Using Sparse Matrix and Solver Routines from Intel MKL

SciNet User Group Meeting, March 2013

Deepak Chandan

Department of Physics, University of Toronto

March 13, 2013

Outline

What are Sparse Matrices

Definition

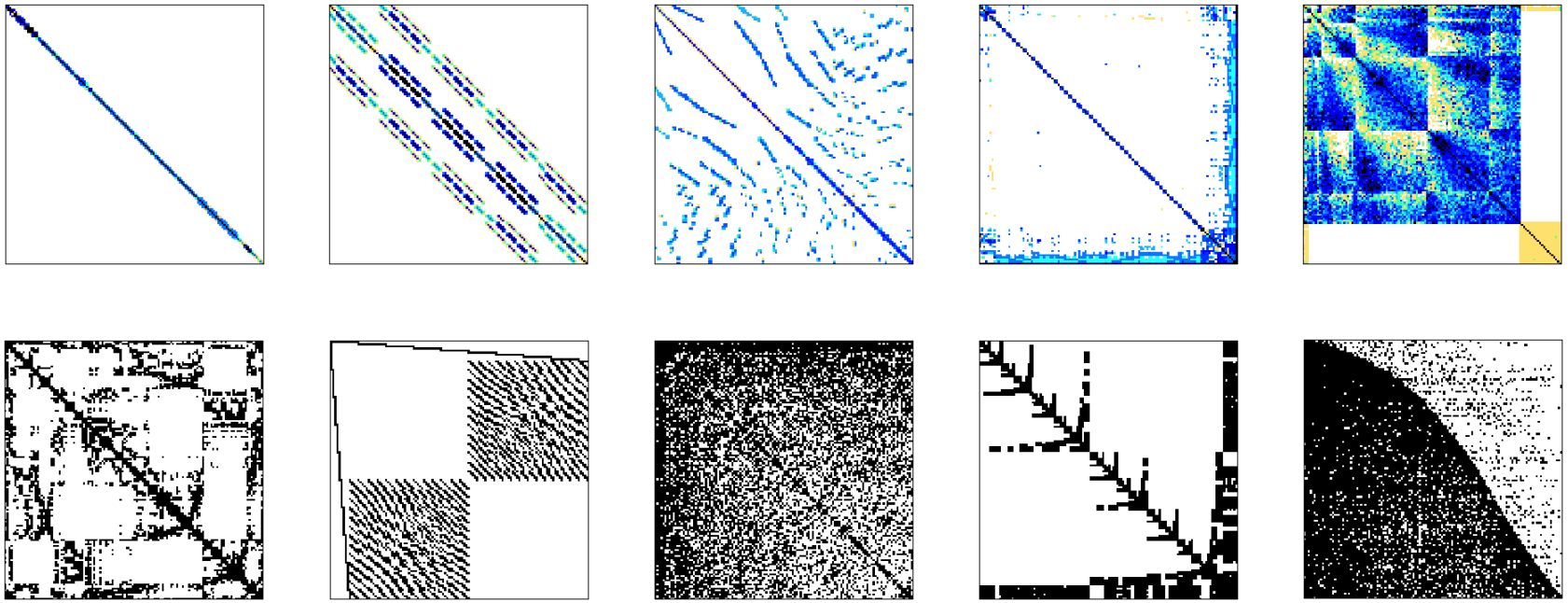
A matrix populated primarily by zeros.

What are Sparse Matrices

Definition

A matrix populated primarily by zeros.

Examples



Outline

Storage Formats

Reference

- The material in this section is taken from Intel MKL, Appendix A.
- To aid you in future reference of the manual I have used the manual's naming scheme and variable names.

Coordinate Format (COO)

Let's begin with an example sparse matrix

$$\begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -3 & * & 6 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

Coordinate Format (COO)

Let's begin with an example sparse matrix

$$\begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -3 & * & 6 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

- Represent only non-zero values by their coordinate indices
- Three vector representation

Coordinate Format (COO)

Let's begin with an example sparse matrix

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & -1 & * & -3 & * \\ 1 & -2 & 5 & * & * & * \\ 2 & * & * & 4 & 6 & 4 \\ 3 & -3 & * & 6 & 7 & * \\ 4 & * & 8 & * & * & -5 \end{array}$$

- Represent only non-zero values by their coordinate indices
- Three vector representation

values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
rows	=	(	0	0	0	1	1	2	2	2	3	3	3	4	4	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)

Note: I have used 0-based indices; can also use 1-based indices

Coordinate Format (COO)

- Although COO improvement over dense matrices, can do better in terms of storage.
- There are duplicate values for the indices.

