SparseLibrary

Version 1.6.0

Generated by Doxygen 1.8.6

Wed Nov 19 2014 17:03:58

Contents

1 Sparse Library **1**

2 Hierarchical Index **3**

2.1 Class Hierarchy 3

3 Class Index **5**

3.1 Class List 5

4 File Index **9**

4.1 File List 9

5 Class Documentation **11**

5.1 BetaHilbert< T > Class Template Reference 11

5.1.1 Detailed Description 14

5.1.2 Constructor & Destructor Documentation 14

5.1.2.1 BetaHilbert 14

5.1.2.2 BetaHilbert 14

5.1.2.3 ~BetaHilbert 14

5.1.3 Member Function Documentation 14

5.1.3.1 bytesUsed 14

5.1.3.2 collectY 15

5.1.3.3 end 15

5.1.3.4 getFirstIndexPair 15

5.1.3.5 load 15

5.1.3.6 mv 16

5.1.3.7 set_p_translate 17

5.1.3.8 synchronise 17

5.1.3.9 thread 17

5.1.3.10 wait 18

5.1.3.11 zax 18

5.1.3.12 zax 18

5.1.3.13 zxa 18

5.1.3.14 zxa 18

5.2 BICRS< _t_value, _i_value > Class Template Reference 19
5.2.1 Detailed Description 21
5.2.2 Constructor & Destructor Documentation 21
5.2.2.1 ∼BICRS 21
5.2.2.2 BICRS 21
5.2.2.3 BICRS 21
5.2.2.4 BICRS 21
5.2.2.5 BICRS 22
5.2.3 Member Function Documentation 22
5.2.3.1 bytesUsed 22
5.2.3.2 getFirstIndexPair 22
5.2.3.3 load 22
5.2.3.4 load 23
5.2.3.5 zax 23
5.2.3.6 zax_fb 23
5.2.3.7 zxa 23
5.2.4 Member Data Documentation 24
5.2.4.1 c_end 24
5.2.4.2 c_inc 24
5.2.4.3 c_start 24
5.2.4.4 jumps 24
5.2.4.5 ntt 24
5.2.4.6 r_end 24
5.2.4.7 r_inc 25
5.2.4.8 r_start 25
5.2.4.9 vals 25
5.3 BigInt Struct Reference 25
5.3.1 Detailed Description 26
5.3.2 Constructor & Destructor Documentation 26
5.3.2.1 BigInt 26
5.3.2.2 BigInt 26
5.3.3 Member Function Documentation 26
5.3.3.1 operator unsigned char 26
5.3.3.2 operator unsigned int 26
5.3.3.3 operator unsigned long int 26
5.3.3.4 operator unsigned short int 26
5.3.3.5 operator= 26
5.3.4 Member Data Documentation 27
5.3.4.1 low 27
5.4 BisectionHilbert< T > Class Template Reference 27

5.4.1 Detailed Description . . . 28
5.4.2 Constructor & Destructor Documentation . . . 29
5.4.2.1 ∼BisectionHilbert . . . 29
5.4.2.2 BisectionHilbert . . . 29
5.4.2.3 BisectionHilbert . . . 29
5.4.2.4 BisectionHilbert . . . 29
5.5 BlockCRS< T > Class Template Reference . . . 29
5.5.1 Detailed Description . . . 30
5.5.2 Member Function Documentation . . . 30
5.5.2.1 in_readout . . . 31
5.5.2.2 post_readout . . . 31
5.5.2.3 pre_readout . . . 31
5.6 BlockHilbert< T > Class Template Reference . . . 31
5.6.1 Detailed Description . . . 34
5.6.2 Constructor & Destructor Documentation . . . 34
5.6.2.1 ∼BlockHilbert . . . 34
5.6.2.2 BlockHilbert . . . 34
5.6.2.3 BlockHilbert . . . 34
5.6.2.4 BlockHilbert . . . 35
5.6.2.5 BlockHilbert . . . 35
5.6.3 Member Function Documentation . . . 35
5.6.3.1 bisect . . . 35
5.6.3.2 bisect . . . 36
5.6.3.3 buildBisectionBlocks . . . 37
5.6.3.4 bytesUsed . . . 37
5.6.3.5 cmp . . . 37
5.6.3.6 find . . . 37
5.6.3.7 getFirstIndexPair . . . 38
5.6.3.8 load . . . 38
5.6.3.9 zax . . . 38
5.6.3.10 zxa . . . 38
5.6.4 Member Data Documentation . . . 39
5.6.4.1 DS . . . 39
5.6.4.2 MAX_DEPTH . . . 39
5.7 BlockOrderer< T > Class Template Reference . . . 39
5.7.1 Detailed Description . . . 41
5.7.2 Member Function Documentation . . . 41
5.7.2.1 in_height . . . 41
5.7.2.2 in_readout . . . 41
5.7.2.3 infix . . . 41

5.7.2.4 post_height . . . 42
5.7.2.5 post_readout . . . 42
5.7.2.6 postfix . . . 42
5.7.2.7 pre_height . . . 42
5.7.2.8 pre_readout . . . 42
5.7.2.9 prefix . . . 42
5.7.2.10 traverse . . . 42
5.7.3 Member Data Documentation . . . 43
5.7.3.1 cur_height . . . 43
5.7.3.2 datatype . . . 43
5.7.3.3 height . . . 43
5.7.3.4 items . . . 43
5.7.3.5 leftpass . . . 43
5.7.3.6 output . . . 43
5.7.3.7 READOUT . . . 43
5.7.3.8 rightpass . . . 44
5.7.3.9 TRAVERSE_HEIGHT . . . 44
5.7.3.10 traverse_mode . . . 44
5.7.3.11 tree . . . 44
5.8 CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value > Class Template Reference . . . 44
5.8.1 Detailed Description . . . 47
5.8.2 Constructor & Destructor Documentation . . . 47
5.8.2.1 ∼CBICRS . . . 47
5.8.2.2 CBICRS . . . 47
5.8.2.3 CBICRS . . . 47
5.8.2.4 CBICRS . . . 48
5.8.2.5 CBICRS . . . 48
5.8.3 Member Function Documentation . . . 48
5.8.3.1 bytesUsed . . . 48
5.8.3.2 getFirstIndexPair . . . 48
5.8.3.3 getMemoryUsage . . . 49
5.8.3.4 getMemoryUsage . . . 49
5.8.3.5 getNumberOfOverflows . . . 49
5.8.3.6 load . . . 49
5.8.3.7 load . . . 50
5.8.3.8 memoryUsage . . . 50
5.8.3.9 zax . . . 50
5.8.3.10 zxa . . . 51
5.8.4 Member Data Documentation . . . 51

5.8.4.1 bytes . . . 51
5.8.4.2 c_inc . . . 51
5.8.4.3 c_start . . . 51
5.8.4.4 mask1 . . . 52
5.8.4.5 mask2 . . . 52
5.8.4.6 ntt . . . 52
5.8.4.7 r_inc . . . 52
5.8.4.8 r_start . . . 52
5.8.4.9 vals . . . 53
5.9 CBICRS_factory< _t_value > Class Template Reference . . . 53
5.9.1 Detailed Description . . . 53
5.9.2 Member Function Documentation . . . 54
5.9.2.1 getCBICCS . . . 54
5.9.2.2 getCBICCS . . . 54
5.9.2.3 getCBICRS . . . 54
5.9.2.4 getCBICRS . . . 55
5.10 CCSWrapper< T, SparseMatrixType, IND > Class Template Reference . . . 56
5.10.1 Detailed Description . . . 58
5.10.2 Constructor & Destructor Documentation . . . 58
5.10.2.1 CCSWrapper . . . 58
5.10.2.2 CCSWrapper . . . 58
5.10.2.3 CCSWrapper . . . 58
5.10.2.4 CCSWrapper . . . 58
5.10.2.5 ∼CCSWrapper . . . 58
5.10.3 Member Function Documentation . . . 58
5.10.3.1 bytesUsed . . . 59
5.10.3.2 getFirstIndexPair . . . 59
5.10.3.3 load . . . 59
5.10.3.4 loadFromFile . . . 59
5.10.3.5 m . . . 59
5.10.3.6 mv . . . 59
5.10.3.7 n . . . 60
5.10.3.8 nzs . . . 60
5.10.3.9 transposeVector . . . 60
5.10.3.10 zax . . . 60
5.10.3.11 zxa . . . 60
5.10.4 Member Data Documentation . . . 61
5.10.4.1 ds . . . 61
5.11 CompressedHilbert< T > Class Template Reference . . . 61
5.11.1 Detailed Description . . . 62

5.11.2 Constructor & Destructor Documentation . . . 63
5.11.2.1 ∼CompressedHilbert . . . 63
5.11.2.2 CompressedHilbert . . . 63
5.11.2.3 CompressedHilbert . . . 63
5.11.2.4 CompressedHilbert . . . 63
5.11.3 Member Function Documentation . . . 63
5.11.3.1 bytesUsed . . . 63
5.11.3.2 getFirstIndexPair . . . 63
5.11.3.3 load . . . 63
5.11.3.4 zax . . . 64
5.11.3.5 zxa . . . 64
5.12 CRS< T > Class Template Reference . . . 64
5.12.1 Detailed Description . . . 66
5.12.2 Constructor & Destructor Documentation . . . 67
5.12.2.1 CRS . . . 67
5.12.2.2 CRS . . . 67
5.12.2.3 CRS . . . 67
5.12.2.4 CRS . . . 67
5.12.2.5 CRS . . . 67
5.12.2.6 ∼CRS . . . 68
5.12.3 Member Function Documentation . . . 68
5.12.3.1 bytesUsed . . . 68
5.12.3.2 columnIndices . . . 68
5.12.3.3 find . . . 68
5.12.3.4 getFirstIndexPair . . . 68
5.12.3.5 load . . . 69
5.12.3.6 random_access . . . 69
5.12.3.7 rowJump . . . 69
5.12.3.8 values . . . 69
5.12.3.9 zax . . . 69
5.12.3.10 ZaX . . . 70
5.12.3.11 ZXa . . . 70
5.12.4 Member Data Documentation . . . 70
5.12.4.1 col_ind . . . 70
5.12.4.2 ds . . . 70
5.12.4.3 row_start . . . 70
5.13 CuHyb Class Reference . . . 70
5.13.1 Detailed Description . . . 72
5.13.2 Constructor & Destructor Documentation . . . 72
5.13.2.1 CuHyb . . . 72

5.13.2.2 CuHyb . . . 72
5.13.2.3 CuHyb . . . 73
5.13.2.4 CuHyb . . . 73
5.13.2.5 ∼CuHyb . . . 73
5.13.3 Member Function Documentation . . . 73
5.13.3.1 bytesUsed . . . 73
5.13.3.2 getFirstIndexPair . . . 73
5.13.3.3 load . . . 74
5.13.3.4 zax . . . 74
5.13.3.5 zax . . . 74
5.13.4 Member Data Documentation . . . 74
5.13.4.1 descrA . . . 74
5.13.4.2 GPUx . . . 74
5.13.4.3 GPUz . . . 75
5.13.4.4 handle . . . 75
5.13.4.5 hybA . . . 75
5.13.4.6 i . . . 75
5.13.4.7 j . . . 75
5.14 DD_MATRIX< T, number_of_diagonals, diagonal_offsets > Class Template Reference . . . 75
5.14.1 Detailed Description . . . 77
5.14.2 Constructor & Destructor Documentation . . . 77
5.14.2.1 DD_MATRIX . . . 77
5.14.2.2 DD_MATRIX . . . 78
5.14.2.3 DD_MATRIX . . . 78
5.14.2.4 DD_MATRIX . . . 78
5.14.2.5 ∼DD_MATRIX . . . 78
5.14.3 Member Function Documentation . . . 79
5.14.3.1 bytesUsed . . . 79
5.14.3.2 getFirstIndexPair . . . 79
5.14.3.3 load . . . 79
5.14.3.4 zax . . . 79
5.14.3.5 zxa . . . 79
5.14.4 Member Data Documentation . . . 80
5.14.4.1 bytes . . . 80
5.14.4.2 d . . . 80
5.14.4.3 full . . . 80
5.14.4.4 nzs . . . 80
5.14.4.5 SELF_ALLOCATED . . . 80
5.15 Duck< T > Class Template Reference . . . 81
5.15.1 Detailed Description . . . 81

5.15.2 Member Function Documentation . . . 82
5.15.2.1 in_readout . . . 82
5.15.2.2 post_readout . . . 82
5.15.2.3 pre_readout . . . 82
5.16 FBICRS< _t_value, _i_value, _sub_ds, logBeta > Class Template Reference . . . 82
5.16.1 Detailed Description . . . 85
5.16.2 Constructor & Destructor Documentation . . . 85
5.16.2.1 ∼FBICRS . . . 85
5.16.2.2 FBICRS . . . 85
5.16.2.3 FBICRS . . . 86
5.16.2.4 FBICRS . . . 86
5.16.2.5 FBICRS . . . 86
5.16.2.6 FBICRS . . . 86
5.16.3 Member Function Documentation . . . 86
5.16.3.1 bytesUsed . . . 86
5.16.3.2 getFirstIndexPair . . . 86
5.16.3.3 load . . . 87
5.16.3.4 load . . . 87
5.16.3.5 load . . . 87
5.16.3.6 zax . . . 88
5.16.3.7 ZaX . . . 88
5.16.3.8 zax_fb . . . 88
5.16.3.9 zxa . . . 89
5.16.3.10 ZXa . . . 89
5.16.3.11 zxa_fb . . . 89
5.16.4 Member Data Documentation . . . 90
5.16.4.1 c_end . . . 90
5.16.4.2 c_inc . . . 90
5.16.4.3 c_start . . . 90
5.16.4.4 fillIn . . . 90
5.16.4.5 jumps . . . 90
5.16.4.6 r_end . . . 90
5.16.4.7 r_inc . . . 91
5.16.4.8 r_start . . . 91
5.17 FileToVT Class Reference . . . 91
5.17.1 Detailed Description . . . 91
5.18 HBICRS< _t_value > Class Template Reference . . . 92
5.18.1 Detailed Description . . . 93
5.18.2 Constructor & Destructor Documentation . . . 94
5.18.2.1 ∼HBICRS . . . 94

5.18.2.2 HBICRS . . . 94
5.18.2.3 HBICRS . . . 94
5.18.2.4 HBICRS . . . 94
5.18.2.5 HBICRS . . . 94
5.18.3 Member Function Documentation . . . 94
5.18.3.1 bytesUsed . . . 94
5.18.3.2 getFirstIndexPair . . . 95
5.18.3.3 load . . . 95
5.18.3.4 load . . . 95
5.18.3.5 load . . . 95
5.18.3.6 zax . . . 95
5.18.3.7 zxa . . . 95
5.18.4 Member Data Documentation . . . 96
5.18.4.1 c_inc . . . 96
5.18.4.2 jumps . . . 96
5.18.4.3 ntt . . . 96
5.18.4.4 r_inc . . . 96
5.18.4.5 vals . . . 96
5.19 Hilbert< T > Class Template Reference . . . 96
5.19.1 Detailed Description . . . 98
5.19.2 Constructor & Destructor Documentation . . . 98
5.19.2.1 ∼Hilbert . . . 98
5.19.2.2 Hilbert . . . 98
5.19.2.3 Hilbert . . . 98
5.19.2.4 Hilbert . . . 99
5.19.3 Member Function Documentation . . . 99
5.19.3.1 bytesUsed . . . 99
5.19.3.2 getDataStructure . . . 99
5.19.3.3 getFirstIndexPair . . . 99
5.19.3.4 load . . . 99
5.19.3.5 saveBinary . . . 100
5.19.3.6 zax . . . 100
5.19.3.7 zxa . . . 100
5.19.4 Member Data Documentation . . . 100
5.19.4.1 ads . . . 100
5.19.4.2 ds . . . 101
5.20 HilbertArray< T, I, hI, mI > Class Template Reference . . . 101
5.20.1 Detailed Description . . . 102
5.20.2 Constructor & Destructor Documentation . . . 103
5.20.2.1 HilbertArray . . . 103

5.20.2.2 ~HilbertArray . . . 103
5.20.3 Member Function Documentation . . . 103
5.20.3.1 bytesUsed . . . 103
5.20.3.2 decode . . . 103
5.20.3.3 decode . . . 103
5.20.3.4 getFirstColumnIndex . . . 104
5.20.3.5 getFirstRowIndex . . . 104
5.20.3.6 moveToNext . . . 104
5.20.3.7 moveToStart . . . 104
5.20.3.8 zax . . . 105
5.20.3.9 zxa . . . 105
5.20.4 Member Data Documentation . . . 105
5.20.4.1 array . . . 105
5.20.4.2 bytes . . . 106
5.20.4.3 curcol . . . 106
5.20.4.4 curcoor . . . 106
5.20.4.5 curpos . . . 106
5.20.4.6 currow . . . 106
5.20.4.7 start_coor . . . 106
5.21 HilbertArrayInterface< T > Class Template Reference . . . 107
5.21.1 Detailed Description . . . 107
5.21.2 Constructor & Destructor Documentation . . . 107
5.21.2.1 ~HilbertArrayInterface . . . 107
5.21.3 Member Function Documentation . . . 108
5.21.3.1 bytesUsed . . . 108
5.21.3.2 getFirstColumnIndex . . . 108
5.21.3.3 getFirstRowIndex . . . 108
5.21.3.4 moveToNext . . . 108
5.21.3.5 moveToStart . . . 108
5.21.3.6 zax . . . 108
5.21.3.7 zxa . . . 108
5.22 HilbertTriplet< T > Class Template Reference . . . 108
5.22.1 Detailed Description . . . 110
5.22.2 Constructor & Destructor Documentation . . . 110
5.22.2.1 HilbertTriplet . . . 110
5.22.2.2 HilbertTriplet . . . 110
5.22.3 Member Function Documentation . . . 110
5.22.3.1 getHilbertCoordinate . . . 110
5.22.3.2 getLeastSignificantHilbertBits . . . 110
5.22.3.3 getMostSignificantHilbertBits . . . 111

5.22.3.4 i . . . 111
5.22.3.5 j . . . 111
5.22.3.6 save . . . 111
5.22.3.7 save . . . 111
5.22.4 Member Data Documentation . . . 112
5.22.4.1 column . . . 112
5.22.4.2 hilbert1 . . . 112
5.22.4.3 hilbert2 . . . 112
5.22.4.4 row . . . 112
5.22.4.5 value . . . 112
5.23 HilbertTripletCompare< T > Class Template Reference . . . 112
5.23.1 Detailed Description . . . 113
5.23.2 Member Function Documentation . . . 113
5.23.2.1 operator() . . . 113
5.24 HTS< T > Class Template Reference . . . 113
5.24.1 Detailed Description . . . 115
5.24.2 Constructor & Destructor Documentation . . . 115
5.24.2.1 HTS . . . 115
5.24.2.2 HTS . . . 115
5.24.2.3 HTS . . . 115
5.24.3 Member Function Documentation . . . 115
5.24.3.1 bytesUsed . . . 115
5.24.3.2 cmp . . . 116
5.24.3.3 find . . . 116
5.24.3.4 getFirstIndexPair . . . 116
5.24.3.5 load . . . 116
5.24.3.6 saveBinary . . . 116
5.24.3.7 zax . . . 117
5.24.3.8 zxa . . . 117
5.24.4 Member Data Documentation . . . 117
5.24.4.1 ds . . . 117
5.25 ICRS< T, _i_value > Class Template Reference . . . 117
5.25.1 Detailed Description . . . 119
5.25.2 Constructor & Destructor Documentation . . . 120
5.25.2.1 ICRS . . . 120
5.25.2.2 ICRS . . . 120
5.25.2.3 ICRS . . . 120
5.25.2.4 ICRS . . . 120
5.25.2.5 ICRS . . . 120
5.25.2.6 ∼ICRS . . . 121

5.25.3 Member Function Documentation . . . 121
5.25.3.1 bytesUsed . . . 121
5.25.3.2 compareTriplets . . . 121
5.25.3.3 getFirstIndexPair . . . 121
5.25.3.4 load . . . 121
5.25.3.5 setStartingPos . . . 122
5.25.3.6 zax . . . 122
5.25.3.7 ZaX . . . 122
5.25.3.8 zxa . . . 122
5.25.3.9 ZXa . . . 123
5.25.4 Member Data Documentation . . . 123
5.25.4.1 bytes . . . 123
5.25.4.2 c_ind . . . 123
5.25.4.3 ds . . . 123
5.25.4.4 fillIn . . . 123
5.25.4.5 r_ind . . . 123
5.26 MachineInfo Class Reference . . . 124
5.26.1 Detailed Description . . . 124
5.27 Matrix< T > Class Template Reference . . . 124
5.27.1 Detailed Description . . . 126
5.27.2 Constructor & Destructor Documentation . . . 126
5.27.2.1 Matrix . . . 126
5.27.2.2 ∼Matrix . . . 126
5.27.3 Member Function Documentation . . . 126
5.27.3.1 bytesUsed . . . 126
5.27.3.2 m . . . 127
5.27.3.3 mv . . . 127
5.27.3.4 n . . . 127
5.27.3.5 nzs . . . 127
5.27.3.6 zax . . . 128
5.27.3.7 zax . . . 129
5.27.3.8 ZaX . . . 129
5.27.3.9 zxa . . . 130
5.27.3.10 ZXa . . . 131
5.27.3.11 zxa . . . 131
5.28 Matrix2HilbertCoordinates Class Reference . . . 131
5.28.1 Detailed Description . . . 132
5.28.2 Member Function Documentation . . . 132
5.28.2.1 IntegerToHilbert . . . 132
5.28.3 Member Data Documentation . . . 132

5.28.3.1 BITWIDTH . . . 132
5.29 McCRS< T > Class Template Reference . . . 133
5.29.1 Detailed Description . . . 134
5.29.2 Constructor & Destructor Documentation . . . 135
5.29.2.1 McCRS . . . 135
5.29.2.2 McCRS . . . 135
5.29.2.3 McCRS . . . 135
5.29.2.4 McCRS . . . 135
5.29.2.5 McCRS . . . 135
5.29.2.6 ∼McCRS . . . 136
5.29.3 Member Function Documentation . . . 136
5.29.3.1 bytesUsed . . . 136
5.29.3.2 columnIndices . . . 136
5.29.3.3 find . . . 136
5.29.3.4 getFirstIndexPair . . . 137
5.29.3.5 load . . . 137
5.29.3.6 mv . . . 137
5.29.3.7 random_access . . . 137
5.29.3.8 rowJump . . . 137
5.29.3.9 values . . . 137
5.29.3.10 zax . . . 138
5.29.4 Member Data Documentation . . . 139
5.29.4.1 col_ind . . . 139
5.29.4.2 ds . . . 139
5.29.4.3 row_start . . . 139
5.30 MinCCS< T > Class Template Reference . . . 139
5.30.1 Detailed Description . . . 140
5.30.2 Member Function Documentation . . . 140
5.30.2.1 in_readout . . . 140
5.30.2.2 post_readout . . . 140
5.30.2.3 pre_readout . . . 141
5.31 MinCRS< T > Class Template Reference . . . 141
5.31.1 Detailed Description . . . 142
5.31.2 Member Function Documentation . . . 142
5.31.2.1 in_readout . . . 142
5.31.2.2 post_readout . . . 142
5.31.2.3 pre_readout . . . 142
5.32 MKLCRS< T > Class Template Reference . . . 143
5.32.1 Detailed Description . . . 144
5.32.2 Constructor & Destructor Documentation . . . 145

5.32.2.1 MKLCRS . . . 145
5.32.2.2 MKLCRS . . . 145
5.32.2.3 MKLCRS . . . 145
5.32.2.4 MKLCRS . . . 145
5.32.2.5 MKLCRS . . . 145
5.32.2.6 ∼MKLCRS . . . 146
5.32.3 Member Function Documentation . . . 146
5.32.3.1 mv . . . 146
5.32.3.2 zax . . . 146
5.32.4 Member Data Documentation . . . 146
5.32.4.1 _col_ind . . . 146
5.32.4.2 _m . . . 146
5.32.4.3 _n . . . 146
5.32.4.4 _one . . . 147
5.32.4.5 _row_start . . . 147
5.32.4.6 descr . . . 147
5.32.4.7 trans . . . 147
5.33 RDB_shared_data< T > Class Template Reference . . . 147
5.33.1 Detailed Description . . . 148
5.33.2 Constructor & Destructor Documentation . . . 148
5.33.2.1 RDB_shared_data . . . 148
5.33.3 Member Data Documentation . . . 149
5.33.3.1 cond . . . 149
5.33.3.2 end_cond . . . 149
5.33.3.3 end_mutex . . . 149
5.33.3.4 end_sync . . . 149
5.33.3.5 id . . . 149
5.33.3.6 input . . . 149
5.33.3.7 local_y . . . 149
5.33.3.8 mode . . . 149
5.33.3.9 mutex . . . 149
5.33.3.10 original . . . 150
5.33.3.11 output . . . 150
5.33.3.12 output_vector_offset . . . 150
5.33.3.13 output_vector_size . . . 150
5.33.3.14 P . . . 150
5.33.3.15 sync . . . 150
5.34 RDBHilbert< T, MatrixType > Class Template Reference . . . 150
5.34.1 Detailed Description . . . 153
5.34.2 Constructor & Destructor Documentation . . . 153

5.34.2.1 RDBHilbert . . . 153
5.34.2.2 RDBHilbert . . . 153
5.34.2.3 ∼RDBHilbert . . . 153
5.34.3 Member Function Documentation . . . 153
5.34.3.1 bytesUsed . . . 153
5.34.3.2 collectY . . . 153
5.34.3.3 end . . . 154
5.34.3.4 getFirstIndexPair . . . 154
5.34.3.5 load . . . 154
5.34.3.6 mv . . . 154
5.34.3.7 reset . . . 154
5.34.3.8 set_p_translate . . . 155
5.34.3.9 synchronise . . . 155
5.34.3.10 thread . . . 155
5.34.3.11 wait . . . 155
5.34.3.12 zax . . . 156
5.34.3.13 zax . . . 156
5.34.3.14 zxa . . . 156
5.34.3.15 zxa . . . 156
5.34.4 Member Data Documentation . . . 156
5.34.4.1 threads . . . 156
5.35 RDCSB< T > Class Template Reference . . . 157
5.35.1 Detailed Description . . . 159
5.35.2 Member Function Documentation . . . 159
5.35.2.1 getFirstIndexPair . . . 159
5.35.2.2 load . . . 159
5.35.2.3 mv . . . 159
5.36 RDCSB_shared_data< T > Class Template Reference . . . 159
5.36.1 Detailed Description . . . 160
5.36.2 Constructor & Destructor Documentation . . . 160
5.36.2.1 RDCSB_shared_data . . . 160
5.37 RDScheme< T, DS > Class Template Reference . . . 161
5.37.1 Detailed Description . . . 163
5.37.2 Constructor & Destructor Documentation . . . 163
5.37.2.1 RDScheme . . . 163
5.37.2.2 RDScheme . . . 163
5.37.2.3 ∼RDScheme . . . 163
5.37.3 Member Function Documentation . . . 163
5.37.3.1 bytesUsed . . . 163
5.37.3.2 collectY . . . 163

5.37.3.3 end . . . 164
5.37.3.4 getFirstIndexPair . . . 164
5.37.3.5 load . . . 164
5.37.3.6 mv . . . 164
5.37.3.7 synchronise . . . 165
5.37.3.8 thread . . . 165
5.37.3.9 zxa . . . 165
5.37.3.10 zxa . . . 165
5.38 RDScheme_shared_data< T > Class Template Reference . . . 165
5.38.1 Detailed Description . . . 166
5.38.2 Constructor & Destructor Documentation . . . 167
5.38.2.1 RDScheme_shared_data . . . 167
5.38.2.2 RDScheme_shared_data . . . 167
5.38.3 Member Data Documentation . . . 167
5.38.3.1 bytes . . . 167
5.38.3.2 cond . . . 167
5.38.3.3 end_cond . . . 167
5.38.3.4 end_mutex . . . 167
5.38.3.5 end_sync . . . 168
5.38.3.6 local_y . . . 168
5.38.3.7 mutex . . . 168
5.38.3.8 original . . . 168
5.38.3.9 output_vector_offset . . . 168
5.38.3.10 output_vector_size . . . 168
5.38.3.11 sync . . . 168
5.39 SBDTree Class Reference . . . 168
5.39.1 Detailed Description . . . 170
5.39.2 Constructor & Destructor Documentation . . . 170
5.39.2.1 SBDTree . . . 170
5.39.2.2 ∼SBDTree . . . 170
5.39.3 Member Function Documentation . . . 171
5.39.3.1 build . . . 171
5.39.3.2 columnBounds . . . 172
5.39.3.3 getSeparatorBB . . . 172
5.39.3.4 left . . . 172
5.39.3.5 right . . . 172
5.39.3.6 rowBounds . . . 173
5.39.3.7 up . . . 173
5.39.4 Member Data Documentation . . . 173
5.39.4.1 c_hi . . . 173

5.39.4.2 c_lo . . . 173
5.39.4.3 left_child . . . 173
5.39.4.4 NO_SUCH_NODE . . . 173
5.39.4.5 nodes . . . 173
5.39.4.6 parent . . . 174
5.39.4.7 r_hi . . . 174
5.39.4.8 r_lo . . . 174
5.39.4.9 right_child . . . 174
5.39.4.10 root . . . 174
5.39.4.11 root_is_set . . . 174
5.40 SepLast< T > Class Template Reference . . . 174
5.40.1 Detailed Description . . . 175
5.40.2 Member Function Documentation . . . 175
5.40.2.1 in_readout . . . 176
5.40.2.2 post_readout . . . 176
5.40.2.3 pre_readout . . . 176
5.41 shared_data< T > Class Template Reference . . . 176
5.41.1 Detailed Description . . . 177
5.41.2 Constructor & Destructor Documentation . . . 177
5.41.2.1 shared_data . . . 177
5.41.2.2 shared_data . . . 178
5.41.3 Member Data Documentation . . . 179
5.41.3.1 cond . . . 179
5.41.3.2 end_cond . . . 179
5.41.3.3 end_mutex . . . 179
5.41.3.4 end_sync . . . 179
5.41.3.5 input . . . 179
5.41.3.6 local_y . . . 179
5.41.3.7 mutex . . . 180
5.41.3.8 original . . . 180
5.41.3.9 output . . . 180
5.41.3.10 output_vector_offset . . . 180
5.41.3.11 output_vector_size . . . 180
5.41.3.12 sync . . . 180
5.42 SparseMatrix< T, IND > Class Template Reference . . . 180
5.42.1 Detailed Description . . . 182
5.42.2 Constructor & Destructor Documentation . . . 182
5.42.2.1 SparseMatrix . . . 182
5.42.2.2 SparseMatrix . . . 182
5.42.2.3 ∼SparseMatrix . . . 182

5.42.3 Member Function Documentation . . . 182
5.42.3.1 getFirstIndexPair . . . 182
5.42.3.2 load . . . 183
5.42.3.3 loadFromFile . . . 183
5.42.3.4 m . . . 183
5.42.3.5 mv . . . 184
5.42.3.6 n . . . 184
5.42.3.7 nzs . . . 184
5.42.3.8 zax . . . 184
5.42.3.9 zxa . . . 185
5.42.4 Member Data Documentation . . . 185
5.42.4.1 nnz . . . 185
5.42.4.2 nor . . . 185
5.42.4.3 zero_element . . . 185
5.43 SVM< T > Class Template Reference . . . 186
5.43.1 Detailed Description . . . 187
5.43.2 Constructor & Destructor Documentation . . . 188
5.43.2.1 SVM . . . 188
5.43.2.2 SVM . . . 188
5.43.2.3 SVM . . . 188
5.43.2.4 SVM . . . 188
5.43.2.5 SVM . . . 188
5.43.2.6 ∼SVM . . . 189
5.43.3 Member Function Documentation . . . 189
5.43.3.1 bytesUsed . . . 189
5.43.3.2 find . . . 189
5.43.3.3 getData . . . 189
5.43.3.4 getFirstIndexPair . . . 189
5.43.3.5 load . . . 190
5.43.3.6 load . . . 190
5.43.3.7 random_access . . . 190
5.43.3.8 zax . . . 190
5.43.3.9 zxa . . . 191
5.44 Upscaler< T >::treeInOrderIterator Class Reference . . . 191
5.44.1 Detailed Description . . . 192
5.44.2 Constructor & Destructor Documentation . . . 192
5.44.2.1 treeInOrderIterator . . . 192
5.44.3 Member Function Documentation . . . 192
5.44.3.1 next . . . 192
5.45 Upscaler< T >::treeIterator Class Reference . . . 193

