Intel® Math Kernel Library Sparse Matrix Vector Multiply Format Prototype Package

Reference Manual

LEGAL INFORMATION

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

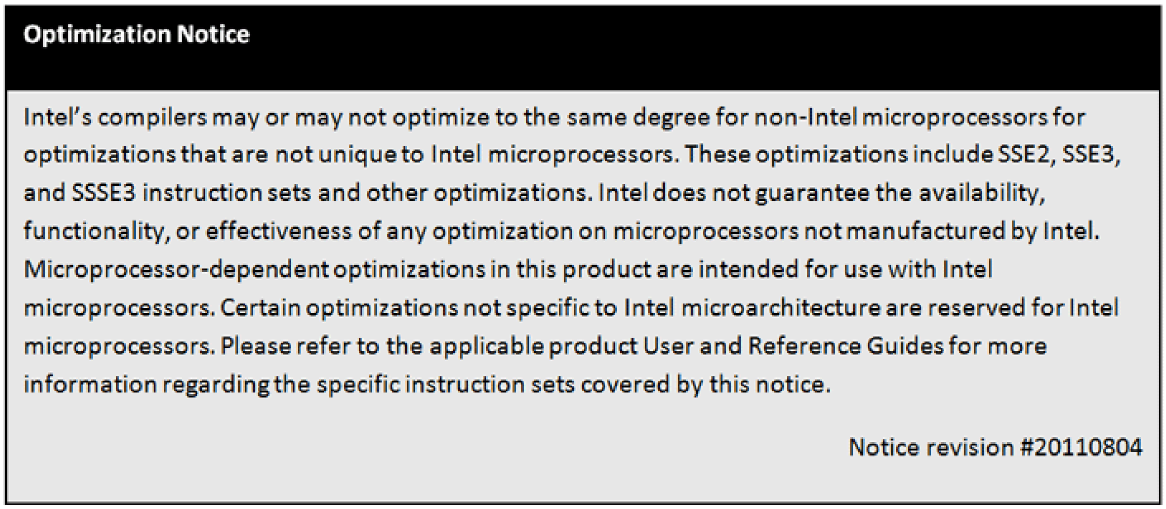
Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

This document contains information on products in the design phase of development.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skoool, the skoool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.



Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

TABLE OF CONTENTS

1 Introducing the SpMV Format Package

The *Intel® Math Kernel Library Sparse Matrix Vector Multiply Format Prototype Package (Intel® MKL SpMV Format Prototype Package)* described in this document contains several implementations of SpMV routines which are optimized for Intel® Xeon Phi™ coprocessors. Interfaces for this library are mostly similar to NVIDIA* CUDA* Sparse Matrix library (CuSPARSE) interfaces [1].

Two sparse formats are currently supported in this library:

- CSR (Column Sparse-Row format), see Sparse Matrix Storage Formats section in [2] for details.
- ESB (ELLPACK Sparse Row format), see [3] for details.

The SpMV functionality implemented is the following general non-transposed case for double precision only:

y := *alpha*A*x+beta*y*,

where *alpha* and *beta* are scalars, *A* is a sparse matrix in the CSR or ESB format, and *x* and *y* are dense vectors.

The sparse matrix for this functionality is stored in an opaque structure passed by a handle, which can be created from an external sparse matrix in the CSR format. Note that the input data is duplicated in the internal array representation. An input matrix can use either 0-based or 1-based indexing.

The library currently provides only C interfaces.

2 Supported Data Types

The following table lists supported data types and possible values for each type:

Data Type	Possible Values	Description
`sparse Status_t`		A status that is returned by library functions.
	`SPARSE_STATUS_SUCCESS`	The operation completed successfully
	`SPARSE_STATUS_NOT_INITIALIZED`	Some of the structures were not created or initialized properly before use. Check the sequence of function calls and statuses of completed operations.
	`SPARSE_STATUS_ALLOC_FAILED`	Allocation of internal memory inside the library failed. A task which is too large is probably being solved or some memory was not released.
	`SPARSE_STATUS_INVALID_VALUE`	Some parameters in the function call have unsupported values. For example: the size of a matrix is negative.