Coordinate Format (COO)

- Although COO improvement over dense matrices, can do better in terms of storage.
- There are duplicate values for the indices.

Duplicate rows

values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
rows	=	(	0	0	0	1	1	2	2	2	3	3	3	4	4	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)

Coordinate Format (COO)

- Although COO improvement over dense matrices, can do better in terms of storage.
- There are duplicate values for the indices.

Duplicate rows

values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
rows	=	(	0	0	0	1	1	2	2	2	3	3	3	4	4	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)

Duplicate columns

values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
rows	=	(	0	0	0	1	1	2	2	2	3	3	3	4	4	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)

Compressed Sparse Row (CSR)

As the name suggests, the rows are stored in compressed form.

- Stores only non-zero values
- Matrix indexing can be zero (C style) or one (Fortran style) based.
- Available in three or four vector formats. In Intel MKL
  - Four vector format is also called NIST Blas format
  - Three vector format is called CSR3

Let's see the difference between the two formats.

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & \boxed{8} & & & \boxed{-5} \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & \boxed{8} & & & \boxed{-5} \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	$\boxed{13}$	)							

Number of Non-Zero Elements

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Four vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
pointerB	=	(	0	3	5	8	11	)								
pointerE	=	(	3	5	8	11	13	)								

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

		0	1	2	3	4	5	6	7	8	9	10	11	12
values	= (	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5)
columns	= (	0	1	3	0	1	2	3	4	0	2	3	1	4)
rowIndex	= (	0	3	5	8	11	13)							

Four vector version

		0	1	2	3	4	5	6	7	8	9	10	11	12
values	= (	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5)
columns	= (	0	1	3	0	1	2	3	4	0	2	3	1	4)
pointerB	= (	0	3	5	8	11)								
pointerE	= (	3	5	8	11	13)								

Note: The values and columns arrays are the same in both formats

Compressed Sparse Row (CSR)

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

Three vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
rowIndex	=	(	0	3	5	8	11	13	)							

Four vector version

			0	1	2	3	4	5	6	7	8	9	10	11	12	
values	=	(	1	-1	-3	-2	5	4	6	4	-3	6	7	8	-5	)
columns	=	(	0	1	3	0	1	2	3	4	0	2	3	1	4	)
pointerB	=	(	0	3	5	8	11	)								
pointerE	=	(	3	5	8	11	13	)								

Note: Connection between pointerB and pointerE.

Compressed Sparse Row (CSR)

- You will be fine just using the three vector format
- I will only use the three vector format in the rest of the talk
- What happens when a row is entirely empty??

Compressed Sparse Row (CSR)

$$\begin{bmatrix} & & & & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

		0	1	2	3	4	5	6	7	8	9
values	=	-2	5	4	6	4	-3	6	7	8	-5
columns	=	0	1	2	3	4	0	2	3	1	4
rowIndex	=	0	0	2	5	8	10				

$$\begin{bmatrix} 1 & -1 & & -3 & \\ & & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & 8 & & & -5 \end{bmatrix}$$

		0	1	2	3	4	5	6	7	8	9	10
values	=	1	-1	-3	4	6	4	-3	6	7	8	-5
columns	=	0	1	3	2	3	4	0	2	3	1	4
rowIndex	=	0	3	3	6	9	11					

$$\begin{bmatrix} 1 & -1 & & -3 & \\ -2 & 5 & & & \\ & & 4 & 6 & 4 \\ -3 & & 6 & 7 & \\ & & & & \end{bmatrix}$$

		0	1	2	3	4	5	6	7	8	9	10
values	=	1	-1	-3	-2	5	4	6	4	-3	6	7
columns	=	0	1	3	0	1	2	3	4	0	2	3
rowIndex	=	0	3	5	8	11	11					

Compressed Sparse Column (CSC)

Similar to CSR, but instead the columns are stored in compressed form.