5.45.1 Detailed Description . . . 193
5.45.2 Constructor & Destructor Documentation . . . 193
5.45.2.1 treeIterator . . . 193
5.45.3 Member Function Documentation . . . 194
5.45.3.1 next . . . 194
5.45.3.2 position . . . 194
5.45.4 Member Data Documentation . . . 194
5.45.4.1 ID . . . 194
5.45.4.2 p . . . 194
5.45.4.3 p_processed . . . 194
5.45.4.4 processed . . . 195
5.45.4.5 walk . . . 195
5.46 Upscaler< T >::treePostOrderIterator Class Reference . . . 195
5.46.1 Detailed Description . . . 196
5.46.2 Constructor & Destructor Documentation . . . 196
5.46.2.1 treePostOrderIterator . . . 196
5.46.3 Member Function Documentation . . . 196
5.46.3.1 next . . . 196
5.47 Triplet< T > Class Template Reference . . . 196
5.47.1 Detailed Description . . . 198
5.47.2 Constructor & Destructor Documentation . . . 198
5.47.2.1 Triplet . . . 198
5.47.2.2 Triplet . . . 198
5.47.2.3 Triplet . . . 198
5.47.3 Member Function Documentation . . . 198
5.47.3.1 i . . . 198
5.47.3.2 j . . . 199
5.47.3.3 load . . . 199
5.47.3.4 loadCRS . . . 199
5.47.3.5 rowOffset . . . 200
5.47.3.6 save . . . 200
5.47.3.7 save . . . 200
5.47.3.8 setColumnPosition . . . 200
5.47.3.9 setPosition . . . 200
5.47.3.10 setRowPosition . . . 200
5.47.3.11 transpose . . . 201
5.47.4 Member Data Documentation . . . 201
5.47.4.1 column . . . 201
5.47.4.2 row . . . 201
5.47.4.3 value . . . 201

5.48 TS< T > Class Template Reference . . . 201
5.48.1 Detailed Description . . . 203
5.48.2 Constructor & Destructor Documentation . . . 203
5.48.2.1 TS . . . 203
5.48.2.2 TS . . . 203
5.48.2.3 TS . . . 203
5.48.2.4 ∼TS . . . 203
5.48.3 Member Function Documentation . . . 203
5.48.3.1 bytesUsed . . . 203
5.48.3.2 getFirstIndexPair . . . 204
5.48.3.3 load . . . 204
5.48.3.4 zax . . . 204
5.48.3.5 ZaX . . . 204
5.48.3.6 zxa . . . 204
5.48.3.7 ZXa . . . 204
5.48.4 Member Data Documentation . . . 205
5.48.4.1 ds . . . 205
5.48.4.2 i . . . 205
5.48.4.3 j . . . 205
5.49 U2< T > Class Template Reference . . . 205
5.49.1 Detailed Description . . . 206
5.49.2 Member Function Documentation . . . 206
5.49.2.1 in_readout . . . 206
5.49.2.2 post_readout . . . 206
5.49.2.3 pre_readout . . . 207
5.50 Upscaler< T > Class Template Reference . . . 207
5.50.1 Detailed Description . . . 209
5.50.2 Constructor & Destructor Documentation . . . 209
5.50.2.1 Upscaler . . . 209
5.50.2.2 ∼Upscaler . . . 209
5.50.3 Member Function Documentation . . . 209
5.50.3.1 determineEmpty . . . 209
5.50.3.2 getSubTree . . . 210
5.50.3.3 getSubTree . . . 210
5.50.3.4 readout . . . 211
5.50.3.5 updateMinMax . . . 211
5.50.4 Member Data Documentation . . . 211
5.50.4.1 containsPID . . . 211
5.50.4.2 nonzeroes . . . 211
5.51 vecBICRS< T, block_length_row, block_length_column, _i_value > Class Template Reference . . . 212

5.51.1 Detailed Description . . . 214
5.51.2 Constructor & Destructor Documentation . . . 214
5.51.2.1 vecBICRS . . . 214
5.51.2.2 vecBICRS . . . 214
5.51.2.3 vecBICRS . . . 215
5.51.2.4 vecBICRS . . . 215
5.51.2.5 vecBICRS . . . 215
5.51.2.6 ∼vecBICRS . . . 216
5.51.3 Member Function Documentation . . . 216
5.51.3.1 addPadding . . . 216
5.51.3.2 allocate . . . 216
5.51.3.3 bytesUsed . . . 216
5.51.3.4 compareTriplets . . . 217
5.51.3.5 getFirstIndexPair . . . 217
5.51.3.6 load . . . 217
5.51.3.7 postProcessRowIncrements . . . 217
5.51.3.8 prepareBlockIteration . . . 218
5.51.3.9 zax . . . 218
5.51.3.10 zxa . . . 219
5.51.4 Member Data Documentation . . . 219
5.51.4.1 allocsize . . . 219
5.51.4.2 block_length . . . 219
5.51.4.3 bytes . . . 219
5.51.4.4 c_ind . . . 219
5.51.4.5 ds . . . 220
5.51.4.6 fillIn . . . 220
5.51.4.7 jumpsize . . . 220
5.51.4.8 r_ind . . . 220
5.52 ZZ_CRS< T, _i_value > Class Template Reference . . . 220
5.52.1 Detailed Description . . . 222
5.52.2 Constructor & Destructor Documentation . . . 222
5.52.2.1 ZZ_CRS . . . 222
5.52.2.2 ZZ_CRS . . . 222
5.52.2.3 ZZ_CRS . . . 223
5.52.2.4 ZZ_CRS . . . 223
5.52.2.5 ZZ_CRS . . . 223
5.52.2.6 ∼ZZ_CRS . . . 223
5.52.3 Member Function Documentation . . . 223
5.52.3.1 bytesUsed . . . 224
5.52.3.2 compareTripletsLTR . . . 224

5.52.3.3 compareTripletsRTL . . . 224
5.52.3.4 getFirstIndexPair . . . 224
5.52.3.5 load . . . 224
5.52.3.6 zax . . . 224
5.52.3.7 zxa . . . 225
5.52.4 Member Data Documentation . . . 225
5.52.4.1 col_ind . . . 225
5.52.4.2 ds . . . 225
5.52.4.3 row_start . . . 225
5.53 ZZ_ICRS< T > Class Template Reference . . . 225
5.53.1 Detailed Description . . . 227
5.53.2 Constructor & Destructor Documentation . . . 227
5.53.2.1 ZZ_ICRS . . . 227
5.53.2.2 ZZ_ICRS . . . 227
5.53.2.3 ZZ_ICRS . . . 228
5.53.2.4 ZZ_ICRS . . . 228
5.53.2.5 ZZ_ICRS . . . 228
5.53.2.6 ∼ZZ_ICRS . . . 228
5.53.3 Member Function Documentation . . . 228
5.53.3.1 bytesUsed . . . 228
5.53.3.2 compareTriplets . . . 229
5.53.3.3 getFirstIndexPair . . . 229
5.53.3.4 load . . . 229
5.53.3.5 zax . . . 229
5.53.3.6 zxa . . . 229
5.53.4 Member Data Documentation . . . 230
5.53.4.1 c_ind . . . 230
5.53.4.2 ds . . . 230
5.53.4.3 jumps . . . 230
5.53.4.4 r_ind . . . 230

6 File Documentation **231**

6.1 ICRS.hpp File Reference . . . 231
6.1.1 Detailed Description . . . 232
6.2 Matrix.hpp File Reference . . . 232
6.2.1 Detailed Description . . . 233

Index **234**

Chapter 1

Sparse Library

This is the doxygen code documentation of the sparse library. The latest version can be found at http://albert-jan.yzelman.net/software#SL

Contact info is available via the same link. This doxygen is not exhaustive. A good starting point on usage info is **SparseMatrix** (p. **??**). Which implementing subclass (i.e., which sparse matrix storage format) is most suitable for you depends on your application and usage scenario. For sequential computations a good baseline might be **CBICRS** (p. **??**), for parallel computations the **RDBHilbert** (p. **??**) class performs best overall.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BigInt . . . ??
BlockOrderer< T > . . . ??
- BlockCRS< T > . . . ??
- Duck< T > . . . ??
- MinCCS< T > . . . ??
- MinCRS< T > . . . ??
- SepLast< T > . . . ??
- U2< T > . . . ??

CBICRS_factory< _t_value > . . . ??
FileToVT . . . ??
HilbertArrayInterface< T > . . . ??
- HilbertArray< T, I, hI, mI > . . . ??

HilbertTriplet< T > . . . ??
HilbertTripletCompare< T > . . . ??
MachineInfo . . . ??
Matrix< T > . . . ??
- SparseMatrix< T, IND > . . . ??
  - CCSWrapper< T, SparseMatrixType, IND > . . . ??
- SparseMatrix< T, LI > . . . ??
  - BlockHilbert< T > . . . ??
    - BisectionHilbert< T > . . . ??
  - HBICRS< T > . . . ??
  - ZZ_CRS< T, _i_value > . . . ??
  - ZZ_ICRS< T > . . . ??
- SparseMatrix< T, ULI > . . . ??
  - BetaHilbert< T > . . . ??
  - CompressedHilbert< T > . . . ??
  - CRS< T > . . . ??
    - MKLCRS< T > . . . ??
  - Hilbert< T > . . . ??
  - HTS< T > . . . ??
  - ICRS< T, _i_value > . . . ??
  - McCRS< T > . . . ??
  - RDBHilbert< T, MatrixType > . . . ??
  - RDCSB< T > . . . ??
  - RDScheme< T, DS > . . . ??
  - SVM< T > . . . ??

TS< T > . . . . . ??
vecBICRS< T, block_length_row, block_length_column, _i_value > . . . . . ??
SparseMatrix< T, unsigned long int > . . . . . ??
DD_MATRIX< T, number_of_diagonals, diagonal_offsets > . . . . . ??
Matrix2HilbertCoordinates . . . . . ??
Matrix< _t_value > . . . . . ??
SparseMatrix< _t_value, _i_value > . . . . . ??
FBICRS< _t_value, _i_value, _sub_ds, logBeta > . . . . . ??
SparseMatrix< _t_value, LI > . . . . . ??
HBICRS< _t_value > . . . . . ??
SparseMatrix< _t_value, ULI > . . . . . ??
BICRS< _t_value, _i_value > . . . . . ??
CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value > . . . . . ??
Matrix< double > . . . . . ??
SparseMatrix< double, size_t > . . . . . ??
CuHyb . . . . . ??
RDB_shared_data< T > . . . . . ??
RDCSB_shared_data< T > . . . . . ??
RDScheme_shared_data< T > . . . . . ??
SBDTree . . . . . ??
Upscaler< T > . . . . . ??
shared_data< T > . . . . . ??
Upscaler< T >::treeIterator . . . . . ??
Upscaler< T >::treeInOrderIterator . . . . . ??
Upscaler< T >::treePostOrderIterator . . . . . ??
Triplet< T > . . . . . ??

Chapter 3

Class Index

3.1 Class List

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

BetaHilbert< **T** >
The Beta **Hilbert** (p. **??**) triplet scheme . . . . . **??**

BICRS< **_t_value, _i_value** >
Bi-directional Incremental Compressed Row Storage scheme . . . . . **??**

BigInt
A 128-bit integer, with overloaded comparison operators . . . . . **??**

BisectionHilbert< **T** >
The Bisection **Hilbert** (p. **??**) triplet scheme . . . . . **??**

BlockCRS< **T** >
Codes the **CRS** (p. **??**) block order . . . . . **??**

BlockHilbert< **T** >
The Block **Hilbert** (p. **??**) triplet scheme . . . . . **??**

BlockOrderer< **T** >
Induces a block order by fully traversing an **SBDTree** (p. **??**) . . . . . **??**

CBICRS< **_t_value, _master_i_value, _master_j_value, _i_value, _j_value** >
Compressed Bi-directional Incremental Compressed Row Storage (**BICRS** (p. **??**)) scheme . . **??**

CBICRS_factory< **_t_value** >
Factory for the Compressed Bi-directional Incremental Compressed Row Storage scheme . . . **??**

CCSWrapper< **T, SparseMatrixType, IND** >
Automatically transforms a row-major scheme into an column-major scheme . . . . . **??**

CompressedHilbert< **T** >
A **Hilbert** (p. **??**) scheme backed by a specialised storage format . . . . . **??**

CRS< **T** >
The compressed row storage sparse matrix data structure . . . . . **??**

CuHyb
Wrapper class for the CuSparse HYB data structure for CUDA C . . . . . **??**

DD_MATRIX< **T, number_of_diagonals, diagonal_offsets** >
The dense diagonal matrix scheme; a storage scheme for sparse matrices consisting of only dense diagonals . . . . . **??**

Duck< **T** >
Codes the Minimal CCS block order . . . . . **??**

FBICRS< **_t_value, _i_value, _sub_ds, logBeta** >
Hierarchical **BICRS** (p. **??**) with fixed subblock size and distribution . . . . . **??**

FileToVT
Class responsible for reading in matrix market files and converting them to vector< Triplet > format . . . . . **??**

HBICRS< **_t_value** >
Hierarchical Bi-directional Incremental Compressed Row Storage scheme . . . . . **??**

Hilbert< **T** >
The **Hilbert** (p. **??**) scheme backed by (C)**BICRS** (p. **??**) . . . . . **??**
HilbertArray< **T, I, hI, mI** >
Actual storage implementation . . . . . **??**
HilbertArrayInterface< **T** >
Class providing an interface to an efficient storage of a 1D array of **Hilbert** (p. **??**) coordinate increments . . . . . **??**
HilbertTriplet< **T** >
Hilbert-coordinate-aware triplet . . . . . **??**
HilbertTripletCompare< **T** >
Class-comparator of **HilbertTriplet** (p. **??**) . . . . . **??**
HTS< **T** >
The **Hilbert** (p. **??**) triplet scheme . . . . . **??**
ICRS< **T, _i_value** >
The *incremental* compressed row storage sparse matrix data structure . . . . . **??**
MachineInfo
Singleton class to get info on the current system . . . . . **??**
Matrix< **T** >
Defines operations common to all matrices, which are implemented in this library . . . . . **??**
Matrix2HilbertCoordinates
Class which maps coordinates to 1D **Hilbert** (p. **??**) Coordinates . . . . . **??**
McCRS< **T** >
The compressed row storage sparse matrix data structure . . . . . **??**
MinCCS< **T** >
Codes the Minimal CCS block order . . . . . **??**
MinCRS< **T** >
Codes the Minimal **CRS** (p. **??**) block order . . . . . **??**
MKLCRS< **T** >
The compressed row storage sparse matrix data structure . . . . . **??**
RDB_shared_data< **T** >
Shared data for **RDBHilbert** (p. **??**) threads . . . . . **??**
RDBHilbert< **T, MatrixType** >
The Beta **Hilbert** (p. **??**) triplet scheme . . . . . **??**
RDCSB< **T** >
Full parallel row-distributed SpMV, based on CSB (**BlockCRS** (p. **??**) + Morton curve + Cilk) and PThreads . . . . . **??**
RDCSB_shared_data< **T** >
Shared data for **RDCSB** (p. **??**) threads . . . . . **??**
RDScheme< **T, DS** >
Full parallel row-distributed SpMV, based on CSB (Morton curve + Cilk) and PThreads . . . . . **??**
RDScheme_shared_data< **T** >
Shared data for **RDScheme** (p. **??**) threads . . . . . **??**
SBDTree
Models a Separated Block Diagonal tree structure . . . . . **??**
SepLast< **T** >
Codes the Minimal CCS block order . . . . . **??**
shared_data< **T** >
Shared data for **BetaHilbert** (p. **??**) threads . . . . . **??**
SparseMatrix< **T, IND** >
Interface common to all sparse matrix storage schemes . . . . . **??**
SVM< **T** >
The sparse vector matrix format . . . . . **??**
Upscaler< **T** >**::treeInOrderIterator**
Same as **treeIterator** (p. **??**), but does in-order traversal instead of pre-order . . . . . **??**
Upscaler< **T** >**::treeIterator**
Pre-order tree iterator . . . . . **??**
Upscaler< **T** >**::treePostOrderIterator**
Same as **treeIterator** (p. **??**), but does post-order traversal instead of pre-order . . . . . **??**

Triplet< **T** >
A single triplet value . . . . . **??**
TS< **T** >
The triplet scheme; a storage scheme for sparse matrices using triplets . . . . . **??**
U2< **T** >
Codes the Minimal CCS block order . . . . . **??**
Upscaler< **T** >
Transforms SBD-structures over q parts into an SBD structure over p parts, with q>p, and q,p both powers of two . . . . . **??**
vecBICRS< **T, block_length_row, block_length_column, _i_value** >
The *incremental* compressed row storage sparse matrix data structure . . . . . **??**
ZZ_CRS< **T, _i_value** >
The zig-zag compressed row storage sparse matrix data structure . . . . . **??**
ZZ_ICRS< **T** >
The zig-zag incremental compressed row storage sparse matrix data structure . . . . . **??**

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

alignment.hpp . . . . . **??**
BetaHilbert.hpp . . . . . **??**
BICRS.hpp . . . . . **??**
BigInt.hpp . . . . . **??**
BisectionHilbert.hpp . . . . . **??**
BlockCRS.hpp . . . . . **??**
BlockHilbert.hpp . . . . . **??**
BlockOrderer.hpp . . . . . **??**
CBICRS.hpp . . . . . **??**
CCSWrapper.hpp . . . . . **??**
CompressedHilbert.hpp . . . . . **??**
CRS.hpp . . . . . **??**
CuHyb.hpp . . . . . **??**
custom_quicksort.hpp . . . . . **??**
DD_MATRIX.hpp . . . . . **??**
Duck.hpp . . . . . **??**
FBICRS.hpp . . . . . **??**
FileToVT.hpp . . . . . **??**
HBICRS.hpp . . . . . **??**
Hilbert.hpp . . . . . **??**
HilbertArray.hpp . . . . . **??**
HilbertTriplet.hpp . . . . . **??**
HilbertTripletCompare.hpp . . . . . **??**
HTS.hpp . . . . . **??**
ICRS.hpp
File created by: A . . . . . **??**
MachineInfo.hpp . . . . . **??**
Matrix.hpp
File created by: A . . . . . **??**
Matrix2HilbertCoordinates.hpp . . . . . **??**
McCRS.hpp . . . . . **??**
MinCCS.hpp . . . . . **??**
MinCRS.hpp . . . . . **??**
MKLCRS.hpp . . . . . **??**
mmio.h . . . . . **??**
RDBHilbert.hpp . . . . . **??**
RDCSB.hpp . . . . . **??**
RDScheme.hpp . . . . . **??**

SBDTree.hpp . . . . . **??**
SepLast.hpp . . . . . **??**
SparseMatrix.hpp . . . . . **??**
SVM.hpp . . . . . **??**
Triplet.hpp . . . . . **??**
TS.hpp . . . . . **??**
U2.hpp . . . . . **??**
ULIdef.hpp . . . . . **??**
Upscaler.hpp . . . . . **??**
util.hpp . . . . . **??**
vecBICRS.hpp . . . . . **??**
ZZ_CRS.hpp . . . . . **??**
ZZ_ICRS.hpp . . . . . **??**

Chapter 5

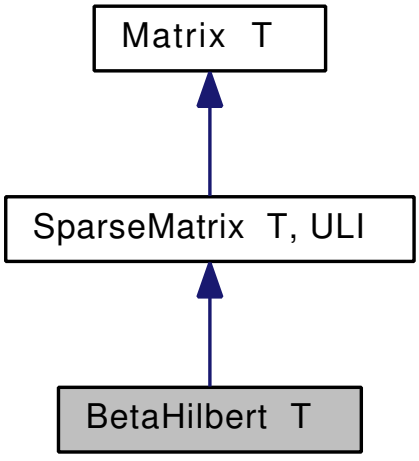
Class Documentation

5.1 BetaHilbert< T > Class Template Reference

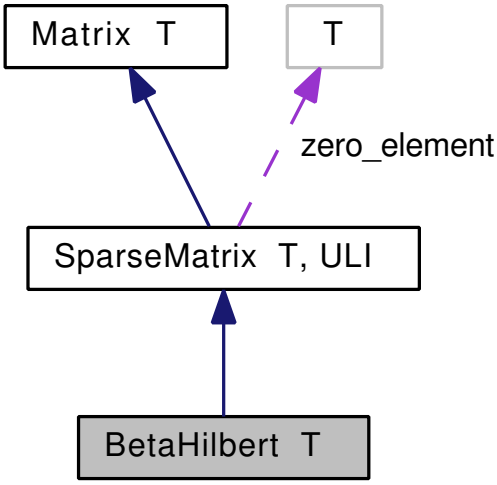
The Beta **Hilbert** (p. **??**) triplet scheme.

`#include <BetaHilbert.hpp>`

Inheritance diagram for BetaHilbert< T >:



Collaboration diagram for BetaHilbert< T >:



Public Member Functions

- **BetaHilbert** (const std::string file, T zero=0, std::vector< unsigned short int > *_p_translate=NULL)

  File-based constructor.
- **BetaHilbert** (std::vector< **Triplet**< T > > &**input**, ULI **m**, ULI **n**, T zero=0, std::vector< unsigned short int > *_p_translate=NULL)

  COO-based constructor.
- virtual ~**BetaHilbert** ()

  Base deconstructor.
- void **wait** ()

  Callee will Wait for end of SpMV.
- virtual void **load** (std::vector< **Triplet**< T > > &**input**, const ULI **m**, const ULI **n**, const T zero)

  Loads from input COO matrix.
- virtual void **zxa** (const T *x, T *z)

  Computes z=xA in place.
- virtual void **zxa** (const T *x, T *z, const size_t repeat)

  Computes z=xA in place, a given number of times successively.
- void **reset** ()

  Reset all local output vectors to zero.
- virtual T * **mv** (const T *x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zax** (const T *x, T *z)

  Computes z=Ax in place.
- virtual void **zax** (const T *x, T *z, const size_t repeat, const clockid_t clock_id, double *elapsed_time)

  Computes z=Ax in place, a given number of times in succession, and measures the time taken.
- virtual void **getFirstIndexPair** (ULI &i, ULI &j)
- virtual size_t **bytesUsed** ()

Static Public Member Functions

- static void **end** (pthread_mutex_t *__**mutex**__, pthread_cond_t *__**cond**__, size_t *__**sync**__, const size_t **P**)

  End synchronisation function.
- static void **synchronise** (pthread_mutex_t *__**mutex**__, pthread_cond_t *__**cond**__, size_t *__**sync**__, const size_t **P**)

Thread synchronisation function.

- static void * **thread** (void *data)

  SPMD code for each thread doing an SpMV.

- static void **collectY** (**shared_data**< T > *shared)

  Code that collects a distributed output vector.

Protected Member Functions

- void **set_p_translate** (std::vector< unsigned short int > *_p_translate)

  Sets p_translate to 0..P-1 by default, or equal to the optionally supplied vector.

Protected Attributes

- std::vector< unsigned short int > **p_translate**

  Which processors to pin threads to.

- const T * **input**

  Input vector.

- T * **output**

  Output vector.

- pthread_t * **threads**

  p_threads associated to this data strcuture

- **shared_data**< T > * **thread_data**

  array of initial thread data

- pthread_mutex_t **mutex**

  Stop/continue mechanism: mutex.

- pthread_cond_t **cond**

  Stop/continue mechanism: condition.

- pthread_mutex_t **end_mutex**

  Wait for end mechanism: mutex.

- pthread_cond_t **end_cond**

  Wait for end mechanism: condition.

- size_t **sync**

  Used for synchronising threads.

- size_t **end_sync**

  Used for construction end signal.

Static Protected Attributes

- static size_t **P** = 0

  Number of threads to fire up.

- static clockid_t **global_clock_id** = 0

  Clock type used for thread-local timing.

- static const ULI **max_n** = **FBICRS**< T >::beta_n

  *Given **FBICRS** (p. **??**), the maximum value for columnwise matrix size.*

- static const ULI **max_m** = **FBICRS**< T >::beta_m

  *Given **FBICRS** (p. **??**), the maximum value for the rowwise matrix size, assuming short ints on **ICRS** (p. **??**) at the lower level.*

Additional Inherited Members

5.1.1 Detailed Description

template<typename T>class BetaHilbert< T >

The Beta **Hilbert** (p. **??**) triplet scheme.

Full parallel SpMV, based on Blocked **Hilbert** (p. **??**) and PThreads. Inspired by Aydin & Gilbert's CSB, and comments by Patrick Amestoy on the **BICRS** (p. **??**) **Hilbert** (p. **??**) scheme.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 template<typename T> BetaHilbert< T >::BetaHilbert (const std::string *file*, T *zero* = 0, std::vector< unsigned short int > * *_p_translate* = NULL) [inline]

File-based constructor.

Parameters

file	Filename to read from.
zero	Zero element of the sparse matrix.
_p_translate	Thread pinning array.

References SparseMatrix< T, ULI >::loadFromFile(), and BetaHilbert< T >::set_p_translate().

5.1.2.2 template<typename T> BetaHilbert< T >::BetaHilbert (std::vector< Triplet< T > > & *input*, ULI *m*, ULI *n*, T *zero* = 0, std::vector< unsigned short int > * *_p_translate* = NULL) [inline]

COO-based constructor.

Parameters

input	COO matrix.
m	The number of rows in the input matrix.
n	The number of columns in the input matrix.
zero	Zero element of the sparse matrix.
_p_translate	Thread pinning array.

References BetaHilbert< T >::input, BetaHilbert< T >::load(), and BetaHilbert< T >::set_p_translate().

5.1.2.3 template<typename T> virtual BetaHilbert< T >::~BetaHilbert ( ) [inline], [virtual]

Base deconstructor.

References BetaHilbert< T >::cond, BetaHilbert< T >::mutex, BetaHilbert< T >::P, BetaHilbert< T >::thread_data, and BetaHilbert< T >::threads.

5.1.3 Member Function Documentation

5.1.3.1 template<typename T> virtual size_t BetaHilbert< T >::bytesUsed ( ) [inline], [virtual]

Returns

The number of bytes used by all threads corresponding to this scheme.

Implements **Matrix< T >** (p. **??**).

References BetaHilbert< T >::P, and BetaHilbert< T >::thread_data.

5.1.3.2 template<typename T> static void BetaHilbert< T >::collectY (shared_data< T > * *shared*) [inline], [static]

Code that collects a distributed output vector.

Parameters

shared	The local thread data.

References shared_data< T >::cond, shared_data< T >::id, shared_data< T >::local_y, SparseMatrix< T, ULI >::m(), shared_data< T >::mutex, shared_data< T >::output, shared_data< T >::output_vector_size, shared_data< T >::P, shared_data< T >::sync, and BetaHilbert< T >::synchronise().

Referenced by BetaHilbert< T >::thread().

5.1.3.3 template<typename T> static void BetaHilbert< T >::end (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

End synchronisation function.

Parameters

mutex	Which mutex to sync with.
cond	Which condition to sync with.
sync	Sync counter to be used.
P	How many SPMD processes are running.

Referenced by BetaHilbert< T >::thread().

5.1.3.4 template<typename T> virtual void BetaHilbert< T >::getFirstIndexPair (ULI & *i,* ULI & *j*) [inline], [virtual]

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

5.1.3.5 template<typename T> virtual void BetaHilbert< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

Loads from input COO matrix.

Parameters

input	The input COO matrix.
m	Input matrix row dimension.
n	Input matrix column dimension.
zero	Zero element of the input matrix.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References BetaHilbert< T >::cond, BetaHilbert< T >::end_cond, BetaHilbert< T >::end_mutex, BetaHilbert< T >::end_sync, BetaHilbert< T >::input, SparseMatrix< T, ULI >::m(), BetaHilbert< T >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, BetaHilbert< T >::output, BetaHilbert< T >::P, BetaHilbert< T >::p_translate, BetaHilbert< T >::sync, BetaHilbert< T >::thread(), BetaHilbert< T >::thread_data, BetaHilbert< T >::threads, BetaHilbert< T >::wait(), and SparseMatrix< T, ULI >::zero_element.

Referenced by BetaHilbert< T >::BetaHilbert().

5.1.3.6 template<typename T> virtual T* BetaHilbert< T >::mv (const T * x) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Parameters

x	The input vector of length n.

Returns

Newly allocated output vector of length m.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, BetaHilbert< T >::reset(), BetaHilbert< T >::zax(), and SparseMatrix< T, ULI >::zero_element.

5.1.3.7 template<typename T> void BetaHilbert< T >::set_p_translate (std::vector< unsigned short int > * *_p_translate*) [inline], [protected]

Sets p_translate to 0..P-1 by default, or equal to the optionally supplied vector.

References MachineInfo::cores(), MachineInfo::getInstance(), BetaHilbert< T >::P, and BetaHilbert< T >::p_translate.

Referenced by BetaHilbert< T >::BetaHilbert().

5.1.3.8 template<typename T> static void BetaHilbert< T >::synchronise (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

Thread synchronisation function.

Parameters

mutex	Mutex used for synchronisation.
cond	Condition used for synchronisation.
sync	Synchronisation counter.
P	The number of SPMD processes running.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.1.3.9 template<typename T> static void* BetaHilbert< T >::thread (void * *data*) [inline], [static]

SPMD code for each thread doing an SpMV.

Parameters

data	Pointer to the thread-local data.

Returns

Pointer to the thread output data.

References shared_data< T >::bytes, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::bytesUsed(), BetaHilbert< T >::collectY(), shared_data< T >::cond, BetaHilbert< T >::cond, BetaHilbert< T >::end(), shared_data< T >::end_cond, shared_data< T >::end_mutex, shared_data< T >::end_sync, BetaHilbert< T >::global_clock_id, Triplet< T >::i(), shared_data< T >::id, shared_data< T >::input, Matrix2HilbertCoordinates::IntegerToHilbert(), shared_data< T >::local_y, SparseMatrix< T, ULI >::m(), BetaHilbert< T >::max_m, BetaHilbert< T >::max_n, shared_data< T >::mode, shared_data< T >::mutex, BetaHilbert< T >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, shared_data< T >::nzb, shared_data< T >::nzc, shared_data< T >::original, shared_data< T >::output, shared_data< T >::output_vector_size, shared_data< T >::P, BetaHilbert< T >::P, shared_data< T >::repeat, shared_data< T >::sync, BetaHilbert< T >::synchronise(), shared_data< T >::time, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

Referenced by BetaHilbert< T >::load().

5.1.3.10 template<typename T> void BetaHilbert< T >::wait ( ) [inline]

Callee will Wait for end of SpMV.

References BetaHilbert< T >::end_cond, and BetaHilbert< T >::end_mutex.

Referenced by BetaHilbert< T >::load(), BetaHilbert< T >::reset(), BetaHilbert< T >::zax(), and BetaHilbert< T >::zxa().

5.1.3.11 template<typename T> virtual void BetaHilbert< T >::zax (const T * *x*, T * *z*) [inline], [virtual]

Computes z=Ax in place.

Parameters

x	The input vector of length n.
z	The output vector of length m.

Referenced by BetaHilbert< T >::mv().

5.1.3.12 template<typename T> virtual void BetaHilbert< T >::zax (const T * *x*, T * *z*, const size_t *repeat*, const clockid_t *clock_id*, double * *elapsed_time*) [inline], [virtual]

Computes z=Ax in place, a given number of times in succession, and measures the time taken.

Parameters

x	The input vector of length n.
z	The output vector of length m.
repeat	The number of times this SpMV will be repeated.
clock_id	Which system clock to use for timing.
elapsed_time	Where to add the elapsed time to.

References BetaHilbert< T >::cond, BetaHilbert< T >::end_mutex, BetaHilbert< T >::global_clock_id, BetaHilbert< T >::input, BetaHilbert< T >::mutex, SparseMatrix< T, ULI >::nor, BetaHilbert< T >::output, BetaHilbert< T >::P, BetaHilbert< T >::thread_data, and BetaHilbert< T >::wait().

5.1.3.13 template<typename T> virtual void BetaHilbert< T >::zxa (const T * *x*, T * *z*) [inline], [virtual]

Computes z=xA in place.

Parameters

x	The input vector of length m.
z	The output vector of length n.

5.1.3.14 template<typename T> virtual void BetaHilbert< T >::zxa (const T * *x*, T * *z*, const size_t *repeat*) [inline], [virtual]

Computes z=xA in place, a given number of times successively.

Parameters

x	The input vector of length m.
z	The output vector of length n.

repeat	The number of times to repeat the in-place SpMV.

References BetaHilbert< T >::cond, BetaHilbert< T >::end_mutex, BetaHilbert< T >::input, BetaHilbert< T >::mutex, SparseMatrix< T, ULI >::nor, BetaHilbert< T >::output, BetaHilbert< T >::P, BetaHilbert< T >::thread_data, and BetaHilbert< T >::wait().

