A Sparse Matrix Library in C++ for High Performance Architectures*

Jack Dongarra[§‡], Andrew Lumsdaine[◊], Xinhiu Niu[◊]
Roldan Pozo[‡], Karin Remington[§]

[§]Oak Ridge National Laboratory, Mathematical Sciences Section
[‡]University of Tennessee, Dept. of Computer Science
[◊]University of Notre Dame, Dept. of Computer Science & Engineering

Abstract

We describe an object oriented sparse matrix library in C++ built upon the Level 3 Sparse BLAS proposal [5] for portability and performance across a wide class of machine architectures. The C++ library includes algorithms for various iterative methods and supports the most common sparse data storage formats used in practice. Besides simplifying the subroutine interface, the object oriented design allows the same driving code to be used for various sparse matrix formats, thus addressing many of the difficulties encountered with the typical approach to sparse matrix libraries. We emphasize the fundamental design issues of the C++ sparse matrix classes and illustrate their usage with the preconditioned conjugate gradient (PCG) method as an example. Performance results illustrate that these codes are competitive with optimized Fortran 77. We discuss the impact of our design on elegance, performance, maintainability, portability, and robustness.

1 Introduction

Sparse matrices are pervasive in application codes which use finite difference, finite element or finite volume discretizations of PDEs for problems in computational fluid dynamics, structural mechanics, semiconductor simulation and other scientific and engineering applications. Nevertheless, comprehensive libraries for sparse matrix computations have not been developed and integrated to the same degree as those for dense matrices. Several factors contribute to the difficulty of designing such a comprehensive library. Different computer architectures, as well as different applications, call for different sparse matrix data formats in order to best exploit registers, data locality, pipelining, and parallel processing. Furthermore, code involving sparse matrices tends to be very complicated, and not easily portable, because the details of the underlying data formats are invariably entangled within the application code.

To address these difficulties, it is essential to develop codes which are as "data format free" as possible, thus providing the greatest flexibility for using the given algorithm (library routine) in various architecture/application combinations. In fact, the selection of an appropriate data structure can typically be deferred until link or run time. We describe an object oriented C++ library for sparse matrix computations which provides a unified interface for various iterative solution techniques across a variety of sparse data formats.

The design of the library is based on the following principles:

Clarity: Implementations of numerical algorithms should resemble the mathematical algorithms on which they are based. This is in contrast to Fortran 77, which can require complicated subroutine calls, often with parameter lists that stretch over several lines.

Reuse: A particular algorithm should only need to be coded once, with identical code used for all matrix representations.

*This project was supported in part by the Defense Advanced Research Projects Agency under contract DAAL03-91-C-0047, administered by the Army Research Office, the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400, and by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, and NSF grant No. CCR-9209815.

Portability: Implementations of numerical algorithms should be directly portable across machine platforms.

High Performance: The object oriented library code should perform as well as optimized data-format-specific code written in C or Fortran.

To achieve these goals the sparse matrix library has to be more than just an unrelated collection of matrix objects. Adhering to true object oriented design philosophies, *inheritance* and *polymorphism* are used to build a matrix hierarchy in which the same codes can be used for various dense and sparse linear algebra computations across sequential and parallel architectures.

2 Iterative Solvers

The library provides algorithms for various iterative methods as described in Barrett *et. al.* [1], together with their preconditioned counterparts:

- Jacobi SOR (SOR)
- Conjugate Gradient (CG)
- Conjugate Gradient on Normal Equations (CGNE, CGNR)
- Generalized Minimal Residual (GMRES)
- Minimum Residual (MINRES)
- Quasi-Minimal Residual (QMR)
- Chebyshev Iteration (Cheb)
- Conjugate Gradient Squared (CGS)
- Biconjugate Gradient (BiCG)
- Biconjugate Gradient Stabilized (Bi-CGSTAB)

Although iterative methods have provided much of the motivation for this library, many of the same operations and design issues are addressed for direct methods as well. In particular, some of the most popular preconditioners, such as Incomplete LU Factorization (ILU) [1] have components quite similar to direct methods.

One motivation for this work is that the high level algorithms found in [1] can be easily implemented in C++. For example, take a preconditioned conjugate gradient algorithm, used to solve $Ax = b$, with preconditioner M. The comparison between the pseudo-code and the C++ listing appears in figure 1.

Here the operators such as `*` and `+=` have been overloaded to work with matrix and vectors formats. This code fragment works for all of the supported sparse storage classes and makes use of the Level 3 Sparse BLAS in the matrix-vector multiply `A*p`.

3 Sparse Matrix types

We have concentrated on the most commonly used data structures which account for a large portion of application codes. The library can be arbitrarily extended to user-specific structures and will eventually grow. (We hope to incorporate user-contributed extensions in future versions of the software.) Matrix classes supported in the initial version of the library include

`Sparse_Vector`: List of nonzero elements with their index locations. It assumes no particular ordering of elements.

