a `vector-collection` object; m then corresponds to the number of variable-length vectors which will follow. In the above example, the following would be the size line: `4 3 7 2` (in the case of distributed-matrix, coordinate).

1.3. **Distributed information.** Next, in the case of a distributed object (that is, `distributed-matrix` or `distributed-vector`) in `original` or `local` view, a Pstart vector is expected, signifying which nonzeroes are distributed to which processor. This data is *not* expected while in the `global` view. The vector is of size $P + 1$. Its first value is always 1, and its last value is always $nz + 1$ (or $m \cdot n$ or m in the appropiate cases of `array` and `distributed-vector`). The bounds of the ith partition is Pstart[i] (lower, inclusive) to Pstart[i+1] (higher, exclusive), where Pstart= $(\text{Pstart}_1, \text{Pstart}_2, \ldots, \text{Pstart}_{P+1})$; that is, we start counting from 1. Using the above example, the following would be in file (assuming parts are stored in increasing part IDs):

```
1
4
8
```

1.4. **Values.** Next follow the matrix values. In case of `array`, exactly $m \cdot n$ values are expected. This is only recommended for dense matrices, or vectors. In the case of `coordinate`, expected are `nz` triplets of the form

`i j v`,

where `i` gives the row index of the nonzero, `j` its column index, and `v` its value. *All index values are, like in the original Matrix-Market, expected to be* 1 *based.* Multiple triplets for the same coordinates (i, j) *are* allowed; in such a case the values are taken added cumulatively. In the case of a `vector-collection`, the data consists of multiple vectors; for each of these vectors, the vector data is preceded by its own size line.

Recall that in the case of the `global` view, a processor number is added after the numerical values; in the case of the coordinate scheme and the array scheme, this will look as follows:

`i j v p`, or `v p`.

1.5. **Example: vector-collection.** Since the `vector-collection` type is new, we review this object type in more detail here.

```
%%Extended-MatrixMarket vector-collection array integer general original
3
2
6
2
3
9
12
4
2
9
7
```

The above vector-collection yields a vector containing 3 other vectors (first size line, line 2). The first vector contains 2 elements (first size line of the first vector element, line 3); which are $(6, 2)$. Line 6 gives the size of the second vector, 3, followed by the data $(9, 12, 4)$. Similary, a vector of length 2 follows, yielding the following "matrix":

$$\begin{pmatrix} 6 & 2 & \\ 9 & 12 & 4 \\ 9 & 7 & \end{pmatrix}.$$

2. Adding extra information

When the objective of the matrix distribution is a SpMV multiplication, input and output vectors have to be distributed as well. These distributions can be stored as follows. First, to signal more information follows, the following additive header is defined:

`%%name object format field symmetry view`.

This is followed by size and value information as defined above.

2.1. **Complete problem descriptions.** Not related to distributed programming a priori, the scheme described here extends to more general problem descriptions as well. For example, a problem may consist of finding a vector x such that $(A+C)x = Mb$;: the matrices C, M and the vector b can be stored in the same file describing the matrix A by use of the method described above:

```
%%EMM matrix pattern general coordinate original
(dimensions, data for A)
%%C matrix real general coordinate original
(dimensions, data for C)
%%M matrix real symmetric coordinate original
(dimensions, data for M)
%%b vector real general array original
(length, data for b)
```

This method allows a single file to represent an entire problem.

3. Mondriaan-related information

This section will describe all information the Mondriaan package [4] writes in Extended-MatrixMarket format after partitioning an input matrix. The main object will be the `distributed-matrix` in `original` view. Other information, such as a local view or corresponding vector distributions, are found within the same file as described in the following. Information on permutations of course is *not* found in the output when the permute option is turned off.

3.1. **Local view of distributed matrix.** Usually, a sparse matrix–vector multiplication (SpMV) algorithm requires that processor-local matrices are indeed local; it is wasteful and memory-wise not scalable to store $m \times n$ matrices at each of the p processors during SpMV (with m, n the size of the original input matrix A). Hence, parallel SpMV usually transformed the distributed matrix in original view to a local view, and, while calculating, also derived other useful index arrays to be used during parallel SpMV. Now, Mondriaan does this translation itself. A local view is written to the matrix object `Local-A`, for which its full header is:
`%%Local-A distributed-matrix coordinate real general local`

The original Mondriaan also gave this information in the Cartesian submatrices file, in a format alike the original Matrix-Market form. The local view described here supersedes that file format. Mondriaan additionally produced another "kind of" global view; in one of the output files the nonzero entries were replaced with the processor numbers. To be complete and potentially downwards compatible with applications using Mondriaan, this information is also saved under the name `Global-A` (in `original` view, since the processor indices *replace* the nonzero values instead of appending them after the nonzeroes as would have been the case in true `global` view).