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

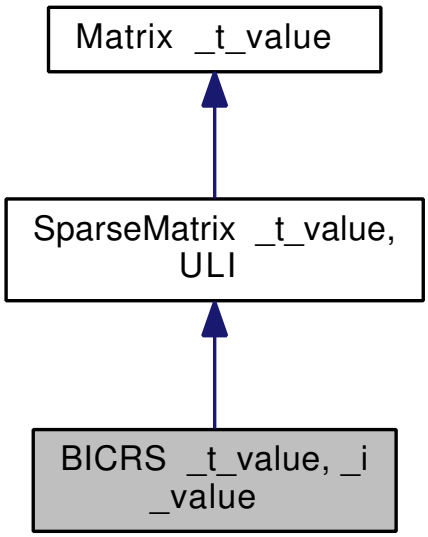
- BetaHilbert.hpp

5.2 BICRS< _t_value, _i_value > Class Template Reference

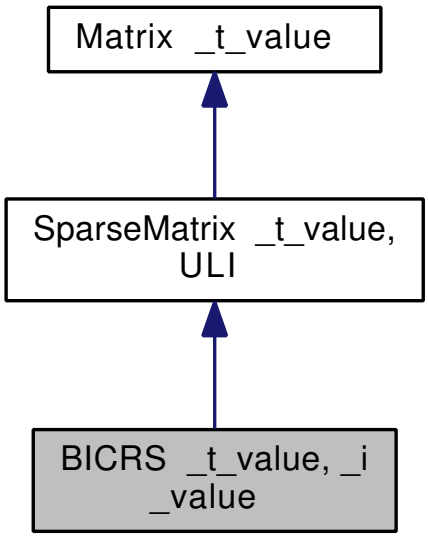
Bi-directional Incremental Compressed Row Storage scheme.

`#include <BICRS.hpp>`

Inheritance diagram for BICRS< _t_value, _i_value >:



Collaboration diagram for BICRS< _t_value, _i_value >:



Public Member Functions

- virtual ~**BICRS** ()

  Base deconstructor.
- **BICRS** ()

  Base constructor.
- **BICRS** (std::string file, _t_value zero=0)

  Base constructor.
- **BICRS** (_i_value *row, _i_value *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)

  Base constructor.
- **BICRS** (std::vector< **Triplet**< _t_value > > &input, ULI **m**, ULI **n**, _t_value zero=0)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< _t_value > > &input, ULI **m**, ULI **n**, _t_value zero)

  This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.
- void **load** (_i_value *row, _i_value *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=xA, but does not allocate y itself.
- virtual void **zax** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=Ax, but does not allocate y itself.
- virtual void **zax_fb** (_t_value *__restrict__ x_f, _t_value *__restrict__ y_f)

  Calculates y=Ax, but does not allocate y itself.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Protected Attributes

- ULI **r_start**

  Stores the row start position.
- ULI **c_start**

  Stores the column start position.
- ULI **r_end**

  Stores the row end position.
- ULI **c_end**

  Stores the column end position.
- ULI **jumps**

  Stores the number of row jumps.
- _i_value * **r_inc**

  Stores the row jumps; size is at maximum the number of nonzeros.
- _i_value * **c_inc**

  Stores the column jumps; size is exactly the number of nonzeros.
- _t_value * **vals**

  Stores the values of the individual nonzeros.
- _i_value **ntt**

  Caches n times two.

Additional Inherited Members

5.2.1 Detailed Description

template<typename _t_value, typename _i_value = LI>class BICRS< _t_value, _i_value >

Bi-directional Incremental Compressed Row Storage scheme.

Supports jumping back and forward within columns. Supports jumping back and forward between rows. Main storage direction in column-wise. Storage requirements are 2nz plus the number of row jumps required. Many row jumps are disadvantageous to storage as well as speed.

Parameters

_t_value	The type of the nonzeros in the matrix.

Warning: this class uses assertions! For optimal performance, define the NDEBUG flag (e.g., pass -DNDEBUG as a compiler flag).

5.2.2 Constructor & Destructor Documentation

5.2.2.1 template<typename _t_value, typename _i_value = LI> virtual BICRS< _t_value, _i_value >::~BICRS ( ) [inline],[virtual]

Base deconstructor.

References BICRS< _t_value, _i_value >::c_inc, BICRS< _t_value, _i_value >::r_inc, and BICRS< _t_value, _i_value >::vals.

5.2.2.2 template<typename _t_value, typename _i_value = LI> BICRS< _t_value, _i_value >::BICRS ( ) [inline]

Base constructor.

5.2.2.3 template<typename _t_value, typename _i_value = LI> BICRS< _t_value, _i_value >::BICRS (std::string *file,* _t_value *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< _t_value, ULI >::loadFromFile().

5.2.2.4 template<typename _t_value, typename _i_value = LI> BICRS< _t_value, _i_value >::BICRS (_i_value * *row,* _i_value * *col,* _t_value * *val,* ULI *m,* ULI *n,* ULI *nz,* _t_value *zero*) [inline]

Base constructor.

Stores triplets in exactly the same order as passed to this constructor.

Parameters

row	The row numbers of the individual nonzeros.
col	The column numbers of the individual nonzeros.
val	The values of the nonzeros.
m	Number of matrix rows.
n	Number of matrix columns.
nz	Number of nonzeros.
zero	Which value is to be regarded zero here.

References BICRS< _t_value, _i_value >::load().

5.2.2.5 template<typename _t_value, typename _i_value = LI> BICRS< _t_value, _i_value >::BICRS (std::vector< Triplet< _t_value > > & *input,* ULI *m,* ULI *n,* _t_value *zero* = 0) [inline]

Base constructor.

See Also

SparseMatrix::SparseMatrix(input, m, n, zero)

References BICRS< _t_value, _i_value >::load().

5.2.3 Member Function Documentation

5.2.3.1 template<typename _t_value, typename _i_value = LI> virtual size_t BICRS< _t_value, _i_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< _t_value >** (p. **??**).

References BICRS< _t_value, _i_value >::jumps, and SparseMatrix< _t_value, ULI >::nnz.

5.2.3.2 template<typename _t_value, typename _i_value = LI> virtual void BICRS< _t_value, _i_value >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References BICRS< _t_value, _i_value >::c_start, and BICRS< _t_value, _i_value >::r_start.

5.2.3.3 template<typename _t_value, typename _i_value = LI> virtual void BICRS< _t_value, _i_value >::load (std::vector< Triplet< _t_value > > & *input,* ULI *m,* ULI *n,* _t_value *zero*) [inline], [virtual]

This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.

See Also

load(row, col, val, m, n, nz)
SparseMatrix::load (p. **??**)

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References BICRS< _t_value, _i_value >::vals.

Referenced by BICRS< _t_value, _i_value >::BICRS().

5.2.3.4 template<typename _t_value, typename _i_value = LI> void BICRS< _t_value, _i_value >::load (_i_value * *row*, _i_value * *col*, _t_value * *val*, ULI *m*, ULI *n*, ULI *nz*, _t_value *zero*) [inline]

See Also

BICRS(row, col, val, m, n, nz) (p. **??**)

References BICRS< _t_value, _i_value >::c_end, BICRS< _t_value, _i_value >::c_inc, BICRS< _t_value, _i_value >::c_start, BICRS< _t_value, _i_value >::jumps, SparseMatrix< _t_value, ULI >::m(), SparseMatrix< _t_value, ULI >::n(), SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, BICRS< _t_value, _i_value >::ntt, BICRS< _t_value, _i_value >::r_end, BICRS< _t_value, _i_value >::r_inc, BICRS< _t_value, _i_value >::r_start, BICRS< _t_value, _i_value >::vals, and SparseMatrix< _t_value, ULI >::zero_element.

5.2.3.5 template<typename _t_value, typename _i_value = LI> virtual void BICRS< _t_value, _i_value >::zax (const _t_value *__restrict__ *x_p*, _t_value *__restrict__ *y_p*) [inline], [virtual]

Calculates y=Ax, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References BICRS< _t_value, _i_value >::c_inc, BICRS< _t_value, _i_value >::c_start, SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, BICRS< _t_value, _i_value >::ntt, BICRS< _t_value, _i_value >::r_inc, BICRS< _t_value, _i_value >::r_start, and BICRS< _t_value, _i_value >::vals.

5.2.3.6 template<typename _t_value, typename _i_value = LI> virtual void BICRS< _t_value, _i_value >::zax_fb (_t_value *__restrict__ *x_f*, _t_value *__restrict__ *y_f*) [inline], [virtual]

Calculates y=Ax, but does not allocate y itself.

Does a front-to-back **Hilbert** (p. **??**) traversal.

Parameters

x_f	The input vector should be initialised and of correct measurements.
y_f	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

References BICRS< _t_value, _i_value >::c_end, BICRS< _t_value, _i_value >::c_inc, BICRS< _t_value, _i_value >::c_start, BICRS< _t_value, _i_value >::jumps, SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, BICRS< _t_value, _i_value >::ntt, BICRS< _t_value, _i_value >::r_end, BICRS< _t_value, _i_value >::r_inc, BICRS< _t_value, _i_value >::r_start, and BICRS< _t_value, _i_value >::vals.

5.2.3.7 template<typename _t_value, typename _i_value = LI> virtual void BICRS< _t_value, _i_value >::zxa (const _t_value *__restrict__ *x_p*, _t_value *__restrict__ *y_p*) [inline], [virtual]

Calculates y=xA, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References BICRS< _t_value, _i_value >::c_inc, BICRS< _t_value, _i_value >::c_start, SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, BICRS< _t_value, _i_value >::ntt, BICRS< _t_value, _i_value >::r_inc, BICRS< _t_value, _i_value >::r_start, and BICRS< _t_value, _i_value >::vals.

5.2.4 Member Data Documentation

5.2.4.1 template<typename _t_value, typename _i_value = LI> ULI BICRS< _t_value, _i_value >::c_end [protected]

Stores the column end position.

Referenced by BICRS< _t_value, _i_value >::load(), and BICRS< _t_value, _i_value >::zax_fb().

5.2.4.2 template<typename _t_value, typename _i_value = LI> _i_value* BICRS< _t_value, _i_value >::c_inc [protected]

Stores the column jumps; size is exactly the number of nonzeros.

Referenced by BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), BICRS< _t_value, _i_value >::zxa(), and BICRS< _t_value, _i_value >::~BICRS().

5.2.4.3 template<typename _t_value, typename _i_value = LI> ULI BICRS< _t_value, _i_value >::c_start [protected]

Stores the column start position.

Referenced by BICRS< _t_value, _i_value >::getFirstIndexPair(), BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), and BICRS< _t_value, _i_value >::zxa().

5.2.4.4 template<typename _t_value, typename _i_value = LI> ULI BICRS< _t_value, _i_value >::jumps [protected]

Stores the number of row jumps.

Referenced by BICRS< _t_value, _i_value >::bytesUsed(), BICRS< _t_value, _i_value >::load(), and BICRS< _t_value, _i_value >::zax_fb().

5.2.4.5 template<typename _t_value, typename _i_value = LI> _i_value BICRS< _t_value, _i_value >::ntt [protected]

Caches n times two.

Referenced by BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), and BICRS< _t_value, _i_value >::zxa().

5.2.4.6 template<typename _t_value, typename _i_value = LI> ULI BICRS< _t_value, _i_value >::r_end [protected]

Stores the row end position.

Referenced by BICRS< _t_value, _i_value >::load(), and BICRS< _t_value, _i_value >::zax_fb().

5.2.4.7 template<typename _t_value, typename _i_value = LI> _i_value* BICRS< _t_value, _i_value >::r_inc [protected]

Stores the row jumps; size is *at maximum* the number of nonzeros.

Referenced by BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), BICRS< _t_value, _i_value >::zxa(), and BICRS< _t_value, _i_value >::~BICRS().

5.2.4.8 template<typename _t_value, typename _i_value = LI> ULI BICRS< _t_value, _i_value >::r_start [protected]

Stores the row start position.

Referenced by BICRS< _t_value, _i_value >::getFirstIndexPair(), BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), and BICRS< _t_value, _i_value >::zxa().

5.2.4.9 template<typename _t_value, typename _i_value = LI> _t_value* BICRS< _t_value, _i_value >::vals [protected]

Stores the values of the individual nonzeros.

Size is exactly the number of nonzeros.

Referenced by BICRS< _t_value, _i_value >::load(), BICRS< _t_value, _i_value >::zax(), BICRS< _t_value, _i_value >::zax_fb(), BICRS< _t_value, _i_value >::zxa(), and BICRS< _t_value, _i_value >::~BICRS().

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

- BICRS.hpp

5.3 BigInt Struct Reference

A 128-bit integer, with overloaded comparison operators.

`#include <BigInt.hpp>`

Public Member Functions

- **BigInt** ()

  Default constructor (sets fields to zero).
- **BigInt** (unsigned long int a, unsigned long int b)

  Direct constructor.
- void **operator=** (const **BigInt** &other)

  Assignment operator.
- **operator unsigned long int** ()
- **operator unsigned int** ()
- **operator unsigned short int** ()
- **operator unsigned char** ()

Public Attributes

- unsigned long int **high**

  Most significant bits.
- unsigned long int **low**

  Least significant bits.

5.3.1 Detailed Description

A 128-bit integer, with overloaded comparison operators.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 BigInt::BigInt ( ) [inline]

Default constructor (sets fields to zero).

5.3.2.2 BigInt::BigInt (unsigned long int *a*, unsigned long int *b*) [inline]

Direct constructor.

Parameters

a	Most significant bits.
b	Least significant bits.

5.3.3 Member Function Documentation

5.3.3.1 BigInt::operator unsigned char ( ) [inline]

Returns

unsigned char cast of this integer.

References high, and low.

5.3.3.2 BigInt::operator unsigned int ( ) [inline]

Returns

unsinged int cast of this integer.

References high, and low.

5.3.3.3 BigInt::operator unsigned long int ( ) [inline]

Returns

unsinged long int cast of this integer.

References high, and low.

5.3.3.4 BigInt::operator unsigned short int ( ) [inline]

Returns

unsigned short int cast of this integer.

References high, and low.

5.3.3.5 void BigInt::operator= (const BigInt & *other*) [inline]

Assignment operator.

Parameters

other	the integer this integer should equal.

References high, and low.

5.3.4 Member Data Documentation

5.3.4.1 unsigned long int BigInt::low

Least significant bits.

Referenced by operator unsigned char(), operator unsigned int(), operator unsigned long int(), operator unsigned short int(), and operator=().

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

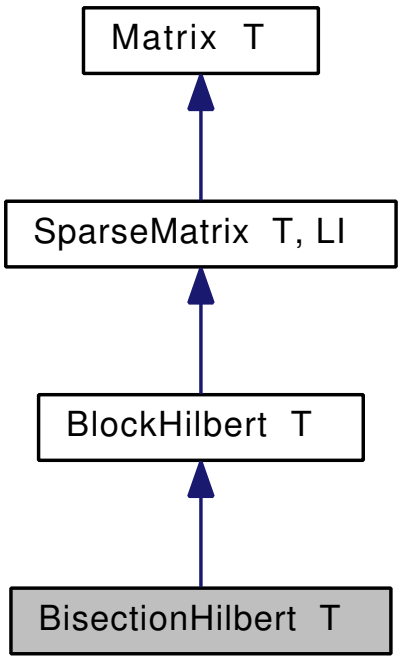
- BigInt.hpp

5.4 BisectionHilbert< T > Class Template Reference

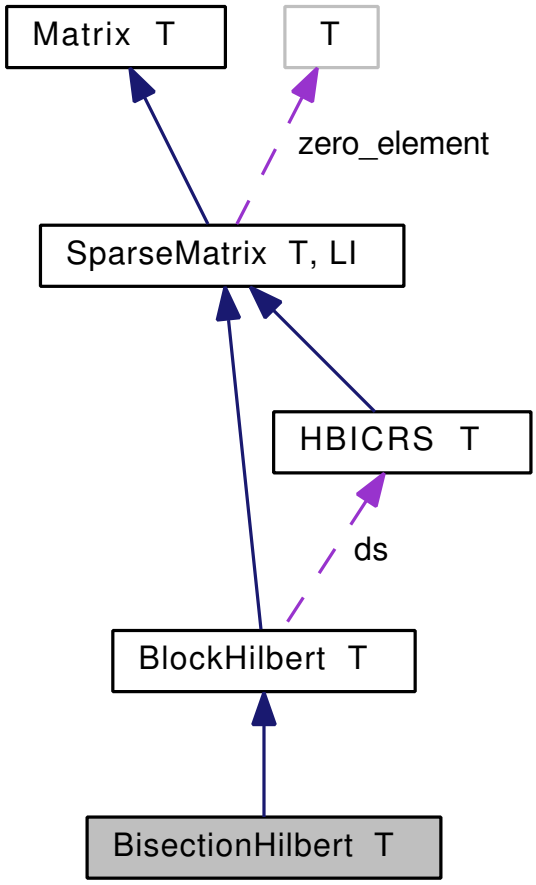
The Bisection **Hilbert** (p. **??**) triplet scheme.

`#include <BisectionHilbert.hpp>`

Inheritance diagram for BisectionHilbert< T >:



Collaboration diagram for BisectionHilbert< T >:



Public Member Functions

- virtual ~**BisectionHilbert** ()

  Base deconstructor.
- **BisectionHilbert** ()

  Base constructor.
- **BisectionHilbert** (std::string file, T zero=0)

  Base constructor.
- **BisectionHilbert** (std::vector< **Triplet**< T > > &input, unsigned long int **m**, unsigned long int **n**, T zero)

  Base constructor.

Additional Inherited Members

5.4.1 Detailed Description

template<typename T>class BisectionHilbert< T >

The Bisection **Hilbert** (p. **??**) triplet scheme.

In effect similar to (HierarchicalBICRS), but uses **Hilbert** (p. **??**) coordinates to determine the order of the blocks, and a bisection algorithm to construct the individual blocks. Wraps around the **BisectionHilbert** (p. **??**) class which already implements this scheme.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 template<typename T> virtual BisectionHilbert< T >::~BisectionHilbert ( ) [inline],[virtual]

Base deconstructor.

5.4.2.2 template<typename T> BisectionHilbert< T >::BisectionHilbert ( ) [inline]

Base constructor.

References BlockHilbert< T >::bisection.

5.4.2.3 template<typename T> BisectionHilbert< T >::BisectionHilbert (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

Parameters

file	The input filename.
zero	What is considered zero for this sparse matrix instance.

See Also

SparseMatrix::SparseMatrix(file, zero)

References BlockHilbert< T >::bisection, and SparseMatrix< T, LI >::loadFromFile().

5.4.2.4 template<typename T> BisectionHilbert< T >::BisectionHilbert (std::vector< Triplet< T > > & *input*, unsigned long int *m*, unsigned long int *n*, T *zero*) [inline]

Base constructor.

Warning: the zero parameter is currently NOT USED!

Parameters

input	Raw input of normal triplets.
m	Total number of rows.
n	Total number of columns.
zero	What elements is considered to-be zero.

References BlockHilbert< T >::bisection, and BlockHilbert< T >::load().

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

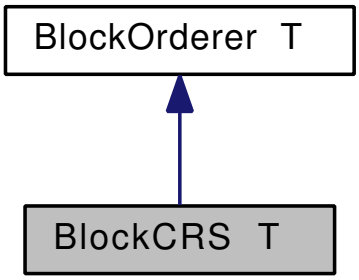
- BisectionHilbert.hpp

5.5 BlockCRS< T > Class Template Reference

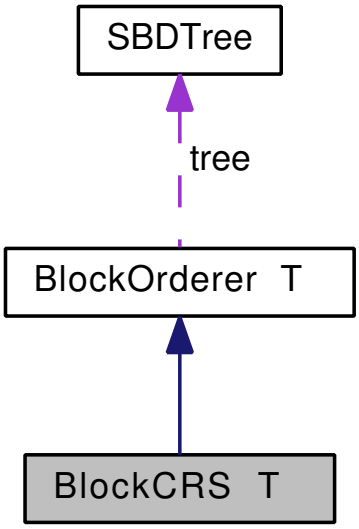
Codes the **CRS** (p. **??**) block order.

`#include <BlockCRS.hpp>`

Inheritance diagram for BlockCRS< T >:



Collaboration diagram for BlockCRS< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)

  Prefix operations during SBD tree traversal.
- virtual void **in_readout** (const unsigned long int index)

  Infix operations during SBD tree traversal.
- virtual void **post_readout** (const unsigned long int index)

  Postfix operations during SBD tree traversal.

Additional Inherited Members

5.5.1 Detailed Description

template<typename T>class BlockCRS< T >

Codes the **CRS** (p. **??**) block order.

5.5.2 Member Function Documentation

5.5.2.1 template<typename T > virtual void BlockCRS< T >::in_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Infix operations during SBD tree traversal.

Implements **BlockOrderer**< **T** > (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

5.5.2.2 template<typename T > virtual void BlockCRS< T >::post_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Postfix operations during SBD tree traversal.

Implements **BlockOrderer**< **T** > (p. **??**).

5.5.2.3 template<typename T > virtual void BlockCRS< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix operations during SBD tree traversal.

Implements **BlockOrderer**< **T** > (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, and BlockOrderer< T >::tree.

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

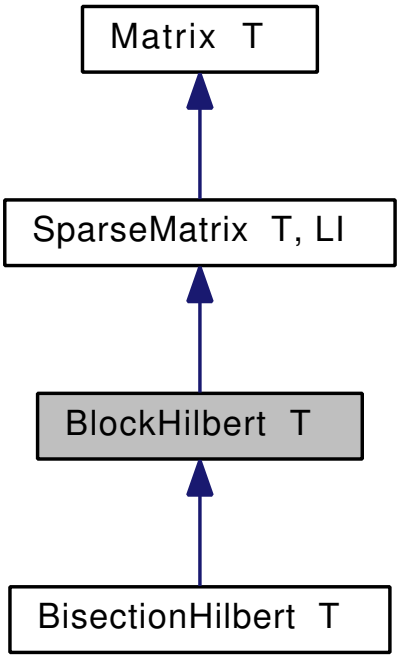
- BlockCRS.hpp

5.6 BlockHilbert< T > Class Template Reference

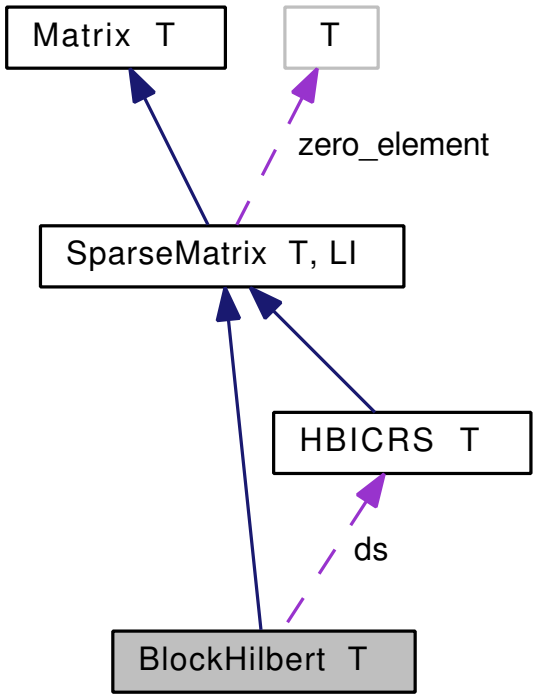
The Block **Hilbert** (p. **??**) triplet scheme.

`#include <BlockHilbert.hpp>`

Inheritance diagram for BlockHilbert< T >:



Collaboration diagram for BlockHilbert< T >:



Public Member Functions

- virtual ~**BlockHilbert** ()

  Base deconstructor.
- **BlockHilbert** ()

Base constructor.

- **BlockHilbert** (std::string file, T zero=0, char **bisect**=0)

  Base constructor.

- **BlockHilbert** (std::vector< **Triplet**< T > > &input, LI **m**, LI **n**, T zero)

  Base constructor.

- **BlockHilbert** (std::vector< **Triplet**< T > > &input, LI **m**, LI **n**, char **bisect**, T zero)

  Base constructor.

- virtual void **load** (std::vector< **Triplet**< T > > &input, const LI **m**, const LI **n**, const T zero)
- virtual void **getFirstIndexPair** (LI &row, LI &col)

  Returns the first nonzero index, per reference.

- virtual void **zxa** (const T *x, T *z)

  Calculates z=xA.

- virtual void **zax** (const T *x, T *z)

  Calculates z=Ax.

- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Protected Member Functions

- char **bisect** (std::vector< **HilbertTriplet**< std::vector< **Triplet**< T > > > > &build, std::vector< **Triplet**< T > > &input, const unsigned long int blocki, const unsigned long int blockj, const unsigned char depth)

  Recursive bisection, leaf case.

- void **bisect** (std::vector< **HilbertTriplet**< std::vector< **Triplet**< T > > > > &build)

  Automatically performs recursive bisection on the input nonzeroes to achieve sparse blocking.

- std::vector< **HilbertTriplet**
  < std::vector< **Triplet**< T > > > > **buildBisectionBlocks** (std::vector< **Triplet**< T > > &input)

  Builds block array by bisection.

- std::vector< **HilbertTriplet**
  < std::vector< **Triplet**< T > > > > **buildBlocks** (std::vector< **Triplet**< T > > &input)

  Builds block array by ordering them in fixed-size sparse submatrices.

- bool **cmp** (**HilbertTriplet**< std::vector< **Triplet**< T > > > &left, **HilbertTriplet**< std::vector< **Triplet**< T > > > &right)

  HilbertCoordinate comparison function.

- unsigned long int **find** (**HilbertTriplet**< std::vector< **Triplet**< T > > > &x, ULI &left, ULI &right, ULI &**minexp**)

  Binary search for finding a given triplet in a given range.

Protected Attributes

- char **bisection**

  Whether we use fixed block grid or a bisection-based grid.

- ULI **minexp**

  Minimum number of expansions.

- std::vector< **HilbertTriplet**
  < std::vector< **Triplet**< T > > > > **hilbert**

  *Vector storing the block indices and their **Hilbert** (p. **??**) coordinates.*

- **HBICRS**< T > * **ds**

  The actual data structure.

Static Protected Attributes

- static const ULI **BLOCK_M** = 2048

  Minimum number of rows in a block.
- static const ULI **BLOCK_N** = 2048

  Minimum number of columns in a block.
- static const ULI **MAX_BLOCKS** = 1024∗128

  Maximum number of blocks.
- static const ULI **MAX_DEPTH** = 64

  Maximum depth of bisection.
- static const signed char **DS** = 2

  Which data structure to use for each block.

Additional Inherited Members

5.6.1 Detailed Description

template<typename T>class BlockHilbert< T >

The Block **Hilbert** (p. **??**) triplet scheme.

In effect similar to (HierarchicalBICRS), but uses **Hilbert** (p. **??**) coordinates to determine the order of the blocks, and a bisection algorithm to construct the individual blocks.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 template<typename T> virtual BlockHilbert< T >::~BlockHilbert ( ) [inline], [virtual]

Base deconstructor.

References BlockHilbert< T >::ds.

5.6.2.2 template<typename T> BlockHilbert< T >::BlockHilbert ( ) [inline]

Base constructor.

References BlockHilbert< T >::bisection.

5.6.2.3 template<typename T> BlockHilbert< T >::BlockHilbert (std::string *file*, T *zero* = 0, char *bisect* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

Parameters

file	The input sparse matrix file.
zero	What element constitutes a matrix zero.
bisect	Whether bisection-based blocking should be used.

See Also

SparseMatrix::SparseMatrix(file, zero)

References BlockHilbert< T >::bisect(), BlockHilbert< T >::bisection, and SparseMatrix< T, LI >::loadFromFile().

5.6.2.4 template<typename T> BlockHilbert< T >::BlockHilbert (std::vector< Triplet< T > > & *input*, LI *m*, LI *n*, T *zero*) [inline]

Base constructor.

Warning: the zero parameter is currently NOT USED!

Parameters

input	Raw input of normal triplets.
m	Total number of rows.
n	Total number of columns.
zero	What elements is considered to-be zero.

References BlockHilbert< T >::bisection, and BlockHilbert< T >::load().

5.6.2.5 template<typename T> BlockHilbert< T >::BlockHilbert (std::vector< Triplet< T > > & *input*, LI *m*, LI *n*, char *bisect*, T *zero*) [inline]

Base constructor.

Warning: the zero parameter is currently NOT USED!

Parameters

input	Raw input of normal triplets.
m	Total number of rows.
n	Total number of columns.
bisect	Whether bisection-based blocking should be used.
zero	What elements is considered to-be zero.

References BlockHilbert< T >::bisect(), BlockHilbert< T >::bisection, and BlockHilbert< T >::load().

5.6.3 Member Function Documentation

5.6.3.1 template<typename T> char BlockHilbert< T >::bisect (std::vector< HilbertTriplet< std::vector< Triplet< T > > > > & *build*, std::vector< Triplet< T > > & *input*, const unsigned long int *blocki*, const unsigned long int *blockj*, const unsigned char *depth*) [inline], [protected]

Recursive bisection, leaf case.

Parameters

build	A series of sparse blocks with their **Hilbert** (p. **??**) coordinates.
input	The input series of nonzeroes.
blocki	The row blocking size.
blockj	The column blocking size.
depth	Current recursion depth.

Returns

Exit code: 0=failed split, 1=OK, 2=not split.

References BlockHilbert< T >::BLOCK_M, BlockHilbert< T >::BLOCK_N, BlockHilbert< T >::MAX_DEPTH, SparseMatrix< T, LI >::noc, and SparseMatrix< T, LI >::nor.

Referenced by BlockHilbert< T >::bisect(), BlockHilbert< T >::BlockHilbert(), and BlockHilbert< T >::buildBisectionBlocks().

5.6.3.2 template<typename T> void BlockHilbert< T >::bisect (std::vector< HilbertTriplet< std::vector< Triplet< T > > > > & *build*) [inline],[protected]

Automatically performs recursive bisection on the input nonzeroes to achieve sparse blocking.

Parameters

build	A series of sparse blocks with their **Hilbert** (p. **??**) coordinates.

References BlockHilbert< T >::bisect(), BlockHilbert< T >::MAX_BLOCKS, and BlockHilbert< T >::MAX_DEPTH.

5.6.3.3 template<typename T> std::vector< HilbertTriplet< std::vector< Triplet< T > > > > BlockHilbert< T >::buildBisectionBlocks (std::vector< Triplet< T > > & *input*) [inline], [protected]

Builds block array by bisection.

resulting in variably-sized submatrices.

Parameters

input	Original nonzero input.

Returns

A series of sparse blocks (with **Hilbert** (p. **??**) coordinates).

References BlockHilbert< T >::bisect(), and SparseMatrix< T, LI >::nnz.

Referenced by BlockHilbert< T >::load().

5.6.3.4 template<typename T> virtual size_t BlockHilbert< T >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References HBICRS< _t_value >::bytesUsed(), and BlockHilbert< T >::ds.

5.6.3.5 template<typename T> bool BlockHilbert< T >::cmp (HilbertTriplet< std::vector< Triplet< T > > > & *left,* HilbertTriplet< std::vector< Triplet< T > > > & *right*) [inline], [protected]

HilbertCoordinate comparison function.

References SparseMatrix< T, LI >::noc, and SparseMatrix< T, LI >::nor.

Referenced by BlockHilbert< T >::find().

5.6.3.6 template<typename T> unsigned long int BlockHilbert< T >::find (HilbertTriplet< std::vector< Triplet< T > > > & *x,* ULI & *left,* ULI & *right,* ULI & *minexp*) [inline], [protected]

Binary search for finding a given triplet in a given range.

Parameters

x	triplet to-be found.
left	Left bound of the range to search in.

right	Right bound of the range to search in.
minexp	The minimum amount of **Hilbert** (p. **??**) coordinate expansions necessary for successful comparison during binary search.

Returns

Index of the triplet searched.

References BlockHilbert< T >::cmp(), and BlockHilbert< T >::hilbert.

5.6.3.7 template<typename T> virtual void BlockHilbert< T >::getFirstIndexPair (LI & *row,* LI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, LI >** (p. **??**).

References BlockHilbert< T >::ds, and HBICRS< _t_value >::getFirstIndexPair().

5.6.3.8 template<typename T> virtual void BlockHilbert< T >::load (std::vector< Triplet< T > > & *input,* const LI *m,* const LI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, LI >** (p. **??**).

References BlockHilbert< T >::bisection, BlockHilbert< T >::buildBisectionBlocks(), BlockHilbert< T >::buildBlocks(), BlockHilbert< T >::DS, BlockHilbert< T >::ds, BlockHilbert< T >::hilbert, SparseMatrix< T, LI >::m(), SparseMatrix< T, LI >::n(), SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, and SparseMatrix< T, LI >::zero_element.

