Run-timeCacheSimulator Reference Manual

Alpha

Generated by Doxygen 1.3.9.1

Fri Aug 15 18:12:22 2008

Contents

Chapter 1

Cache Simulator – perfect cache simulator

The cache simulator uses the **CACHE**(p. 5), **CS_ELEMENT**(p. 17), **CS_ARRAY**(p. 10), **CS_MATRIX**(p. 22), **CS_CRS**(p. 13) and **CS_ICRS**(p. 19) classes to simulate cache behaviour in order to obtain cache performance statistics.

This is *not* a trace-driven cache simulator. Applications simply use the classes provided by this library instead of normal datatypes.

This library is developed as a means to obtain cache hit/miss statistics in the area of sparse matrix operations, using our own idealised cache model:

-The cache is k-way set-associative cache.

-There is only one cache in between the CPU and main memory; i.e., we assume a single-level cache architecture.

-The cache is always full; that is, cache lines always store valid data. Hence a cache miss always results in a cache eviction.

-The replacement policy when choosing between k cache lines is the Least Recently Used (LRU) policy.

-Reading in n cache lines from main memory to the cache takes O(l_m+nb_m) time, where l_m,b_m in (0,infinity) constant.

-Reading in n cache lines from cache to CPU takes O(l_c+nb_c) time, where l_c,b_c in (0,infinity) constant.

-Analogously: writing n cache lines from CPU to cache takes O(l_c+nb_c) time, whereas writing from cache to main memory takes O(l_m+nb_m) time.

The current framework only keeps statistics on the number of cache hits and misses.

Chapter 2

Run-timeCacheSimulator Class Index

2.1 Run-timeCacheSimulator Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Run-timeCacheSimulator Class Documentation

3.1 CACHE Class Reference

The cache simulator main class.

`#include <CACHE.hpp>`

Collaboration diagram for CACHE:

Public Member Functions

- void **clearStatistics** ()

  Resets all counters.

- const **STATUS** & **getStatus** ()

  Returns the current cache status.

- **CACHE** (unsigned long int l, unsigned long int s, unsigned long int k)

  Main constructor.

- bool **hitOrMiss** (void *pointer, unsigned long int size)

  Detects a cache hit or miss.

- void **access** (void *pointer, const unsigned long int size, unsigned long int &FAC)

  Simulates a read.

- std::string **properties** ()

  Prints cache properties to output string.

- void **clear** ()

  Clears all cache contents.

Static Public Member Functions

- **CACHE** * **getInstance** ()

  Returns the singleton cache.

- void **createInstance** (unsigned long int l, unsigned long int s, unsigned long int k)

  Creates an instance with specific cache parameters.

- void **recreateInstance** (unsigned long int l, unsigned long int s, unsigned long int k)

  Deletes the current cache instance and creates a new one with the given cache parameters.

Protected Member Functions

- **CACHE_SET_COLLECTION** * **getCacheLines** (unsigned long int setID)

  Internal function which get the cache set collection corresponding to a given pointer.

Protected Attributes

- unsigned long int **REINIT**

  Reinitialisation counter.

- unsigned long int **_l**

  Total number of cache lines.

- unsigned long int **_s**

  Total cache size.

- unsigned long int **_k**

  Number of subcaches.

- unsigned long int **_sdl**

  Caches _s / _l (cache line size LS).

- unsigned long int **_ldk**

  Caches _l / _k (number of cache lines per subcache).

- **CACHE_SET_COLLECTION** ** **_csccollections**

  The _l / _k different cache set collections.

- **STATUS** **_status**

  Keeps track of cache statistics.

3.1.1 Detailed Description

The cache simulator main class.

3.1.2 Member Function Documentation

3.1.2.1 CACHE * CACHE::getInstance () [static]

Returns the singleton cache.

If not already initialised, the singleton will be set to the default cache with parameters: l=2^{26} s=2^{32} k=2^4 (Corresponding to an Intel Core 2 4MB L2 cache)

3.1.2.2 bool CACHE::hitOrMiss (void * *pointer*, unsigned long int *size*)

Detects a cache hit or miss.

Does not update statistics!

The documentation for this class was generated from the following files:

- CACHE.hpp
- CACHE.cpp

3.2 CACHE_LINE Class Reference

Represents a single cache line.

