Sparse BLAS, Toolkits and Primitives

Michael A. Heroux
Scalable Computing and Algorithms Group
Applications Division
Cray Research, Inc.

Introduction

Outline:

I. Overview of Toolkit.

II. Latest Changes.

  A. No Characters.

  B. "Floating base" pointer and index vectors.

  C. PNTR generalized to BPNTR and EPNTR.

III. Sparse BLAS Primitives.

IV. Sparse BLAS Layers.

Overview: Scope of the Sparse BLAS Toolkit

- Matrix-matrix products:

$$C \leftarrow \alpha AB + \beta C$$

$$C \leftarrow \alpha A^T B + \beta C$$

- Solution of triangular systems (supported for all but the COO and BCO data structures):

$$C \leftarrow \alpha DA^{-1} B + \beta C$$

$$C \leftarrow \alpha DA^{-T} B + \beta C$$

$$C \leftarrow \alpha A^{-1} DB + \beta C$$

$$C \leftarrow \alpha A^{-T} DB + \beta C$$

- Right permutation of a sparse matrix (presently for JAD data structure only):

$$A \leftarrow AP$$

$$A \leftarrow AP^T$$

Overview: Data Structures

- Point entry:
  - COO - Coordinate
  - CSC - Compressed sparse column
  - CSR - Compressed sparse row
  - DIA - Sparse diagonal
  - ELL - Ellpack/Itpack
  - JAD - Jagged diagonal
  - SKY - Skyline
- Block entry:
  - BCO - Block coordinate
  - BSC - Block compressed sparse column
  - BSR - Block compressed sparse row
  - BDI - Block sparse diagonal
  - BEL - Block Ellpack/Itpack
  - VBR - Variable block compressed sparse row

Latest Features: Language Independence

These changes eliminate language dependencies in the interface and data structures.

- No characters:
  - Early proposals used Fortran 77 character arrays as descriptors.
  - All character variables are now integers.
- Floating base pointers and index vectors:
  - DESCRA has entries for pointer and index vector offsets.
  - Fortran: (1,1)   C: (0,0)

Latest Features: Floating Base Example

$$A = \begin{pmatrix} 11 & 0 & 13 & 14 & 0 \\ 0 & 22 & 23 & 24 & 0 \\ 31 & 0 & 33 & 34 & 0 \\ 0 & 42 & 0 & 44 & 0 \\ 51 & 52 & 0 & 0 & 55 \end{pmatrix},$$

CSC representation with 1-based pointer and index:

$$\begin{array}{rcl} \mathrm{VAL} & = & (\ 11\ \ 31\ \ 51\ \ 22\ \ 42\ \ 52\ \ 13\ \ 23\ \ 33 \\ & & \ \ 14\ \ 24\ \ 34\ \ 44\ \ 55\ ), \\ \mathrm{INDX} & = & (\ 1\ \ 3\ \ 5\ \ 2\ \ 4\ \ 5\ \ 1\ \ 2\ \ 3 \\ & & \ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ ), \\ \mathrm{PNTR} & = & (\ 1\ \ 4\ \ 7\ \ 10\ \ 14\ \ 15\ ), \end{array}$$

CSC representation with 0-based pointer and index:

$$\begin{array}{rcl} \mathrm{INDX} & = & (\ 0\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 0\ \ 1\ \ 2 \\ & & \ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ ), \\ \mathrm{PNTR} & = & (\ 0\ \ 3\ \ 6\ \ 9\ \ 13\ \ 14\ ). \end{array}$$

Latest Features: BPNTR and EPNTR

- The argument PNTR is replace by BPNTR and EPNTR.
  - BPNTR(i) = start of row/column i in VAL and INDX.
  - EPNTR(i) $- 1$ = end of row/column i in VAL and INDX.
- Backward compatible with PNTR (use PNTR($1:n$) for BPNTR, and PNTR($2:n+1$) for EPNTR).
- Can extract triangular matrices.
- Removes storage association between contiguous rows.

Latest Features: BPNTR and EPNTR

$$A = \begin{pmatrix} 11 & 0 & 13 & 14 & 0 \\ 0 & 22 & 23 & 24 & 0 \\ 31 & 0 & 33 & 34 & 0 \\ 0 & 42 & 0 & 44 & 0 \\ 51 & 52 & 0 & 0 & 55 \end{pmatrix},$$

CSC representation with 1-based pointer and index:

$$\begin{array}{rcllllllllll} \text{VAL} & = & (& 11 & 31 & 51 & 22 & 42 & 52 & 13 & 23 & 33 \\ & & & 14 & 24 & 34 & 44 & 55 &), & & & \\ \text{INDX} & = & (& 1 & 3 & 5 & 2 & 4 & 5 & 1 & 2 & 3 \\ & & & 1 & 2 & 3 & 4 & 5 &), & & & \\ \text{PNTR1} & = & (& 1 & 4 & 7 & 10 & 14 & 15 &), & & \\ \text{PNTR2} & = & (& 1 & 4 & 9 & 13 & 14 &). & & & \end{array}$$

Compute $y = y + Ax$:

```
      do j = 1, n
        do i = bpntr(j), epntr(j) - 1
          y(indx(i)) = y(indx(i)) + val(i) * x(j)
        end do
      end do
```

```
call SCSCMM(..., pntr1, pntr1(2),...)  ! Full MV

call SCSCMM(..., pntr1, pntr2,...)     ! Upper Tri MV
```

Sparse BLAS Primitives

Observation: For any given matrix operation and data structure (and transposes) all computation is expressed by a few essential loops with minor variations.

Example: Compressed Row/Column MV uses two basic loop sets.

Set 1:

```
      do j = 1, n
        istrt = fn1(j)
        istop = fn2(j)
        do i = istrt, istop
          y(indx(i)) = y(indx(i)) + val(i) * x(j)
        end do
      end do
```

Set 2:

```
      do i = 1, m
        jstrt = fn1(i)
        jstop = fn2(i)
        do j = jstrt, jstop
          y(i) = y(i) + val(j) * x(indx(j))
        end do
      end do
```

Sparse BLAS Primitives

- Example: `fn1(j) = pntr(j)`, `fn2(j) = pntr(j) - 1`.
- Note: Loop optimizations are usually independent of `fn1` and `fn2`.
- Definitions for `fn1` and `fn2` can be enumerated.

The Sparse BLAS Primitives:

- A *very* small set of macro dependent loop sets that support the Sparse BLAS Toolkit.
- Unexpanded (macro) version of loops can be optimized for particular architecture since definition of macros does not impact optimization strategies.
- Minimizes the actual amount of code hardware vendors must optimize.

Sparse (and other) BLAS Layers

- High Level Interfaces:
  - Object oriented.
  - Data structure neutral/independent.
  - Language dependent.
  - Complicated data structures: parallelism, matrix features, preprocessing.
  - Portable code, portable performance (if lower level libs available).
- Mid Level Interfaces:
  - Data structure specific.
  - Language independent.
  - Simple data structures: No parallel, no preprocessing.
  - Portable code, perhaps performance.
- Low Level Interfaces:
  - High performance implementation.
  - Not portable.
  - Vendor provided.