Referenced by BisectionHilbert< T >::BisectionHilbert(), and BlockHilbert< T >::BlockHilbert().

5.6.3.9 template<typename T> virtual void BlockHilbert< T >::zax (const T * *x,* T * *z*) [inline], [virtual]

Calculates z=Ax.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References BlockHilbert< T >::ds, and HBICRS< _t_value >::zax().

5.6.3.10 template<typename T> virtual void BlockHilbert< T >::zxa (const T * *x,* T * *z*) [inline], [virtual]

Calculates z=xA.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References BlockHilbert< T >::ds, and HBICRS< _t_value >::zxa().

5.6.4 Member Data Documentation

5.6.4.1 template<typename T> const signed char BlockHilbert< T >::DS = 2 [static], [protected]

Which data structure to use for each block.

Referenced by BlockHilbert< T >::load().

5.6.4.2 template<typename T> const ULI BlockHilbert< T >::MAX_DEPTH = 64 [static], [protected]

Maximum depth of bisection.

Referenced by BlockHilbert< T >::bisect().

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

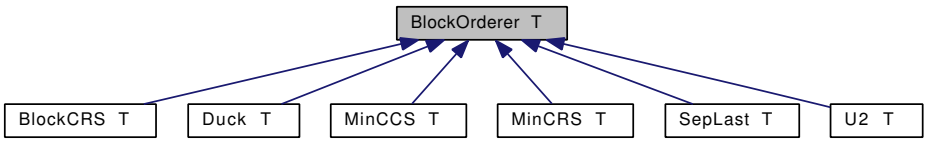
- BlockHilbert.hpp

5.7 BlockOrderer< T > Class Template Reference

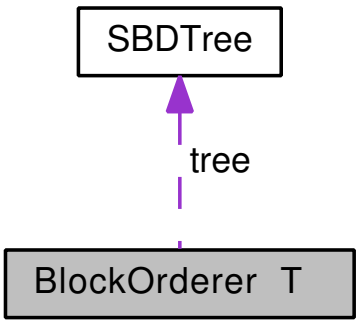
Induces a block order by fully traversing an **SBDTree** (p. **??**).

`#include <BlockOrderer.hpp>`

Inheritance diagram for BlockOrderer< T >:



Collaboration diagram for BlockOrderer< T >:



Public Member Functions

- **BlockOrderer** ()

  Base constructor.

- virtual ~**BlockOrderer** ()

  Base destructor.
- std::vector< std::vector < **Triplet**< T > > > **induce** (const std::vector< **Triplet**< T > > &input, std::vector< unsigned long int > &r_hierarchy, std::vector< unsigned long int > &c_hierarchy, std::vector< unsigned long int > &r_bounds, std::vector< unsigned long int > &c_bounds, std::vector< signed char > ∗**datatype**)

  Induces this ordering on a set of nonzeroes.

Protected Member Functions

- void **prefix** (const unsigned long int index)

  Prefix operation during SBD tree traversal.
- void **infix** (const unsigned long int index)

  Infix operation during SBD tree traversal.
- void **postfix** (const unsigned long int index)

  Postfix operation during SBD tree traversal.
- void **traverse** ()

  Traverses the SBD tree, makes call to prefix, infix, and postfix during traversal.
- void **pre_height** (const unsigned long int index)

  Prefix code for height-determining traversal.
- void **in_height** (const unsigned long int index)

  Infix code for height-determining traversal.
- void **post_height** (const unsigned long int index)

  Postfix code for height-determining traversal.
- virtual void **pre_readout** (const unsigned long int index)=0

  Prefix traversal code.
- virtual void **in_readout** (const unsigned long int index)=0

  Infix traversal code.
- virtual void **post_readout** (const unsigned long int index)=0

  Postfix traversal code.
- bool **left_horizontal** (const unsigned long int index, const **Triplet**< T > triplet)

  Helper function for determining place of a nonzero within a separator cross.
- bool **right_horizontal** (const unsigned long int index, const **Triplet**< T > triplet)

  Helper function for determining place of a nonzero within a separator cross.
- bool **upper_vertical** (const unsigned long int index, const **Triplet**< T > triplet)

  Helper function for determining place of a nonzero within a separator cross.
- bool **lower_vertical** (const unsigned long int index, const **Triplet**< T > triplet)

  Helper function for determining place of a nonzero within a separator cross.
- bool **middle** (const unsigned long int index, const **Triplet**< T > triplet)

  Helper function for determining place of a nonzero within a separator cross.

Protected Attributes

- **SBDTree** ∗ **tree**

  The SBD tree to order the blocks of.
- std::vector< char > **leftpass**

  Stores whether a node has been traversed from the left.
- std::vector< char > **rightpass**

  Stores whether a node has been traversed from the right.
- char **traverse_mode**

Sets the traversal mode.

- unsigned long int * **height**

  Stores the height of a SBD node.

- unsigned long int **cur_height**

  The current height at the traversal position.

- std::vector< **Triplet**< T > > * **items**

  Nonzero storage at each node.

- std::vector< std::vector < **Triplet**< T > > > * **output**

  Output structure after SBD readout (series of blocks in the correct order).

- std::vector< signed char > * **datatype**

  The data type of each block.

Static Protected Attributes

- static const char **TRAVERSE_HEIGHT** = 0

  Mode flag for height-determining traversal.

- static const char **READOUT** = 1

  Mode flag for SBD tree readouts.

5.7.1 Detailed Description

template<typename T>class BlockOrderer< T >

Induces a block order by fully traversing an **SBDTree** (p. **??**).

5.7.2 Member Function Documentation

5.7.2.1 template<typename T> void BlockOrderer< T >::in_height (const unsigned long int *index*) [protected]

Infix code for height-determining traversal.

5.7.2.2 template<typename T> virtual void BlockOrderer< T >::in_readout (const unsigned long int *index*) [protected], [pure virtual]

Infix traversal code.

Implemented in **MinCCS**< **T** > (p. **??**), **MinCRS**< **T** > (p. **??**), **BlockCRS**< **T** > (p. **??**), **U2**< **T** > (p. **??**), **Duck**< **T** > (p. **??**), and **SepLast**< **T** > (p. **??**).

Referenced by BlockOrderer< T >::infix().

5.7.2.3 template<typename T> void BlockOrderer< T >::infix (const unsigned long int *index*) [inline], [protected]

Infix operation during SBD tree traversal.

References BlockOrderer< T >::cur_height, BlockOrderer< T >::height, BlockOrderer< T >::in_readout(), BlockOrderer< T >::READOUT, BlockOrderer< T >::TRAVERSE_HEIGHT, and BlockOrderer< T >::traverse_mode.

Referenced by BlockOrderer< T >::traverse().

5.7.2.4 template<typename T> void BlockOrderer< T >::post_height (const unsigned long int *index*) [protected]

Postfix code for height-determining traversal.

5.7.2.5 template<typename T> virtual void BlockOrderer< T >::post_readout (const unsigned long int *index*) [protected], [pure virtual]

Postfix traversal code.

Implemented in **MinCCS< T >** (p. **??**), **MinCRS< T >** (p. **??**), **BlockCRS< T >** (p. **??**), **U2< T >** (p. **??**), **Duck< T >** (p. **??**), and **SepLast< T >** (p. **??**).

Referenced by BlockOrderer< T >::postfix().

5.7.2.6 template<typename T> void BlockOrderer< T >::postfix (const unsigned long int *index*) [inline], [protected]

Postfix operation during SBD tree traversal.

References BlockOrderer< T >::cur_height, BlockOrderer< T >::post_readout(), BlockOrderer< T >::READOUT, BlockOrderer< T >::TRAVERSE_HEIGHT, and BlockOrderer< T >::traverse_mode.

Referenced by BlockOrderer< T >::traverse().

5.7.2.7 template<typename T> void BlockOrderer< T >::pre_height (const unsigned long int *index*) [protected]

Prefix code for height-determining traversal.

5.7.2.8 template<typename T> virtual void BlockOrderer< T >::pre_readout (const unsigned long int *index*) [protected], [pure virtual]

Prefix traversal code.

Implemented in **BlockCRS< T >** (p. **??**), **MinCCS< T >** (p. **??**), **MinCRS< T >** (p. **??**), **U2< T >** (p. **??**), **Duck< T >** (p. **??**), and **SepLast< T >** (p. **??**).

Referenced by BlockOrderer< T >::prefix().

5.7.2.9 template<typename T> void BlockOrderer< T >::prefix (const unsigned long int *index*) [inline], [protected]

Prefix operation during SBD tree traversal.

References BlockOrderer< T >::cur_height, BlockOrderer< T >::pre_readout(), BlockOrderer< T >::READOUT, BlockOrderer< T >::TRAVERSE_HEIGHT, and BlockOrderer< T >::traverse_mode.

Referenced by BlockOrderer< T >::traverse().

5.7.2.10 template<typename T> void BlockOrderer< T >::traverse ( ) [inline], [protected]

Traverses the SBD tree, makes call to prefix, infix, and postfix during traversal.

References SBDTree::getRoot(), BlockOrderer< T >::infix(), SBDTree::isLeaf(), SBDTree::left(), BlockOrderer< T >::leftpass, BlockOrderer< T >::postfix(), BlockOrderer< T >::prefix(), SBDTree::right(), BlockOrderer< T >::rightpass, SBDTree::size(), BlockOrderer< T >::tree, and SBDTree::up().

Referenced by BlockOrderer< T >::induce().

5.7.3 Member Data Documentation

5.7.3.1 template<typename T> unsigned long int BlockOrderer< T >::cur_height [protected]

The current height at the traversal position.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::infix(), BlockOrderer< T >::postfix(), and BlockOrderer< T >::prefix().

5.7.3.2 template<typename T> std::vector< signed char >* BlockOrderer< T >::datatype [protected]

The data type of each block.

Referenced by Duck< T >::in_readout(), BlockCRS< T >::in_readout(), U2< T >::in_readout(), MinCCS< T >::in_readout(), MinCRS< T >::in_readout(), BlockOrderer< T >::induce(), SepLast< T >::post_readout(), Duck< T >::post_readout(), U2< T >::post_readout(), MinCCS< T >::post_readout(), MinCRS< T >::post_readout(), Duck< T >::pre_readout(), SepLast< T >::pre_readout(), MinCCS< T >::pre_readout(), MinCRS< T >::pre_readout(), U2< T >::pre_readout(), and BlockCRS< T >::pre_readout().

5.7.3.3 template<typename T> unsigned long int* BlockOrderer< T >::height [protected]

Stores the height of a SBD node.

Referenced by BlockOrderer< T >::induce(), and BlockOrderer< T >::infix().

5.7.3.4 template<typename T> std::vector< Triplet< T > >* BlockOrderer< T >::items [protected]

Nonzero storage at each node.

Referenced by Duck< T >::in_readout(), BlockCRS< T >::in_readout(), U2< T >::in_readout(), MinCCS< T >::in_readout(), MinCRS< T >::in_readout(), BlockOrderer< T >::induce(), SepLast< T >::post_readout(), Duck< T >::post_readout(), U2< T >::post_readout(), MinCCS< T >::post_readout(), MinCRS< T >::post_readout(), Duck< T >::pre_readout(), SepLast< T >::pre_readout(), MinCCS< T >::pre_readout(), MinCRS< T >::pre_readout(), U2< T >::pre_readout(), and BlockCRS< T >::pre_readout().

5.7.3.5 template<typename T> std::vector< char > BlockOrderer< T >::leftpass [protected]

Stores whether a node has been traversed from the left.

Referenced by BlockOrderer< T >::traverse().

5.7.3.6 template<typename T> std::vector< std::vector< Triplet< T > > >* BlockOrderer< T >::output [protected]

Output structure after SBD readout (series of blocks in the correct order).

Referenced by Duck< T >::in_readout(), U2< T >::in_readout(), BlockCRS< T >::in_readout(), MinCCS< T >::in_readout(), MinCRS< T >::in_readout(), BlockOrderer< T >::induce(), SepLast< T >::post_readout(), Duck< T >::post_readout(), U2< T >::post_readout(), MinCCS< T >::post_readout(), MinCRS< T >::post_readout(), SepLast< T >::pre_readout(), Duck< T >::pre_readout(), MinCCS< T >::pre_readout(), BlockCRS< T >::pre_readout(), MinCRS< T >::pre_readout(), U2< T >::pre_readout(), and BlockOrderer< T >::~BlockOrderer().

5.7.3.7 template<typename T> const char BlockOrderer< T >::READOUT = 1 [static], [protected]

Mode flag for SBD tree readouts.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::infix(), BlockOrderer< T >::postfix(), and BlockOrderer< T >::prefix().

5.7.3.8 template<typename T> std::vector< char > BlockOrderer< T >::rightpass [protected]

Stores whether a node has been traversed from the right.

Referenced by BlockOrderer< T >::traverse().

5.7.3.9 template<typename T> const char BlockOrderer< T >::TRAVERSE_HEIGHT = 0 [static], [protected]

Mode flag for height-determining traversal.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::infix(), BlockOrderer< T >::postfix(), and BlockOrderer< T >::prefix().

5.7.3.10 template<typename T> char BlockOrderer< T >::traverse_mode [protected]

Sets the traversal mode.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::infix(), BlockOrderer< T >::postfix(), and BlockOrderer< T >::prefix().

5.7.3.11 template<typename T> SBDTree* BlockOrderer< T >::tree [protected]

The SBD tree to order the blocks of.

Referenced by Duck< T >::in_readout(), BlockCRS< T >::in_readout(), U2< T >::in_readout(), MinCRS< T >::in_readout(), MinCCS< T >::in_readout(), BlockOrderer< T >::induce(), BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), SepLast< T >::post_readout(), Duck< T >::post_readout(), U2< T >::post_readout(), MinCRS< T >::post_readout(), MinCCS< T >::post_readout(), Duck< T >::pre_readout(), SepLast< T >::pre_readout(), MinCCS< T >::pre_readout(), U2< T >::pre_readout(), BlockCRS< T >::pre_readout(), MinCRS< T >::pre_readout(), BlockOrderer< T >::right_horizontal(), BlockOrderer< T >::traverse(), and BlockOrderer< T >::upper_vertical().

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

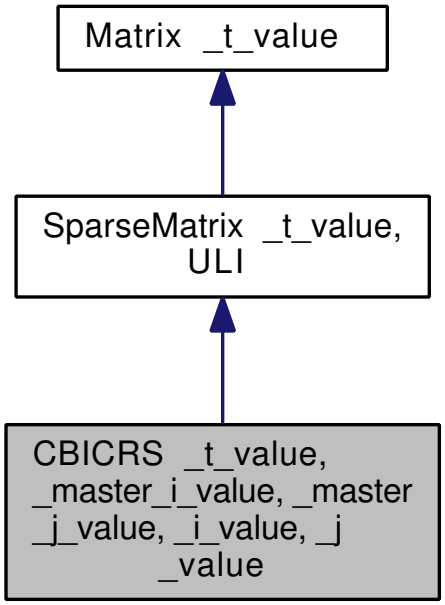
- BlockOrderer.hpp

5.8 CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value > Class Template Reference

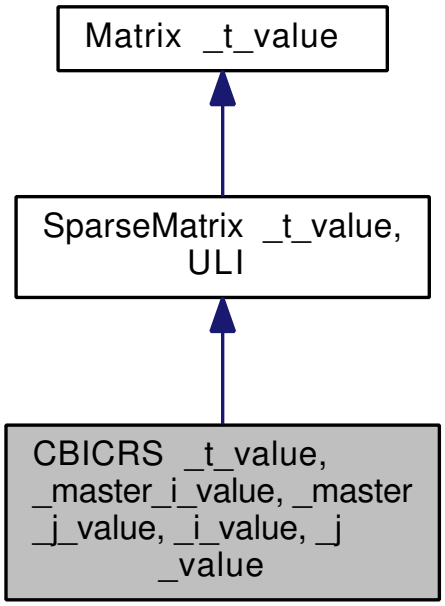
Compressed Bi-directional Incremental Compressed Row Storage (**BICRS** (p. **??**)) scheme.

`#include <CBICRS.hpp>`

Inheritance diagram for CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >:



Collaboration diagram for CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >:



Public Member Functions

- virtual ~**CBICRS** ()

  Base deconstructor.
- **CBICRS** ()

  Base constructor.
- **CBICRS** (std::string file, _t_value zero=0)

  Base constructor.
- **CBICRS** (ULI *row, ULI *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)

Base constructor.

- **CBICRS** (std::vector< **Triplet**< _t_value > > &input, ULI **m**, ULI **n**, _t_value zero=0)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< _t_value > > &input, ULI **m**, ULI **n**, _t_value zero)

  This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.
- void **load** (ULI *row, ULI *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=xA, but does not allocate y itself.
- virtual void **zax** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=Ax, but does not allocate y itself.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Static Public Member Functions

- static unsigned long int **getMemoryUsage** (ULI *row, ULI *col, const ULI nz, const ULI **m**, const ULI **n**)

  Calculates and returns the number of bytes used when employing this data structure.
- static unsigned long int **getMemoryUsage** (std::vector< **Triplet**< _t_value > > &input, const ULI **m**, const ULI **n**)

  Calculates and returns the number of bytes used when employing this data structure.

Static Protected Member Functions

- static void **getNumberOfOverflows** (const ULI **nnz**, ULI *const row, ULI *const col, const ULI **ntt**, ULI &row_overflows, ULI &col_overflows, ULI &sim_overflows, ULI &jumps)

  Calculates the number of overflows given a triplet-form input.
- static unsigned long int **memoryUsage** (const ULI **nnz**, const ULI jumps, const ULI row_o, const ULI col_o, const ULI sim_o)

  Estimates the number of bytes required by this data structure.

Protected Attributes

- _master_i_value * **r_start**

  Stores the row chunk start increments; size is the number of nonzeros plus one.
- _master_i_value * **c_start**

  Stores the column chunk start increments; size is the number of nonzeros plus one.
- _i_value * **r_inc**

  Stores the row jumps; size is the number of nonzeros plus 2.
- _i_value * **c_inc**

  Stores the column jumps; size is exactly the number of nonzeros.
- unsigned char * **mask1**

  Bitmask used for switching between c_start and c_ind.
- unsigned char * **mask2**

  Bitmask used for switching between r_start and r_ind.
- _t_value * **vals**

  Stores the values of the individual nonzeros.
- size_t **bytes**

  Stores the number of bytes used for storage.
- _master_j_value **ntt**

  Caches n times two.

Additional Inherited Members

5.8.1 Detailed Description

template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI>class CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >

Compressed Bi-directional Incremental Compressed Row Storage (**BICRS** (p. **??**)) scheme.

Compression is done by using less large data types for storing increments; e.g., 8-bit signed chars instead of 64-bit signed long ints. The exact data type used are input parameters.

Refer to **CBICRS_factory** (p. **??**) for an auto-tuned selection.

Parameters

_t_value	The type of the nonzeros in the matrix.

Warning: this class uses assertions! For optimal performance, define the NDEBUG flag (e.g., pass -DNDEBUG as a compiler flag).

5.8.2 Constructor & Destructor Documentation

5.8.2.1 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS ( ) [inline], [virtual]

Base deconstructor.

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start, and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals.

5.8.2.2 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS ( ) [inline]

Base constructor.

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start, and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals.

5.8.2.3 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS (std::string *file*, _t_value *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< _t_value, ULI >::loadFromFile().

5.8.2.4 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS (ULI * *row,* ULI * *col,* _t_value * *val,* ULI *m,* ULI *n,* ULI *nz,* _t_value *zero*) [inline]

Base constructor.

Stores triplets in exactly the same order as passed to this constructor.

Parameters

row	The row numbers of the individual nonzeros.
col	The column numbers of the individual nonzeros.
val	The values of the nonzeros.
m	Number of matrix rows.
n	Number of matrix columns.
nz	Number of nonzeros.
zero	Which value is to be regarded zero here.

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load().

5.8.2.5 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS (std::vector< Triplet< _t_value > > & *input,* ULI *m,* ULI *n,* _t_value *zero* = 0) [inline]

Base constructor.

See Also

SparseMatrix::SparseMatrix(input, m, n, zero)

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load().

5.8.3 Member Function Documentation

5.8.3.1 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual size_t CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< _t_value >** (p. **??**).

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::bytes.

5.8.3.2 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start.

5.8.3.3 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> static unsigned long int CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage (ULI * *row,* ULI * *col,* const ULI *nz,* const ULI *m,* const ULI *n*) [inline], [static]

Calculates and returns the number of bytes used when employing this data structure.

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getNumberOfOverflows(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::memoryUsage().

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage(), CBICRS_factory< _t_value >::investigate(), and CBICRS_factory< _t_value >::investigateCCS().

5.8.3.4 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> static unsigned long int CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage (std::vector< Triplet< _t_value > > & *input,* const ULI *m,* const ULI *n*) [inline], [static]

Calculates and returns the number of bytes used when employing this data structure.

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage().

5.8.3.5 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> static void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getNumberOfOverflows (const ULI *nnz,* ULI *const *row,* ULI *const *col,* const ULI *ntt,* ULI & *row_overflows,* ULI & *col_overflows,* ULI & *sim_overflows,* ULI & *jumps*) [inline], [static], [protected]

Calculates the number of overflows given a triplet-form input.

Parameters

nnz	The number of nonzeroes in the COO input.
row	The COO row indices array.
col	The COO column indices array.
ntt	Should equal the number of columns times two (n times two)
row_overflows	Where to store the number of row overflows.
col_overflows	Where to store the number of column overflows.
sim_overflows	Where to store the number of simultaneous row and column overflows.
jumps	Where to store the number of row jumps;

References SparseMatrix< _t_value, ULI >::nnz.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load().

5.8.3.6 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load (std::vector< Triplet< _t_value > > & *input,* ULI *m,* ULI *n,* _t_value *zero*) [inline], [virtual]

This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.

See Also

load(row, col, val, m, n, nz)
SparseMatrix::load (p. **??**)

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS().

5.8.3.7 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load (ULI * *row,* ULI * *col,* _t_value * *val,* ULI *m,* ULI *n,* ULI *nz,* _t_value *zero*) [inline]

See Also

CBICRS(row, col, val, m, n, nz) (p. **??**)

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::bytes, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getNumberOfOverflows(), SparseMatrix< _t_value, ULI >::m(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::memoryUsage(), SparseMatrix< _t_value, ULI >::n(), SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::ntt, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals, and SparseMatrix< _t_value, ULI >::zero_element.

5.8.3.8 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> static unsigned long int CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::memoryUsage (const ULI *nnz,* const ULI *jumps,* const ULI *row_o,* const ULI *col_o,* const ULI *sim_o*) [inline], [static], [protected]

Estimates the number of bytes required by this data structure.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getMemoryUsage(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load().

5.8.3.9 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax (const _t_value *__restrict__ *x_p,* _t_value *__restrict__ *y_p*) [inline], [virtual]

Calculates y=Ax, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2, SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::ntt, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc, CBICRS< _t_value, _master_i_value,

_master_j_value, _i_value, _j_value >::r_start, and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals.

5.8.3.10 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> virtual void CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p) [inline],[virtual]

Calculates y=xA, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix< _t_value, ULI >** (p. **??**).

References CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2, SparseMatrix< _t_value, ULI >::nnz, SparseMatrix< _t_value, ULI >::noc, SparseMatrix< _t_value, ULI >::nor, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::ntt, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc, CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start, and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals.

5.8.4 Member Data Documentation

5.8.4.1 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> size_t CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::bytes [protected]

Stores the number of bytes used for storage.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::bytesUsed(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load().

5.8.4.2 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _i_value* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_inc [protected]

Stores the column jumps; size is exactly the number of nonzeros.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.3 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _master_i_value* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::c_start [protected]

Stores the column chunk start increments; size is the number of nonzeros plus one.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getFirstIndexPair(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master-

_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.4 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> unsigned char* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask1 [protected]

Bitmask used for switching between c_start and c_ind.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.5 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> unsigned char* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::mask2 [protected]

Bitmask used for switching between r_start and r_ind.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.6 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _master_j_value CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::ntt [protected]

Caches n times two.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa().

5.8.4.7 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _i_value* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_inc [protected]

Stores the row jumps; size is the number of nonzeros plus 2.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.8 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _master_i_value* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::r_start [protected]

Stores the row chunk start increments; size is the number of nonzeros plus one.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::getFirstIndexPair(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

5.8.4.9 template<typename _t_value, typename _master_i_value = signed long int, typename _master_j_value = signed long int, typename _i_value = LI, typename _j_value = LI> _t_value* CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::vals [protected]

Stores the values of the individual nonzeros.

Size is exactly the number of nonzeros.

Referenced by CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::CBICRS(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::load(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zax(), CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::zxa(), and CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >::~CBICRS().

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

- CBICRS.hpp

5.9 CBICRS_factory< _t_value > Class Template Reference

Factory for the Compressed Bi-directional Incremental Compressed Row Storage scheme.

`#include <CBICRS.hpp>`

Static Public Member Functions

- static **Matrix**< _t_value > * **getCBICRS** (std::string file, _t_value zero=0)

  *Factory function for row-based compressed **BICRS** (p. **??**) functions, file-based.*
- static **Matrix**< _t_value > * **getCBICCS** (std::string file, _t_value zero=0)

  *Factory function for column-based compressed **BICRS** (p. **??**) functions, file-based.*
- static **Matrix**< _t_value > * **getCBICRS** (std::vector< **Triplet**< _t_value > > &triplets, unsigned long int m, unsigned long int n, _t_value zero=0)

  *Factory function for row-based compressed **BICRS** (p. **??**) functions, Triplet-based.*
- static **Matrix**< _t_value > * **getCBICCS** (std::vector< **Triplet**< _t_value > > &triplets, unsigned long int m, unsigned long int n, _t_value zero=0)

  *Factory function for row-based compressed **BICRS** (p. **??**) functions, Triplet-based.*

Static Protected Member Functions

- template<typename _master_i_value , typename _master_j_value , typename _i_value , typename _j_value >
  static void **investigateCCS** (const std::string tn, std::vector< **Triplet**< _t_value > > input, unsigned long int m, unsigned long int n, unsigned long int &usage, unsigned long int &zle_usage)

  Used for auto-tunes the index type.
- template<typename _master_i_value , typename _master_j_value , typename _i_value , typename _j_value >
  static void **investigate** (const std::string tn, std::vector< **Triplet**< _t_value > > triplets, ULI m, ULI n, unsigned long int &usage, unsigned long int &zle_usage)

  Used for auto-tuning of the index type.

5.9.1 Detailed Description

template<typename _t_value>class CBICRS_factory< _t_value >

Factory for the Compressed Bi-directional Incremental Compressed Row Storage scheme.

Auto-tunes index type (signed char, short int, int or long int). May revert back to a plain **BICRS** (p. **??**) implementation.

Warning: assertions may be used! For optimal performance, define the NDEBUG flag (pass -DNDEBUG as a compiler flag).

5.9.2 Member Function Documentation

5.9.2.1 template<typename _t_value > static Matrix< _t_value >* CBICRS_factory< _t_value >::getCBICCS (std::string *file*, _t_value *zero* = 0) [inline],[static]

Factory function for column-based compressed **BICRS** (p. **??**) functions, file-based.

Parameters

file	Input file path.
zero	What is considered zero for the resulting sparse matrix.

Returns

A compressed sparse matrix representation.

References FileToVT::parse().

5.9.2.2 template<typename _t_value > static Matrix< _t_value >* CBICRS_factory< _t_value >::getCBICCS (std::vector< Triplet< _t_value > > & *triplets*, unsigned long int *m*, unsigned long int *n*, _t_value *zero* = 0) [inline],[static]

Factory function for row-based compressed **BICRS** (p. **??**) functions, Triplet-based.

Parameters

triplets	Triplet-based input matrix.
m	The number of rows in the input matrix.
n	The number of columns in the input matrix.
zero	What is considered zero for the resulting sparse matrix.

Returns

A compressed sparse matrix representation.

5.9.2.3 template<typename _t_value > static Matrix< _t_value >* CBICRS_factory< _t_value >::getCBICRS (std::string *file*, _t_value *zero* = 0) [inline],[static]

Factory function for row-based compressed **BICRS** (p. **??**) functions, file-based.

Parameters

file	Input file path.
zero	What is considered zero for the resulting sparse matrix.

Returns

A compressed sparse matrix representation.

References FileToVT::parse().

Referenced by Hilbert< T >::getDataStructure().

5.9.2.4 template<typename _t_value > static Matrix< _t_value >* CBICRS_factory< _t_value >::getCBICRS (std::vector< Triplet< _t_value > > & *triplets,* unsigned long int *m,* unsigned long int *n,* _t_value *zero* = 0) [inline],[static]

Factory function for row-based compressed **BICRS** (p. **??**) functions, Triplet-based.

Parameters

triplets	Triplet-based input matrix.
m	The number of rows in the input matrix.
n	The number of columns in the input matrix.
zero	What is considered zero for the resulting sparse matrix.

Returns

A compressed sparse matrix representation.

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

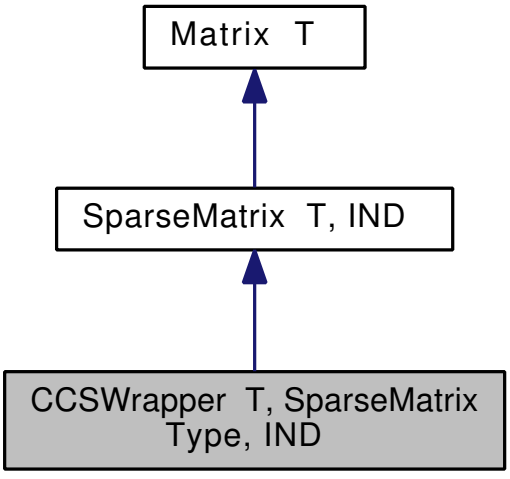
- CBICRS.hpp

5.10 CCSWrapper< T, SparseMatrixType, IND > Class Template Reference

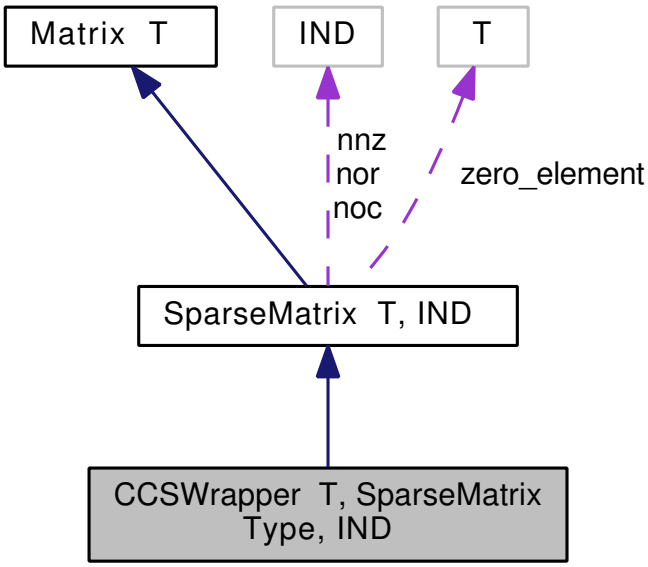
Automatically transforms a row-major scheme into an column-major scheme.

`#include <CCSWrapper.hpp>`

Inheritance diagram for CCSWrapper< T, SparseMatrixType, IND >:



Collaboration diagram for CCSWrapper< T, SparseMatrixType, IND >:



Public Member Functions

- **CCSWrapper** ()
  Default constructor (initialises with invalid data).
- **CCSWrapper** (std::string file, T zero=0)
  Base file-based constructor.
- **CCSWrapper** (const IND **nnz**, const IND **nor**, const IND **noc**, T zero)
  Base empty matrix constructor (sets nnz, rows, columns only).
- **CCSWrapper** (std::vector< **Triplet**< T > > &input, IND **m**, IND **n**, T zero)
  Base Triplet-based constructor.
- virtual ~**CCSWrapper** ()
  Base destructor.
- virtual void **load** (std::vector< **Triplet**< T > > &input, IND **m**, IND **n**, T zero)
  Triplet-based loader; first transposes, then calls nested constructor.
- virtual void **loadFromFile** (const std::string file, const T zero=0)
  File-based loader; reads file, then passes to Triplet-based loader.
- virtual ULI **m** ()
  Returns the number of matrix rows (taking into account transposition).
- virtual ULI **n** ()
  Returns the number of matrix columns (taking into account transposition).
- virtual ULI **nzs** ()
  Returns the number of nonzeroes.
- virtual void **getFirstIndexPair** (IND &row, IND &col)
- virtual T * **mv** (const T *x)
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)
- virtual size_t **bytesUsed** ()

Protected Member Functions

- void **transposeVector** (std::vector< **Triplet**< T > > &input)
  *Helper function that transposes an input matrix in **Triplet*** (p. **??**) *format, in-place.*

Protected Attributes

- SparseMatrixType * **ds**

  Pointer to the underlying data structure.