3.2. **Mapping local matrix entries to the original indices.** Intuitively, the biggest requirement when having access to local representations of a distributed matrix, is to be able to map the local entries to the locations in the original matrix. However, the locally stored submatrices do not have to be of the same size; generally, they vary around the optimal size $m/p \times n/p$. Moreover, a single row or column can be mapped to multiple processors through different nonzeroes; adding up all local submatrix sizes is likely to exceed $m \times n$. This entails that the local-row-to-original-row (and local-column-to-original-column) index translations cannot be intuitively stored in a $p \times m/p$ ($p \times n/p$) dense matrix, nor in a distributed vector of size m.

The newly defined `vector-collection` object provides relief; p vectors of variable size can be stored this way. Two of these objects are stored under the following headers:
`%%LocalRow2Global vector-collection array integer general original`
`%%LocalCol2Global vector-collection array integer general original`

These two collections of index translation vectors give full information regarding the correspondence of local matrix entries with respect to their location in the original, nondistributed, matrix.

3.3. **Vector distributions.** In the case of the input and output vector distribution, see, for example, [1], we have the following object:
`%%Input-vector distributed-vector array integer general global`
`1 n`
followed by the (line-separated) distribution vector, indicating which entry is distributed where. Note that Mondriaan does not read in any vector information; the input vector distribution thus only maps *indices* to processors (and not the actual input vector elements to processors). Similarly for the output distribution, but then with the following header:
`%%Output-vector vector array integer general global`.

The global view does not immediately make clear how many elements from the input or output vector are distributed to a given processor. For convenience, this data is written to file as well; the relevant size vectors (of length p) are available under the names `InputVectorLengths` and `OutputVectorLengths`.

3.4. **Required translations during parallel SpMV.** Looking from the perspective of a single processor out of the p available, each processor requires, apart from a local representation of the distributed matrix, information on the nonzeroes distributed to its fellow processors. To be precise, for each local matrix column, information is required whether the corresponding element from the input vector is local, and if not, at which processor it resides. Furthermore, for all local matrix columns, the index of the corresponding element from the input vector at the processor the element resides at, must be known. Two similar translation arrays must be known for the local matrix rows in relation to the output vector distribution. Mondriaan now writes this information in Extended-MatrixMarket format, using again the `vector-collection` object types, under the following headers:
`%%LocalRow2Processor vector-collection array integer general original`
`%%LocalRow2Index vector-collection array integer general original`
`%%LocalCol2Processor vector-collection array integer general original`
`%%LocalCol2Index vector-collection array integer general original`.

3.5. **Permutation vectors.** To permute the input matrix into, e.g., doubly SBD form [5, 6], information on which row (column) should be first in the permuted form is stored in the *permutation vectors*. Similarly to the vector distributions, these can be stored after the additive header
`%%Row-permutation vector array integer general original`,
or,
`%%Column-permutation vector array integer general original`,
respectively.

3.6. **Permuted matrix.** The above row and column permutations yield the possibility of permuting the input matrix A according to PAQ, where P, Q are permutation matrices derived from the row and column permutations, respectively. This matrix can also be directly stored under the name `PAQ`, as a `matrix` object in `original` view, with the remaining flags identical to those of the original matrix.

In its permuted form, matrix separator blocks may be seen in-between two partition blocks (in SBD mode) or before or after those partition blocks (reverseBBD or BBD). The starting index of each separator block and partition block is stored in the `Row-boundaries` and `Column-boundaries` integer vectors. Since Mondriaan partitions by using recursive bipartitioning, a hierarchy of (separator) blocks can be derived, in the form of a binary tree. See also [6]. Each block of nonzeroes in the permuted matrix thus has a single parent, and each element (except for the very last one) from the boundaries vector defined above corresponds to a single such block. The tree therefore can be stored by setting a vector of integers pointing to parent block indices as they appear in the boundary vectors, or 0 if the current block corresponds to the root of the tree. We thus end up with an additional two integer vectors: `Row-hierarchy` and `Column-hierarchy`.

References

[1] R. H. Bisseling and W. Meesen, *Communication balancing in parallel sparse matrix-vector multiplication*, Electronic Transactions on Numerical Analysis, 21 (2005), pp. 47–65.

[2] Ronald F. Boisvert, Roldan Pozo, Karen A. Remington, Richard F. Barrett, and Jack J. Dongarra, *Matrix market: a web resource for test matrix collections*, Proceedings of the IFIP TC2 (1997), pp. 125–137.