`COOR_Matrix`: Coordinate Storage Matrix. List of nonzero elements with their respective row and column indices. This is the most general sparse matrix format, but it is not very space or computationally efficient. It assumes no ordering of nonzero matrix values.

`CRS_Matrix` : Compressed Row Storage Matrix. Subsequent nonzeros of the matrix rows are stored in contiguous memory locations and an additional integer arrays specifies where each row begins. It assumes no ordering among nonzero values within each row, but rows are stored in consecutive order.

`CCS_Matrix`: Compressed Column Storage (also commonly known as the *Harwell-Boeing* sparse matrix format [4]). This is a variant of CRS storage where columns, rather rows, are stored contiguously. Note that the CCS ordering of A is the same as the CRS of A^T.

`CDS_Matrix`: Compressed Diagonal Storage. Designed primarily for matrices with relatively constant bandwidth, the sub and super-diagonals are stored in contiguous memory locations.

`JDS_Matrix`: Jagged Diagonal Storage. Also know as `ITPACK` storage. More space efficient than CDS

Initial $r^{(0)} = b - Ax^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence;
end

```
r = b - Ax;
for (int i=1; i<maxiter; i++){
    z = M.solve(r);
    rho = r * z;
    if (i==1)
       p = z;
    else{
       beta = rho1/ rho0;
       p = z + p * beta;
    }
    q = A*p;
    alpha = rho1 / (p*q);
    x += alpha * p;
    r -= alpha * q;
    if (norm(r)/normb < tol) break;
}
```

Figure 1: Psuedocode and C++ comparison of a preconditioned conjugate gradient method in IML++ package

matrices at the cost of a gather/scatter operation.

BCRS_Matrix: Block Compressed Row Storage. Useful when the sparse matrix is comprised of square dense blocks of nonzeros in some regular pattern. The savings in storage and reduced indirect addressing over CRS can be significant for matrices with large block sizes.

SKS_Matrix: Skyline Storage. Also for variable band or profile matrices. Mainly used in direct solvers, but can also be used for handling the diagonal blocks in block matrix factorizations.

In addition, symmetric and Hermitian versions of most of these sparse formats will also be supported. In such cases only an upper (or lower) triangular portion of the matrix is stored. The trade-off is a more complicated algorithm with a somewhat different pattern of data access. Further details of each data storage format are given in [1] and [6].

Our library contains the common computational kernels required for solving linear systems by many direct and iterative methods. The internal data structures of these kernels are compatible with the proposed Level 3 Sparse BLAS, thus providing the user with large software base of Fortran 77 module and application libraries. Just as the dense Level 3 BLAS [3] have allowed for higher performance kernels on hierarchical memory architectures, the Sparse BLAS allow vendors to provide optimized routines taking advantage of indirect addressing hardware, registers, pipelining, caches, memory management, and parallelism on their particular architecture. Standardizing the Sparse BLAS will not only provide efficient codes, but will also ensure portable computational kernels with a common interface.

There are two types of C++ interfaces to basic kernels. The first utilizes simple binary operators for multiplication and addition, and the second is a functional interfaces which can group triad and more complex operations. The binary operators provide for a simpler interface, e.g. $y = A * x$ denotes a sparse matrix-vector multiply, but may produce less efficient code. The computational kernels include:

- sparse matrix products, $C \leftarrow \alpha\ op(A)\ B + \beta C$.
- solution of triangular systems, $C \leftarrow \alpha D\ op(A)^{-1}\ B + \beta C$
- reordering of a sparse matrix (permutations), $A \leftarrow A\ op(P)$
- conversion of one data format to another, $A' \leftarrow A$,

where α and β are scalars, B and C are rectangular matrices, D is a (block) diagonal matrix, A and A' are sparse matrices, and $op(A)$ is either A or A^T.

4 Sparse Matrix Construction and I/O

In dealing with issues of I/O, the C++ library is presently designed to support reading and writing to Harwell-Boeing format sparse matrix files [4]. These files are inherently in compressed column storage; however, since sparse matrices in the library can be transformed between various data formats, this is not a severe limitation. File input is embedded as another form of a sparse matrix constructor; a file can be read and transformed into another format using conversions and the `iostream` operators. In the future, the library will also support other matrix file formats, such as a MATLAB^{TM} compatible format, and IEEE binary formats. Sparse matrices can be also be initialized from conventional data and index vectors, thus allowing for a universal interface to import data from C or Fortran modules.