Additional Inherited Members

5.10.1 Detailed Description

template<typename T, typename SparseMatrixType, typename IND>class CCSWrapper< T, SparseMatrixType, IND >

Automatically transforms a row-major scheme into an column-major scheme.

Can wrap around any **SparseMatrix** (p. **??**) type, and automatically switches input row indices with input column indices, and switches the z=Ax operation with z=xA, and vice versa.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 template<typename T, typename SparseMatrixType, typename IND> CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper ( ) [inline]

Default constructor (initialises with invalid data).

5.10.2.2 template<typename T, typename SparseMatrixType, typename IND> CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper (std::string *file,* T *zero* = 0) [inline]

Base file-based constructor.

References CCSWrapper< T, SparseMatrixType, IND >::loadFromFile().

5.10.2.3 template<typename T, typename SparseMatrixType, typename IND> CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper (const IND *nnz,* const IND *nor,* const IND *noc,* T *zero*) [inline]

Base empty matrix constructor (sets nnz, rows, columns only).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.2.4 template<typename T, typename SparseMatrixType, typename IND> CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper (std::vector< Triplet< T > > & *input,* IND *m,* IND *n,* T *zero*) [inline]

Base Triplet-based constructor.

References CCSWrapper< T, SparseMatrixType, IND >::load().

5.10.2.5 template<typename T, typename SparseMatrixType, typename IND> virtual CCSWrapper< T, SparseMatrixType, IND >::~CCSWrapper ( ) [inline], [virtual]

Base destructor.

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.3 Member Function Documentation

5.10.3.1 template<typename T, typename SparseMatrixType, typename IND> virtual size_t CCSWrapper< T, SparseMatrixType, IND >::bytesUsed ( ) [inline], [virtual]

See Also

Matrix::bytesUsed (p. **??**)

Implements **Matrix< T >** (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.3.2 template<typename T, typename SparseMatrixType, typename IND> virtual void CCSWrapper< T, SparseMatrixType, IND >::getFirstIndexPair (IND & *row,* IND & *col*) [inline], [virtual]

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix< T, IND >** (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.3.3 template<typename T, typename SparseMatrixType, typename IND> virtual void CCSWrapper< T, SparseMatrixType, IND >::load (std::vector< Triplet< T > > & *input,* IND *m,* IND *n,* T *zero*) [inline], [virtual]

Triplet-based loader; first transposes, then calls nested constructor.

Implements **SparseMatrix< T, IND >** (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds, and CCSWrapper< T, SparseMatrixType, IND >::transposeVector().

Referenced by CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper(), and CCSWrapper< T, SparseMatrixType, IND >::loadFromFile().

5.10.3.4 template<typename T, typename SparseMatrixType, typename IND> virtual void CCSWrapper< T, SparseMatrixType, IND >::loadFromFile (const std::string *file,* const T *zero* = 0) [inline], [virtual]

File-based loader; reads file, then passes to Triplet-based loader.

References CCSWrapper< T, SparseMatrixType, IND >::load(), CCSWrapper< T, SparseMatrixType, IND >::m(), CCSWrapper< T, SparseMatrixType, IND >::n(), and FileToVT::parse().

Referenced by CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper().

5.10.3.5 template<typename T, typename SparseMatrixType, typename IND> virtual ULI CCSWrapper< T, SparseMatrixType, IND >::m ( ) [inline], [virtual]

Returns the number of matrix rows (taking into account transposition).

Reimplemented from **SparseMatrix< T, IND >** (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

Referenced by CCSWrapper< T, SparseMatrixType, IND >::loadFromFile(), and CCSWrapper< T, SparseMatrixType, IND >::mv().

5.10.3.6 template<typename T, typename SparseMatrixType, typename IND> virtual T* CCSWrapper< T, SparseMatrixType, IND >::mv (const T * *x*) [inline], [virtual]

See Also

Matrix::mv (p. **??**)

Reimplemented from **SparseMatrix**< **T, IND** > (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds, and CCSWrapper< T, SparseMatrixType, IND >::m().

5.10.3.7 template<typename T, typename SparseMatrixType, typename IND> virtual ULI CCSWrapper< T, SparseMatrixType, IND >::n ( ) [inline], [virtual]

Returns the number of matrix columns (taking into account transposition).

Reimplemented from **SparseMatrix**< **T, IND** > (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

Referenced by CCSWrapper< T, SparseMatrixType, IND >::loadFromFile().

5.10.3.8 template<typename T, typename SparseMatrixType, typename IND> virtual ULI CCSWrapper< T, SparseMatrixType, IND >::nzs ( ) [inline], [virtual]

Returns the number of nonzeroes.

Reimplemented from **SparseMatrix**< **T, IND** > (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.3.9 template<typename T, typename SparseMatrixType, typename IND> void CCSWrapper< T, SparseMatrixType, IND >::transposeVector (std::vector< Triplet< T > > & *input*) [inline], [protected]

Helper function that transposes an input matrix in **Triplet** (p. **??**) format, in-place.

Parameters

input	The input matrix.

Referenced by CCSWrapper< T, SparseMatrixType, IND >::load().

5.10.3.10 template<typename T, typename SparseMatrixType, typename IND> virtual void CCSWrapper< T, SparseMatrixType, IND >::zax (const T *__restrict__ *x*, T *__restrict__ *z*) [inline], [virtual]

See Also

Matrix::zax (p. **??**) (takes into account transposition)

Implements **SparseMatrix**< **T, IND** > (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.3.11 template<typename T, typename SparseMatrixType, typename IND> virtual void CCSWrapper< T, SparseMatrixType, IND >::zxa (const T *__restrict__ *x*, T *__restrict__ *z*) [inline], [virtual]

See Also

Matrix::zxa (p. **??**) (takes into account transposition)

Implements **SparseMatrix**< **T, IND** > (p. **??**).

References CCSWrapper< T, SparseMatrixType, IND >::ds.

5.10.4 Member Data Documentation

5.10.4.1 template<typename T, typename SparseMatrixType, typename IND> SparseMatrixType* CCSWrapper< T, SparseMatrixType, IND >::ds [protected]

Pointer to the underlying data structure.

Referenced by CCSWrapper< T, SparseMatrixType, IND >::bytesUsed(), CCSWrapper< T, SparseMatrixType, IND >::CCSWrapper(), CCSWrapper< T, SparseMatrixType, IND >::getFirstIndexPair(), CCSWrapper< T, SparseMatrixType, IND >::load(), CCSWrapper< T, SparseMatrixType, IND >::m(), CCSWrapper< T, SparseMatrixType, IND >::mv(), CCSWrapper< T, SparseMatrixType, IND >::n(), CCSWrapper< T, SparseMatrixType, IND >::nzs(), CCSWrapper< T, SparseMatrixType, IND >::zax(), CCSWrapper< T, SparseMatrixType, IND >::zxa(), and CCSWrapper< T, SparseMatrixType, IND >::~CCSWrapper().

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

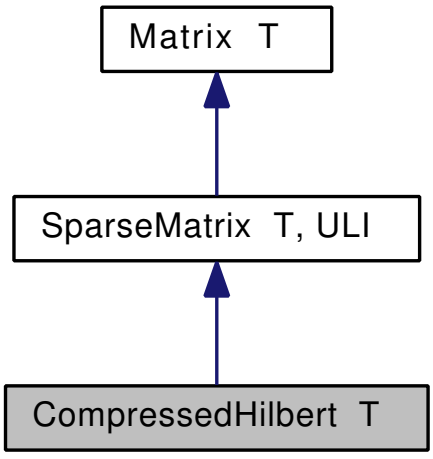
- CCSWrapper.hpp

5.11 CompressedHilbert< T > Class Template Reference

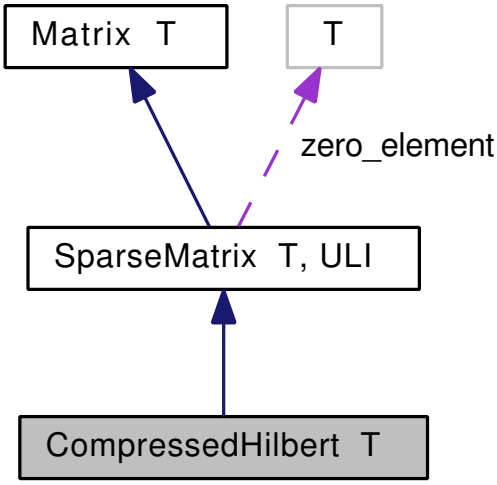
A **Hilbert** (p. **??**) scheme backed by a specialised storage format.

`#include <CompressedHilbert.hpp>`

Inheritance diagram for CompressedHilbert< T >:



Collaboration diagram for CompressedHilbert< T >:



Public Member Functions

- virtual ~**CompressedHilbert** ()

  Base deconstructor.
- **CompressedHilbert** ()

  Base constructor.
- **CompressedHilbert** (std::string file, T zero=0)

  Base constructor initialising from file.
- **CompressedHilbert** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero=0)

  Base constructor initialising from direct input.
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *x, T *z)

  Calculates z=xA.
- virtual void **zax** (const T *x, T *z)

  Calculates z=Ax.
- virtual size_t **bytesUsed** ()

Protected Attributes

- T * **values**

  Array of nonzero values.
- **HilbertArrayInterface**< T > * **indices**

  Pointer to indices in Hilbert-coordinate form.

Additional Inherited Members

5.11.1 Detailed Description

template<typename T>class CompressedHilbert< T >

A **Hilbert** (p. **??**) scheme backed by a specialised storage format.

This implementation does not use blocking, as that would require a two-fold datastructure (COO or **BICRS** (p. **??**) on the lower end). The block size would then best be of the size of a char (256x256) or of a short int (65536x65536).

5.11.2 Constructor & Destructor Documentation

5.11.2.1 template<typename T > virtual CompressedHilbert< T >::~CompressedHilbert ( ) [inline], [virtual]

Base deconstructor.

References CompressedHilbert< T >::indices, and CompressedHilbert< T >::values.

5.11.2.2 template<typename T > CompressedHilbert< T >::CompressedHilbert ( ) [inline]

Base constructor.

References CompressedHilbert< T >::indices, and CompressedHilbert< T >::values.

5.11.2.3 template<typename T > CompressedHilbert< T >::CompressedHilbert (std::string *file,* T *zero* = 0) [inline]

Base constructor initialising from file.

References SparseMatrix< T, ULI >::loadFromFile().

5.11.2.4 template<typename T > CompressedHilbert< T >::CompressedHilbert (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero* = 0) [inline]

Base constructor initialising from direct input.

References CompressedHilbert< T >::load().

5.11.3 Member Function Documentation

5.11.3.1 template<typename T > virtual size_t CompressedHilbert< T >::bytesUsed ( ) [inline], [virtual]

Returns

The number of bytes used by this storage scheme.

Implements **Matrix< T >** (p. **??**).

References CompressedHilbert< T >::indices.

5.11.3.2 template<typename T > virtual void CompressedHilbert< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References CompressedHilbert< T >::indices.

5.11.3.3 template<typename T > virtual void CompressedHilbert< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References CompressedHilbert< T >::indices, Matrix2HilbertCoordinates::IntegerToHilbert(), SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, CompressedHilbert< T >::values, and SparseMatrix< T, ULI >::zero_element.

Referenced by CompressedHilbert< T >::CompressedHilbert().

5.11.3.4 template<typename T > virtual void CompressedHilbert< T >::zax (const T * x, T * z) [inline], [virtual]

Calculates z=Ax.

Note z is not implicitly zeroed before multiplication.

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References CompressedHilbert< T >::indices, SparseMatrix< T, ULI >::nnz, and CompressedHilbert< T >::values.

5.11.3.5 template<typename T > virtual void CompressedHilbert< T >::zxa (const T * x, T * z) [inline], [virtual]

Calculates z=xA.

Note z is not implicitly zeroed before multiplication.

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References CompressedHilbert< T >::indices, SparseMatrix< T, ULI >::nnz, and CompressedHilbert< T >::values.

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

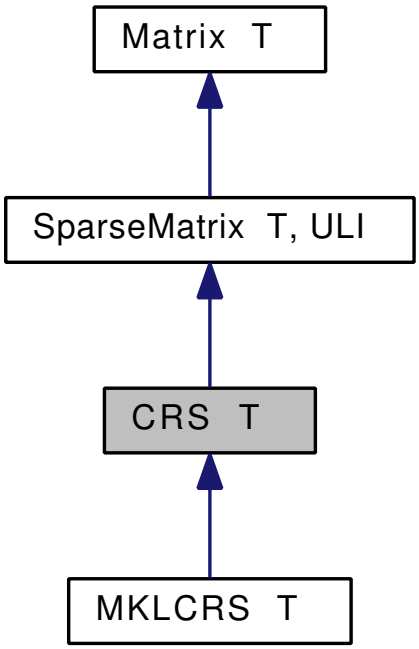
- CompressedHilbert.hpp

5.12 CRS< T > Class Template Reference

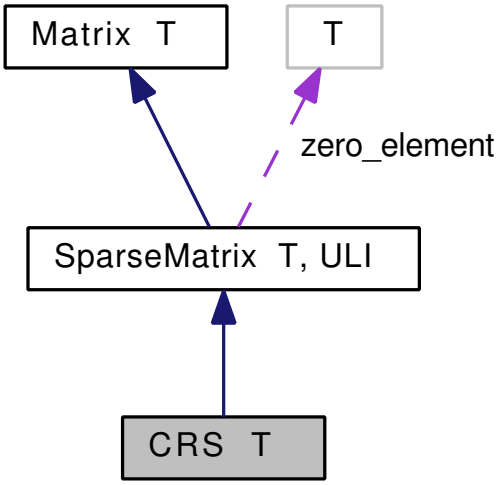
The compressed row storage sparse matrix data structure.

`#include <CRS.hpp>`

Inheritance diagram for CRS< T >:



Collaboration diagram for CRS< T >:



Public Member Functions

- **CRS** ()

  Base constructor.
- **CRS** (std::string file, T zero=0)

  Base constructor.
- **CRS** (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.
- **CRS** (**CRS**< T > &toCopy)

  Copy constructor.
- **CRS** (std::vector< **Triplet**< T > > input, ULI **m**, ULI **n**, T zero)

*Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.*

- virtual void **load** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero)
- T & **random_access** (ULI i, ULI j)

  Method which provides random matrix access to the stored sparse matrix.
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=xA function.
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=Ax function.
- template<size_t k>
  void **ZaX** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- template<size_t k>
  void **ZXa** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- ULI * **rowJump** ()

  Returns pointer to the row_start vector.
- ULI * **columnIndices** ()

  Returns pointer to the column index vector.
- T * **values** ()

  Returns pointer to the matrix nonzeros vector.
- virtual ~**CRS** ()

  Base deconstructor.
- virtual size_t **bytesUsed** ()

Protected Member Functions

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

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

Static Protected Member Functions

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

  Sorts 1D columnwise.

Protected Attributes

- ULI * **row_start**

  Array keeping track of individual row starting indices.
- T * **ds**

  Array containing the actual nnz non-zeros.
- ULI * **col_ind**

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

Additional Inherited Members

5.12.1 Detailed Description

template<typename T>class CRS< T >

The compressed row storage sparse matrix data structure.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 template<typename T> CRS< T >::CRS ( ) [inline]

Base constructor.

5.12.2.2 template<typename T> CRS< T >::CRS (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.12.2.3 template<typename T> CRS< T >::CRS (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** (p. **??**) 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 to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns of the matrix.
zero	The element considered to be zero.

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, CRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.12.2.4 template<typename T> CRS< T >::CRS (CRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **CRS** (p. **??**) datastructure to copy.

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::nor, CRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.12.2.5 template<typename T> CRS< T >::CRS (std::vector< Triplet< T > > *input*, ULI *m*, ULI *n*, T *zero*) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	The element considered to be zero.

References CRS< T >::load().

5.12.2.6 template<typename T> virtual CRS< T >::~CRS ( ) [inline], [virtual]

Base deconstructor.

References CRS< T >::col_ind, CRS< T >::ds, and CRS< T >::row_start.

5.12.3 Member Function Documentation

5.12.3.1 template<typename T> virtual size_t CRS< T >::bytesUsed ( ) [inline], [virtual]

See Also

Matrix::bytesUsed() (p. **??**)

Implements **Matrix< T >** (p. **??**).

References SparseMatrix< T, ULI >::nnz, and SparseMatrix< T, ULI >::nor.

5.12.3.2 template<typename T> ULI* CRS< T >::columnIndices ( ) [inline]

Returns pointer to the column index vector.

References CRS< T >::col_ind.

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

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

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.

References CRS< T >::col_ind.

Referenced by CRS< T >::random_access().

5.12.3.4 template<typename T> virtual void CRS< T >::getFirstIndexPair (ULI & *row*, ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References CRS< T >::col_ind, and CRS< T >::row_start.

5.12.3.5 template<typename T> virtual void CRS< T >::load (std::vector< Triplet< T > > & *input,* ULI *m,* ULI *n,* T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References CRS< T >::col_ind, CRS< T >::compareTriplets(), CRS< T >::ds, Triplet< T >::j(), SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, CRS< T >::row_start, Triplet< T >::value, and SparseMatrix< T, ULI >::zero_element.

Referenced by CRS< T >::CRS(), and MKLCRS< T >::MKLCRS().

5.12.3.6 template<typename T> T& 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 (p. **??**) valuei at (i,j).

References CRS< T >::ds, CRS< T >::find(), CRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.12.3.7 template<typename T> ULI* CRS< T >::rowJump ( ) [inline]

Returns pointer to the row_start vector.

References CRS< T >::row_start.

5.12.3.8 template<typename T> T* CRS< T >::values ( ) [inline]

Returns pointer to the matrix nonzeros vector.

References CRS< T >::ds.

5.12.3.9 template<typename T> virtual void CRS< T >::zax (const T *__restrict__ *x,* T *__restrict__ *z*) [inline], [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< T, ULI >** (p. **??**).

Reimplemented in **MKLCRS< T >** (p. **??**).

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nor, and CRS< T >::row_start.

5.12.3.10 template<typename T> template<size_t k> void CRS< T >::ZaX (const T *__restrict__ const *__restrict__ const *X*, T *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZaX (p. **??**)

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and CRS< T >::row_start.

5.12.3.11 template<typename T> template<size_t k> void CRS< T >::ZXa (const T *__restrict__ const *__restrict__ const *X*, T *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZXa (p. **??**)

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nor, and CRS< T >::row_start.

5.12.4 Member Data Documentation

5.12.4.1 template<typename T> ULI* CRS< T >::col_ind [protected]

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

Referenced by CRS< T >::columnIndices(), CRS< T >::CRS(), CRS< T >::find(), CRS< T >::getFirstIndexPair(), CRS< T >::load(), MKLCRS< T >::MKLCRS(), MKLCRS< T >::prepare(), CRS< T >::zax(), CRS< T >::ZaX(), CRS< T >::zxa(), CRS< T >::ZXa(), and CRS< T >::~CRS().

5.12.4.2 template<typename T> T* CRS< T >::ds [protected]

Array containing the actual nnz non-zeros.

Referenced by CRS< T >::CRS(), CRS< T >::load(), MKLCRS< T >::MKLCRS(), CRS< T >::random_access(), CRS< T >::values(), MKLCRS< T >::zax(), CRS< T >::zax(), CRS< T >::ZaX(), CRS< T >::zxa(), CRS< T >::ZXa(), and CRS< T >::~CRS().

5.12.4.3 template<typename T> ULI* CRS< T >::row_start [protected]

Array keeping track of individual row starting indices.

Referenced by CRS< T >::CRS(), CRS< T >::getFirstIndexPair(), CRS< T >::load(), MKLCRS< T >::MKLCRS(), MKLCRS< T >::prepare(), CRS< T >::random_access(), CRS< T >::rowJump(), CRS< T >::zax(), CRS< T >::ZaX(), CRS< T >::zxa(), CRS< T >::ZXa(), and CRS< T >::~CRS().

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

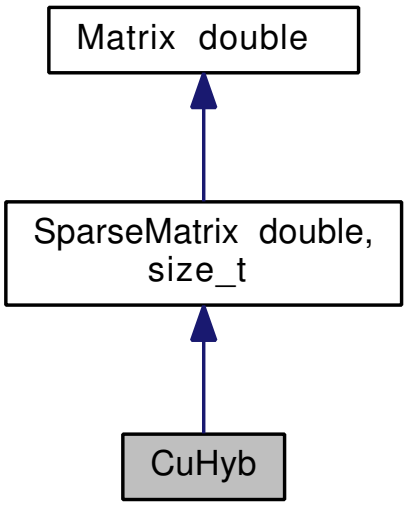
- CRS.hpp

5.13 CuHyb Class Reference

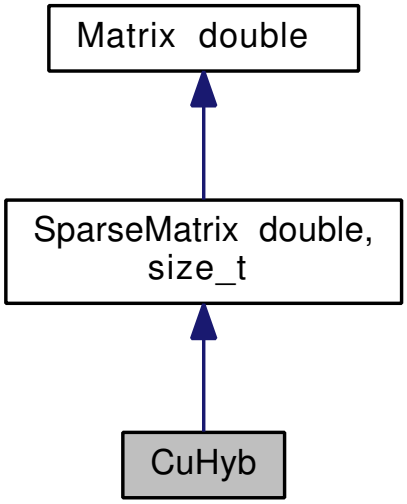
Wrapper class for the CuSparse HYB data structure for CUDA C.

`#include <CuHyb.hpp>`

Inheritance diagram for CuHyb:



Collaboration diagram for CuHyb:



Public Member Functions

- **CuHyb** ()

  Base constructor.
- **CuHyb** (std::string file, double zero=0)

  Base constructor.
- **CuHyb** (const ULI number_of_nonzeros, const ULI number_of_rows, const ULI number_of_cols, double zero)

  Base constructor which only initialises the internal arrays.
- **CuHyb** (std::vector< **Triplet**< double > > input, ULI **m**, ULI **n**, double zero)

  *Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.*
- virtual ~**CuHyb** ()

  Base deconstructor.
- virtual void **load** (std::vector< **Triplet**< double > > &input, ULI **m**, ULI **n**, double zero)
- virtual void **zxa** (const double *__restrict__ x, double *__restrict__ z)

In-place z=xA function.

- virtual void **zax** (const double *__restrict__ x, double *__restrict__ z)

  In-place z=Ax function.

- virtual void **zax** (const double *__restrict__ x, double *__restrict__ z, const unsigned long int repeat, const clockid_t clock_id=0, double *elapsed_time=NULL)

  In-place z=Ax function that supports repition and timing.

- virtual size_t **bytesUsed** ()
- virtual void **getFirstIndexPair** (size_t &**i**, size_t &**j**)

Static Protected Member Functions

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

  Sorts 1D columnwise.

Protected Attributes

- cusparseHandle_t **handle**

  Handle to CuSparse.

- cusparseMatDescr_t **descrA**

  CuSparse matrix descriptor.

- cusparseHybMat_t **hybA**

  GPU-local matrix.

- double * **GPUx**

  GPU-local buffer to the input vector.

- double * **GPUz**

  GPU-local buffer to the output vector.

- size_t **i**

  Top-left row coordinate.

- size_t **j**

  Top-right column coordinate.

Additional Inherited Members

5.13.1 Detailed Description

Wrapper class for the CuSparse HYB data structure for CUDA C.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 CuHyb::CuHyb ( )

Base constructor.

5.13.2.2 CuHyb::CuHyb (std::string *file,* double *zero =* 0)

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< double, size_t >::loadFromFile().

5.13.2.3 CuHyb::CuHyb (const ULI *number_of_nonzeros,* const ULI *number_of_rows,* const ULI *number_of_cols,* double *zero*)

Base constructor which only initialises the internal arrays.

Note that to gain a valid structure, these arrays have to be filled by some external mechanism, and this mechanism needs to call CUDA and CuSparse initialisation routines.

Parameters

number_of_nonzeros	The number of non-zeros to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns of the matrix.
zero	The element considered to be zero.

References SparseMatrix< double, size_t >::nnz, SparseMatrix< double, size_t >::noc, SparseMatrix< double, size_t >::nor, and SparseMatrix< double, size_t >::zero_element.

5.13.2.4 CuHyb::CuHyb (std::vector< **Triplet**< double > > *input,* ULI *m,* ULI *n,* double *zero*)

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	The element considered to be zero.

References load().

5.13.2.5 CuHyb::~CuHyb ( ) [virtual]

Base deconstructor.

References descrA, GPUx, GPUz, handle, and hybA.

5.13.3 Member Function Documentation

5.13.3.1 size_t CuHyb::bytesUsed ( ) [virtual]

See Also

Matrix::bytesUsed (p. **??**)

Implements **Matrix**< **double** > (p. **??**).

5.13.3.2 void CuHyb::getFirstIndexPair (size_t & *i,* size_t & *j*) [virtual]

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix**< **double, size_t** > (p. **??**).

References i, and j.

5.13.3.3 void CuHyb::load (std::vector< Triplet< double > > & *input*, ULI *m*, ULI *n*, double *zero*) [virtual]

See Also

SparseMatrix::load (p. **??**)

References compareTriplets(), descrA, GPUx, GPUz, handle, hybA, i, j, Triplet< T >::j(), SparseMatrix< double, size_t >::m(), SparseMatrix< double, size_t >::n(), SparseMatrix< double, size_t >::nnz, SparseMatrix< double, size_t >::noc, SparseMatrix< double, size_t >::nor, Triplet< T >::value, and SparseMatrix< double, size_t >::zero_element.

Referenced by CuHyb().

5.13.3.4 void CuHyb::zax (const double *__restrict__ *x*, double *__restrict__ *z*) [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< double, size_t >** (p. **??**).

References descrA, GPUx, GPUz, handle, hybA, SparseMatrix< double, size_t >::noc, and SparseMatrix< double, size_t >::nor.

5.13.3.5 void CuHyb::zax (const double *__restrict__ *x*, double *__restrict__ *z*, const unsigned long int *repeat*, const clockid_t *clock_id* = 0, double * *elapsed_time* = NULL) [virtual]

In-place z=Ax function that supports repition and timing.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.
repeat	How many times to repeat the SpMV, essentially computing z=A^{repeat}x.
clock_id	Which POSIX realtime clock to use for timing.
elapsed_time	Where to add the elapsed time to.

References descrA, GPUx, GPUz, handle, hybA, i, SparseMatrix< double, size_t >::noc, and SparseMatrix< double, size_t >::nor.

5.13.4 Member Data Documentation

5.13.4.1 cusparseMatDescr_t CuHyb::descrA [protected]

CuSparse matrix descriptor.

Referenced by load(), zax(), and ~CuHyb().

5.13.4.2 double* CuHyb::GPUx [protected]

GPU-local buffer to the input vector.

Referenced by load(), zax(), and ~CuHyb().

5.13.4.3 double* CuHyb::GPUz [protected]

GPU-local buffer to the output vector.

Referenced by load(), zax(), and ~CuHyb().

5.13.4.4 cusparseHandle_t CuHyb::handle [protected]

Handle to CuSparse.

Referenced by load(), zax(), and ~CuHyb().

5.13.4.5 cusparseHybMat_t CuHyb::hybA [protected]

GPU-local matrix.

Referenced by load(), zax(), and ~CuHyb().

5.13.4.6 size_t CuHyb::i [protected]

Top-left row coordinate.

Referenced by getFirstIndexPair(), load(), and zax().

5.13.4.7 size_t CuHyb::j [protected]

Top-right column coordinate.

Referenced by getFirstIndexPair(), and load().

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

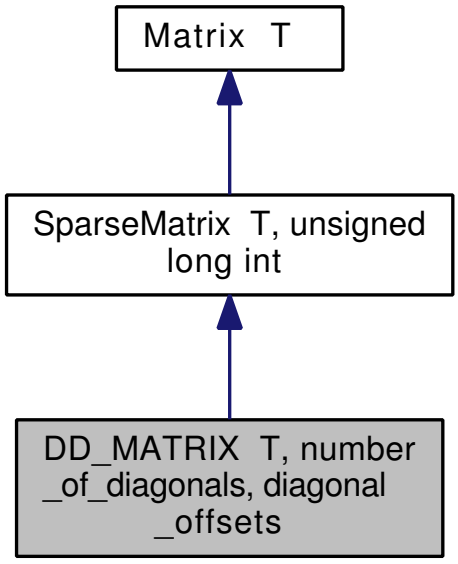
- CuHyb.hpp
- CuHyb.cpp

5.14 DD_MATRIX< T, number_of_diagonals, diagonal_offsets > Class Template Reference

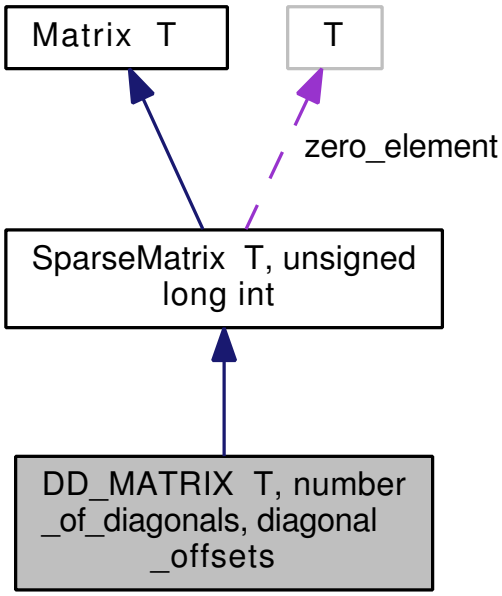
The dense diagonal matrix scheme; a storage scheme for sparse matrices consisting of only dense diagonals.

`#include <DD_MATRIX.hpp>`

Inheritance diagram for DD_MATRIX< T, number_of_diagonals, diagonal_offsets >:



Collaboration diagram for DD_MATRIX< T, number_of_diagonals, diagonal_offsets >:



Public Member Functions

- **DD_MATRIX** ()

  Base constructor.
- **DD_MATRIX** (std::string file, T zero=0)

  Base constructor.
- **DD_MATRIX** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero)

  Base constructor.
- **DD_MATRIX** (T **nonzeroes, ULI **m**, ULI **n**, T zero)

  Dense diagonal matrix specific constructor.

- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (unsigned long int &row, unsigned long int &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *x, T *z)

  In-place z=xA calculation algorithm.
- virtual void **zax** (const T *x, T *z)

  In-place z=Ax calculation algorithm.
- size_t **bytesUsed** ()
- ~**DD_MATRIX** ()

  Base destructor.

Protected Attributes

- T ** **nzs**

  The values of the nonzeros.
- ULI **full**

  How many full length diagonals this (possible not square matrix) contains.
- ULI **d**

  What the main diagonal length is (longest diagonal).
- bool **SELF_ALLOCATED**

  Whether or not nzs was allocated by this instance itself.
- size_t **bytes**

  Keeps track of the number of bytes spent for this matrix.

Additional Inherited Members

5.14.1 Detailed Description

template<typename T, int number_of_diagonals, int diagonal_offsets>class DD_MATRIX< T, number_of_diagonals, diagonal_offsets >

The dense diagonal matrix scheme; a storage scheme for sparse matrices consisting of only dense diagonals.

Parameters

T	type of numerical values to store in the matrix (int, unsigned int, float, double, ...)
number_of_diagonals	number of (assumed) dense diagonals in the matrix
diagonal_offsets	must be defined global in calling code, so that it is ensured constant at compiletime (necessary when template is used)

5.14.2 Constructor & Destructor Documentation

5.14.2.1 template<typename T , int number_of_diagonals, int diagonal_offsets> DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX ( ) [inline]

Base constructor.

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load().

5.14.2.2 template<typename T , int number_of_diagonals, int diagonal_offsets> DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, unsigned long int >::loadFromFile().

5.14.2.3 template<typename T , int number_of_diagonals, int diagonal_offsets> DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX (std::vector< Triplet< T > > & *input,* ULI *m,* ULI *n,* T *zero*) [inline]

Base constructor.

Parameters

input	std::vector of triplets to be stored in this scheme.
m	total number of rows.
n	total number of columns.
zero	what is to be considered the zero element.

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load().

5.14.2.4 template<typename T , int number_of_diagonals, int diagonal_offsets> DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX (T ** *nonzeroes,* ULI *m,* ULI *n,* T *zero*) [inline]

Dense diagonal matrix specific constructor.

Parameters

nonzeroes	The to this matrix belonging k times max(m,n) dense matrix representation.
m	Number of matrix rows.
n	Number of matrix columns.
zero	What is considered to be the zero element.

See Also

DD_MATRIX(input, m, n, zero) (p. **??**) for specifics on the rest of the parameters.

