საქართველოს მეცნიერებათა ეროვნული აკადემიის მოამბე, ტ. 14, №3, 2020
BULLETIN OF THE GEORGIAN NATIONAL ACADEMY OF SCIENCES, vol. 14, no. 3, 2020

Informatics

Sparse Matrix Formats with Two Types for Real Values

Koba Gelashvili[*] and Vakhtang Laluashvili[**]

[*] *School of Business, Computing and Social Sciences, St. Andrew the First-Called Georgian University of the Patriarchate of Georgia, Tbilisi, Georgia*
[**] *Public Service Development Agency, Tbilisi, Georgia*

(Presented by Academy Member Ramaz Khurodze)

In existing sparse matrix formats, the only one real type is being used for all values. In practice, as for matrices taken from different collections, values belong to two or more types of real numbers, thereby creating the opportunity to save used memory (up to 50-75%). The presented paper describes main implementation details of sparse formats with two value-types, which without fail save the used memory without significantly complicating the program code. Both, static and dynamic data structures are considered. Intentionally, the case of symmetric, positive-definite sparse matrices used in conjugate gradient method is considered. © *2020 Bull. Georg. Natl. Acad. Sci.*

Sparse matrix, mapped matrix, compressed sparse row, boost library, conjugate gradient method, matrix-vector multiplication, b+ tree

The existing set of sparse formats is quite large. In the development the mathematical side was dominant. The software side has been developing as much as it was necessary for the software support of the mathematical idea and implementation of the formats. In the presented paper, the possibility of developing of even more memory-efficient sparse formats is considered, based on the well-known approaches on data types and data structures.

Currently used sparse formats are monotypic: either every value of matrix is stored with a single precision or with double precision, or using some other precision (e.g., half or four-fold precision etc). In our experiments (see material at https://github.com/kobage) some of the widely spread formats are modified. Both for static and dynamic formats, modifications use two real value-types and are more memory-saving in any possible scenario. But the matrix-vector multiplication algorithm becomes slower. In the case of static sparse formats, we consider CSR [1]. In the case of dynamic formats besides of using two value-types, we dicuss other, more efficient way of obtaining more economical formats using recent advances in the implementation of associative ordered containers [2]. Even though we have used modern features of C++, the design and the code is pretty simple to understand.

The second section is devoted to the recognition of types (float or double) of real numbers which are being read from the file. This task can not be solved by some elementary method. Therefore we had to modify one of the standard library function of the C language.

In the third section, we discuss CSR format and its two modifications. We outline what components the particular format should consist of and the key issues of their implementations.

The fourth section is devoted to sparse formats, based on dynamic data structures. Namely, we discuss several ways of making Mapped Matrix format from boost library [3] more memory-saving. Mapped Matrix and similar formats are useful in situations where the usage of CSR format is problematic (huge matrices or/and there is the frequent necessity to change the entries).

The last section makes conclusions and considers the possible directions of future research.

Recognition the Types of Entries, Being Read from the File

Matrices from the famous libraries [4, 5] with arrangement of non-zero entries not subjected to any regularity, are considered in our experiments. The values of these matrices are located inside the file with extension ".mtx". In order to use two real types (float and double) for the matrix values, when building the matrix class object from data contained in the file we have to do very frequent answer to the question: could a given real number be stored with the type real or float? The question is fairly simple, but not its solution. Any library function, transforming string to real number, generates numbers of some predefined type. This type is indicated in the function's name. In C language unary functions: atof, atoi, atol, strtod – the type is indicated by the last symbol in function name. E.g., after the following statements

```
char buffer[256] = "45.1";
double x = atof(buffer);
```

the x variable occupies at least two-fold more memory than it needs in reality. The type float (or even half) would be better suited to this end.

We took atof from http://www.beedub.com/Sprite093/src/lib/c/stdlib/atof.c and add the second boolean parameter, named isFloat to its open source implementation. isFloat is used to determine the type of the real number, transformed from string. E.g., for the same buffer variable with the same "45.1" value after the following call:

```
double x = atof(buffer, isFloat);
```

isFloat will be true. Now we are able to take this fact into account when filling the sparse matrix, passing x as a single precision number. Furthermore, if buffer would have the value "45.0902391", or "45.0e+102", then the variable isFloat would have stayed false. Consequently, x should be passed into sparse format as the double precision number.

The original atof is a C-language function. Because its modification is frequently used to fill sparse matrix, we took the advantage of C++ language's features and pass the parameter "buffer" by reference, which allows us to process the next entry immediately after the function is returned. Otherwise, the buffer variable would have stayed on the first character of the given entry and we would have had to traverse this string again to find the next entry.