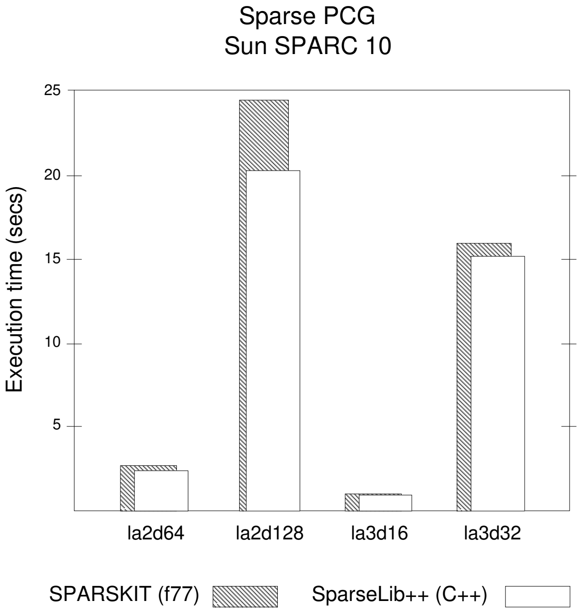

Figure 2: Performance comparison of C++ (g++) vs. optimized Fortran codes on a Sun Sparc 10.

5 Performance

To compare the efficiency of our C++ class designs, we tested the performance our library modules against optimized Fortran sparse matrix packages. Figures 2 and 3 illustrate the performance of the PCG method with diagonal preconditioning on various common example problems (2D and 3D Laplacian operators) between our C++ library and the f77 Sparskit [?] package. In all cases we utilized full optimization of each compiler.

The C++ modules are slightly more efficient since the low-level Sparse BLAS have been fined tuned for each platform. This shows that it is possible to have an elegant coding interface (as shown in figure 1) and still maintain competitive performance (if not better) with conventional Fortran modules.

It should be pointed out that our intent is not to compare ourselves with a specific library. Sparskit is an excellent Fortran library whose focus is more on portability than ultimate performance. Our goal is simply to demonstrate that the C++ codes can also achieve good performance, compared to Fortran. If both libraries relied on vendor-supplied Sparse BLAS, then their performance difference would be undetectable.

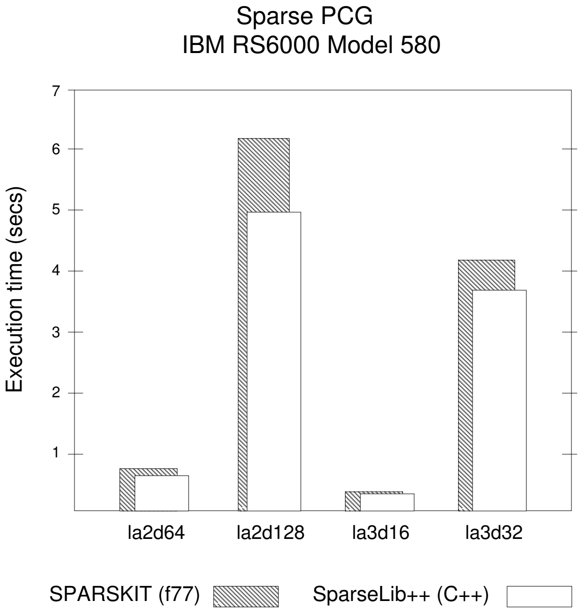

Figure 3: Performance comparison of C++ (xlC) vs. optimized Fortran codes (xlf -O) on an IBM RS/6000 Model 580,

6 Conclusion

Using C++ for numerical computation can greatly enhance clarity, reuse, and portability. In our C++ sparse matrix library, SparseLib++, the details of the underlying sparse matrix data format are completely hidden at the algorithm level. These results in iterative algorithm codes which closely resemble their mathematical denotation. Also, since the library is built upon the Level 3 Sparse BLAS, it provides performance comparable to optimized Fortran.

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM Press, 1994.

[2] J. Dongarra, R. Pozo, D. Walker, "LAPACK++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra," Proceedings of Supercomputing '93, IEEE Press, 1993, pp. 162-171.

[3] J. Dongarra, J. Du Croz, I. S. Duff, S. Hammarling, "A set of level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, Vol. 16, 1990, pp. 1-17.

[4] I. Duff, R. Grimes, J. Lewis, "Sparse Matrix Test Problems," *ACM Trans. Math. Soft.*, Vol. 15, 1989, pp. 1-14.

[5] I. Duff, M. Marrone, G. Radicati, *A Proposal for User Level Sparse BLAS*, CERFACS Technical Report TR/PA/92/85, 1992.

[6] M. A. Heroux, *A Proposal for a Sparse BLAS Toolkit*, CERFACS Technical Report TR/PA/92/90, 1992.

[7] Y. Saad, *Sparskit: a basic toolkit for sparse matrix computations*, May, 1990.