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytes, DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::d, DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::full, SparseMatrix< T, unsigned long int >::m(), SparseMatrix< T, unsigned long int >::n(), SparseMatrix< T, unsigned long int >::nnz, SparseMatrix< T, unsigned long int >::noc, SparseMatrix< T, unsigned long int >::nor, SparseMatrix< T, unsigned long int >::nzs(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::SELF_ALLOCATED, and SparseMatrix< T, unsigned long int >::zero_element.

5.14.2.5 template<typename T , int number_of_diagonals, int diagonal_offsets> DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::~DD_MATRIX ( ) [inline]

Base destructor.

References SparseMatrix< T, unsigned long int >::nzs(), and DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::SELF_ALLOCATED.

5.14.3 Member Function Documentation

5.14.3.1 template<typename T , int number_of_diagonals, int diagonal_offsets> size_t DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytesUsed ( ) [inline], [virtual]

Returns

The number of bytes used to store this data structure.

Implements **Matrix< T >** (p. **??**).

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytes.

5.14.3.2 template<typename T , int number_of_diagonals, int diagonal_offsets> virtual void DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::getFirstIndexPair (unsigned long int & *row,* unsigned long int & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, unsigned long int >** (p. **??**).

5.14.3.3 template<typename T , int number_of_diagonals, int diagonal_offsets> virtual void DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, unsigned long int >** (p. **??**).

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytes, DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::d, DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::full, SparseMatrix< T, unsigned long int >::m(), SparseMatrix< T, unsigned long int >::n(), SparseMatrix< T, unsigned long int >::nnz, SparseMatrix< T, unsigned long int >::noc, SparseMatrix< T, unsigned long int >::nor, SparseMatrix< T, unsigned long int >::nzs(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::SELF_ALLOCATED, and SparseMatrix< T, unsigned long int >::zero_element.

Referenced by DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX().

5.14.3.4 template<typename T , int number_of_diagonals, int diagonal_offsets> virtual void DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::zax (const T * *x,* T * *z*) [inline], [virtual]

In-place z=Ax calculation algorithm.

Parameters

x	The vector x supplied for calculation of Ax.
z	The result vector z. Should be pre-allocated with entries set to zero.

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::d, SparseMatrix< T, unsigned long int >::noc, and SparseMatrix< T, unsigned long int >::nzs().

5.14.3.5 template<typename T , int number_of_diagonals, int diagonal_offsets> virtual void DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::zxa (const T * *x,* T * *z*) [inline], [virtual]

In-place z=xA calculation algorithm.

Parameters

x	The vector x supplied for calculation of xA.
z	The result vector z. Should be pre-allocated with entries set to zero.

References DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::d, SparseMatrix< T, unsigned long int >::nor, and SparseMatrix< T, unsigned long int >::nzs().

5.14.4 Member Data Documentation

5.14.4.1 template<typename T , int number_of_diagonals, int diagonal_offsets> size_t DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytes [protected]

Keeps track of the number of bytes spent for this matrix.

Referenced by DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::bytesUsed(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX(), and DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load().

5.14.4.2 template<typename T , int number_of_diagonals, int diagonal_offsets> ULI DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::d [protected]

What the main diagonal length is (longest diagonal).

Referenced by DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::zax(), and DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::zxa().

5.14.4.3 template<typename T , int number_of_diagonals, int diagonal_offsets> ULI DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::full [protected]

How many full length diagonals this (possible not square matrix) contains.

Referenced by DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX(), and DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load().

5.14.4.4 template<typename T , int number_of_diagonals, int diagonal_offsets> T** DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::nzs [protected]

The values of the nonzeros.

5.14.4.5 template<typename T , int number_of_diagonals, int diagonal_offsets> bool DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::SELF_ALLOCATED [protected]

Whether or not nzs was allocated by this instance itself.

Referenced by DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::DD_MATRIX(), DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::load(), and DD_MATRIX< T, number_of_diagonals, diagonal_offsets >::~DD_MATRIX().

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

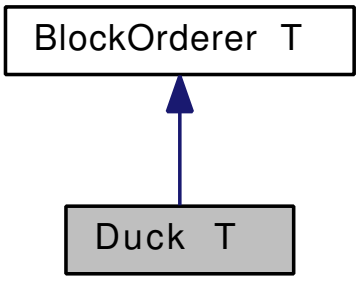
- DD_MATRIX.hpp

5.15 Duck< T > Class Template Reference

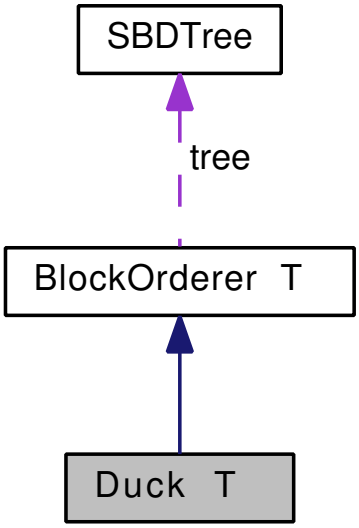
Codes the Minimal CCS block order.

`#include <Duck.hpp>`

Inheritance diagram for Duck< T >:



Collaboration diagram for Duck< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)
  Prefix traversal code.
- virtual void **in_readout** (const unsigned long int index)
  Infix traversal code.
- virtual void **post_readout** (const unsigned long int index)
  Postfix traversal code.

Additional Inherited Members

5.15.1 Detailed Description

template<typename T>class Duck< T >

Codes the Minimal CCS block order.

5.15.2 Member Function Documentation

5.15.2.1 template<typename T > virtual void Duck< T >::in_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Infix traversal code.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

5.15.2.2 template<typename T > virtual void Duck< T >::post_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Postfix traversal code.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::output, and BlockOrderer< T >::tree.

5.15.2.3 template<typename T > virtual void Duck< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix traversal code.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, and BlockOrderer< T >::tree.

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

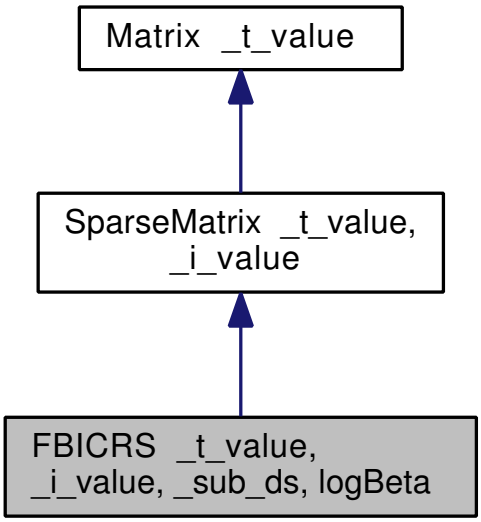
- Duck.hpp

5.16 FBICRS< _t_value, _i_value, _sub_ds, logBeta > Class Template Reference

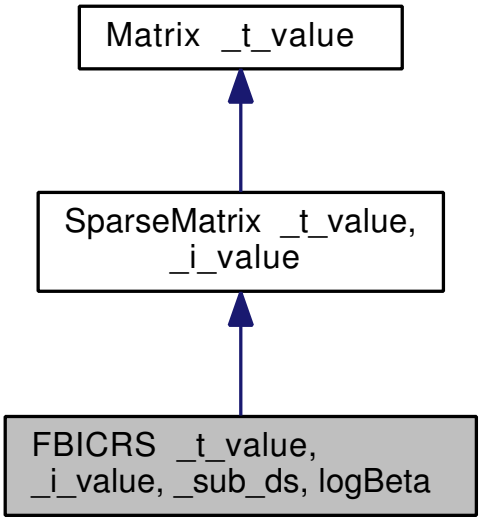
Hierarchical **BICRS** (p. **??**) with fixed subblock size and distribution.

`#include <FBICRS.hpp>`

Inheritance diagram for FBICRS< _t_value, _i_value, _sub_ds, logBeta >:



Collaboration diagram for FBICRS< _t_value, _i_value, _sub_ds, logBeta >:



Public Member Functions

- virtual ~**FBICRS** ()

  Default destructor.
- **FBICRS** ()

  Base constructor (initialises with invalid data).
- **FBICRS** (std::string file, _t_value zero=0)

  Base file-based constructor.
- **FBICRS** (_i_value *row, _i_value *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)

  Base COO-based constructor.
- **FBICRS** (std::vector< **Triplet**< _t_value > > &input, ULI **m**, ULI **n**, _t_value zero=0)

  Base Triplet-based constructor.
- **FBICRS** (std::vector< std::vector< **Triplet**< _t_value > > > &input, ULI **m**, ULI **n**, _t_value zero=0)

Base hierarchical constructor.

- virtual void **load** (std::vector< **Triplet**< _t_value > > &input, _i_value **m**, _i_value **n**, _t_value zero)

  *Builds input matrix from **Triplet** (p. **??**) input.*
- void **load** (_i_value *row, _i_value *col, _t_value *val, ULI **m**, ULI **n**, ULI nz, _t_value zero)

  Builds input matrix from COO input.
- void **load** (std::vector< std::vector< **Triplet**< _t_value > > > &input, ULI **m**, ULI **n**, _t_value zero)

  *Builds input matrix from **Triplet** (p. **??**) input.*
- virtual void **getFirstIndexPair** (_i_value &row, _i_value &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  In-place z=xA function.
- template<size_t k>
  void **ZXa** (const _t_value *__restrict__ const *__restrict__ const X, _t_value *__restrict__ const *__restrict__ const Z)
- virtual void **zxa_fb** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  *Interleaved zxa kernel for use with the **BetaHilbert** (p. **??**) scheme.*
- virtual void **zax** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  In-place z=Ax function.
- template<size_t k>
  void **ZaX** (const _t_value *__restrict__ const *__restrict__ const X, _t_value *__restrict__ const *__restrict__ const Z)
- virtual void **zax_fb** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  *Interleaved zax kernel for use with the **BetaHilbert** (p. **??**) scheme.*
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Public Attributes

- size_t **fillIn**

  Stores the total fillIn.
- _sub_ds ** **dss**

  Stores the lower-level data structures.
- ULI **n_overflow**

  Caches the overflow forcing value.

Static Public Attributes

- static const ULI **beta_m** = 1l << (logBeta-1)

  Maximum matrix size for _sub_ds with the above data type; row size.
- static const ULI **beta_n** = **beta_m**

  Maximum matrix size for _sub_ds with the above data type; column size.

Protected Attributes

- ULI **r_start**

  Row index of the first nonzero.
- ULI **c_start**

  Column index of the first nonzero.
- ULI **r_end**

  Row index of the last nonzero.
- ULI **c_end**

Column index of the last nonzero.

- ULI **jumps**

  How many row jumps are stored.

- _i_value * **r_inc**

  Row increment array.

- _i_value * **c_inc**

  Column increment array.

5.16.1 Detailed Description

template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15>class FBICRS< _t_value, _i_value, _sub_ds, logBeta >

Hierarchical **BICRS** (p. **??**) with fixed subblock size and distribution.

Uses compression on upper and lower level. Currently handles matrices with at most 39 bits indices (signed ints on upper level with unsigned chars on the lower level; it was either that or 72 bits, but the latter cannot efficiently be represented on 64-bit architectures).

Template Parameters

_sub_ds	Defaults chooses a vectorised **BICRS** (p. **??**) structure with unsigned short int index type as sub data type. Other options: typedef ICRS< _t_value, unsigned char > _sub_ds; typedef ICRS< _t_value, uint16_t > _sub_ds; typedef vecBICRS< _t_value, 8, 1, int16_t > _sub_ds; typedef ICRS< _t_value, unsigned int > _sub_ds; (default) typedef CRS< _t_value > _sub_ds; (using the latter with logBeta = 64 boils down to a block row distributed CRS-based parallel SpMV)
logBeta	The log of the size of the maximum block size represented by _sub_ds. Options: static const unsigned char logBeta = 7; //=max ICRS/unsigned char static const unsigned char logBeta = 11; //=2k static const unsigned char logBeta = 12; //=4k static const unsigned char logBeta = 14; //=max vecBICRS/int16_t=16k static const unsigned char logBeta = 15; //=max ICRS/uint16_t=32k static const unsigned char logBeta = 17; //=128k static const unsigned char logBeta = 31; //=max ICRS/unsigned short static const unsigned char logBeta = 64; //=max **CRS** (p. **??**)

5.16.2 Constructor & Destructor Documentation

5.16.2.1 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual FBICRS< _t_value, _i_value, _sub_ds, logBeta >::~FBICRS ( ) [inline], [virtual]

Default destructor.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, SparseMatrix< _t_value, _i_value >::nnz, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc.

5.16.2.2 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS ( ) [inline]

Base constructor (initialises with invalid data).

5.16.2.3 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS (std::string *file,* _t_value *zero* = 0) [inline]

Base file-based constructor.

References SparseMatrix< _t_value, _i_value >::loadFromFile().

5.16.2.4 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS (_i_value * *row,* _i_value * *col,* _t_value * *val,* ULI *m,* ULI *n,* ULI *nz,* _t_value *zero*) [inline]

Base COO-based constructor.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.2.5 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS (std::vector< Triplet< _t_value > > & *input,* ULI *m,* ULI *n,* _t_value *zero* = 0) [inline]

Base Triplet-based constructor.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.2.6 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS (std::vector< std::vector< Triplet< _t_value > > > & *input,* ULI *m,* ULI *n,* _t_value *zero* = 0) [inline]

Base hierarchical constructor.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.3 Member Function Documentation

5.16.3.1 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual size_t FBICRS< _t_value, _i_value, _sub_ds, logBeta >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< _t_value >** (p. **??**).

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::jumps, and SparseMatrix< _t_value, _i_value >::nnz.

Referenced by BetaHilbert< T >::thread().

5.16.3.2 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::getFirstIndexPair (_i_value & *row,* _i_value & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< _t_value, _i_value >** (p. **??**).

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start.

5.16.3.3 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load (std::vector< Triplet< _t_value > > & *input*, _i_value *m*, _i_value *n*, _t_value *zero*) [inline], [virtual]

Builds input matrix from **Triplet** (p. **??**) input.

Parameters

input	Input matrix in triplet form.
m	Number of rows in the input matrix.
n	Number of columns in the input matrix.
zero	The zero element of this matrix instance.

Implements **SparseMatrix< _t_value, _i_value >** (p. **??**).

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::beta_m, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::beta_n, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, SparseMatrix< _t_value, _i_value >::m(), SparseMatrix< _t_value, _i_value >::n(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, SparseMatrix< _t_value, _i_value >::nor, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start, and SparseMatrix< _t_value, _i_value >::zero_element.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::FBICRS(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.3.4 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load (_i_value * *row*, _i_value * *col*, _t_value * *val*, ULI *m*, ULI *n*, ULI *nz*, _t_value *zero*) [inline]

Builds input matrix from COO input.

Parameters

row	COO row index array.
col	COO column index array.
val	COO nonzero value array.
m	Number of rows in the input matrix.
n	Number of columns in the input matrix.
nz	The number of nonzeroes in the COO input.
zero	The zero element of this matrix instance.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.3.5 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load (std::vector< std::vector< Triplet< _t_value > > > & *input*, ULI *m*, ULI *n*, _t_value *zero*) [inline]

Builds input matrix from **Triplet** (p. **??**) input.

Parameters

input	Hierarchical input matrix in triplet form.

m	Number of rows in the input matrix.
n	Number of columns in the input matrix.
zero	The zero element of this matrix instance.

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::beta_m, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::beta_n, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::fillIn, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::jumps, SparseMatrix< _t_value, _i_value >::m(), SparseMatrix< _t_value, _i_value >::n(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, SparseMatrix< _t_value, _i_value >::nor, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start, and SparseMatrix< _t_value, _i_value >::zero_element.

5.16.3.6 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax (const _t_value *__restrict__ *x*, _t_value *__restrict__ *z*) [inline], [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< _t_value, _i_value >** (p. **??**).

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb().

5.16.3.7 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> template<size_t k> void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZaX (const _t_value *__restrict__ const *__restrict__ const *X*, _t_value *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZaX (p. **??**)

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start.

5.16.3.8 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb (const _t_value *__restrict__ *x_p*, _t_value *__restrict__ *y_p*) [inline], [virtual]

Interleaved zax kernel for use with the **BetaHilbert** (p. **??**) scheme.

See Also

Matrix::zax (p. **??**)

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds,

logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::jumps, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax().

Referenced by BetaHilbert< T >::thread().

5.16.3.9 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa (const _t_value *__restrict__ x, _t_value *__restrict__ z) [inline], [virtual]

In-place z=xA function.

Parameters

x	The x vector to apply in left-multiplication to A
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< _t_value, _i_value >** (p. **??**).

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

5.16.3.10 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> template<size_t k> void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZXa (const _t_value *__restrict__ const *__restrict__ const X, _t_value *__restrict__ const *__restrict__ const Z) [inline]

See Also

Matrix::ZXa (p. **??**)

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start.

5.16.3.11 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> virtual void FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p) [inline], [virtual]

Interleaved zxa kernel for use with the **BetaHilbert** (p. **??**) scheme.

See Also

Matrix::zxa (p. **??**)

References FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::dss, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::jumps, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::n_overflow, SparseMatrix< _t_value, _i_value >::nnz, SparseMatrix< _t_value, _i_value >::noc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_end, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc, FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start, and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa().

Referenced by BetaHilbert< T >::thread().

5.16.4 Member Data Documentation

5.16.4.1 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> ULI FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_end [protected]

Column index of the last nonzero.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

5.16.4.2 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> _i_value* FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_inc [protected]

Column increment array.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZaX(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZXa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::~FBICRS().

5.16.4.3 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> ULI FBICRS< _t_value, _i_value, _sub_ds, logBeta >::c_start [protected]

Column index of the first nonzero.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::getFirstIndexPair(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZaX(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZXa(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

5.16.4.4 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> size_t FBICRS< _t_value, _i_value, _sub_ds, logBeta >::fillIn

Stores the total fillIn.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load().

5.16.4.5 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> ULI FBICRS< _t_value, _i_value, _sub_ds, logBeta >::jumps [protected]

How many row jumps are stored.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::bytesUsed(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

5.16.4.6 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> ULI FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_end [protected]

Row index of the last nonzero.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

5.16.4.7 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> _i_value* FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_inc [protected]

Row increment array.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZaX(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZXa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::~FBICRS().

5.16.4.8 template<typename _t_value, typename _i_value = LI, typename _sub_ds = ICRS< _t_value, uint16_t >, unsigned char logBeta = 15> ULI FBICRS< _t_value, _i_value, _sub_ds, logBeta >::r_start [protected]

Row index of the first nonzero.

Referenced by FBICRS< _t_value, _i_value, _sub_ds, logBeta >::getFirstIndexPair(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::load(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZaX(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zax_fb(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa(), FBICRS< _t_value, _i_value, _sub_ds, logBeta >::ZXa(), and FBICRS< _t_value, _i_value, _sub_ds, logBeta >::zxa_fb().

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

- FBICRS.hpp

5.17 FileToVT Class Reference

Class responsible for reading in matrix market files and converting them to vector< Triplet > format.

`#include <FileToVT.hpp>`

Static Public Member Functions

- static std::vector< **Triplet** < double > > **parse** (std::string filename)

  Parses a matrix-market input file.

- static std::vector< **Triplet** < double > > **parse** (std::string filename, ULI &m, ULI &n)

  Parses a matrix-market input file.

- static std::vector< **Triplet** < double > > **parse** (std::string filename, ULI &m, ULI &n, unsigned long int &nnz)

  Parses a matrix-market input file.

5.17.1 Detailed Description

Class responsible for reading in matrix market files and converting them to vector< Triplet > format.

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

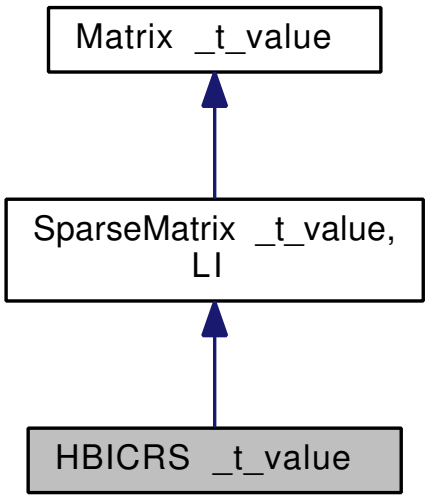
- FileToVT.hpp
- FileToVT.cpp

5.18 HBICRS< _t_value > Class Template Reference

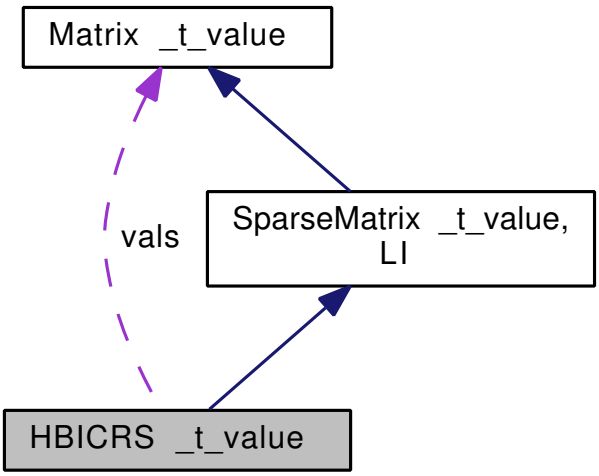
Hierarchical Bi-directional Incremental Compressed Row Storage scheme.

`#include <HBICRS.hpp>`

Inheritance diagram for HBICRS< _t_value >:



Collaboration diagram for HBICRS< _t_value >:



Public Member Functions

- ~**HBICRS** ()

  Base deconstructor.
- **HBICRS** ()

  Base constructor.
- **HBICRS** (std::string file, _t_value zero=0)

  Base constructor.
- **HBICRS** (std::vector< **Triplet**< _t_value > > &input, LI **m**, LI **n**, _t_value zero)

  Base constructor.

- **HBICRS** (std::vector< std::vector< **Triplet**< _t_value > > > &input, signed char *group_type, LI **m**, LI **n**, _t_value zero=0)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< _t_value > > &input, LI **m**, LI **n**, _t_value zero)

  This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.
- virtual void **load** (std::vector< std::vector< **Triplet**< _t_value > > > &input, signed char *group_type, LI **m**, LI **n**, _t_value zero)

  Constructs the hierarchical part.
- void **load** (LI *row, LI *col, LI **m**, LI **n**, LI nb)

  *Builds the **BICRS** (p. **??**) structure.*
- virtual void **getFirstIndexPair** (LI &i, LI &j)
- virtual void **zxa** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=xA, but does not allocate y itself.
- virtual void **zax** (const _t_value *__restrict__ x_p, _t_value *__restrict__ y_p)

  Calculates y=Ax, but does not allocate y itself.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Protected Attributes

- size_t **jumps**

  The number of row jumps.
- LI * **r_inc**

  Stores the row jumps; size is at maximum the number of nonzeros.
- LI * **c_inc**

  Stores the column jumps; size is exactly the number of nonzeros.
- **Matrix**< _t_value > ** **vals**

  Stores the values individual storage schemes.
- ULI **ntt**

  Caches n times two.

Additional Inherited Members

5.18.1 Detailed Description

template<typename _t_value>class HBICRS< _t_value >

Hierarchical Bi-directional Incremental Compressed Row Storage scheme.

Stores other **SparseMatrix** (p. **??**) data structures, can include other **HBICRS** (p. **??**) schemes, but this is not recommended with regards to efficiency. Supports jumping back and forward within columns. Supports jumping back and forward between rows. Main storage direction in column-wise. Storage requirements are 2nz plus the number of row jumps required, plus the storage requirements for each stored data structure, of course. Many row jumps are disadvantageous to storage as well as speed.

Parameters

_t_value	The type of the nonzeros in the matrix.

This class is based on the **BICRS** (p. **??**) as per revision 75. **BICRS** (p. **??**) was chosen since it is an all-round improvement over Triplets, with the additional advantage of being pointer based (no jumps required when going recursive into deeper storage schemes).

Warning: this class uses assertions! For optimal performance, define the NDEBUG flag (e.g., pass -DNDEBUG as a compiler flag).

5.18.2 Constructor & Destructor Documentation

5.18.2.1 template<typename _t_value> HBICRS< _t_value >::~HBICRS ( ) [inline]

Base deconstructor.

5.18.2.2 template<typename _t_value> HBICRS< _t_value >::HBICRS ( ) [inline]

Base constructor.

5.18.2.3 template<typename _t_value> HBICRS< _t_value >::HBICRS (std::string *file,* _t_value *zero =* 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

5.18.2.4 template<typename _t_value> HBICRS< _t_value >::HBICRS (std::vector< Triplet< _t_value > > & *input,* LI *m,* LI *n,* _t_value *zero*) [inline]

Base constructor.

See Also

SparseMatrix::SparseMatrix(input, m, n, zero)

5.18.2.5 template<typename _t_value> HBICRS< _t_value >::HBICRS (std::vector< std::vector< Triplet< _t_value > > > & *input,* signed char * *group_type,* LI *m,* LI *n,* _t_value *zero =* 0) [inline]

Base constructor.

See Also

SparseMatrix::SparseMatrix(input, m, n, zero)

5.18.3 Member Function Documentation

5.18.3.1 template<typename _t_value> virtual size_t HBICRS< _t_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< _t_value >** (p. **??**).

Referenced by HBICRS< T >::bytesUsed(), and BlockHilbert< T >::bytesUsed().

5.18.3.2 template<typename _t_value> virtual void HBICRS< _t_value >::getFirstIndexPair (LI & *i,* LI & *j*) [inline], [virtual]

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix**< **_t_value, LI** > (p. **??**).

Referenced by BlockHilbert< T >::getFirstIndexPair().

5.18.3.3 template<typename _t_value> virtual void HBICRS< _t_value >::load (std::vector< **Triplet**< _t_value > > & *input,* LI *m,* LI *n,* _t_value *zero*) [inline], [virtual]

This function will rewrite the std::vector< Triplet > structure to one suitable for the other load function.

See Also

load(row, col, val, m, n, nz)
SparseMatrix::load (p. **??**)

Implements **SparseMatrix**< **_t_value, LI** > (p. **??**).

Referenced by HBICRS< T >::HBICRS(), and HBICRS< T >::load().

5.18.3.4 template<typename _t_value> virtual void HBICRS< _t_value >::load (std::vector< std::vector< **Triplet**< _t_value > > > & *input,* signed char * *group_type,* LI *m,* LI *n,* _t_value *zero*) [inline], [virtual]

Constructs the hierarchical part.

Note that this function does not do anything with offsets, due to lack of support in the other datastructures used.

5.18.3.5 template<typename _t_value> void HBICRS< _t_value >::load (LI * *row,* LI * *col,* LI *m,* LI *n,* LI *nb*) [inline]

Builds the **BICRS** (p. **??**) structure.

5.18.3.6 template<typename _t_value> virtual void HBICRS< _t_value >::zax (const _t_value *__restrict__ *x_p,* _t_value *__restrict__ *y_p*) [inline], [virtual]

Calculates y=Ax, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix**< **_t_value, LI** > (p. **??**).

Referenced by BlockHilbert< T >::zax().

5.18.3.7 template<typename _t_value> virtual void HBICRS< _t_value >::zxa (const _t_value *__restrict__ *x_p,* _t_value *__restrict__ *y_p*) [inline], [virtual]

Calculates y=xA, but does not allocate y itself.

Parameters

x_p	The input vector should be initialised and of correct measurements.
y_p	The output vector should be preallocated and of size m. Furthermore, y[i]=0 for all i, 0<=i<m.

Implements **SparseMatrix< _t_value, LI >** (p. **??**).

Referenced by BlockHilbert< T >::zxa().

5.18.4 Member Data Documentation

5.18.4.1 template<typename _t_value> LI* HBICRS< _t_value >::c_inc [protected]

Stores the column jumps; size is exactly the number of nonzeros.

Referenced by HBICRS< T >::getFirstIndexPair(), HBICRS< T >::load(), HBICRS< T >::zax(), HBICRS< T >::zxa(), and HBICRS< T >::~HBICRS().

5.18.4.2 template<typename _t_value> size_t HBICRS< _t_value >::jumps [protected]

The number of row jumps.

Referenced by HBICRS< T >::bytesUsed(), and HBICRS< T >::load().

5.18.4.3 template<typename _t_value> ULI HBICRS< _t_value >::ntt [protected]

Caches n times two.

Referenced by HBICRS< T >::load(), HBICRS< T >::zax(), and HBICRS< T >::zxa().

5.18.4.4 template<typename _t_value> LI* HBICRS< _t_value >::r_inc [protected]

Stores the row jumps; size is *at maximum* the number of nonzeros.

Referenced by HBICRS< T >::getFirstIndexPair(), HBICRS< T >::load(), HBICRS< T >::zax(), HBICRS< T >::zxa(), and HBICRS< T >::~HBICRS().

5.18.4.5 template<typename _t_value> Matrix< _t_value >** HBICRS< _t_value >::vals [protected]

Stores the values individual storage schemes.

Referenced by HBICRS< T >::bytesUsed(), HBICRS< T >::load(), HBICRS< T >::zax(), HBICRS< T >::zxa(), and HBICRS< T >::~HBICRS().

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

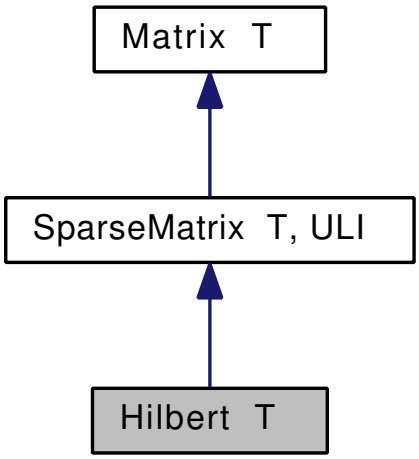
- HBICRS.hpp

5.19 Hilbert< T > Class Template Reference

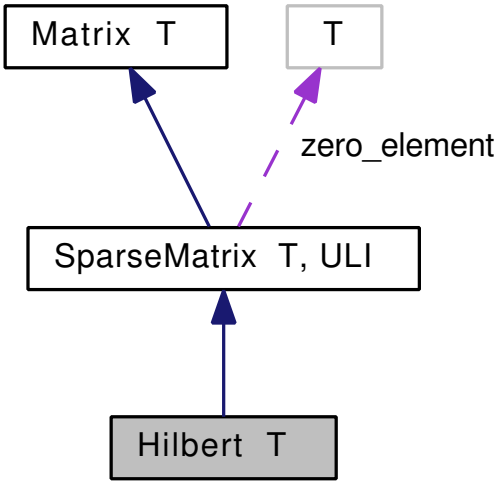
The **Hilbert** (p. **??**) scheme backed by (C)**BICRS** (p. **??**).

`#include <Hilbert.hpp>`

Inheritance diagram for Hilbert< T >:



Collaboration diagram for Hilbert< T >:



Public Member Functions

- virtual ~**Hilbert** ()

  Base deconstructor.
- **Hilbert** ()

  Base constructor.
- **Hilbert** (std::string file, T zero=0)

  Base constructor.
- **Hilbert** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *x, T *z)

  Calculates z=xA.
- virtual void **zax** (const T *x, T *z)

Calculates z=Ax.

- virtual size_t **bytesUsed** ()

  Gets the number of bytes used by this storage scheme.

- void **saveBinary** (const std::string fn)

  *Saves the current **Hilbert** (p. **??**) structure in binary triplet form.*

- void **loadBinary** (const std::string fn)

  *Loads from binary triplets, assumes **Hilbert** (p. **??**) ordering already done.*

Static Protected Member Functions

- static **Matrix**< T > * **getDataStructure** (std::vector< **Triplet**< T > > &tds, const ULI **m**, const ULI **n**, T zero)

  Gets the data structure.

Protected Attributes

- ULI **minexp**

  Minimum number of expansions.

- std::vector< **HilbertTriplet**< T > > **ds**

  Vector storing the non-zeros and their locations.

- **Matrix**< T > * **ads**

  Actual data structure.

Additional Inherited Members

5.19.1 Detailed Description

template<typename T>class Hilbert< T >

The **Hilbert** (p. **??**) scheme backed by (C)**BICRS** (p. **??**).

In effect similar to the **Hilbert** (p. **??**) **Triplet** (p. **??**) scheme (**HTS** (p. **??**)), but uses **BICRS** (p. **??**) to store the nonzeroes.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 template<typename T> virtual Hilbert< T >::~Hilbert ( ) [inline], [virtual]

Base deconstructor.

References Hilbert< T >::ads.

5.19.2.2 template<typename T> Hilbert< T >::Hilbert ( ) [inline]

Base constructor.

References Hilbert< T >::ads.