Modifications of the CSR Format, Adapted for Usage with Two Value Types

Sparse matrix – vector multiplication is one of the frequently used computational procedure. Hence, all formats support it, but CSR (compressed sparse row) is the most efficient [6]. Beside, CSR is one of the most economical ones from the point of view of the used memory. There are generalizations of CSR, including the ones concerning parallel programming [7, 8]. But CSR format has two drawbacks: it is slow in applications that frequently

change elements in matrix, and is unusable for the huge sparse matrices, because it may not be possible to allocate the required memory. Existing generalizations of CSR inherit the second drawback. For simplicity purposes, we will stay in the serial case during our considerations.

CSR format stores the whole information about matrix A in three arrays AA, JA, IA. AA consists of non-zero elements of matrix A, elements of array JA represent column indexes in matrix A of the corresponding elements of AA array. Both arrays have length nnz, where nnz denotes the number of non-zero entries. Array IA is defined recursively: IA[0] = 0 and IA[i] = IA[i - 1]+(number of non-zero elements on the (i - 1)-th row). The total number of elements in the arrays is 2 *nnz + n + 1. To calculate needed memory, it is essential that the last two arrays are of integer type. One of the classes in our project is implemented exactly as described here. int32_t is used as the integer type. This more than necessary in most cases. double is used as the real type (for values of the array AA), In every modification of CSR, the matrix-vector multiplication is provided by virtual functions. This function does not return, the result is written in the different vector, passed as a parameter.

Different classes correspond to different modifications of CSR. Each class has the additional virtual function, which writes the matrix diagonal in a dense vector. This function is used to create the simplest preconditioner. Our set of tests, where CG solves an algebraic equation system, is not large, but the matrices are large in size and without preconditioning some tests failed to solve. Hence, we think that sparse matrix libraries should propose preconditioners, as it is unrealistic to expect that consumers themselves will be able to program the preconditioner. It should be noted here that the right-hand sides of the systems of equations consists only of ones, as in [9].

We are interested in such a modification of CSR, that gives us opportunities of choosing the types for values. On the other hand, the used memory ought to be decreased (or stayed the same) without fail. The diversity of types is ensured by using the templates of functions and classes. This is a strong aspect of the C++ language and recently it has been frequently used in creating sparse matrix libraries [10].

From a programming standpoint, it is better to store lengthes of rows in IA array even in the classical realization of CSR. This complicates nothing in the implementation. Instead, for the vast majority of sparse matrices it is completely sufficient to use type int16_t and often even int8_t to store lengthes of rows. Therefore, it is advisable to use a class template in CSR implementations. With respect to modified CSRs with two value-types, the usage of the template parameter is an absolute necessity, since we need to simultaneously store two attributes in a single integer number – the quantity of the floats and the quantity of doubles for each row.

Let us denote the variant of CSR which distinguishes the types of values by lm_CSR, emphasizing the fact that we are going to use less memory. We need the following arrays for this format: AAd – to store real elements of the double precision; AAs – to store real elements of the single precision; JAd – to store indexes of columns of doubles; JAs – to store indexes of columns of floats; IA – to store the quantities of doubles and floats for each row.

Let us discuss the change of the used memory. Firstly, two pointers were added to two new arrays. This is O(1) change and we may pay no attention to it. First four arrays contain exactly the same number of elements as AA and JA arrays (of CSR). IA array has (n + 1) elements and consists of int32_t type numbers. IA array in lm_CSR has n elements and and its elements types are determined when reading the elements from a file into the buffer. E.g., if the maximum number of non-zero elements in rows is less than 16, then the uint8_t is enough. If the number is less than 255, then the uint16_t is enough, but if the number is less than 65 535, then we would require uint32_t type. When creating the

sparse matrix object, the parser located in the constructor determines the template parameter in this way using a rather old and experienced technic: If the constructor decides that the number of bits in the class template type should be equal to "bits", then for each row of the matrix we count quantities of doubles and floats, denote them by rd and rs respectively, and the corresponding element in IA becomes (rd << (bits / 2)) + rs. In the sequel, the number of doubles and floats are easily deducted from elements of IA using easy arithmetic.

The experiments show that the modified format is unconditionally economical in terms of saving memory. However, it is slower when multiplying symmetric positive-definite matrix by dense vector. This is caused by the fact that for each symmetric matrix, only the upper triangular part is stored, and therefore it is necessary to find the diagonal elements and process them separately, which is a rather expensive operation.

To reach a compromise between memory and speed, the diagonal in the modified format should be stored in the seaparate array (all elements with double precision). Anything else should remain as in lm_CSR. In terms of memory, this approach is definitely better than both CSR and lm_CSR, because the diagonal array does not need indexes. Speed is indeed compromise.

Besides, the last modification reduces even more the main drawback of CSR, that creation of too long arrays is problematic. Indeed, now the elements of two arrays will be divided into five arrays.

