

Sparse Matrices in Matlab*P

Final Report

Submitted by: **Stu Blair**

Date: **8 May 2003**

Introduction and Motivation

The purpose of this project was to provide sparse matrix functionality to the users of MATLAB*P. It would be difficult to exaggerate the importance of sparse matrices in the realm of large-scale scientific computing. For problems such as those that require the discrete approximation to partial differential equations¹, very large and sparse matrices naturally arise. It is important, and in many cases essential, that the sparsity of these matrices is taken advantage of in the solution process. While MATLAB*P provides a rather wide set of tools for the creation and manipulation of dense matrices, no support hitherto was provided for sparse matrices².

While it was at the project outset (and still is) viewed as unrealistic to provide the complete sparse matrix functionality that is enjoyed by users of the commercial, serial version of Matlab™³ in a short amount of time, it was hoped that, at least a solid foundation for the further development of sparse matrices in MATLAB*P could be established. To that end, a set of sparse matrix construction and manipulation routines were designed and implemented. Additionally, the necessary routines were created to allow explicit transfer of whole sparse matrices between the client and server processes.

Sparse Matrix Data Structure

Ron Choy provided the basic sparse matrix object. The class: PPSparseMatrix is publicly derived from PPMatrix – the same as PPDenseMatrix, the class that underlies

¹ Read: many models of almost any natural physical system

² For a detailed description and history of the design and architecture of MATLAB*P, please see references [1]-[3].

³ Matlab is a trademark of The Mathworks, Natick, MA.

the `ddense` Matlab class⁴. The data is stored as a linear array of linked lists – one list for each column. Support is provided at this time only for column-wise distribution. The linked lists are exactly those provided by the Standard Template Library. The parameterized data type in the linked list is a structure that contains an integer row index and either a single or double precision variable that holds the value of each non-zero sparse matrix element. (complex values are also supported)

Implemented Functionality

The following function calls are provided --- starting with matrix construction:

1. `sparse(M,N*P)`: While Matlab supports numerous versions of this constructor, `MATLAB*P` as of this writing only supports this one version. As can be inferred, the columns are distributed among the processors. Use of this call results in an all-zeros sparse matrix.
2. `sprand(M,N*P,density)` and `sprandn(M,N*P,density)`: These function calls are similar to those provided in Matlab. A sparse, column-distributed matrix is constructed. The approximate percentage of non-zero entries is determined by the argument ‘density’, with the non-zero value chosen from at random – with the distribution depending on whether `sprand` or `sprandn` is called.
3. `speye(M,N*P)`: This constructor returns a sparse identity matrix of size $M \times N$. Despite the use of both M and N , it is required by the function call that both M and N be equal, and that the matrix be column distributed.

The following matrix manipulation routines are provided:

4. `spones(A)`: For this function, A is a distributed sparse matrix. The result being, as in Matlab, all of the non-zero entries of A are changed to ones⁵.
5. `full(A)`: Where A is a distributed sparse matrix, the returned object is a `ddense` matrix with the same dimensions and values as the corresponding sparse matrix.
6. `plus`: The matrix addition operator was overloaded for sparse matrices. For this project, the mixed case was not implemented. Only sparse matrices may participate. Comments on the design of mixed-case operators will be discussed in later sections. The result of this operation is, obviously, a `dsparse` object representing the sum of two `dsparse` operands.
7. `minus`: The same as `plus`, but with subtraction.

The following auxiliary routines are provided:

⁴ Despite the usual hesitance to provide specific program implementation details, I thought it would be useful to provide some specific information for the benefit of any potential readers who may consider building upon (or even replacing completely) the work reported here.

⁵ Not the most challenging of functions, but part of the interface nonetheless, and a nice break from the more difficult functions.