`#include <CACHE_LINE.hpp>`

Public Member Functions

- **CACHE_LINE** ()
  Base constructor.
- **CACHE_LINE** (void *pointer)
  Base constructor.

Public Attributes

- void * **_pointer**
  Pointer this line starts at.
- bool **_dirty**
  Represents if memory here has been changed in cache.

3.2.1 Detailed Description

Represents a single cache line.

The documentation for this class was generated from the following files:

- CACHE_LINE.hpp
- CACHE_LINE.cpp

3.3 CACHE_SET_COLLECTION Class Reference

Represents a single cache set collection, i.e., the line C(i,:).

`#include <CACHE_SET_COLLECTION.hpp>`

Collaboration diagram for CACHE_SET_COLLECTION:

Public Member Functions

- **CACHE_SET_COLLECTION** (unsigned int _k)
  Base constructor.
- **~CACHE_SET_COLLECTION** ()
  Base deconstructor.
- bool **bringIntoCache** (unsigned long int pointer)
  Brings a memory line into cache.
- void **clear** ()
  Clears the LRU stack of all entries.

Protected Attributes

- unsigned int **k**
  Number of different setIDs available.
- **LRU_STACK** * **stack**
  The LRU stack datastructure used for simulation.

3.3.1 Detailed Description

Represents a single cache set collection, i.e., the line C(i,:).

3.3.2 Member Function Documentation

3.3.2.1 bool CACHE_SET_COLLECTION::bringIntoCache (unsigned long int *pointer*)

Brings a memory line into cache.

Returns:
Whether or not the data was cached.

The documentation for this class was generated from the following files:

- CACHE_SET_COLLECTION.hpp
- CACHE_SET_COLLECTION.cpp

3.4 CS_ARRAY< T > Class Template Reference

Denotes the base simulated datatype.

`#include <CS_ARRAY.hpp>`

Collaboration diagram for CS_ARRAY< T >:

Public Member Functions

- **CS_ARRAY** (unsigned long int size, std::string descr)
  The only valid constructor.
- unsigned long int **size** ()
  Returns the size of the array.
- T & **operator[]** (unsigned long int i)
  []-overloading.
- const T & **const_access** (unsigned long int i)
  Const variant of **access()**(p. 10).
- T & **unrecorded_access** (const unsigned long int i)
  Performs unrecorded (not-simulated) access to stored values.
- const T & **unrecorded_const_access** (const unsigned long int i) const
  Const variant of unrecorded_ access.
- T & **access** (unsigned long int i)
  Function alias of the []-operator.
- unsigned long int **getSize** () const
  Gets the array size.
- void **unrecordedCopyTo** (T *target)
  Copies caption of current array to target, non-simulated array.
- void **unrecordedCopyFrom** (T *target)
  Copies caption from a target, non-simulated array into current array.
- void **yaxpy** (const **CS_MATRIX**< T > &A, const **CS_ARRAY**< T > &x)
  Performs an yaxpy operation.
- void **zaxpy** (const **CS_MATRIX**< T > &A, const **CS_ARRAY**< T > &x, const **CS_-ARRAY**< T > &y)
  Performs a zaxpy operation.
- void **zax** (const **CS_MATRIX**< T > &A, const **CS_ARRAY**< T > &x)
  Performs a yax operation.

Static Public Member Functions

- std::string **fDim** (const **CS_ARRAY** &obj)
  Get the dimensions of the array in string format.

Public Attributes

- std::string **_descr**
  Description of the current variable.

Protected Attributes

- T * **_array**
  Contains the actual data.
- unsigned long int **_size**
  Size of _array.
- unsigned long int **FAC**
  Variable used to check if data has been brought into cache before.

3.4.1 Detailed Description

template<typename T> class CS_ARRAY< T >

Denotes the base simulated datatype.

3.4.2 Member Function Documentation

3.4.2.1 template<typename T> std::string CS_ARRAY< T >::fDim (const CS_ARRAY< T > & *obj*) [inline, static]

Get the dimensions of the array in string format.

Parameters:
obj The array to get the dimensions from.

Returns:
The dimensions in string format.

3.4.2.2 template<typename T> void CS_ARRAY< T >::yaxpy (const CS_MATRIX< T > & *A*, const CS_ARRAY< T > & *x*) [inline]

Performs an yaxpy operation.