- Stores only non-zero values
- Matrix indexing can be zero (C style) or one (Fortran style) based.

$$\begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -3 & * & 6 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

- Represent only non-zero values by their coordinate indices
- Three or Four vector representation

This is the four-vector format

values	=	(	1	-2	-3	-1	5	8	4	6	-3	6	7	4	-5	)
rows	=	(	0	1	3	0	1	4	2	3	0	2	3	1	4	)
pointerB	=	(	0	3	6	8	11	)								
pointerE	=	(	3	6	8	11	13	)								

Outline

Sparse BLAS

BLAS

Basic Linear Algebra Subroutines

Most (if not all) of you may be already familiar with BLAS routines operating on dense arrays. They come in three levels:

- BLAS Level 1 - Vector-Vector Operations
- BLAS Level 2 - Matrix-Vector Operations
- BLAS Level 3 - Matrix-Matrix Operations

MKL provides an extension to BLAS for sparse matrices called Sparse BLAS

Sparse BLAS

BLAS

Basic Linear Algebra Subroutines

Most (if not all) of you may be already familiar with BLAS routines operating on dense arrays. They come in three levels:

- BLAS Level 1 - Vector-Vector Operations
- BLAS Level 2 - Matrix-Vector Operations
- BLAS Level 3 - Matrix-Matrix Operations

MKL provides an extension to BLAS for sparse matrices called Sparse BLAS

Sparse Vectors

Before discussing Sparse BLAS Level 1, we need a brief digression into sparse-vectors.

Sparse Vectors

The Intel MKL uses the following terms to describe vectors for BLAS routines:

Compressed sparse vectors

In this form only non-zero elements and their indices are stored. I.e. there a total of two arrays are required to describe the sparse-vector - one containing the elements and the other containing the indices.

Full-storage vectors

This is just the usual dense form of the vector.

Sparse BLAS Level 1

Naming Conventions

If the Sparse BLAS routine is an extension of a regular BLAS routine then it's name is derived by appending an *i* at the end of the corresponding dense BLAS routine.

Examples

Dense BLAS	Sparse BLAS	Description
saxpy	saxpyi	Single-Precision scalar-vector product plus vector
ddot	ddoti	Double-Precision Dot product
sdot	sdoti	Single-Precision Dot product

Note: There are some dense BLAS Level 1 routines that will work with compressed-form array (with the parameter $incx = 1$), such as *dcopy*. See Manual.

Sparse BLAS Level 2 and Level 3

Useful

- Level 2 and 3 routines are useful for building iterative solvers for large sparse systems.
- Such solvers are based on Reverse Communication Interface (RCI).

Terminology

In the next few slides the following terms will be used:

- **Typical/Conventional interface** Interface similar to that of NIST Sparse BLAS library.
- **Simplified interface** Essentially supports CSR3.

Sparse BLAS Level 2 and Level 3

Naming Scheme

Each routine has a six- or eight-character base name prefixed by either `mkl_` or `mkl_cspblas_`.

Typical Interface

The routines have the following template:

```
mkl_<character><data><operation>
```

Simplified Interface

The routines have the following template:

For one-based indexing :-

```
mkl_<character><data><mtype><operation>
```

For zero-based indexing :-

```
mkl_cspblas_<character><data><mtype><operation>
```

Sparse BLAS Level 2 and Level 3

`<character>` can be:

`s`	real single precision
`c`	complex single precision
`d`	real double precision
`z`	complex double precision

Sparse BLAS Level 2 and Level 3

`<character>` can be:

`s`	real single precision
`c`	complex single precision
`d`	real double precision
`z`	complex double precision

`<data>` can be:

`coo`	coordinate format
`csr`	CSR and variants
`csc`	CSC and variants
`dia`	diagonal format
`sky`	skyline format
`bsr`	block sparse row format

Sparse BLAS Level 2 and Level 3

`<character>` can be:

s	real single precision
c	complex single precision
d	real double precision
z	complex double precision