8. `nnz(A)`: This function returns the total number of non-zero entries in the distributed sparse matrix `A`.
9. `pp2matlab(A)`: This function transfers to the front-end, the `dsparse` matrix `A`. It is then assigned to a conventional Matlab sparse matrix, or output to the terminal.
10. `matlab2pp(A)`: This function accepts a Matlab sparse matrix `A`, transfers this matrix to the server back-end and assigns it to an already constructed, appropriately dimensioned distributed sparse matrix. Currently this function does not work for complex sparse matrices `A`.

The implementation of the ‘plus’ and ‘minus’ operators revealed the requirement to make minor modifications to the `MatrixManager` object. The functionality was not provided for creating a copy of an existing `dsparse` matrix object – or, for copying the values of a `dsparse` matrix into a second, existing `dsparse` matrix. This capability was added as a part of this project.

The implementation of the above calls required, in general two separate operations. First, the appropriate Matlab scripts were provided and/or modified. This is done in conformance with the ability (or, in this case requirement) to overload certain operations on user-defined data-types (`layout`, `ddense` and `dsparse`). While these Matlab scripts will not be described in detail in this report, the proper design implementation of these scripts are important in ensuring that the proper back-end functions are called⁶. Secondly, the back-end functions are written. Most of the back-end functions that pertain to sparse matrices are contained within the package `PPSparse.cc`.

For the case of `matlab2pp(..)` modification was also required of the `ppclient.cc` function. The issue that arises is that the sparse matrix provides a different interface through the `ppclient .mex` file as it is passed back to the server. In addition to the pointers to the data in the array, two additional pointers are provided: One to an array of integers that represents the row index of the non-zero data members, the second is to an array of length $n+1$ ⁷ that contains the cumulative number of non-zero entries (counted column wise) in the matrix. Using this data it is possible to completely re-construct the sparse matrix.

⁶ An illustrative example of the full sequence of events that occurs upon the call of `sprandn(..)` with a `layout` argument is provided in the mid-term report on this project, available on the project web-site [6].

⁷ Where `n` is the number of columns in the matrix.

Plus and Minus Operators

The functionality provided was more than enough to allow the creation of sparse matrices, and the testing of the effectiveness of the only arithmetical algorithm that was implemented. In addition to creating a set of random matrices and testing the arithmetic operations on them, a sample of test matrices was imported from the University of Florida Sparse Matrix collection. The basic idea of this test is to check the performance of the plus and minus operator as compared to serial Matlab. In all, the results were disappointing. The matrix addition operation was significantly slower than that which can be obtained using the serial version of Matlab on a single machine. If anything good can be said about this performance it is that: it scales well. However, this is hardly surprising given the fact that the actual operations are embarrassingly parallel. The results are given in Figure 1. A benefit is obtained from the fact that larger matrices can be constructed and stored on the server, than that which can be formed on a single serial machine. Perhaps with the adoption of more efficient algorithms (or algorithms and data structures) the performance of the server will be improved so that large-scale sparse matrix operations become useable.