[3] Ronald F. Boisvert, Roldan Pozo, and Karin A. Remington, *The Matrix Market Exchange Formats: Initial Design*, NISTIR, 5935 (1996).
[4] B. Vastenhouw and R. H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, SIAM Rev., 47 (2005), pp. 67–95.
[5] A. N. Yzelman and Rob H. Bisseling, *Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods*, SIAM Journal on Scientific Computing, 31 (2009), pp. 3128–3154.
[6] A. N. Yzelman and Rob H. Bisseling, *Two-dimensional cache-oblivious sparse matrix–vector multiplication*, Pre-print, October 2010. URL: `http://www.math.uu.nl/people/bisseling/`

Appendix A. Example Extended Matrix-Market files

Here we give listings of a very small example Extended Matrix-Market matrix. Both the original form and a distributed form (after processing by Mondriaan) are given.

Original (Matrix-Market)

```
%%MatrixMarket matrix coordinate real general
% This is the example matrix from Fig. 4.4 (p. 177) in:
% Parallel Scientific Computation: A structured approach using BSP and MPI
% by Rob H. Bisseling (Oxford University Press 2004)
5 5 13
1 2 3.0
1 5 1.0
2 1 4.0
2 2 1.0
3 2 5.0
3 3 9.0
3 4 2.0
4 1 6.0
4 4 5.0
4 5 3.0
5 3 5.0
5 4 8.0
5 5 9.0
```

Original (EMM)

```
%%Extended-MatrixMarket matrix coordinate real general original
% This is the example matrix from Fig. 4.4 (p. 177) in:
% Parallel Scientific Computation: A structured approach using BSP and MPI
% by Rob H. Bisseling (Oxford University Press 2004)
5 5 13
1 2 3.0
1 5 1.0
2 1 4.0
2 2 1.0
3 2 5.0
3 3 9.0
3 4 2.0
4 1 6.0
4 4 5.0
4 5 3.0
5 3 5.0
5 4 8.0
5 5 9.0
```

Mondriaan output

```
%%Extended-MatrixMarket distributed-matrix coordinate real general original
% This is the example matrix from Fig. 4.4 (p. 177) in:
% Parallel Scientific Computation: A structured approach using BSP and MPI
% by Rob H. Bisseling (Oxford University Press 2004)
% File generated by running Mondriaan on A=../tests/test_Mondriaan.mtx with p=2 and \epsilon=0.2
5 5 13 2
1
```

```
8
14
1 2 3
1 5 1
2 1 4
2 2 1
4 1 6
4 4 5
4 5 3
5 5 9
5 4 8
5 3 5
3 4 2
3 3 9
3 2 5
%%PAQ matrix coordinate real general original
5 5 13
1 2 3
1 3 1
2 1 4
2 2 1
3 1 6
3 4 5
3 3 3
4 3 9
4 4 8
4 5 5
5 4 2
5 5 9
5 2 5
%%Row-boundaries vector array integer general original
4
1
4
4
6
%%Row-hierarchy vector array integer general original
3
2
0
2
%%Col-boundaries vector array integer general original
4
1
2
5
6
%%Col-hierarchy vector array integer general original
3
2
0
2
%%Local-A distributed-matrix coordinate real general local
```

```
5 5 13 2
1
8
14
1 1 3
1 2 1
2 3 4
2 1 1
3 3 6
3 4 5
3 2 3
1 1 9
1 2 8
1 3 5
2 2 2
2 3 9
2 4 5
%%LocalRow2Global vector-collection array integer general original
2
3
1
2
4
2
5
3
%%LocalCol2Global vector-collection array integer general original
2
4
2
5
1
4
4
5
4
3
2
%%Global-A matrix coordinate integer general original
5 5 13
1 2 1
1 5 1
2 1 1
2 2 1
4 1 1
4 4 1
4 5 1
5 5 2
5 4 2
5 3 2
3 4 2
3 3 2
3 2 2
```

```
%%Row-permutation vector array integer general original
5
1
2
4
5
3
%%Column-permutation vector array integer general original
5
1
2
5
4
3
%%Input-vector distributed-vector array integer general global
5 2
1 1
2 2
3 2
4 1
5 1
%%Output-vector distributed-vector array integer general global
5 2
1 1
2 1
3 2
4 1
5 2
%%OutputVectorLengths vector array integer general original
2
4
3
%%LocalRow2Processor vector-collection array integer general original
2
3
1
1
1
2
2
2
%%LocalRow2Index vector-collection array integer general original
2
3
1
2
3
2
2
1
%%InputVectorLengths vector array integer general original
2
4
```

```
3
%%LocalCol2Processor vector-collection array integer general original
2
4
2
1
1
1
4
1
1
2
2
%%LocalCol2Index vector-collection array integer general original
2
4
1
3
1
2
4
3
2
2
1
```