Sparse BLAS Issues in C/C++

Roldan Pozo
Karin Remington

National Institute of Standards and Technology

Andrew Lumsdaine

Unversity of Notre Dame

Proposed Level 3 Sparse Blas

- **sparse matrix products,**

$$C \leftarrow \alpha \; op(A) \; B + \beta C$$

- **solution of triangular systems,**

$$C \leftarrow \alpha D \; op(A)^{-1} \; B + \beta C$$
$$C \leftarrow \alpha \; op(A)^{-1} \; DB + \beta C$$

- **reordering of a sparse matrix (permutations),**

$$A \leftarrow A \; op(P)$$

- **conversion of one data format to another,**

$$A' \leftarrow A$$

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

Proposed F77 Interface

ex: `DCRSMM(TRANS,  M, N, K, ALPHA, DESCRA, VAL, INDX, PNTR, B, LDB, BETA, LDC, WORK, LWORK)`

Argument list

1. Arguments specifying options
2. arguments specifying problem dimensions
3. input scalar associated with input matrices
4. description of sparse input matrices
   (a) Descriptor array
   (b) array containing the nonzero entries of input matrix
   (c) first dimension of value array
   (d) index array(s) corresponding to input matrices
      i. ordered in decreasing length
      ii. same length arrays ordered with row indices
   (e) length or first dimension of value/index arrays
   (f) pointer array(s)
      i. ordered in descreasing length
      ii. same length arrays ordered: row, column, diag
   (g) length or first dimension of pointer arrays
   (h) (left/right) permutation arrrays of A
   (i) block entry dimension
5. description of dense input matrices
6. input scalar associated with input-output matrix
7. description of input-output matrix
8. error processing information
9. workspace
10. length or workspace

Interface issues in C/C++

- more general matrix structures (e.g. compressed rows/columns need not be single vector.)
- 0/1 based indexing (argument shifting not sufficient)
- simplification of current interface
  - remove work arrays
  - remove character string arguments
  - replace `DESCRA` matrix type encoding with self-contained matrix objects
  - utilize function overloading to remove other matrix-type specific subroutine parameters
  - utilize optional arguments for common defaults (e.g. `CompCol_MatVec_Product(C, A, x)`)
- validation tests at object construction reduces the need for repeated integrity checking.
- modification of sparse matrix structure via methods reduces the possibility of corruption

Interface Examples

C++

```
y = A * x;

BLAS_Mat_Vec_Mult(y, A, x);

BLAS_Mat_Vec_Mult(C, alpha, A, B, beta);
```

Fortran 77

```
DESCRA(1) = 0
DCSRMM(TRANS, M, N, K, 1.0, DESCRA, VAL, INDX, PNTR
   X, N, 1.0, LCC, WORK, LWORK)

DCSRMM(TRANS, M, N, K, ALPHA, DESCRA, VAL, INDX, PNTR,
   B, LDB, BETA, LCC, WORK, LWORK)
```

C++ Enhancements

- **pre-processing information structures as optional arguments**
- **support for block matrix variants via templates (e.g.** `CompCol_Mat<double>`, `CompCol_Mat<block4x4>`**)**
- **generic types, e.g.** `CompCol_Mat<MyObj>`.
- **representations for diagonal and triangular views**

Experience with the C Sparse BLAS

- *not* generated via f2c()
- initial 1.0 release in March '95 (bundled with SparseLib++)
- v. 1.2 now standalone in ANSI C: `http://math.nist.gov/pozo/sparseblas.html`
- total # of access to SparseLib++ and Sparse BLAS: over 2,000
- Sparse BLAS kernels on Sun Sparc 20, IBM RS6K, and Intel Pentium platforms

C Sparse BLAS on various platforms

File	Size	# nonzeros	Watcom C/C++ 10.0 Dell Pentium 90 (Mflops)	Sun C 3.0.1 Sun SPARC 20 (Mflops)
la1d32.hb	(32×32)	94	6.0	8.3
alt2d16.hb	(256×256)	1216	6.5	9.4
pores_3.rua	(532×532)	3474	8.2	8.9
orani678.rua	(2529×2529)	90,158	4.1	8.1
shyy161.rua	$(76,480 \times 76,480)$	329,762	4.7	5.5
psmg1.hb	(3140×3140)	543,162	6.3	6.7

Table 1: Performance of optimized sparse matrix/vector products for various Harwell-Boeing matrices. (Watcom compile flags: -oneaxt -5r -zp4 -fp5; Sun C compile flags: -O)

File	Size	# nonzeros	Wide Node RS-6000 590 (Mflops)	Thin Node RS-6000 590 (Mflops)
la1d32.hb	(32×32)	94	20.3	12.9
alt2d16.hb	(256×256)	1216	24.1	16.0
pores_3.rua	(532×532)	3474	26.9	12.7
shyy161.rua	$(76,480 \times 76,480)$	329,762	21.2	10.9
psmg1.hb	(3140×3140)	543,162	29.0	13.9

Table 2: Performance of optimized sparse matrix/vector products for various Harwell-Boeing matrices. (xlc compile flags: -O)

Optimized and compiler-generated Sparse BLAS (Compressed Row Matvec)

Microsoft Fortran 32-bit (-O)

```
L2:             mov     eax,ebp
                mov     edx,dword ptr [edi+eax*4]
                fld     qword ptr [ecx+eax*8]
                fmul    qword ptr [esi+edx*8]
                fadd    qword ptr L92
                fstp    qword ptr L92
                inc     ebp
                mov     dword ptr L94,ebp
L3:             mov     ebp,dword ptr L94
                cmp     ebx,ebp
                jge     short L2
```

Watcom C/C++ 10.0a (-ox)

```
L2:             mov     ecx,dword ptr [edx]
                shl     ecx,03H
                add     ecx,dword ptr +14H[ebp]
                fld     qword ptr [ebx]
                fmul    qword ptr [ecx]
                add     ebx,00000008H
                add     edx,00000004H
                inc     eax
                faddp   st(1),st
                cmp     eax,esi
                jl      short L2
```

Microsoft Assembler v. 6.2

```
L2:     fld     qword ptr [val + 8*k + 8]         ; st <- val[k+1]
        fld     qword ptr [val + 8*k ]            ; st <- val[k]
        mov     tmp, dword ptr [row + 4*k]        ; tmp <- row[k]
        fmul    qword ptr [ x + 8*tmp]            ; val[k] * x[row[k]]
        fxch
        mov     tmp, dword ptr [row + 4*k + 4]    ; tmp <- row[k+1]
        fmul    qword ptr [ x + 8*tmp]            ; val[k+1] * x[row[k+1]]
        fxch
        faddp   st(2), st                         ; sum1 +=  k version
        add     k, 2                              ;  k = k + 2
        faddp   st(2), st                         ; sum2 +=  k+1 version
        cmp     k, e_Mem                          ;  if (k<e)
        jl      short L2                          ;      goto L2
```

Product List

- **C Sparse BLAS v. 1.1**
- **SparseLib++ 1.5**
- **IML++ v. 1.2a**
- **MV++ v. 1.5**

Available at `http://math.nist.gov/pozo`

Summary

1. To allow Sparse BLAS interface in C:
   - allow 0-based indexing
   - remove string arguments
2. Seperate C interface
   - simplify parameter list
   - group related parameters into structures
   - use ANSI C prototyping for safety and proper linking
3. Seperate C++ interface
   - make sparse matrix formats seperate classes (SparseLib++)
   - use function overloading to provide single interface call
   - use templates for block structures and matrices of generic types
   - use optional paramters to simplify function calls
   - use operator overloading to replace function calls
   - use exception handling for better error control