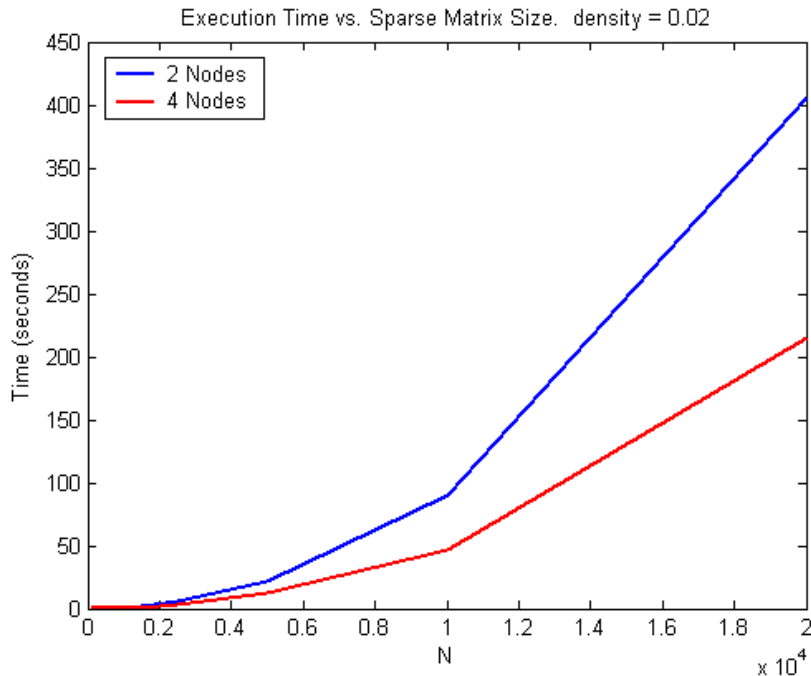


Figure 1: Simple Test of the Plus operator.

Some test matrices of various sizes was obtained from the University of Florida Sparse Matrix Collection⁸ – but timing data was not completed in time for this report. Suffice it to say, however that the ability to load the matrices on to the server, perform the arithmetic and get results off of the server were completed satisfactorily.

Mixed Type Operations

A significant amount of work remains to be done to deal with the issues associated with mixed dense and sparse operation. Obviously such operations should be allowed – since they are allowed in ‘regular’ Matlab – but decisions must be made and enforced carefully within the MATLAB*P code. For the purposes of this project, these issues have been left side-stepped by not allowing mixed dense and sparse arithmetic. In general, it is suggested that, when mixed-type operations are implemented, the guidance provided by reference [4] be followed if for no other reason than that of consistency. The first guiding principal is that: Sparsity should not be automatically inserted without the user specifically asking for it – but that, once sparsity is specified, the sparsity should propagate. Secondly, sparse operands should provide sparse results, except in the case of

⁸ Please see reference [7].

operations that generally destroy sparsity – such as adding a sparse and a full matrix. Additionally, and perhaps most importantly, the *answer* should not depend upon the storage class of the operands⁹. In most respects, these rules are easy to enforce. For example:

$$\gg D2 = D + S$$

Here, ‘D’ is dense and ‘S’ is sparse. Since matrix addition of D to S will generally destroy the sparsity of S, the result will be dense. Likewise:

$$\gg S2 = S ./ D$$

Will be sparse, since the element-wise division by D, in general, has no impact on the sparsity structure of S.

Sometimes, this set of rules leads to some messy situations, for example when the user issues a command such as:

$$\gg B = D ./ S$$

Where, again, ‘D’ is dense and ‘S’ is sparse. In this case, it seems quite logical that ‘B’ should be dense. But what should be done about the (majority) of entries in B that result in a division by zero? Rather than returning an error, or some other compromise result¹⁰, Matlab simply returns a dense ‘B’ with many entries being NaN.

Conclusion

There is a tremendous amount of work to be done with regards to sparse matrices in MATLAB*P. I should probably not waste too much time guessing as to the particular work that will/should be done to build further on to the sparse matrix functionality – but I would say that I would be happy to contribute to that end in any way possible.

⁹ In this context, the term ‘storage class’ means – is it dense or sparse?

¹⁰ See reference [4] for a full discussion of the contemplated solutions to this and other issues that arise in mixed type (mixed storage class) operations.

References

- [1] Parry Husbands and Charles Isbell. "The Parallel Problems Server: A Client-Server Model for Interactive Large Scale Scientific Computation," In *Proceedings of VECPAR98*, June 1998.
- [2] Parry Husbands, "Interactive Supercomputing," MIT THESIS, EECS, 1999.
- [3] Ron Choy's master's thesis
- [4] J.Gilbert, C.Moler, R.Schreiber, " Sparse Matrices in MATLAB: Design and Implementation".
- [5] D.Hanselman, B.Littlefield,"Mastering Matlab: A Comprehensive Tutorial and Reference," Prentice Hall, Upper Saddle River, New Jersey, 2001.
- [6] My Project Web-site
- [7] T. Davis, University of Florida Sparse Matrix Collection,
<http://www.cise.ufl.edu/research/sparse/matrices>
<http://www.netlib.org/na-digest-html/94/v94n42.html>