5.19.2.3 template<typename T> Hilbert< T >::Hilbert (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References Hilbert< T >::ads, and SparseMatrix< T, ULI >::loadFromFile().

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

Base constructor.

Warning: the zero parameter is currently NOT USED!

Parameters

input	Raw input of normal triplets.
m	Total number of rows.
n	Total number of columns.
zero	What elements is considered to-be zero.

References Hilbert< T >::ads, and Hilbert< T >::load().

5.19.3 Member Function Documentation

5.19.3.1 template<typename T> virtual size_t Hilbert< T >::bytesUsed ( ) [inline], [virtual]

Gets the number of bytes used by this storage scheme.

Implements **Matrix< T >** (p. **??**).

References Hilbert< T >::ads.

5.19.3.2 template<typename T> static Matrix< T >* Hilbert< T >::getDataStructure (std::vector< Triplet< T > > & *tds,* const ULI *m,* const ULI *n,* T *zero*) [inline], [static], [protected]

Gets the data structure.

Convience function, enables quick changes in **Hilbert** (p. **??**) backing structure.

References CBICRS_factory< _t_value >::getCBICRS(), SparseMatrix< T, ULI >::m(), and SparseMatrix< T, ULI >::n().

Referenced by Hilbert< T >::load(), and Hilbert< T >::loadBinary().

5.19.3.3 template<typename T> virtual void Hilbert< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References Hilbert< T >::ds.

5.19.3.4 template<typename T> virtual void Hilbert< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix**< **T, ULI** > (p. **??**).

References Hilbert< T >::ads, Hilbert< T >::ds, Hilbert< T >::getDataStructure(), SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and SparseMatrix< T, ULI >::zero_element.

Referenced by Hilbert< T >::Hilbert().

5.19.3.5 template<typename T> void Hilbert< T >::saveBinary (const std::string *fn*) [inline]

Saves the current **Hilbert** (p. **??**) structure in binary triplet form.

Parameters

fn	Filename to save to.

See Also

HilbertTriplet< **T** >**::save** (p. **??**) ‘

References Hilbert< T >::ds, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and HilbertTriplet< T >::save().

5.19.3.6 template<typename T> virtual void Hilbert< T >::zax (const T * *x*, T * *z*) [inline], [virtual]

Calculates z=Ax.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References Hilbert< T >::ads.

5.19.3.7 template<typename T> virtual void Hilbert< T >::zxa (const T * *x*, T * *z*) [inline], [virtual]

Calculates z=xA.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References Hilbert< T >::ads.

5.19.4 Member Data Documentation

5.19.4.1 template<typename T> Matrix< T >* Hilbert< T >::ads [protected]

Actual data structure.

Referenced by Hilbert< T >::bytesUsed(), Hilbert< T >::Hilbert(), Hilbert< T >::load(), Hilbert< T >::loadBinary(), Hilbert< T >::zax(), Hilbert< T >::zxa(), and Hilbert< T >::~Hilbert().

5.19.4.2 template<typename T> std::vector< HilbertTriplet< T > > Hilbert< T >::ds [protected]

Vector storing the non-zeros and their locations.

Referenced by Hilbert< T >::getFirstIndexPair(), Hilbert< T >::load(), and Hilbert< T >::saveBinary().

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

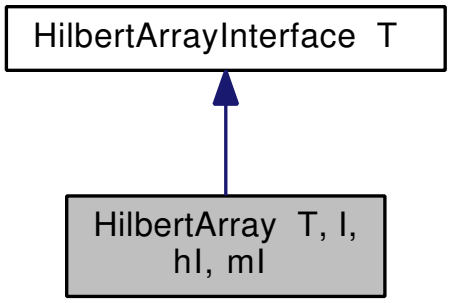
- Hilbert.hpp

5.20 HilbertArray< T, I, hI, mI > Class Template Reference

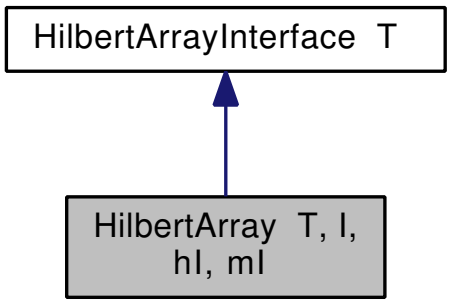
Actual storage implementation.

`#include <HilbertArray.hpp>`

Inheritance diagram for HilbertArray< T, I, hI, mI >:



Collaboration diagram for HilbertArray< T, I, hI, mI >:



Public Member Functions

- **HilbertArray** (const std::vector< unsigned long int > &input)

  Base constructor.
- virtual ULI **getFirstRowIndex** ()

  Gets the start-location of the first nonzero in this array.
- virtual ULI **getFirstColumnIndex** ()

  Gets the start-location of the first nonzero in this array.
- virtual void **zxa** (const T *__restrict__ const &x, T *__restrict__ const &y, const T *__restrict__ &v, const T *__restrict__ const &v_end)

  Flat implementation of the zax.

- virtual void **zax** (const T *__restrict__ const &x, T *__restrict__ const &y, const T *__restrict__ &v, const T *__restrict__ const &v_end)

  Flat implementation of the zax.
- virtual size_t **bytesUsed** ()
- virtual void **moveToStart** (const T *__restrict__ &x, T *__restrict__ &y, const T &v)

  Moves this interface to the start location and perform a single multiply-add there.
- virtual void **moveToNext** (const T *__restrict__ &x, T *__restrict__ &y, const T &v)

  Moves this interface and the two given vectors to the next position.
- virtual ~**HilbertArray** ()

  Base deconstructor.

Protected Member Functions

- void **decode** (const hI coor, mI &row, mI &col)

  *Translates a **Hilbert** (p. **??**) coordinate into a row and column index.*
- void **decode** (hI &cur, const I &inc, mI &row, mI &col)

  *Translates a **Hilbert** (p. **??**) coordinate into a row and column index.*

Protected Attributes

- hI **start_coor**

  ***Hilbert** (p. **??**) start coordinate.*
- I * **array**

  Index storage array.
- I *__restrict__ **curpos**

  Current position in the array.
- hI **curcoor**

  *Current **Hilbert** (p. **??**) coordinate.*
- mI **currow**

  Current row position.
- mI **curcol**

  Current column position.
- size_t **bytes**

  Keeps track of memory use.

5.20.1 Detailed Description

template<typename T, typename I, typename hI, typename mI>class HilbertArray< T, I, hI, mI >

Actual storage implementation.

Template Parameters

T	Nonzero data type.
I	Hilbert-coordinate increment type.
hI	Full Hilbert-coordinate data type.
mI	Full matrix coordinate data type.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 template<typename T , typename I , typename hI , typename mI > HilbertArray< T, I, hI, mI >::HilbertArray (const std::vector< unsigned long int > & *input*) [inline]

Base constructor.

References HilbertArray< T, I, hI, mI >::array, HilbertArray< T, I, hI, mI >::bytes, HilbertArray< T, I, hI, mI >::curpos, and HilbertArray< T, I, hI, mI >::start_coor.

5.20.2.2 template<typename T , typename I , typename hI , typename mI > virtual HilbertArray< T, I, hI, mI >::~HilbertArray ( ) [inline], [virtual]

Base deconstructor.

References HilbertArray< T, I, hI, mI >::array.

5.20.3 Member Function Documentation

5.20.3.1 template<typename T , typename I , typename hI , typename mI > virtual size_t HilbertArray< T, I, hI, mI >::bytesUsed ( ) [inline], [virtual]

Returns

The number of bytes used by this **Hilbert** (p. **??**) array

Implements **HilbertArrayInterface< T >** (p. **??**).

References HilbertArray< T, I, hI, mI >::bytes.

5.20.3.2 template<typename T , typename I , typename hI , typename mI > void HilbertArray< T, I, hI, mI >::decode (const hI *coor,* mI & *row,* mI & *col*) [inline], [protected]

Translates a **Hilbert** (p. **??**) coordinate into a row and column index.

Parameters

coor	Input **Hilbert** (p. **??**) coordinate.
row	Where to decode the row index.
col	Where to decode the column index.

Referenced by HilbertArray< T, I, hI, mI >::getFirstColumnIndex(), HilbertArray< T, I, hI, mI >::getFirstRowIndex(), HilbertArray< T, I, hI, mI >::moveToNext(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

5.20.3.3 template<typename T , typename I , typename hI , typename mI > void HilbertArray< T, I, hI, mI >::decode (hI & *cur,* const I & *inc,* mI & *row,* mI & *col*) [inline], [protected]

Translates a **Hilbert** (p. **??**) coordinate into a row and column index.

Only updates the bits necessary to be updated, i.e., skips the first startPos bits of row and col. Assumes the bits that will be skipped already have the correct value (of course), i.e., this only works with the incremental scheme!

Parameters

cur	Current input **Hilbert** (p. **??**) coordinate.
inc	**Hilbert** (p. **??**) coordinate increment
row	Where to decode the row index.
col	Where to decode the column index.

5.20.3.4 template<typename T , typename I , typename hI , typename mI > virtual ULI HilbertArray< T, I, hI, mI >::getFirstColumnIndex ( ) [inline], [virtual]

Gets the start-location of the first nonzero in this array.

Returns

The column-position of this location.

Implements **HilbertArrayInterface**< **T** > (p. **??**).

References HilbertArray< T, I, hI, mI >::curcol, HilbertArray< T, I, hI, mI >::currow, HilbertArray< T, I, hI, mI >::decode(), and HilbertArray< T, I, hI, mI >::start_coor.

5.20.3.5 template<typename T , typename I , typename hI , typename mI > virtual ULI HilbertArray< T, I, hI, mI >::getFirstRowIndex ( ) [inline], [virtual]

Gets the start-location of the first nonzero in this array.

Returns

The row-position of this location.

Implements **HilbertArrayInterface**< **T** > (p. **??**).

References HilbertArray< T, I, hI, mI >::curcol, HilbertArray< T, I, hI, mI >::currow, HilbertArray< T, I, hI, mI >::decode(), and HilbertArray< T, I, hI, mI >::start_coor.

5.20.3.6 template<typename T , typename I , typename hI , typename mI > virtual void HilbertArray< T, I, hI, mI >::moveToNext (const T *__restrict__ & *x*, T *__restrict__ & *y*, const T & *v*) [inline], [virtual]

Moves this interface and the two given vectors to the next position.

Parameters

x	Points to the start of the input vector.
y	Points to the start of the output vector.
v	Non-zero value to use.

Implements **HilbertArrayInterface**< **T** > (p. **??**).

References HilbertArray< T, I, hI, mI >::curcol, HilbertArray< T, I, hI, mI >::curcoor, HilbertArray< T, I, hI, mI >::curpos, HilbertArray< T, I, hI, mI >::currow, and HilbertArray< T, I, hI, mI >::decode().

5.20.3.7 template<typename T , typename I , typename hI , typename mI > virtual void HilbertArray< T, I, hI, mI >::moveToStart (const T *__restrict__ & *x*, T *__restrict__ & *y*, const T & *v*) [inline], [virtual]

Moves this interface to the start location and perform a single multiply-add there.

Parameters

x	Points to the start of the input vector.
y	Points to the start of the output vector.
v	Nonzero value to use.

Implements **HilbertArrayInterface< T >** (p. **??**).

References HilbertArray< T, I, hI, ml >::array, HilbertArray< T, I, hI, ml >::curcol, HilbertArray< T, I, hI, ml >::curcoor, HilbertArray< T, I, hI, ml >::curpos, HilbertArray< T, I, hI, ml >::currow, HilbertArray< T, I, hI, ml >::decode(), and HilbertArray< T, I, hI, ml >::start_coor.

5.20.3.8 template<typename T , typename I , typename hI , typename ml > virtual void HilbertArray< T, I, hI, ml >::zax (const T *__restrict__ const & *x*, T *__restrict__ const & *y*, const T *__restrict__ & *v*, const T *__restrict__ const & *v_end*) [inline], [virtual]

Flat implementation of the zax.

Might perform better than using the moveToStart and moveToNext functions from an outside class.

Parameters

x	Pointer to the input vector.
y	Pointer to the output vector.
v	Pointer to the nonzero values array. Warning: the pointer position will be altered!
v_end	End-location of the nonzero values array (equal to v+nnz).

Implements **HilbertArrayInterface< T >** (p. **??**).

References HilbertArray< T, I, hI, ml >::array, HilbertArray< T, I, hI, ml >::curcol, HilbertArray< T, I, hI, ml >::curcoor, HilbertArray< T, I, hI, ml >::curpos, HilbertArray< T, I, hI, ml >::currow, HilbertArray< T, I, hI, ml >::decode(), Matrix2HilbertCoordinates::IntegerToHilbert(), and HilbertArray< T, I, hI, ml >::start_coor.

5.20.3.9 template<typename T , typename I , typename hI , typename ml > virtual void HilbertArray< T, I, hI, ml >::zxa (const T *__restrict__ const & *x*, T *__restrict__ const & *y*, const T *__restrict__ & *v*, const T *__restrict__ const & *v_end*) [inline], [virtual]

Flat implementation of the zax.

Might perform better than using the moveToStart and moveToNext functions from an outside class.

Parameters

x	Pointer to the input vector.
y	Pointer to the output vector.
v	Pointer to the nonzero values array. Warning: the pointer position will be altered!
v_end	End-location of the nonzero values array (equal to v+nnz).

Implements **HilbertArrayInterface< T >** (p. **??**).

References HilbertArray< T, I, hI, ml >::array, HilbertArray< T, I, hI, ml >::curcol, HilbertArray< T, I, hI, ml >::curcoor, HilbertArray< T, I, hI, ml >::curpos, HilbertArray< T, I, hI, ml >::currow, HilbertArray< T, I, hI, ml >::decode(), and HilbertArray< T, I, hI, ml >::start_coor.

5.20.4 Member Data Documentation

5.20.4.1 template<typename T , typename I , typename hI , typename ml > I* HilbertArray< T, I, hI, ml >::array [protected]

Index storage array.

Referenced by HilbertArray< T, I, hI, mI >::HilbertArray(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), HilbertArray< T, I, hI, mI >::zxa(), and HilbertArray< T, I, hI, mI >::~HilbertArray().

5.20.4.2 template<typename T , typename I , typename hI , typename mI > size_t HilbertArray< T, I, hI, mI >::bytes [protected]

Keeps track of memory use.

Referenced by HilbertArray< T, I, hI, mI >::bytesUsed(), and HilbertArray< T, I, hI, mI >::HilbertArray().

5.20.4.3 template<typename T , typename I , typename hI , typename mI > mI HilbertArray< T, I, hI, mI >::curcol [protected]

Current column position.

Referenced by HilbertArray< T, I, hI, mI >::getFirstColumnIndex(), HilbertArray< T, I, hI, mI >::getFirstRowIndex(), HilbertArray< T, I, hI, mI >::moveToNext(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

5.20.4.4 template<typename T , typename I , typename hI , typename mI > hI HilbertArray< T, I, hI, mI >::curcoor [protected]

Current **Hilbert** (p. **??**) coordinate.

Referenced by HilbertArray< T, I, hI, mI >::moveToNext(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

5.20.4.5 template<typename T , typename I , typename hI , typename mI > I* __restrict__ HilbertArray< T, I, hI, mI >::curpos [protected]

Current position in the array.

Referenced by HilbertArray< T, I, hI, mI >::HilbertArray(), HilbertArray< T, I, hI, mI >::moveToNext(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

5.20.4.6 template<typename T , typename I , typename hI , typename mI > mI HilbertArray< T, I, hI, mI >::currow [protected]

Current row position.

Referenced by HilbertArray< T, I, hI, mI >::getFirstColumnIndex(), HilbertArray< T, I, hI, mI >::getFirstRowIndex(), HilbertArray< T, I, hI, mI >::moveToNext(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

5.20.4.7 template<typename T , typename I , typename hI , typename mI > hI HilbertArray< T, I, hI, mI >::start_coor [protected]

Hilbert (p. **??**) start coordinate.

Referenced by HilbertArray< T, I, hI, mI >::getFirstColumnIndex(), HilbertArray< T, I, hI, mI >::getFirstRowIndex(), HilbertArray< T, I, hI, mI >::HilbertArray(), HilbertArray< T, I, hI, mI >::moveToStart(), HilbertArray< T, I, hI, mI >::zax(), and HilbertArray< T, I, hI, mI >::zxa().

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

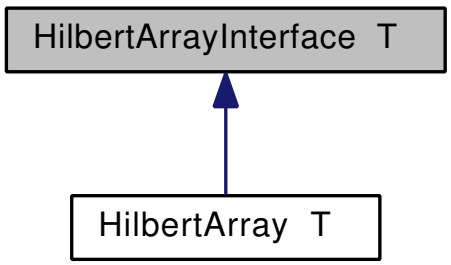
- HilbertArray.hpp

5.21 HilbertArrayInterface< T > Class Template Reference

Class providing an interface to an efficient storage of a 1D array of **Hilbert** (p. **??**) coordinate increments.

`#include <HilbertArray.hpp>`

Inheritance diagram for HilbertArrayInterface< T >:



Public Member Functions

- virtual void **zax** (const T *__restrict__ const &x, T *__restrict__ const &y, const T *__restrict__ &v, const T *__restrict__ const &v_end)=0

  Does a zax-operation using the moveToStart and moveToNext functionalities.
- virtual void **zxa** (const T *__restrict__ const &x, T *__restrict__ const &y, const T *__restrict__ &v, const T *__restrict__ const &v_end)=0

  Does a zxa-operation using the moveToStart and moveToNext functionalities.
- virtual size_t **bytesUsed** ()=0

  Gets the amount of storage used.
- virtual ULI **getFirstColumnIndex** ()=0

  Gets the first column index.
- virtual ULI **getFirstRowIndex** ()=0

  Gets the first row index.
- virtual void **moveToStart** (const T *__restrict__ &x, T *__restrict__ &y, const T &v)=0

  Moves this interface and the two given vectors to the start location.
- virtual void **moveToNext** (const T *__restrict__ &x, T *__restrict__ &y, const T &v)=0

  Moves this interface and the two given vectors to the next position.
- virtual ~**HilbertArrayInterface** ()

  Base deconstructor.

5.21.1 Detailed Description

template<typename T>class HilbertArrayInterface< T >

Class providing an interface to an efficient storage of a 1D array of **Hilbert** (p. **??**) coordinate increments.

Includes functionality to translate increments into 2D movements applied on two vectors of a template type T. This interface is best viewed as an iterator.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 template<typename T > virtual HilbertArrayInterface< T >::~HilbertArrayInterface ( ) [inline], [virtual]

Base deconstructor.

5.21.3 Member Function Documentation

5.21.3.1 template<typename T > virtual size_t HilbertArrayInterface< T >::bytesUsed ( ) [pure virtual]

Gets the amount of storage used.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.2 template<typename T > virtual ULI HilbertArrayInterface< T >::getFirstColumnIndex ( ) [pure virtual]

Gets the first column index.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.3 template<typename T > virtual ULI HilbertArrayInterface< T >::getFirstRowIndex ( ) [pure virtual]

Gets the first row index.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.4 template<typename T > virtual void HilbertArrayInterface< T >::moveToNext (const T *__restrict__ & *x*, T *__restrict__ & *y*, const T & *v*) [pure virtual]

Moves this interface and the two given vectors to the next position.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.5 template<typename T > virtual void HilbertArrayInterface< T >::moveToStart (const T *__restrict__ & *x*, T *__restrict__ & *y*, const T & *v*) [pure virtual]

Moves this interface and the two given vectors to the start location.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.6 template<typename T > virtual void HilbertArrayInterface< T >::zax (const T *__restrict__ const & *x*, T *__restrict__ const & *y*, const T *__restrict__ & *v*, const T *__restrict__ const & *v_end*) [pure virtual]

Does a zax-operation using the moveToStart and moveToNext functionalities.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

5.21.3.7 template<typename T > virtual void HilbertArrayInterface< T >::zxa (const T *__restrict__ const & *x*, T *__restrict__ const & *y*, const T *__restrict__ & *v*, const T *__restrict__ const & *v_end*) [pure virtual]

Does a zxa-operation using the moveToStart and moveToNext functionalities.

Implemented in **HilbertArray< T, I, hI, mI >** (p. **??**).

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

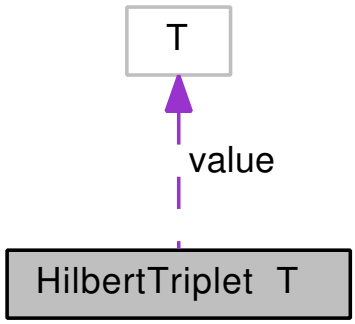
- HilbertArray.hpp

5.22 HilbertTriplet< T > Class Template Reference

Hilbert-coordinate-aware triplet.

`#include <HilbertTriplet.hpp>`

Collaboration diagram for HilbertTriplet< T >:



Public Member Functions

- unsigned long int **i** () const
- unsigned long int **j** () const
- **HilbertTriplet** (unsigned long int **i**, unsigned long int **j**, T val)

  Base constructor.
- **HilbertTriplet** ()

  Base constructor.
- void **calculateHilbertCoordinate** ()

  *Calculates the full **Hilbert** (p. **??**) coordinate.*
- void **getHilbertCoordinate** (size_t &h1, size_t &h2)

  *Gets the **Hilbert** (p. **??**) coordinates.*
- size_t **getMostSignificantHilbertBits** ()
- size_t **getLeastSignificantHilbertBits** ()

Static Public Member Functions

- static void **save** (std::string fn, **HilbertTriplet**< T > *toWrite, const unsigned long int m, const unsigned long int n, const size_t s)

  *Saves an array of **Hilbert** (p. **??**) triplets to a file, in binary format.*
- static void **save** (std::string fn, std::vector< **HilbertTriplet**< T > > &toWrite, const unsigned long int m, const unsigned long int n)

  *Saves a std::vector of **Hilbert** (p. **??**) triplets to a file, in binary format.*

Public Attributes

- T **value**

  Value stored at this triplet.

Protected Attributes

- size_t **row**

  The row coordinate of this triplet.
- size_t **column**

  The column coordinate of this triplet.
- size_t **hilbert1**

*Most significant part of a 128-bits **Hilbert** (p. **??**) coordinate, for one-shot, non-iterative, calculation.*

- size_t **hilbert2**

  *Least significant part of a 128-bits **Hilbert** (p. **??**) coordinate, for one-shot, non-iterative, calculation.*

5.22.1 Detailed Description

template<typename T>class HilbertTriplet< T >

Hilbert-coordinate-aware triplet.

Can save vectors of HilbertTriplets to binary.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 template<typename T> HilbertTriplet< T >::HilbertTriplet (unsigned long int *i*, unsigned long int *j*, T *val*) [inline]

Base constructor.

Parameters

i	row index.
j	column index.
val	nonzero value.

5.22.2.2 template<typename T> HilbertTriplet< T >::HilbertTriplet ( ) [inline]

Base constructor.

Sets all values to zero.

5.22.3 Member Function Documentation

5.22.3.1 template<typename T> void HilbertTriplet< T >::getHilbertCoordinate (size_t & *h1*, size_t & *h2*) [inline]

Gets the **Hilbert** (p. **??**) coordinates.

Does not check if **calculateHilbertCoordinate()** (p. **??**) was called first, otherwise (0,0) will be returned. Note that the **Hilbert** (p. **??**) coordinate is a 1D 128-bits unsigned integer.

Parameters

h1	The first (most significant) 64-bits of the **Hilbert** (p. **??**) coordinate
h2	the remainder of the **Hilbert** (p. **??**) coordinate.

References HilbertTriplet< T >::hilbert1, and HilbertTriplet< T >::hilbert2.

5.22.3.2 template<typename T> size_t HilbertTriplet< T >::getLeastSignificantHilbertBits ( ) [inline]

Returns

h2 of **HilbertTriplet<T>::getHilbertCoordinate()** (p. **??**)

References HilbertTriplet< T >::hilbert2.

Referenced by HilbertTripletCompare< T >::operator()().

5.22.3.3 template<typename T> size_t HilbertTriplet< T >::getMostSignificantHilbertBits ( ) [inline]

Returns

h1 of **HilbertTriplet<T>::getHilbertCoordinate()** (p. **??**)

References HilbertTriplet< T >::hilbert1.

Referenced by HilbertTripletCompare< T >::operator()().

5.22.3.4 template<typename T> unsigned long int HilbertTriplet< T >::i ( ) const [inline]

Returns

Row index of this triplet.

References HilbertTriplet< T >::row.

Referenced by HTS< T >::cmp(), and HilbertTriplet< T >::save().

5.22.3.5 template<typename T> unsigned long int HilbertTriplet< T >::j ( ) const [inline]

Returns

Column index of this triplet.

References HilbertTriplet< T >::column.

Referenced by HTS< T >::cmp(), and HilbertTriplet< T >::save().

5.22.3.6 template<typename T> static void HilbertTriplet< T >::save (std::string *fn,* HilbertTriplet< T > * *toWrite,* const unsigned long int *m,* const unsigned long int *n,* const size_t *s*) [inline], [static]

Saves an array of **Hilbert** (p. **??**) triplets to a file, in binary format.

Does NOT save the **Hilbert** (p. **??**) coordinate! (For reading-in the written file, use the regular **Triplet** (p. **??**) scheme)

Parameters

fn	Filename to save to (overwrite mode).
toWrite	Array of **Hilbert** (p. **??**) triplets to write.
m	Total number of rows in the matrix.
n	Total number of columns in the matrix.
s	Size of the array toWrite.

References HilbertTriplet< T >::i(), HilbertTriplet< T >::j(), and HilbertTriplet< T >::value.

Referenced by HilbertTriplet< T >::save(), Hilbert< T >::saveBinary(), and HTS< T >::saveBinary().

5.22.3.7 template<typename T> static void HilbertTriplet< T >::save (std::string *fn,* std::vector< HilbertTriplet< T > > & *toWrite,* const unsigned long int *m,* const unsigned long int *n*) [inline], [static]

Saves a std::vector of **Hilbert** (p. **??**) triplets to a file, in binary format.

Does NOT save the **Hilbert** (p. **??**) coordinate! (For reading-in the written file, use the regular **Triplet** (p. **??**) scheme)

Parameters

fn	Filename to save to (overwrite mode).
toWrite	Vector of **Hilbert** (p. **??**) triplets to write.
m	Total number of rows in the matrix.
n	Total number of columns in the matrix.

References HilbertTriplet< T >::save().

5.22.4 Member Data Documentation

5.22.4.1 template<typename T> size_t HilbertTriplet< T >::column [protected]

The column coordinate of this triplet.

Referenced by HilbertTriplet< T >::calculateHilbertCoordinate(), and HilbertTriplet< T >::j().

5.22.4.2 template<typename T> size_t HilbertTriplet< T >::hilbert1 [protected]

Most significant part of a 128-bits **Hilbert** (p. **??**) coordinate, for one-shot, non-iterative, calculation.

Referenced by HilbertTriplet< T >::calculateHilbertCoordinate(), HilbertTriplet< T >::getHilbertCoordinate(), and HilbertTriplet< T >::getMostSignificantHilbertBits().

5.22.4.3 template<typename T> size_t HilbertTriplet< T >::hilbert2 [protected]

Least significant part of a 128-bits **Hilbert** (p. **??**) coordinate, for one-shot, non-iterative, calculation.

Referenced by HilbertTriplet< T >::calculateHilbertCoordinate(), HilbertTriplet< T >::getHilbertCoordinate(), and HilbertTriplet< T >::getLeastSignificantHilbertBits().

5.22.4.4 template<typename T> size_t HilbertTriplet< T >::row [protected]

The row coordinate of this triplet.

Referenced by HilbertTriplet< T >::calculateHilbertCoordinate(), and HilbertTriplet< T >::i().

5.22.4.5 template<typename T> T HilbertTriplet< T >::value

Value stored at this triplet.

Referenced by HilbertTriplet< T >::save().

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

- HilbertTriplet.hpp

5.23 HilbertTripletCompare< T > Class Template Reference

Class-comparator of **HilbertTriplet** (p. **??**).

`#include <HilbertTripletCompare.hpp>`

Public Member Functions

- bool **operator()** (**HilbertTriplet**< T > i, **HilbertTriplet**< T > j)

  *Compares two **Hilbert** (p. **??**) triplets.*

5.23.1 Detailed Description

template<typename T>class HilbertTripletCompare< T >

Class-comparator of **HilbertTriplet** (p. **??**).

5.23.2 Member Function Documentation

5.23.2.1 template<typename T> bool HilbertTripletCompare< T >::operator() (HilbertTriplet< T > *i*, HilbertTriplet< T > *j*) [inline]

Compares two **Hilbert** (p. **??**) triplets.

Parameters

i	The first **Hilbert** (p. **??**) triplet.
j	The second **Hilbert** (p. **??**) triplet.

Returns

Whether the **Hilbert** (p. **??**) index of i is smaller than the one of j.

References HilbertTriplet< T >::getLeastSignificantHilbertBits(), and HilbertTriplet< T >::getMostSignificantHilbertBits().

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

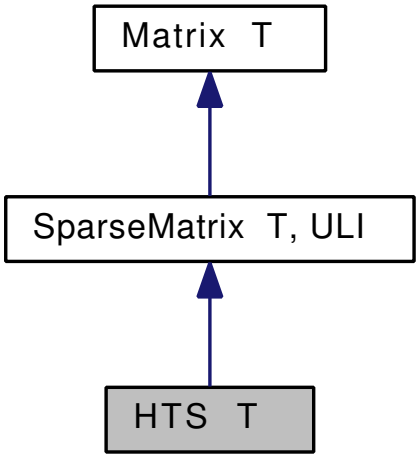
- HilbertTripletCompare.hpp

5.24 HTS< T > Class Template Reference

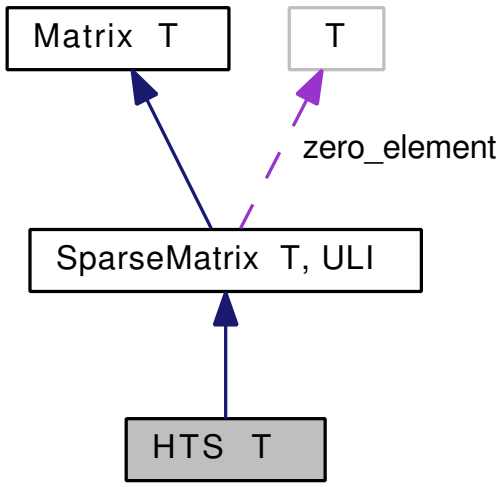
The **Hilbert** (p. **??**) triplet scheme.

`#include <HTS.hpp>`

Inheritance diagram for HTS< T >:



Collaboration diagram for HTS< T >:



Public Member Functions

- **HTS** ()

  Base constructor.
- **HTS** (std::string file, T zero=0)

  Base constructor.
- **HTS** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *x, T *z)

  Calculates z=xA.
- virtual void **zax** (const T *x, T *z)

  Calculates z=Ax.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.
- void **saveBinary** (const std::string fn)

  *Saves the current **HTS** (p. **??**).*
- void **loadBinary** (const std::string fn)

  *Loads from binary triplets, assumes **Hilbert** (p. **??**) ordering already done.*

Protected Member Functions

- bool **cmp** (**HilbertTriplet**< T > &left, **HilbertTriplet**< T > &right)

  HilbertCoordinate comparison function.
- unsigned long int **find** (**HilbertTriplet**< T > &x, ULI &left, ULI &right)

  Binary search for finding a given triplet in a given range.

Protected Attributes

- ULI **minexp**

  Minimum number of expansions.
- std::vector< **HilbertTriplet**< T > > **ds**

  Vector storing the non-zeros and their locations.

Additional Inherited Members

5.24.1 Detailed Description

template<typename T>class HTS< T >

The **Hilbert** (p. **??**) triplet scheme.

In effect similar to the triplet scheme (**TS** (p. **??**)), but uses **Hilbert** (p. **??**) coordinates to determine the order of storage.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 template<typename T> HTS< T >::HTS ( ) [inline]

Base constructor.

5.24.2.2 template<typename T> HTS< T >::HTS (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

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

Base constructor.

Warning: the zero parameter is currently NOT USED!

Parameters

input	Raw input of normal triplets.
m	Total number of rows.
n	Total number of columns.
zero	What elements is considered to-be zero.

References HTS< T >::load().

5.24.3 Member Function Documentation

5.24.3.1 template<typename T> virtual size_t HTS< T >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References SparseMatrix< T, ULI >::nnz.

5.24.3.2 template<typename T> bool HTS< T >::cmp (HilbertTriplet< T > & *left,* HilbertTriplet< T > & *right*) [inline],[protected]

HilbertCoordinate comparison function.