Calculates y = Ax+y, with A a **CS_MATRIX**(p. 22), X a CS_ARRAY and y this CS_ARRAY. Dimensions must agree, this is checked explicitly.

3.4.2.3 template<typename T> void CS_ARRAY< T >::zax (const CS_MATRIX< T > & *A*, const CS_ARRAY< T > & *x*) [inline]

Performs a yax operation.

Calculates z = Ax, with A a **CS_MATRIX**(p. 22), x a CS_ARRAY, and z this CS_ARRAY. Dimensions must agree, this is checked explicitly. TODO Check order of access in non-simulated case and compare!

3.4.2.4 template<typename T> void CS_ARRAY< T >::zaxpy (const CS_MATRIX< T > & *A*, const CS_ARRAY< T > & *x*, const CS_ARRAY< T > & *y*) [inline]

Performs a zaxpy operation.

Calculates z = Ax+y, with A a **CS_MATRIX**(p. 22), x & y CS_ARRAYs and z this CS_-ARRAY. Dimensions must agree, this is checked explicitly. TODO Check order of access in non-simulated case and compare!

The documentation for this class was generated from the following file:

- CS_ARRAY.hpp

3.5 CS_CRS< T > Class Template Reference

The compressed row storage sparse matrix data structure.

`#include <CS_CRS.hpp>`

Collaboration diagram for CS_CRS< T >:

Public Member Functions

- **CS_CRS** (const ULI number_of_nonzeros, const ULI number_of_rows, const ULI number_of_cols, const T zero)

  Base constructor which only initialises the internal arrays.

- **CS_CRS** (CRS< T > &toCopy)

  Copy constructor.

- **CS_CRS** (**CS_CRS**< T > &toCopy)

  Copy constructor.

- **CS_CRS** (std::vector< Triplet< T > > input, unsigned long int currow, unsigned long int curcol, T &zero)

  Constructor which transforms a collection of input triplets to CRS format.

- **CS_CRS** (std::vector< Triplet< T > > input, T &zero)

  Constructor which transforms a collection of input triplets to CRS format.

- T & **random_access** (ULI i, ULI j)

  Method which provides random matrix access to the stored sparse matrix.

- T * **zax** (T *x_orig)

  Calculates and returns z=Ax.

- ~**CS_CRS** ()

  Base deconstructor.

Protected Member Functions

- bool **find** (ULI col_index, ULI search_start, ULI search_end, ULI &ret)

  Helper function which finds a value with a given column index on a given subrange of indeces.

Static Protected Member Functions

- int **compareTriplets** (const void *left, const void *right)

  Comparation function for 1D columnwise sort.

Protected Attributes

- **CS_ELEMENT**< ULI > * **nnz**

  Number of non-zeros.

- **CS_ELEMENT**< ULI > * **nor**

  Number of rows.

- **CS_ELEMENT**< ULI > * **noc**

  Number of columns.

- **CS_ARRAY**< ULI > * **row_start**

  Array keeping track of individual row starting indeces.

- **CS_ARRAY**< T > * **nzs**

  Array containing the actual nnz non-zeros.

- **CS_ARRAY**< ULI > * **col_ind**

  Array containing the column indeces corresponding to the elements in nzs.

- **CS_ELEMENT**< T > * **zero_element**

  Which element is considered as zero.

3.5.1 Detailed Description

template<typename T> class CS_CRS< T >

The compressed row storage sparse matrix data structure.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 template<typename T> CS_CRS< T >::CS_CRS (const ULI *number_of_nonzeros*, const ULI *number_of_rows*, const ULI *number_of_cols*, const T *zero*) [inline]

Base constructor which only initialises the internal arrays.

Note that to gain a valid CRS structure, these arrays have to be filled by some external mechanism (i.e., after calling this constructor, the internal arrays contain garbage, resuling in invalid datastructure).

Parameters:

number_of_nonzeros The number of non-zeros of the matrix to be stored.

number_of_rows The number of rows of the matrix to be stored.

number_of_cols The number of columns of the matrix to be stored.

zero Which element is considered to be the zero element.

3.5.2.2 template<typename T> CS_CRS< T >::CS_CRS (CS_CRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters:

toCopy reference to the CS_CRS datastructure to copy.