	SPARSE_STATUS_EXECUTION_FAILED, SPARSE_STATUS_INTERNAL_ERROR	Usually it means that execution stopped due to unexpected values of some internal variables, which probably occurred as a result of the application of the wrong algorithm.
sparse Operation_t		A matrix operation
	SPARSE_OPERATION_NON_TRANSPOSE	Non-transposed matrix operation.
sparse Schedule_t		A workload scheduling algorithm
	SPARSE_SCHEDULE_STATIC	Statically distribute input data to threads
	SPARSE_SCHEDULE_DYNAMIC	Dynamically distribute input data to threads
	SPARSE_SCHEDULE_BLOCK	Distribute input data to threads as fixed-size blocks
sparseMatrix Type_t		A description of an input matrix
	SPARSE_MATRIX_TYPE_GENERAL	Input matrix is stored in the general representation.

`sparseIndexBase_t`		Indexing of a matrix
	`SPARSE_INDEX_BASE_ZERO`	The matrix has zero-based indexing.
	`SPARSE_INDEX_BASE_ONE`	The matrix has one-based indexing.

3 Auxiliary Functions

sparseCreateESBMatrix

Creates ESB internal matrix structure with default initial values.

Syntax

```
sparseStatus_t sparseCreateESBMatrix (sparseESBMatrix_t *esbA,
sparseSchedule_t schedule);
```

Include Files

`spmv_interface.h`

Input Parameters

`schedule`

Specifies the matrix scheduling algorithm.

Possible values:

`SPARSE_SCHEDULE_STATIC` – statically distribute the input data;

`SPARSE_SCHEDULE_DYNAMIC` – dyntamically distribute the input data.

Output Parameters

`esbA`

Pointer to the created ESB matrix structure.

Return Values

`Status`

Possible values:

`SPARSE_STATUS_SUCCESS` – ESB matrix structure is created successfully,

`SPARSE_STATUS_ALLOC_FAILED` – ESB matrix structure could not be allocated,

`SPARSE_STATUS_INVALID_VALUE` – the schedule parameter has an unsupported value.

Note: *The scheduling algorithm determines workload balancing for the input matrix. SPARSE_SCHEDULE_STATIC means that the input matrix is divided into small chunks which are assigned to threads statically. SPARSE_SCHEDULE_DYNAMIC means that these chunks are assigned to threads dynamically in run time.*

sparseDestroyESBMatrix

Releases memory used by the ESB matrix structure.

Syntax

```
sparseStatus_t sparseDestroyESBMatrix(sparseESBMatrix_t  esbA);
```

Include Files

```
spmv_interface.h
```

Input Parameters

esbA	the matrix for which to release the memory.

Return Values

Status

Possible values:

`SPARSE_STATUS_SUCCESS – ESB` matrix structure is released successfully,

`SPARSE_STATUS_NOT_INITIALIZED` – ESB matrix structure was not created.

sparseCreateCSRMatrix

Creates the CSR internal matrix structure with default initial values.

Syntax

```
sparseStatus_t sparseCreateCSRMatrix (sparseCSRMatrix_t *csrA,
sparseSchedule_t schedule);
```

Include Files

`spmv_interface.h`

Input Parameters

`schedule`

Specifies the matrix scheduling algorithm.

Possible values:

`SPARSE_SCHEDULE_STATIC` – statically distribute the input data;

`SPARSE_SCHEDULE_DYNAMIC` – dynamically distribute the input;

`SPARSE_SCHEDULE_BLOCK` – the input data will be distributed as fixed-size blocks.

Output Parameters

`csrA`

Pointer to the created CSR matrix structure.

Return Values

`Status`

Possible values:

`SPARSE_STATUS_SUCCESS` – CSR matrix structure is created successfully,

`SPARSE_STATUS_ALLOC_FAILED` – the memory for the CSR matrix structure could not be allocated,

`SPARSE_STATUS_INVALID_VALUE` – the schedule parameter has an unsupported value.

Note: *The scheduling algorithm determines workload balancing for the input matrix. SPARSE_SCHEDULE_STATIC means that the input matrix is divided into small chunks which are assigned to threads statically. SPARSE_SCHEDULE_DYNAMIC means that these chunks are assigned to threads dynamically in run time. SPARSE_SCHEDULE_BLOCK means that each thread processes one block of input matrix, all blocks have more or less equal numbers of non-zeroes.*

sparseDestroyCSRMatrix

Releases the memory used by the CSR matrix structure.

Syntax

```
sparseStatus_t sparseDestroyCSRMatrix(sparseCSRMatrix_t  csrA);
```

Include Files

```
spmv_interface.h
```

Input Parameters

`csrA` The matrix for which to release the memory.