Mapped Matrix

Sparse matrix formats based on the dynamic data structures are relatively slow with respect their static counterparts, but are free from two major drawbacks of CSR.

Mapped Matrix [3] format from The Boost library is easier adjustable to two-typed value format than the CSR. Mapped Matrix is a template class, therefore, one object (with template parameter float) will be created for floats, and the second for doubles. It is also easy to program the matrix -vector multiplication and as a result we will get more economical format in a sense of a memory.

The implementation of Mapped Matrix is based on the red-black tree structure. The company Google recently has undertaken a project [2] and has created associated sorted containers, which are created based not on the red-black tree, but the B+ tree. They have the same interface as their counterparts from the C++ STL library (based on red-black tree), but are much more efficient with respect both memory and speed.

Therefore, currently, the first-order task to modernize the Mapped Matrix format is not to increase the number of value types in matrix, but to replace the base data structure itself.

Conclusions

Our experiments show that developing more economical matrix formats with respect used memory, is a promising direction. Using two value-types does not pose substantial impelementation difficulties.

However, in the static case new modifications need further development to improve the speed for the matrix-vector multiplication. There are much more resources available when the sparse format is created in case of dynamic data stucture. In this case, the first order task in the future research is to replace the basic data structure from the red-black tree to b+ tree, then using two or even more value-types.

ინფორმატიკა

მეჩხერი მატრიცების ფორმატები ნამდვილი მნიშვნელობების ორი ტიპით

კ. გელაშვილი* და ვ. ლალუაშვილი**

** საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახ. ქართული უნივერსიტეტი, ბიზნესის, კომპიუტინგისა და სოციალურ მეცნიერებათა სკოლა, თბილისი, საქართველო*
*** სახელმწიფო სერვისების განვითარების სააგენტო, თბილისი, საქართველო*

(წარმოდგენილია აკადემიის წევრის რ. ხუროძის მიერ)

მეჩხერი მატრიცების არსებულ ფორმატებში, ყველა მნიშვნელობისთვის გამოიყენება მხოლოდ ერთი ნამდვილი ტიპი. პრაქტიკაში, როგორც სხვადასხვა კოლექციიდან აღებულ მატრიცებში, მნიშვნელობები ეკუთვნის ნამდვილი რიცხვების ორ ან მეტ ტიპს, რაც ქმნის გამოყენებული მეხსიერების დაზოგვის საშუალებას (50-75%-მდე). წარმოდგენილი ნაშრომი აღწერს მნიშვნელობების ორი ტიპის მქონე მეჩხერი ფორმატების განხორციელების ძირითად დეტალებს, რომლებიც უპირობოდ ზოგავენ გამოყენებულ მეხსიერებას პროგრამული კოდის საგრძნობი გართულების გარეშე. როგორც სტატიკური, ასევე დინამიკურ მონაცემთა სტრუქტურები არის განხილული. ასევე, განხილულია შეუღლებულ გარადიენტთა მეთოდში სიმეტრიული, დადებითად-განსაზღვრული მეჩხერი მატრიცების გამოყენების შემთხვევა.

REFERENCES

1. Saad Y. (2003) Iterative methods for sparse linear systems, 2nd edition, SIAM.
2. cpp-btree. see https://code.google.com/archive/p/cpp-btree/
3. boost library. See http://www.boost.org/doc/libs/1_60_0/libs/numeric/ublas/doc/matrix_sparse.html
4. Davis A., Hu Y. (2011) The University of Florida sparse matrix collection. ACM. *Trans. on Mathematical Software*, 38(1): 1-25.
5. The SuiteSparse matrix collection. See https://sparse.tamu.edu/
6. Chalauri G., Laluashvili V., Gelashvili K. (2018) Jagged non-zero submatrix data structure. *Transactions of A. Razmadze Mathematical Institute,* 172: 7-14.
7. Karakasis V., Gkountouvas T., Kourtis K., Goumas G. and Koziris N. (2013) An extended compression format for the optimization of sparse matrix-vector multiplication. *IEEE Trans. Parallel Distrib. Syst.* (TPDS) 24(10): 1930-1940.
8. Elafrou A., Karakasis V., Gkountouvas T., Kourtis K., Goumas G., Koziris N. (2018) Sparsex: A library for high-performance sparse matrix-vector multiplication on multicore platforms. *ACM Transactions on Mathematical Software* (TOMS) 44 (3): 1-32.
9. Lin CJ., Moré J. (1999) Incomplete Cholesky factorizations with limited memory. *SIAM J. Sci. Comput.,* 21(1): 24-45.
10. Sanderson C., Curtin R. (2019) Practical sparse matrices in C++ with hybrid storage and template-based expression optimisation. *Mathematical and Computational Applications,* 24 (3):70.

Received July, 2020