3.5.2.3 template<typename T> CS_CRS< T >::CS_CRS (std::vector< Triplet< T > > *input*, unsigned long int *currow*, unsigned long int *curcol*, T & *zero*) [inline]

Constructor which transforms a collection of input triplets to CRS format.

The input collection is considered to have at most one triplet with unique pairs of indeces. Unspecified behaviour occurs when this assumption is not met.

Parameters:

input The input collection.

currow The number of rows of the input matrix.

curcol The number of columns of the input matrix.

zero Which element is considered zero.

3.5.2.4 template<typename T> CS_CRS< T >::CS_CRS (std::vector< Triplet< T > > *input*, T & *zero*) [inline]

Constructor which transforms a collection of input triplets to CRS format.

The input collection is considered to have at most one triplet with unique pairs of indeces. Unspecified behaviour occurs when this assumption is not met.

Parameters:

input The input collection.

zero Which element is considered zero.

3.5.3 Member Function Documentation

3.5.3.1 template<typename T> int CS_CRS< T >::compareTriplets (const void * *left*, const void * *right*) [inline, static, protected]

Comparation function for 1D columnwise sort.

Parameters:

left The triplet at the left side of the comparation operator <

right The triplet at the right side of the comparation operator <

Returns:

-1 iff the column of left is less than that of right, 1 if it is larger, 0 if equal.

3.5.3.2 template<typename T> bool CS_CRS< T >::find (ULI *col_index*, ULI *search_start*, ULI *search_end*, ULI & *ret*) [inline, protected]

Helper function which finds a value with a given column index on a given subrange of indeces.

Parameters:

col_index The given column index.

search_start The start index of the subrange (inclusive).

search_end The end index of the subrange (exlusive).

ret Reference to the variable where the return *index* is stored.

Returns:

Whether or not a non-zero value should be returned.

3.5.3.3 template<typename T> T& CS_CRS< T >::random_access (ULI *i*, ULI *j*) [inline]

Method which provides random matrix access to the stored sparse matrix.

Parameters:

i Row index.

j Column index.

Returns:

Matrix valuei at (i,j).

3.5.3.4 template<typename T> T* CS_CRS< T >::zax (T * *x_orig*) [inline]

Calculates and returns z=Ax.

The vectors x should have appropiately set length; this is not enforced. Memory leaks, segmentation faults, the works; one or more of these will occur if dimensions do not make sense. The return array is allocated to a length equal to the number of rows in this function. Cache simulation is only performed on access to elements from A, x or z.

Parameters:

x_orig The input (dense) x-vector.

Returns:

The matrix-vector multiplication Ax, where A corresponds to the currently stored matrix.

The documentation for this class was generated from the following file:

- CS_CRS.hpp

3.6 CS_ELEMENT< T > Class Template Reference

Denotes the base simulated datatype.

`#include <CS_ELEMENT.hpp>`

Collaboration diagram for CS_ELEMENT< T >:

Public Member Functions

- **CS_ELEMENT** ()
  The only valid constructor.
- **CS_ELEMENT** (const T initial_caption)
  Convenience constructor.
- **~CS_ELEMENT** ()
  Base deconstructor.
- const T & **const_access** ()
  Const variant of **access()**(p. 17).
- T & **access** ()
  Access function.
- T & **unrecorded_access** ()
  Unrecorded (non-simulated) access function.
- const T & **unrecorded_const_access** () const
  Const variant of unrecorded_ access.
- **CS_ELEMENT**< T > & **operator=** (const **CS_ELEMENT**< T > &other)
- **CS_ELEMENT**< T > & **operator=** (const T &other)
- **CS_ELEMENT**< T > **operator+** (const **CS_ELEMENT**< T > &other) const
- **CS_ELEMENT**< T > **operator-** (const **CS_ELEMENT**< T > &other) const
- **CS_ELEMENT**< T > **operator** * (const **CS_ELEMENT**< T > &other) const
- **CS_ELEMENT**< T > **operator/** (const **CS_ELEMENT**< T > &other) const
- void **operator++** ()
- bool **operator<** (const T &other) const
- bool **operator<** (const **CS_ELEMENT**< T > &other) const
- bool **operator>** (const T &other) const
- bool **operator>** (const **CS_ELEMENT**< T > &other) const
- bool **operator==** (const T &other) const
- bool **operator==** (const **CS_ELEMENT**< T > &other) const

Protected Attributes

- T * **_array**
  Contains the actual data.