Return Values

`Status` Possible values:

`SPARSE_STATUS_SUCCESS` – the CSR matrix structure is released successfully, SPARSE_STATUS_NOT_INITIALIZED – the CSR matrix structure was not created.

sparseCreateMatDescr

Creates a matrix descriptor and initializes it with default values using the SPARSE_MATRIX_TYPE_GENERAL *and* SPARSE_INDEX_BASE_ZERO *settings.*

Syntax

```
sparseStatus_t sparseCreateMatDescr  (sparseMatDescr_t *descrA);
```

Include Files

```
 spmv_interface.h
```

Output Parameters

descrA	Pointer to the created matrix descrtiptor.

Return Values

Status	Possible values: SPARSE_STATUS_SUCCESS – the matrix descriptor structure is created successfully, SPARSE_STATUS_ALLOC_FAILED – the memory for the matrix descriptor structure could not be allocated.

sparseDestroyMatDescr

Releases the memory used by the matrix descriptor structure.

Syntax

```
sparseStatus_t sparseDestroyMatDescr (sparseMatDescr_t  descrA);
```

Include Files

```
spmv_interface.h
```

Input Parameters

descrA	The matrix descriptor for which to release the memory.

Return Values

Status	Possible values: `SPARSE_STATUS_SUCCESS` – the matrix descriptor is released successfully, `SPARSE_STATUS_NOT_INITIALIZED` – the matrix descriptor was not previously created.

sparseSetMatType

Sets the MatrixType value in the matrix descriptor.

Syntax

```
sparseStatus_t      sparseSetMatType(sparseMatDescr_t descrA,
sparseMatrixType_t type);
```

Include Files

```
spmv_interface.h
```

Input Parameters

descrA	The matrix descriptor to be modified.

`type`	The matrix type. Possible value: `SPARSE_MATRIX_TYPE_GENERAL`, default.

Output Parameters

`descrA`	The matrix descriptor.

Return Values

`descrA`	The matrix descriptor.
`Status`	Possible values: `SPARSE_STATUS_SUCCESS` – *MatrixType* was set successfully, `SPARSE_STATUS_NOT_INITIALIZED` – the *descrA* matrix structure was not initialized properly.

sparseGetMatType

Get the MatrixType value from the matrix descriptor.

Syntax

```
sparseMatrixType_t  sparseGetMatType(const sparseMatDescr_t descrA);
```

Include Files

```
spmv_interface.h
```

Input Parameters

descrA	The matrix descriptor.

Return Values

MatrixType	The value of the matrix type from the matrix descriptor. Possible value: `SPARSE_MATRIX_TYPE_GENERAL`, default.

sparseSetMatIndexBase

Sets the indexing base value in the matrix descriptor.

Syntax

```
sparseStatus_t      sparseSetMatIndexBase(sparseMatDescr_t descrA,
sparseIndexBase_t base);
```

Include Files

```
spmv_interface.h
```

Input Parameters

descrA	The matrix descriptor to be modified.
base	The base to use for indexing arrays. Possible values: `SPARSE_INDEX_BASE_ZERO` (default) and `SPARSE_INDEX_BASE_ONE`.

Output Parameters

descrA	The matrix descriptor.

Return Values

Status

Possible values:

SPARSE_STATUS_SUCCESS – the indexing base was set successfully,

SPARSE_STATUS_NOT_INITIALIZED – the *descrA* matrix structure was not initialized properly.

sparseGetMatIndexBase

Get the indexing base value from the matrix descriptor.

Syntax

```
sparseIndexBase_t   sparseGetMatIndexBase(const sparseMatDescr_t
descrA);
```

Include Files

```
spmv_interface.h
```

Input Parameters

descrA

The matrix descriptor.

Returned Values

IndexBase

The base used for indexing arrays.

Note:

If *descrA* was not initialized, SPARSE_INDEX_BASE_ZERO is returned.

sparseDcsr2esb

Creates the internal ESB matrix structure from the input matrix in the CSR format, described by the descrA descriptor. It also analyzes the sparsity structure of the matrix and prepares data for workload scheduling algorithms.

Syntax

```
sparseStatus_t sparseDcsr2esb(   int m, int n, const sparseMatDescr_t
descrA, const double *csrValA, const int *csrRowPtrA,  const int
*csrColIndA, sparseESBMatrix_t  esbA );
```

Include Files

`spmv_interface.h`

Input Parameters

m

The number of rows of the input matrix *A*.

n

The number of columns of the input matrix *A*.

descrA

The descriptor of the matrix *A*.

csrValA

Array of length *csrRowPtrA [m]* - *csrRowPtrA [0]*. Contains non-zero elements of the matrix A. See the description of the values array in CSR Format for more details.

csrRowPtrA

Array of length *m* + 1. Contains indices of elements in the *csrValA* array such that *csrRowPtrA[I]* is the index in the *csrValA* array of the first non-zero element in the row I. The value of *csrRowPtrA [m] - csrRowPtrA [0]* is equal to the number of non-zero elements. See the description of the rowIndex array in CSR Format for more details.

csrColIndA

Array of length equal to the length of the *csrValA* array. Contains the column indices for each non-zero element of the matrix *A*. See the description of the columns array in CSR Format for more details.