`<operation>` can be:

mv	matrix-vector product (Level 2)
mm	matrix-matrix product (Level 3)
sv	solve single triangular system (Level 2)
sm	solve $\triangle$ system with multiple RHS (Level 3)

`<data>` can be:

coo	coordinate format
csr	CSR and variants
csc	CSC and variants
dia	diagonal format
sky	skyline format
bsr	block sparse row format

Sparse BLAS Level 2 and Level 3

`<character>` can be:

s	real single precision
c	complex single precision
d	real double precision
z	complex double precision

`<operation>` can be:

mv	matrix-vector product (Level 2)
mm	matrix-matrix product (Level 3)
sv	solve single triangular system (Level 2)
sm	solve $\triangle$ system with multiple RHS (Level 3)

`<data>` can be:

coo	coordinate format
csr	CSR and variants
csc	CSC and variants
dia	diagonal format
sky	skyline format
bsr	block sparse row format

`<mtype>` can be:

ge	general matrix
sy	upper or lower $\triangle$ of a symmetric matrix
tr	triangular matrix

Sparse BLAS Level 2 and Level 3

Examples

Matrix-vector product of a sparse general matrix

	0-based	1-based
Simplified	`mkl_cspblas_?csrgemv`	`mkl_?csrgemv`
Typical	`mkl_?csrmv`	`mkl_?csrmv`

Note: Simplified form uses CSR3 and Typical form uses CSR4.

Outline

Solver Solvers

Types

There are two types of solvers available in the MKL:

- Direct Solvers
- Iterative Solvers

Solver Solvers

Types

There are two types of solvers available in the MKL:

- Direct Solvers
- Iterative Solvers

These is a very extensive topic, principally because the routines provide a multitude of customization.

Solver Solvers

Types

There are two types of solvers available in the MKL:

- Direct Solvers
- Iterative Solvers

These is a very extensive topic, principally because the routines provide a multitude of customization.

Only discuss the general operation of these routines.

Reference

See Chapter 8 in the MKL.

Direct Solver - PARDISO

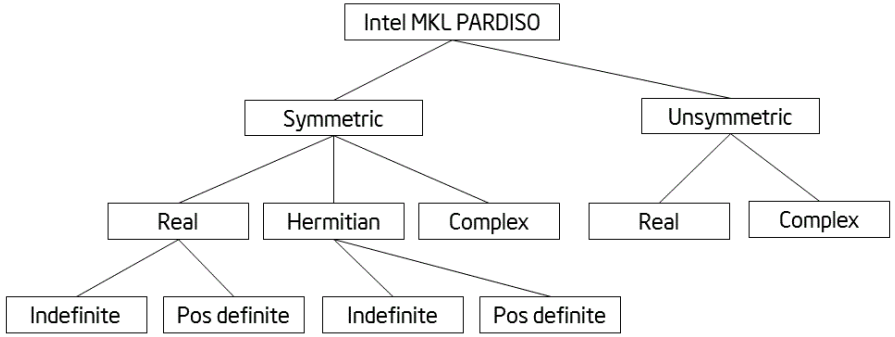
PARDISO stands for **PAR**allel **DI**rect **SO**lver.

> *The Intel MKL PARDISO package is a high-performance, robust, memory efficient, and easy to use software package for solving large sparse linear systems of equations on shared memory multiprocessors. - MKL Manual*

Direct Solver - PARDISO

PARDISO stands for **PAR**allel **DI**rect **SO**lver.

> *The Intel MKL PARDISO package is a high-performance, robust, memory efficient, and easy to use software package for solving large sparse linear systems of equations on shared memory multiprocessors. - MKL Manual*



Matrices that PARDISO can solve.

Direct Solver - PARDISO

PARDISO solving process has multiple "phases". You have to call the same function more than once to process these phases.

The phases are:

- Phase 1: Fill-reduction analysis and symbolic factorization
- Phase 2: Numerical factorization
- Phase 3: Forward and backward solve and optional iterative refinement
- Phase 4: Memory release phase

Direct Solver - PARDISO

PARDISO solving process has multiple "phases". You have to call the same function more than once to process these phases.