- unsigned long int **FAC**

  First access counter.

3.6.1 Detailed Description

template<typename T> class CS_ELEMENT< T >

Denotes the base simulated datatype.

The documentation for this class was generated from the following file:

- CS_ELEMENT.hpp

3.7 CS_ICRS< T > Class Template Reference

The *incremental* compressed row storage sparse matrix data structure.

`#include <CS_ICRS.hpp>`

Collaboration diagram for CS_ICRS< T >:

Public Member Functions

- **CS_ICRS** (const ULI number_of_nonzeros, const ULI number_of_rows, const ULI number_of_cols, const T zero)
  Base constructor which only initialises the internal arrays.
- **CS_ICRS** (**CS_ICRS**< T > &toCopy)
  Copy constructor.
- **CS_ICRS** (std::vector< Triplet< T > > &input, const ULI m, const ULI n, T &zero)
  Constructor which transforms a collection of input triplets to CRS format.
- T * **zax** (T *x_orig)
  Calculates and returns z=Ax.
- T * **zaxtrace** (T *x_orig)
  Calculates and returns z=Ax.
- ~**CS_ICRS** ()
  Base deconstructor.

Static Protected Member Functions

- int **compareTriplets** (const void *left, const void *right)
  Comparison function used for sorting input data.

Protected Attributes

- **CS_ELEMENT**< ULI > **nnz**
  Number of non-zeros.
- **CS_ELEMENT**< ULI > **nor**
  Number of rows.
- **CS_ELEMENT**< ULI > **noc**
  Number of cols.
- **CS_ARRAY**< T > * **nzs**
  Array containing the actual nnz non-zeros.

- **CS_ARRAY< ULI > * c_ind**
  Array containing the column jumps.
- **CS_ARRAY< ULI > * r_ind**
  Array containing the row jumps.
- **CS_ELEMENT< T > zero_element**
  Which element is considered as zero.

3.7.1 Detailed Description

template<typename T> class CS_ICRS< T >

The *incremental* compressed row storage sparse matrix data structure.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 template<typename T> CS_ICRS< T >::CS_ICRS (const ULI *number_of_nonzeros*, const ULI *number_of_rows*, const ULI *number_of_cols*, const T *zero*) [inline]

Base constructor which only initialises the internal arrays.

Note that to gain a valid CRS structure, these arrays have to be filled by some external mechanism (i.e., after calling this constructor, the internal arrays contain garbage, resuling in invalid datastructure).

Parameters:

number_of_nonzeros The number of non-zeros of the matrix to be stored.

number_of_rows The number of rows of the matrix to be stored.

number_of_cols The number of columns of the matrix to be stored.

zero Which element is considered to be the zero element.

3.7.2.2 template<typename T> CS_ICRS< T >::CS_ICRS (CS_ICRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters:

toCopy reference to the CRS datastructure to copy.

3.7.2.3 template<typename T> CS_ICRS< T >::CS_ICRS (std::vector< Triplet< T > > & *input*, const ULI *m*, const ULI *n*, T & *zero*) [inline]

Constructor which transforms a collection of input triplets to CRS format.

The input collection is considered to have at most one triplet with unique pairs of indeces. Unspecified behaviour occurs when this assumption is not met.

Parameters:

input The input collection.

m Number of matrix rows.

n Number of matrix columns.

zero Which element is considered zero.

3.7.3 Member Function Documentation

3.7.3.1 template<typename T> T* CS_ICRS< T >::zax (T * x_orig) [inline]

Calculates and returns z=Ax.

The vector x should have appropiately set length; this is not strictly enforced. Bus errors, segmentation faults, the works; one or more of these will occur if dimensions do not make sense. The return array is allocated to length equal to the number of rows in this function.

Parameters:

x_orig The input (dense) x-vector.

Returns:

The matrix-vector multiplication Ax, where A corresponds to the currently stored matrix.

3.7.3.2 template<typename T> T* CS_ICRS< T >::zaxtrace (T * x_orig) [inline]

Calculates and returns z=Ax.