Output Parameters

esbA

The ESB matrix descriptor.

Return Values

Status

See the type description for possible values.

sparseDcsr2csr

Creates the internal CSR matrix structure from the input matrix in the CSR format, described by the descrA descriptor. It also analyzes the sparsity structure of the matrix and prepares data for workload scheduling algorithms.

Syntax

```
sparseStatus_t sparseDcsr2csr( int m, int n, const sparseMatDescr_t
descrA,const double *csrValA, const int *csrRowPtrA, const int
*csrColIndA, sparseCSRMatrix_t  csrA );
```

Include Files

```
spmv_interface.h
```

Input Parameters

m

The number of rows of the input matrix *A*.

n

The number of columns of the input matrix *A*.

descrA

The descriptor of the matrix *A*.

csrValA

Array of length *csrRowPtrA [m]* - *csrRowPtrA [0]*. Contains non-zero elements of the matrix *A*. See the description of the values array in CSR Format for more details.

csrRowPtrA

Array of length *m* + 1. Contains indices of elements in the array *csrValA* such that *csrRowPtrA[I]* is the index in the array *csrValA* of the first non-zero element in the row I. The value of *csrRowPtrA [m] – csrRowPtrA [0]* is equal to the number of non-zero elements. See the description of the rowIndex array in CSR Format for more details.

csrColIndA

Array of length equal to the length of the array *csrValA*. Contains the column indices for each non-zero element of the matrix *A*. See the description of the columns array in CSR Format for more details.

Output Parameters

csrA

CSR matrix descriptor.

Return Values

Status

See the type description for possible values.

4 Computational Functions

sparseDesbmv

*Performs the matrix-vector operation y:= alpha*A*x+beta*y, where alpha and beta are scalars, x and y are dense vectors, and A is a sparse m-by-n matrix in the internal ESB format.*

Syntax

```
sparseStatus_t sparseDesbmv( sparseOperation_t transA, const double
*alpha, const sparseESBMatrix_t  esbA, const double *x, const double
*beta, double *y);
```

Include Files

```
spmv_interface.h
```

Input Parameters

transA Descriptor of the matrix operation. Only non-transposed operation is currently supported.

alpha The scalar parameter *alpha*.

esbA Sparse matrix in the ESB format. Use the sparseDcsr2esb() convertor to initialize the esbA structure.

x Array of dimension at least *n*. Contains the dense vector *x*.

beta The scalar parameter *beta*.

y Array of dimension at least *m*. Contains the input vector *y*.

Output Parameters

y	Array of dimension at least m. Contains the resulting vector *y*.

Return Values

Status	See the type description for possible values.

sparseDcsrmv

*Performs the matrix-vector operation y:= alpha*A*x+beta*y, where alpha and beta are scalars, x and y are dense vectors and A is a sparse m-by-n matrix in the internal CSR format.*

Syntax

```
sparseStatus_t sparseDcsrmv( sparseOperation_t transA, const double
*alpha, const sparseCSRMatrix_t  csrA, const double *x, const double
*beta, double *y);
```

Include Files

```
spmv_interface.h
```

Input Parameters

transA	The descriptor of the matrix operation. Only non-transposed operation is currently supported.
alpha	The scalar parameter *alpha*.
csrA	Sparse matrix in the CSR format. Use the sparseDcsr2csr() convertor to initialize the csrA

structure.

`x` Array of dimension at least *n*. Contains the vector *x*.

`beta` The scalar parameter beta.

`y` Array of dimension at least *m*. Contains the input vector *y*.

Output Parameters

`y` Array of dimension at least *m*. Contains the resulting vector *y*.

Return Values

`Status` See the type description for possible values.

BIBLIOGRAPHY

1. The NVIDIA CuSPARSE* library documentation
2. Intel® Math Kernel Library Reference Manual
3. Xing Liu, Mikhail Smelyanskiy, Edmond Chow, Pradeep Dubey. “Efficient Sparse Matrix-Vector Multiplication on x86-Based Many-Core Processors”. ICS’13, June 10–14, 2013, Eugene, Oregon, USA