The phases are:

- Phase 1: Fill-reduction analysis and symbolic factorization
- Phase 2: Numerical factorization
- Phase 3: Forward and backward solve and optional iterative refinement
- Phase 4: Memory release phase

See manual for detailes. See code samples for usage.

Iterative Solvers

MKL supports iterative sparse solvers (ISS) based on the reverse communication interface (RCI).

Iterative Solvers

MKL supports iterative sparse solvers (ISS) based on the reverse communication interface (RCI).

What is RCI?

A group of user-callable routines that are used in the step-by-step solving process.

Iterative Solvers

MKL supports iterative sparse solvers (ISS) based on the reverse communication interface (RCI).

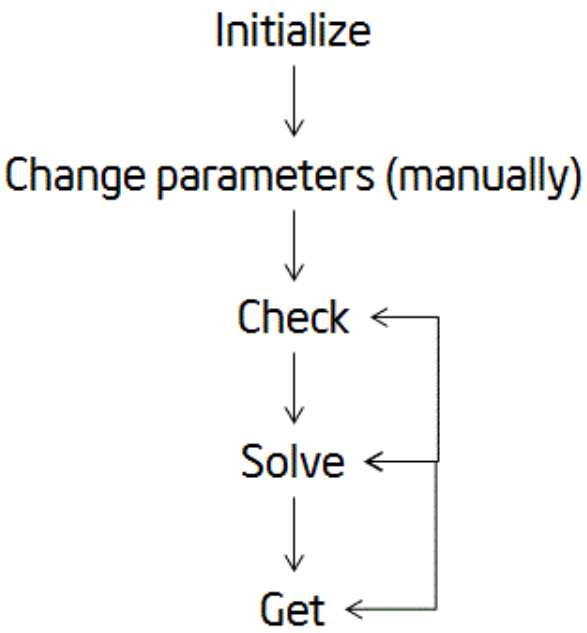
What is RCI?

A group of user-callable routines that are used in the step-by-step solving process.

What can RSS-ISS solve?

- Symmetric Positive Definite System
  Uses Conjugate Gradients (CG)
- Non-symmetric Indefinite System
  Uses Flexible Generalized Minimul Residual Solver (FGMRES)

Iterative Solvers



RCI ISS Interface Routines

Routine	Description
dcg_init, dcgmrhs_init, dfgmres_init	Initializes the solver.
dcg_check, dcgmrhs_check, dfgmres_check	Checks the consistency and correctness of the user defined data.
dcg, dcgmrhs, dfgmres	Computes the approximate solution vector.
dcg_get, dcgmrhs_get, dfgmres_get	Retrieves the number of the current iteration.

Outline

SCXX Library

Description

Stands for *Scientific CXX* is a templated linear algebra library that I have written over the past year that provides:

1. Dense vectors
2. Dense matrices (2D, 3D and 4D)
3. Sparse 2D Matrices
4. Postscript plotting routines (useful for debugging)
5. Computational grids
6. Solvers (coming soon)

The classes that implement the above structures provide:

1. Operator overloading for: `+, -, *, /, =, *=, +=, -=, », «, (), []`
2. Mathematical functions such as `MATMUL, DOT, TRIPLE_PRODUCT, Mag()`
3. C++ exception handling
4. Debugging support
5. Sparse matrix routines

I want to talk briefly about the support for sparse matrices.

SCXX Sparse Module

The library provides the following sparse matrix classes:

`SMatrixSP_COO, SMatrixSP_CSR, SMatrixSP_CSC`

These classes:

1. Provide convenient single object handle for a sparse matrix.
2. Format inter-conversion (via overloaded assignment operator `=`)
3. Sparse Matrix-Dense Vector multiplication
4. Neatly formatted output (via overloaded output operator `«`)
5. Easy insertion of data (via function-call operator)

   ```
   void operator()(Double val, Int32 row, Int32 col)
   ```

   This manages the difficult task of generating the pointers array for the CSR and CSC formats.
6. Easy accessing of data (via fucntion-call operator)

   ```
   Double operator()(Int32 row, Int32 col)
   ```
7. Methods to retrieve internal pointers for use in external routines (such as those of Intel MKL)

THE END!