The vector x should have appropiately set length; this is not strictly enforced. Bus errors, segmentation faults, the works; one or more of these will occur if dimensions do not make sense. This method outputs the row and column indices of matrix elements as they are processed during multiplication to stdout. The return array is allocated to length equal to the number of rows in this function.

Parameters:

x_orig The input (dense) x-vector.

Returns:

The matrix-vector multiplication Ax, where A corresponds to the currently stored matrix.

See also:

zax(p. 21)

The documentation for this class was generated from the following file:

- CS_ICRS.hpp

3.8 CS_MATRIX< T > Class Template Reference

Denotes the base simulated datatype.

`#include <CS_MATRIX.hpp>`

Collaboration diagram for CS_MATRIX< T >:

Public Member Functions

- **CS_MATRIX** (unsigned long int rows, unsigned long int cols, std::string descr)
  The only valid constructor.
- unsigned long int **cols** () const
  Returns the size of the array.
- unsigned long int **rows** () const
  Returns the size of the array.
- T & **operator()** (unsigned long int i, unsigned long int j)
  Matrix accessor.
- T & **access** (unsigned long int i, unsigned long int j)
  Matrix access, function version.
- const T & **const_access** (unsigned long int i, unsigned long int j) const
  Const matrix access version.

Static Public Member Functions

- std::string **fDim** (const **CS_MATRIX** &obj)
  Gets the dimension of a given data matrix.

Public Attributes

- std::string **_descr**
  Description of the current variable.

Protected Attributes

- T ** **_matrix**
  Contains the actual data.
- unsigned long int **_cols**
  column size.
- unsigned long int **_rows**

row size.

3.8.1 Detailed Description

template<typename T> class CS_MATRIX< T >

Denotes the base simulated datatype.

3.8.2 Member Function Documentation

3.8.2.1 template<typename T> std::string CS_MATRIX< T >::fDim (const CS_MATRIX< T > & *obj*) [inline, static]

Gets the dimension of a given data matrix.

Parameters:

obj Reference to the matrix to get the dimensions of.

Returns:

The dimensions formatted as a string.

The documentation for this class was generated from the following file:

- CS_MATRIX.hpp

3.9 CS_Triplet< T > Class Template Reference

Cache-simulated triplet (i,j,v), where i and j are indices and v as a value of template type T.

`#include <CS_Triplet.hpp>`

Collaboration diagram for CS_Triplet< T >:

Public Member Functions

- const unsigned long int **i** ()

  Reads out the row index.

- const unsigned long int **j** ()

  Reads out the column index.

- T & **getValue** ()
- **CS_Triplet** (unsigned long int i, unsigned long int j, T val)

  Base constructor with user-supplied values.

- **CS_Triplet** ()

  Base constructor; sets all values to 0.

Public Attributes

- T **value**

  Gives access to the value.

Protected Attributes

- unsigned long int **row**

  The row index i.

- unsigned long int **column**

  The column index j.

3.9.1 Detailed Description

template<typename T> class CS_Triplet< T >

Cache-simulated triplet (i,j,v), where i and j are indices and v as a value of template type T.

3.9.2 Member Function Documentation

3.9.2.1 template<typename T> T& CS_Triplet< T >::getValue () [inline]

Returns:
A reference to the value.

The documentation for this class was generated from the following file:

- CS_Triplet.hpp

3.10 CS_TS< T > Class Template Reference

Cache-simulated version of the Triplet Scheme method to store sparse matrices.

`#include <CS_TS.hpp>`

Collaboration diagram for CS_TS< T >:

Public Member Functions

- **CS_TS** (std::vector< Triplet< T > > input, ULI m, ULI n, T zero)
  Base constructor.
- T * **MV** (T *x)
  Basic MV.
- ~**CS_TS** ()
  Base destructor.

Protected Attributes

- ULI **nnz**
  Number of non-zeros.
- ULI **nor**
  Number of rows.
- ULI **noc**
  Number of columns.
- **CS_ARRAY**< T > * **nzs**
  Raw array storing the non-zeros and their locations.
- **CS_ARRAY**< ULI > * **row**
  Raw array storing the row-indices.
- **CS_ARRAY**< ULI > * **col**
  Raw array storing the column-indices.
- T **zero_element**
  Which element is considered zero.
- unsigned long int **FAC**
  version control for cache resets

3.10.1 Detailed Description

template<typename T> class CS_TS< T >

Cache-simulated version of the Triplet Scheme method to store sparse matrices.

3.10.2 Member Function Documentation

3.10.2.1 template<typename T> T* CS_TS< T >::MV (T * x) [inline]

Basic MV.

Performs z=Ax.

The documentation for this class was generated from the following file:

- CS_TS.hpp

3.11 CS_ZZ_ICRS< T > Class Template Reference

The zig-zag incremental compressed row storage sparse matrix data structure.

`#include <CS_ZZ_ICRS.hpp>`

Collaboration diagram for CS_ZZ_ICRS< T >:

Public Member Functions

- **CS_ZZ_ICRS** (const ULI number_of_nonzeros, const ULI number_of_rows, const ULI number_of_cols, T zero)

  Base constructor which only initialises the internal arrays.

- **CS_ZZ_ICRS** (**CS_ZZ_ICRS**< T > &toCopy)

  Copy constructor.

- **CS_ZZ_ICRS** (std::vector< Triplet< T > > &input, const ULI m, const ULI n, const T zero)

  Constructor which transforms a collection of input triplets to CRS format.

- T * **zax** (T *x_orig)

  Calculates and returns z=Ax.

- ~**CS_ZZ_ICRS** ()

  Base deconstructor.

Static Protected Member Functions

- bool **compareTriplets** (const Triplet< T > &one, const Triplet< T > &two)

  Comparison function used for sorting input data.

Protected Attributes

- **CS_ELEMENT**< ULI > **nnz**

  Number of non-zeros.

- **CS_ELEMENT**< ULI > **nor**

  Number of rows.

- **CS_ELEMENT**< ULI > **noc**

  Number of cols.

- **CS_ARRAY**< T > * **nzs**

  Array containing the actual nnz non-zeros.

- **CS_ARRAY**< ULI > * **c_ind**

  Array containing the column jumps.

- **CS_ARRAY**< ULI > * **r_ind**

  Array containing the row jumps.

- **CS_ELEMENT**< T > **zero_element**

  Which element is considered as zero.

3.11.1 Detailed Description

template<typename T> class CS_ZZ_ICRS< T >

The zig-zag incremental compressed row storage sparse matrix data structure.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 template<typename T> CS_ZZ_ICRS< T >::CS_ZZ_ICRS (const ULI *number_of_nonzeros*, const ULI *number_of_rows*, const ULI *number_of_cols*, T *zero*) [inline]

Base constructor which only initialises the internal arrays.

Note that to gain a valid CRS structure, these arrays have to be filled by some external mechanism (i.e., after calling this constructor, the internal arrays contain garbage, resuling in invalid datastructure).

Parameters:

number_of_nonzeros The number of non-zeros of the matrix to be stored.

number_of_rows The number of rows of the matrix to be stored.

number_of_cols The number of columns of the matrix to be stored.

zero Which element is considered to be the zero element.

3.11.2.2 template<typename T> CS_ZZ_ICRS< T >::CS_ZZ_ICRS (CS_ZZ_ICRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters:

toCopy reference to the CRS datastructure to copy.

3.11.2.3 template<typename T> CS_ZZ_ICRS< T >::CS_ZZ_ICRS (std::vector< Triplet< T > > & *input*, const ULI *m*, const ULI *n*, const T *zero*) [inline]

Constructor which transforms a collection of input triplets to CRS format.

The input collection is considered to have at most one triplet with unique pairs of indeces. Unspecified behaviour occurs when this assumption is not met.

Parameters:

input The input collection.

m Number of matrix rows.

n Number of matrix columns.

zero Which element is considered zero.

3.11.3 Member Function Documentation

3.11.3.1 template<typename T> T* CS_ZZ_ICRS< T >::zax (T * x_orig) [inline]

Calculates and returns z=Ax.

The vector x should have appropiately set length; this is not strictly enforced. Bus errors, segmentation faults, the works; one or more of these will occur if dimensions do not make sense. The return array is allocated to length equal to the number of rows in this function.

Parameters:

x_orig The input (dense) x-vector.

Returns:

The matrix-vector multiplication Ax, where A corresponds to the currently stored matrix.

The documentation for this class was generated from the following file:

- CS_ZZ_ICRS.hpp

3.12 DIRECT_MAPPED_CACHE Class Reference

Direct Mapped cache using simple modulo as mapping function.

`#include <DIRECT_MAPPED_CACHE.hpp>`

Collaboration diagram for DIRECT_MAPPED_CACHE:

Public Member Functions

- **DIRECT_MAPPED_CACHE** (unsigned long int lines)
  Base constructor.
- ~**DIRECT_MAPPED_CACHE** ()
  Base deconstructor.
- **CACHE_LINE** & **getLine** (void *pointer)
  Gets the cache line belonging to a given pointer.

Protected Attributes

- **CACHE_LINE** * **_lines**
  Number of cache lines available.
- unsigned long int **_size**
  Total cache size.
- unsigned long int **_line_size**
  Total cache line size.

3.12.1 Detailed Description

Direct Mapped cache using simple modulo as mapping function.

3.12.2 Member Data Documentation

3.12.2.1 unsigned long int DIRECT_MAPPED_CACHE::_line_size [protected]

Total cache line size.

Equals _size / _lines

The documentation for this class was generated from the following files:

- DIRECT_MAPPED_CACHE.hpp
- DIRECT_MAPPED_CACHE.cpp

3.13 LRU_STACK Class Reference

A stack-like structure which always has the least recently used item at the bottom, and the most recently used at the top.

`#include <LRU_STACK.hpp>`

Public Member Functions

- **LRU_STACK** (unsigned long int _k)
  Base constructor.
- ~**LRU_STACK** ()
  Base deconstructor.
- bool **exists** (unsigned long int &toCheck)
  Check if supplied value is already stored.
- bool **remove** (unsigned long int &toRemove)
  Remove an element from the stack (prints a warning if not found).
- void **push** (unsigned long int &toPush)
  Puts an element on top of the stack.
- void **clear** ()
  Clear all entries.

Protected Attributes

- unsigned long int * **data**
  Internal array used for datastructure (TODO: swap with singly linked list).
- unsigned long int **start_index**
  Start of stack in array.
- unsigned long int **k**
  Length of _data array.

3.13.1 Detailed Description

A stack-like structure which always has the least recently used item at the bottom, and the most recently used at the top.

The documentation for this class was generated from the following files:

- LRU_STACK.hpp
- LRU_STACK.cpp

3.14 onlyPublic Class Reference

Class providing means to strip all private and protected data members of .h-files, leaving only public members.

Static Public Member Functions

- void **main** (String[] args)
  Main method.

3.14.1 Detailed Description

Class providing means to strip all private and protected data members of .h-files, leaving only public members.

3.14.2 Member Function Documentation

3.14.2.1 void onlyPublic::main (String[] *args*) [inline, static]

Main method.

Parameters:

args The first argument contains the input c++ header file, while the second denotes the output file.

The documentation for this class was generated from the following file:

- onlyPublic.java

3.15 STATUS Class Reference

Class (struct) to simplify keeping cache statistics.

`#include <STATUS.hpp>`

Public Member Functions

- unsigned long int **totalMisses** () const

Public Attributes

- unsigned long int **_lc**

  Number of times cpu-to-cache latency occured.

- unsigned long int **_bc**

  How much units of cpu-to-cache bandwith was needed.

- unsigned long int **_lm**

  Number of times cache-to-main memory latency occured.

- unsigned long int **_bm**

  How much units of cache-to-main memory bandwith was needed.

- unsigned long int **_misses**

  How many cache misses occured.

- unsigned long int **_initialMisses**

  How many initial cache misses occurred.

- unsigned long int **_hits**

  How many cache hits occured.

3.15.1 Detailed Description

Class (struct) to simplify keeping cache statistics.

3.15.2 Member Function Documentation

3.15.2.1 unsigned long int STATUS::totalMisses () const [inline]

Returns:
The number of total misses, that is, the sum of the initial and non-initial misses.

3.15.3 Member Data Documentation

3.15.3.1 unsigned long int STATUS::_initialMisses

How many initial cache misses occurred.

Initial cache misses correspond to data items brought into cache for the first time.

The documentation for this class was generated from the following file:

- STATUS.hpp

Index