References HilbertTriplet< T >::i(), HilbertTriplet< T >::j(), SparseMatrix< T, ULI >::noc, and SparseMatrix< T, ULI >::nor.

Referenced by HTS< T >::find().

5.24.3.3 template<typename T> unsigned long int HTS< T >::find (HilbertTriplet< T > & *x,* ULI & *left,* ULI & *right*) [inline],[protected]

Binary search for finding a given triplet in a given range.

Parameters

x	triplet to-be found.
left	Left bound of the range to search in.
right	Right bound of the range to search in.

Returns

Index of the triplet searched.

References HTS< T >::cmp(), HTS< T >::ds, and HTS< T >::minexp.

5.24.3.4 template<typename T> virtual void HTS< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References HTS< T >::ds.

5.24.3.5 template<typename T> virtual void HTS< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline],[virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References HTS< T >::ds, SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and SparseMatrix< T, ULI >::zero_element.

Referenced by HTS< T >::HTS().

5.24.3.6 template<typename T> void HTS< T >::saveBinary (const std::string *fn*) [inline]

Saves the current **HTS** (p. **??**).

Parameters

fn	Filename to save to.

See Also

HilbertTriplet< T >::save (p. **??**) ‘

References HTS< T >::ds, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and HilbertTriplet< T >::save().

5.24.3.7 template<typename T> virtual void HTS< T >::zax (const T * x, T * z) [inline], [virtual]

Calculates z=Ax.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References HTS< T >::ds, and SparseMatrix< T, ULI >::nnz.

5.24.3.8 template<typename T> virtual void HTS< T >::zxa (const T * x, T * z) [inline], [virtual]

Calculates z=xA.

z is *not* set to 0 at the start of this method!

Parameters

x	The (initialised) input vector.
z	The (initialised) output vector.

References HTS< T >::ds, and SparseMatrix< T, ULI >::nnz.

5.24.4 Member Data Documentation

5.24.4.1 template<typename T> std::vector< HilbertTriplet< T > > HTS< T >::ds [protected]

Vector storing the non-zeros and their locations.

Referenced by HTS< T >::find(), HTS< T >::getFirstIndexPair(), HTS< T >::load(), HTS< T >::loadBinary(), HTS< T >::saveBinary(), HTS< T >::zax(), and HTS< T >::zxa().

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

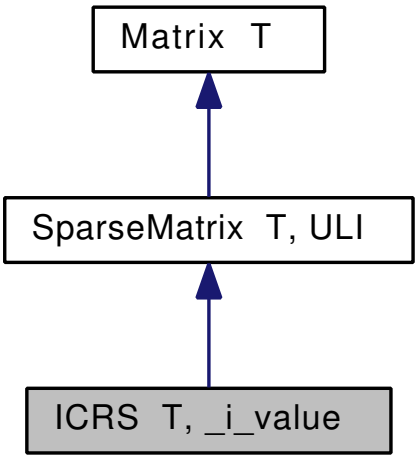
- HTS.hpp

5.25 ICRS< T, _i_value > Class Template Reference

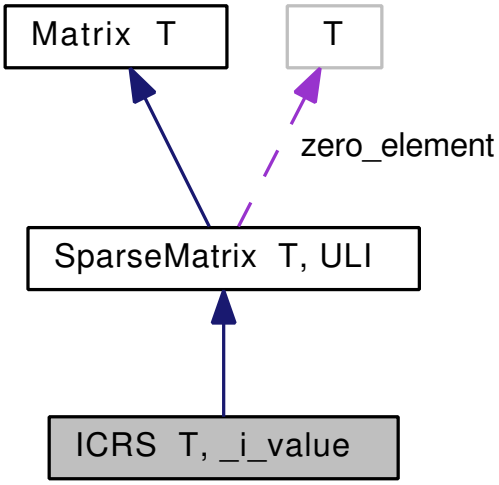
The *incremental* compressed row storage sparse matrix data structure.

`#include <ICRS.hpp>`

Inheritance diagram for ICRS< T, _i_value >:



Collaboration diagram for ICRS< T, _i_value >:



Public Member Functions

- **ICRS** ()

  Base constructor.
- **ICRS** (std::string file, T zero=0)

  Base constructor.
- **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.
- **ICRS** (**ICRS**< T > &toCopy)

  Copy constructor.
- **ICRS** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero=0)

  *Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.*
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- void **getStartingPos** (ULI &row_start, ULI &column_start)

Gets starting position (first nonzero location)

- void **setStartingPos** (const ULI row_start, const ULI column_start)

  Sets starting position of matrix multiplication.

- virtual void **zxa** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=xA function.

- virtual void **zax** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=Ax function.

- template<size_t k>
  void **ZaX** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- template<size_t k>
  void **ZXa** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- ~**ICRS** ()

  Base deconstructor.

- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Static Public Member Functions

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

  Comparison function used for sorting input data.

Static Public Attributes

- static const size_t **fillIn** = 0

  *Fill-in field for interoperability with **vecBICRS*** (p. **??**)*.*

Protected Attributes

- T * **ds**

  Array containing the actual nnz non-zeros.

- ULI **r_start**

  Start position, row.

- ULI **c_start**

  Start position, column.

- _i_value * **c_ind**

  Array containing the column jumps.

- _i_value * **r_ind**

  Array containing the row jumps.

- size_t **bytes**

  Remembers the number of bytes allocated.

Additional Inherited Members

5.25.1 Detailed Description

template<typename T, typename _i_value = ULI>class ICRS< T, _i_value >

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

5.25.2 Constructor & Destructor Documentation

5.25.2.1 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::ICRS ( ) [inline]

Base constructor.

5.25.2.2 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::ICRS (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.25.2.3 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::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 **ICRS** (p. **??**) 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 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.

References ICRS< T, _i_value >::bytes, ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::ds, SparseMatrix< T, ULI >::nnz, and ICRS< T, _i_value >::r_ind.

5.25.2.4 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::ICRS (ICRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	Reference to the **ICRS** (p. **??**) datastructure to copy.

References ICRS< T, _i_value >::bytes, ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, ICRS< T, _i_value >::r_ind, ICRS< T, _i_value >::r_start, and SparseMatrix< T, ULI >::zero_element.

5.25.2.5 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::ICRS (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero* = 0) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.

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

Parameters

input	The input collection of triplets (i,j,val).
m	The number of rows of the input matrix.
n	The number of columns of the input matrix.
zero	Which element is considered zero.

References ICRS< T, _i_value >::load().

5.25.2.6 template<typename T, typename _i_value = ULI> ICRS< T, _i_value >::~ICRS ( ) [inline]

Base deconstructor.

References ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::ds, and ICRS< T, _i_value >::r_ind.

5.25.3 Member Function Documentation

5.25.3.1 template<typename T, typename _i_value = ULI> virtual size_t ICRS< T, _i_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References ICRS< T, _i_value >::bytes.

5.25.3.2 template<typename T, typename _i_value = ULI> static int ICRS< T, _i_value >::compareTriplets (const void * *left,* const void * *right*) [inline], [static]

Comparison function used for sorting input data.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by ICRS< T, _i_value >::load().

5.25.3.3 template<typename T, typename _i_value = ULI> virtual void ICRS< T, _i_value >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References ICRS< T, _i_value >::c_start, and ICRS< T, _i_value >::r_start.

5.25.3.4 template<typename T, typename _i_value = ULI> virtual void ICRS< T, _i_value >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References ICRS< T, _i_value >::bytes, ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::compareTriplets(), ICRS< T, _i_value >::ds, Triplet< T >::i(), Triplet< T >::j(), SparseMatrix< T, ULI

>::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, ICRS< T, _i_value >::r_ind, ICRS< T, _i_value >::r_start, Triplet< T >::value, and SparseMatrix< T, ULI >::zero_element.

Referenced by ICRS< T, _i_value >::ICRS().

5.25.3.5 template<typename T, typename _i_value = ULI> void ICRS< T, _i_value >::setStartingPos (const ULI *row_start,* const ULI *column_start*) [inline]

Sets starting position of matrix multiplication.

(Useful for example when the input/output vectors will be shifted before passed on to this class' zax method.)

Parameters

row_start	New row start location
column_start	New column start location

References ICRS< T, _i_value >::c_start, and ICRS< T, _i_value >::r_start.

5.25.3.6 template<typename T, typename _i_value = ULI> virtual void ICRS< T, _i_value >::zax (const T *__restrict__ *pDataX,* T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=Ax function.

Adapted from the master thesis of Joris Koster, Utrecht University.

Parameters

pDataX	Pointer to array x to multiply by the current matrix (Ax).
pDataZ	Pointer to result array. Must be pre-allocated and its elements set to zero for correct results.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, ICRS< T, _i_value >::r_ind, and ICRS< T, _i_value >::r_start.

5.25.3.7 template<typename T, typename _i_value = ULI> template<size_t k> void ICRS< T, _i_value >::ZaX (const T *__restrict__ const *__restrict__ const *X,* T *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZaX (p. **??**)

References ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::ds, ICRS< T, _i_value >::r_ind, and ICRS< T, _i_value >::r_start.

5.25.3.8 template<typename T, typename _i_value = ULI> virtual void ICRS< T, _i_value >::zxa (const T *__restrict__ *pDataX,* T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=xA function.

Adapted from the master thesis of Joris Koster, Utrecht University.

Parameters

pDataX	Pointer to array x to multiply by the current matrix (Ax).
pDataZ	Pointer to result array. Must be pre-allocated and its elements set to zero for correct results.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, ICRS< T, _i_value >::r_ind, and ICRS< T, _i_value >::r_start.

5.25.3.9 template<typename T, typename _i_value = ULI> template<size_t k> void ICRS< T, _i_value >::ZXa (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z) [inline]

See Also

Matrix::ZXa (p. **??**)

References ICRS< T, _i_value >::c_ind, ICRS< T, _i_value >::c_start, ICRS< T, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, ICRS< T, _i_value >::r_ind, and ICRS< T, _i_value >::r_start.

5.25.4 Member Data Documentation

5.25.4.1 template<typename T, typename _i_value = ULI> size_t ICRS< T, _i_value >::bytes [protected]

Remembers the number of bytes allocated.

Referenced by ICRS< T, _i_value >::bytesUsed(), ICRS< T, _i_value >::ICRS(), and ICRS< T, _i_value >::load().

5.25.4.2 template<typename T, typename _i_value = ULI> _i_value* ICRS< T, _i_value >::c_ind [protected]

Array containing the column jumps.

Referenced by ICRS< T, _i_value >::ICRS(), ICRS< T, _i_value >::load(), ICRS< T, _i_value >::zax(), ICRS< T, _i_value >::ZaX(), ICRS< T, _i_value >::zxa(), ICRS< T, _i_value >::ZXa(), and ICRS< T, _i_value >::~ICRS().

5.25.4.3 template<typename T, typename _i_value = ULI> T* ICRS< T, _i_value >::ds [protected]

Array containing the actual nnz non-zeros.

Referenced by ICRS< T, _i_value >::ICRS(), ICRS< T, _i_value >::load(), ICRS< T, _i_value >::zax(), ICRS< T, _i_value >::ZaX(), ICRS< T, _i_value >::zxa(), ICRS< T, _i_value >::ZXa(), and ICRS< T, _i_value >::~ICRS().

5.25.4.4 template<typename T, typename _i_value = ULI> const size_t ICRS< T, _i_value >::fillIn = 0 [static]

Fill-in field for interoperability with **vecBICRS** (p. **??**).

Fill-in is always 0 for regular **ICRS** (p. **??**).

5.25.4.5 template<typename T, typename _i_value = ULI> _i_value* ICRS< T, _i_value >::r_ind [protected]

Array containing the row jumps.

Referenced by ICRS< T, _i_value >::ICRS(), ICRS< T, _i_value >::load(), ICRS< T, _i_value >::zax(), ICRS< T, _i_value >::ZaX(), ICRS< T, _i_value >::zxa(), ICRS< T, _i_value >::ZXa(), and ICRS< T, _i_value >::~ICRS().

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

- **ICRS.hpp**

5.26 MachineInfo Class Reference

Singleton class to get info on the current system.

`#include <MachineInfo.hpp>`

Public Member Functions

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

  The number of available cores.

Static Public Member Functions

- static const **MachineInfo** & **getInstance** ()

  Gets a singleton instance.

Static Protected Member Functions

- static void **load** ()

  Intialises the singleton instance.

Protected Attributes

- unsigned long int **P**

  The number of available cores on this machine.

5.26.1 Detailed Description

Singleton class to get info on the current system.

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

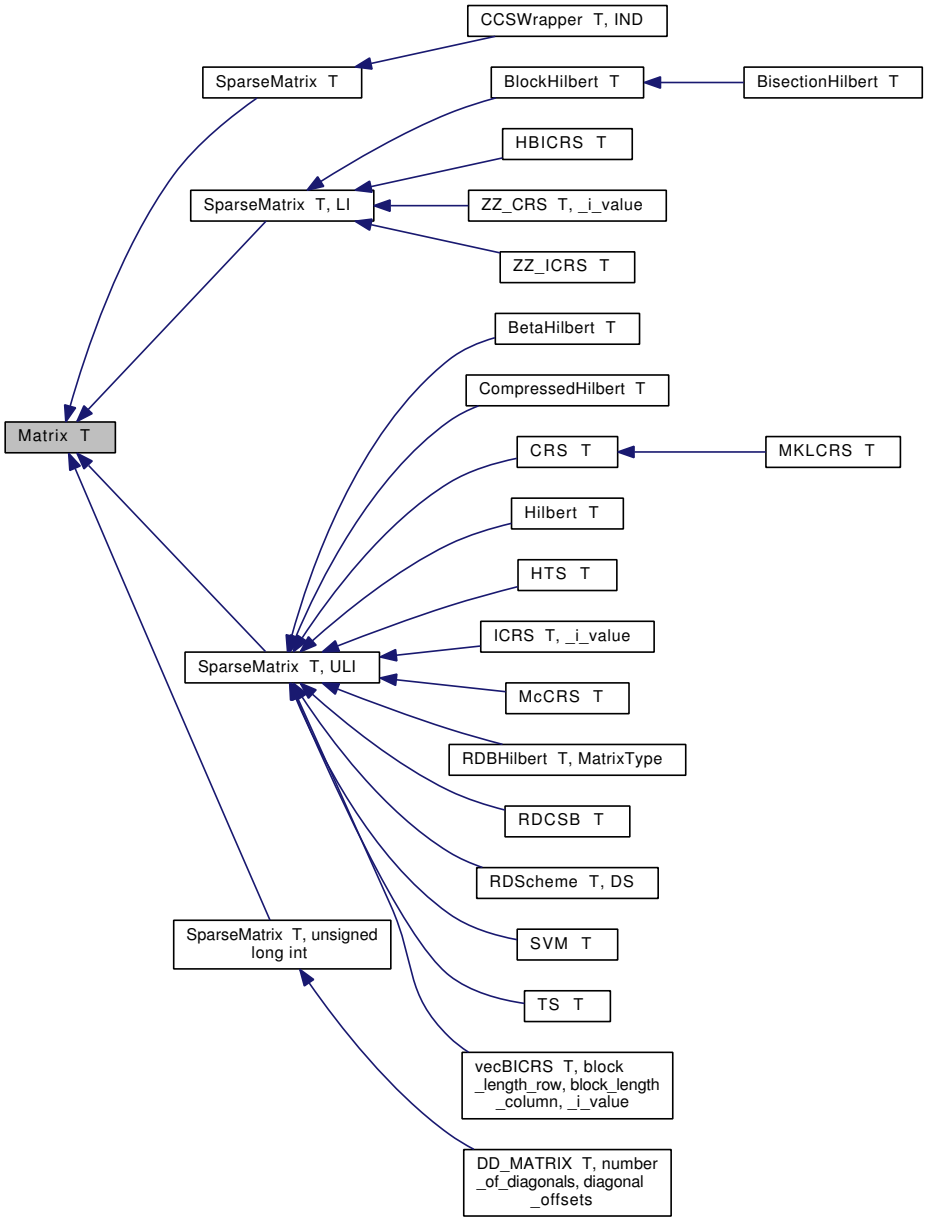
- MachineInfo.hpp
- MachineInfo.cpp

5.27 Matrix< T > Class Template Reference

Defines operations common to all matrices, which are implemented in this library.

`#include <Matrix.hpp>`

Inheritance diagram for Matrix< T >:



Public Member Functions

- **Matrix** ()

  Base constructor.
- virtual ~**Matrix** ()

  Base deconstructor.
- virtual unsigned long int **m** ()=0
- virtual unsigned long int **n** ()=0
- virtual unsigned long int **nzs** ()
- virtual T * **mv** (const T *x)=0

  Calculates z=Ax (where A is this matrix).
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)=0

  In-place z=Ax function.

- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z, const size_t k, const clockid_t clock_id=0, double *elapsed_time=NULL)

  Wrapper function to call the zax kernel multiple times successively, while timing the duration of the operation.

- template<size_t k>
  void **ZaX** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)

  In-place Z=AX function, where A is m x n, Z = m x k, and X is n x k.

- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)=0

  In-place z=xA function.

- template<size_t k>
  void **ZXa** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)

  In-place Z=XA function, where A is m x n, Z = k x n, and X is k x m.

- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z, const unsigned long int repeat, const clockid_t clock_id=0, double *elapsed_time=NULL)

  Wrapper function to call the zxa kernel multiple times successively, while timing the operation duration.

- virtual size_t **bytesUsed** ()=0

  Function to query the amount of storage required by this sparse matrix.

5.27.1 Detailed Description

template<typename T>class Matrix< T >

Defines operations common to all matrices, which are implemented in this library.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 template<typename T> Matrix< T >::Matrix ( ) [inline]

Base constructor.

5.27.2.2 template<typename T> virtual Matrix< T >::~Matrix ( ) [inline], [virtual]

Base deconstructor.

5.27.3 Member Function Documentation

5.27.3.1 template<typename T> virtual size_t Matrix< T >::bytesUsed ( ) [pure virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **BetaHilbert< T >** (p. **??**), **RDBHilbert< T, MatrixType >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **RDScheme< T, DS >** (p. **??**), **BlockHilbert< T >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **CRS< T >** (p. **??**), **McCRS< T >** (p. **??**), **SVM< T >** (p. **??**), **DD_MATRIX< T, number_of_diagonals, diagonal_offsets >** (p. **??**), **ZZ_CRS< T, _i_value >** (p. **??**), **HTS< T >** (p. **??**), **CompressedHilbert< T >** (p. **??**), **TS< T >** (p. **??**), **Hilbert< T >** (p. **??**), **CuHyb** (p. **??**), and **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

5.27.3.2 template<typename T> virtual unsigned long int Matrix< T >::m ( ) [pure virtual]

Returns

The number of rows.

Implemented in **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), and **SparseMatrix< double, size_t >** (p. **??**).

Referenced by Matrix< _t_value >::nzs().

5.27.3.3 template<typename T> virtual T* Matrix< T >::mv (const T * x) [pure virtual]

Calculates z=Ax (where A is this matrix).

Parameters

x	The input vector.

Returns

The output vector z.

See Also

Matrix::zax (p. **??**).

Implemented in **BetaHilbert< T >** (p. **??**), **RDBHilbert< T, MatrixType >** (p. **??**), **RDScheme< T, DS >** (p. **??**), **RDCSB< T >** (p. **??**), **McCRS< T >** (p. **??**), **MKLCRS< T >** (p. **??**), **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), **SparseMatrix< double, size_t >** (p. **??**), and **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

5.27.3.4 template<typename T> virtual unsigned long int Matrix< T >::n ( ) [pure virtual]

Returns

The number of columns.

Implemented in **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), **SparseMatrix< double, size_t >** (p. **??**), and **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

Referenced by Matrix< _t_value >::nzs().

5.27.3.5 template<typename T> virtual unsigned long int Matrix< T >::nzs ( ) [inline], [virtual]

Returns

The number of nonzeros.

Reimplemented in **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), **SparseMatrix< double, size_t >** (p. **??**), and **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

5.27.3.6 template<typename T> virtual void Matrix< T >::zax (const T *__restrict__ *x*, T *__restrict__ *z*) [pure virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **McCRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **SVM< T >** (p. **??**), **CRS< T >** (p. **??**), **MKLCRS< T >** (p. **??**), **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), **SparseMatrix< double, size_t >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), and **CuHyb** (p. **??**).

Referenced by Matrix< _t_value >::zax(), and Matrix< _t_value >::ZaX().

5.27.3.7 template<typename T> virtual void Matrix< T >::zax (const T *__restrict__ *x*, T *__restrict__ *z*, const size_t *k*, const clockid_t *clock_id* = 0, double * *elapsed_time* = NULL) [inline], [virtual]

Wrapper function to call the zax kernel multiple times successively, while timing the duration of the operation.

Essentially computes z=A^k. The timing functionalities are optional.

Parameters

x	The input vector.
z	The output vector.
k	How many times to repeat the z=Ax operation.
clock_id	The POSIX realtime clock ID (optional).
elapsed_time	To which double the time taken should be added (optional; set to NULL to disable timing).

See Also

Matrix::zax (p. **??**)

5.27.3.8 template<typename T> template<size_t k> void Matrix< T >::ZaX (const T *__restrict__ const *__restrict__ const *X*, T *__restrict__ const *__restrict__ const *Z*) [inline]

In-place Z=AX function, where A is m x n, Z = m x k, and X is n x k.

The default implementation is to perform k successive z=Ax operations. Efficient schemes should override this default implementation to achieve higher performance.

Template Parameters

k	How many linear maps to compute. This is a template parameter so that the compiler can optimise better.

Parameters

X	The k input vectors of length n each.
Z	The k output vectors of length m each; each is assumed to be pre-initialised.

See Also

Matrix::zax (p. **??**)

5.27.3.9 template<typename T> virtual void Matrix< T >::zxa (const T *__restrict__ *x*, T *__restrict__ *z*) [pure virtual]

In-place z=xA function.

Parameters

x	The input vector to left-multiply to the current matrix.
z	The output vector. Must be pre-allocated and initialised to zero for correct results.

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **McCRS< T >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **SVM< T >** (p. **??**), **CRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **MKLCRS< T >** (p. **??**), **SparseMatrix< T, IND >** (p. **??**), **SparseMatrix< _t_value, ULI >** (p. **??**), **SparseMatrix< _t_value, LI >** (p. **??**), **SparseMatrix< _t_value, _i_value >** (p. **??**), **SparseMatrix< T, LI >** (p. **??**), **SparseMatrix< T, unsigned long int >** (p. **??**), **SparseMatrix< T, ULI >** (p. **??**), **SparseMatrix< double, size_t >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), and **CuHyb** (p. **??**).

Referenced by Matrix< _t_value >::ZXa(), and Matrix< _t_value >::zxa().

5.27.3.10 template<typename T> template<size_t k> void Matrix< T >::ZXa (const T *__restrict__ const *__restrict__ const *X*, T *__restrict__ const *__restrict__ const *Z*) [inline]

In-place Z=XA function, where A is m x n, Z = k x n, and X is k x m.

Transposed variant of **Matrix::ZaX** (p. **??**), analogue to **Matrix::zxa** (p. **??**) being the transposed operation of **Matrix::zax** (p. **??**).

Template Parameters

k	How many linear maps to compute. This is a template parameter so that the compiler can optimise better.

Parameters

X	The k input vectors of length m each.
Z	The k output vectors of length n each.

See Also

Matrix::ZaX (p. **??**)
Matrix::zxa (p. **??**)

5.27.3.11 template<typename T> virtual void Matrix< T >::zxa (const T *__restrict__ *x*, T *__restrict__ *z*, const unsigned long int *repeat*, const clockid_t *clock_id* = 0, double * *elapsed_time* = NULL) [inline], [virtual]

Wrapper function to call the zxa kernel multiple times successively, while timing the operation duration.

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

- **Matrix.hpp**

5.28 Matrix2HilbertCoordinates Class Reference

Class which maps coordinates to 1D **Hilbert** (p. **??**) Coordinates.

`#include <Matrix2HilbertCoordinates.hpp>`

Static Public Member Functions

- static void **IntegerToHilbert** (const size_t i, const size_t j, size_t &h1, size_t &h2)

  New method, October 2010.

Static Protected Attributes

- static const unsigned char **BITWIDTH** = 8 ∗ sizeof(size_t)

  The amount of bits in a native data word.

5.28.1 Detailed Description

Class which maps coordinates to 1D **Hilbert** (p. **??**) Coordinates.

5.28.2 Member Function Documentation

5.28.2.1 void Matrix2HilbertCoordinates::IntegerToHilbert (const size_t *i,* const size_t *j,* size_t & *h1,* size_t & *h2*) [static]

New method, October 2010.

Maps any 2D coordinate (i,j), with i and j size_t values of length x bits (machine- dependent), to a 1D coordinate of length 2∗x bits. The 2∗x bits 1D coordinate is returned as two size_t values h1 and h2, where h1 stores the most significant part of the 1D coordinate. If both i and j can be stored in half or less the amount of bits of a size_t integer, then h1 will be 0.

Parameters

i	A size_t integer value in one dimension
j	A size_t integer value in the other dimension
h1	First part of the 1D **Hilbert** (p. **??**) coordinate, unsigned integer format (most significant, first 64 bits)
h2	Second part of the 1D **Hilbert** (p. **??**) coordinate (least significant, last 64 bits)

Maps any 2D coordinate (i,j), with i and j 64-bits unsigned integers, to a 1D 128-bits unsigned integer.

Parameters

i	A 64-bits unsigned integer value in one dimension
j	A 64-bits unsigned integer value in the other dimension
h1	First part of the 128-bit **Hilbert** (p. **??**) coordinate, unsigned integer format (most significant, first 64 bits)
h2	Second part of the 128-bit **Hilbert** (p. **??**) coordinate (least significant, last 64 bits)

References BITWIDTH.

Referenced by HilbertTriplet< T >::calculateHilbertCoordinate(), CompressedHilbert< T >::load(), BetaHilbert< T >::thread(), RDBHilbert< T, MatrixType >::thread(), and HilbertArray< T, I, hI, mI >::zax().

5.28.3 Member Data Documentation

5.28.3.1 const unsigned char Matrix2HilbertCoordinates::BITWIDTH = 8 ∗ sizeof(size_t) [static],[protected]

The amount of bits in a native data word.

Referenced by IntegerToHilbert().

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

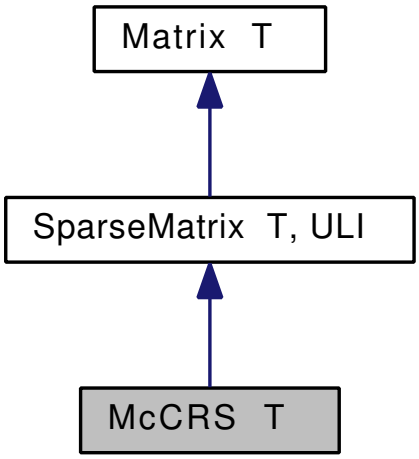
- Matrix2HilbertCoordinates.hpp
- Matrix2HilbertCoordinates.cpp

5.29 McCRS< T > Class Template Reference

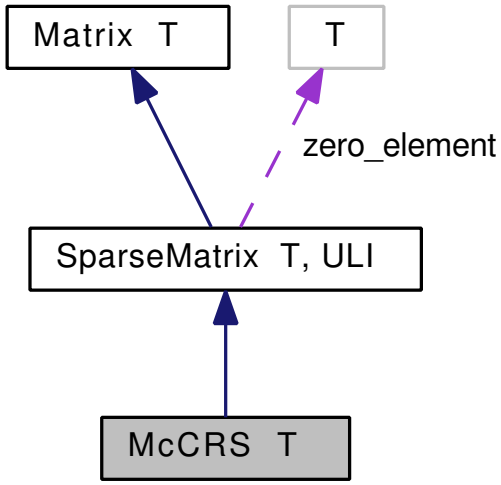
The compressed row storage sparse matrix data structure.

`#include <McCRS.hpp>`

Inheritance diagram for McCRS< T >:



Collaboration diagram for McCRS< T >:



Public Member Functions

- **McCRS** ()
  Base constructor.
- **McCRS** (std::string file, T zero=0)
  Base constructor.
- **McCRS** (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.
- **McCRS** (**McCRS**< T > &toCopy)
  Copy constructor.
- **McCRS** (std::vector< **Triplet**< T > > input, ULI **m**, ULI **n**, T zero)
  *Constructor which transforms a collection of input triplets to **McCRS** (p. **??**) format.*

- virtual void **load** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero)
- T & **random_access** (ULI i, ULI j)

  Method which provides random matrix access to the stored sparse matrix.
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual T * **mv** (const T *x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=xA function.
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=Ax function.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.
- ULI * **rowJump** ()

  Returns pointer to the row_start vector.
- ULI * **columnIndices** ()

  Returns pointer to the column index vector.
- T * **values** ()

  Returns pointer to the matrix nonzeros vector.
- virtual ~**McCRS** ()

  Base deconstructor.

Protected Member Functions

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

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

Static Protected Member Functions

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

  Sorts 1D columnwise.

Protected Attributes

- ULI * **row_start**

  Array keeping track of individual row starting indices.
- T * **ds**

  Array containing the actual nnz non-zeros.
- ULI * **col_ind**

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

Additional Inherited Members

5.29.1 Detailed Description

template<typename T>class McCRS< T >

The compressed row storage sparse matrix data structure.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 template<typename T> McCRS< T >::McCRS ( ) [inline]

Base constructor.

5.29.2.2 template<typename T> McCRS< T >::McCRS (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.29.2.3 template<typename T> McCRS< T >::McCRS (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 **McCRS** (p. **??**) 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 to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns of the matrix.
zero	The element considered to be zero.

References McCRS< T >::col_ind, McCRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, McCRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.29.2.4 template<typename T> McCRS< T >::McCRS (McCRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **McCRS** (p. **??**) datastructure to copy.

References McCRS< T >::col_ind, McCRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::nor, McCRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.29.2.5 template<typename T> McCRS< T >::McCRS (std::vector< Triplet< T > > *input*, ULI *m*, ULI *n*, T *zero*) [inline]

Constructor which transforms a collection of input triplets to **McCRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	The element considered to be zero.

References McCRS< T >::load().

5.29.2.6 template<typename T> virtual McCRS< T >::~McCRS ( ) [inline], [virtual]

Base deconstructor.

References McCRS< T >::col_ind, McCRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::nor, and McCRS< T >::row_start.

5.29.3 Member Function Documentation

5.29.3.1 template<typename T> virtual size_t McCRS< T >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References SparseMatrix< T, ULI >::nnz, and SparseMatrix< T, ULI >::nor.

5.29.3.2 template<typename T> ULI* McCRS< T >::columnIndices ( ) [inline]

Returns pointer to the column index vector.

References McCRS< T >::col_ind.

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

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

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.

References McCRS< T >::col_ind.

Referenced by McCRS< T >::random_access().

5.29.3.4 template<typename T> virtual void McCRS< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References McCRS< T >::col_ind, and McCRS< T >::row_start.

5.29.3.5 template<typename T> virtual void McCRS< T >::load (std::vector< Triplet< T > > & *input,* ULI *m,* ULI *n,* T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References McCRS< T >::col_ind, McCRS< T >::compareTriplets(), McCRS< T >::ds, Triplet< T >::j(), SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, McCRS< T >::row_start, Triplet< T >::value, and SparseMatrix< T, ULI >::zero_element.

Referenced by McCRS< T >::McCRS().

5.29.3.6 template<typename T> virtual T* McCRS< T >::mv (const T * *x*) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, McCRS< T >::zax(), and SparseMatrix< T, ULI >::zero_element.

5.29.3.7 template<typename T> T& McCRS< 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 (p. **??**) valuei at (i,j).

References McCRS< T >::ds, McCRS< T >::find(), McCRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.29.3.8 template<typename T> ULI* McCRS< T >::rowJump ( ) [inline]

Returns pointer to the row_start vector.

References McCRS< T >::row_start.

5.29.3.9 template<typename T> T* McCRS< T >::values ( ) [inline]

Returns pointer to the matrix nonzeros vector.

References McCRS< T >::ds.

5.29.3.10 template<typename T> virtual void McCRS< T >::zax (const T *__restrict__ x, T *__restrict__ z) [inline], [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References McCRS< T >::col_ind, McCRS< T >::ds, SparseMatrix< T, ULI >::nor, and McCRS< T >::row_start.

Referenced by McCRS< T >::mv().

5.29.4 Member Data Documentation

5.29.4.1 template<typename T> ULI* McCRS< T >::col_ind [protected]

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

Referenced by McCRS< T >::columnIndices(), McCRS< T >::find(), McCRS< T >::getFirstIndexPair(), McCRS< T >::load(), McCRS< T >::McCRS(), McCRS< T >::zax(), and McCRS< T >::~McCRS().

5.29.4.2 template<typename T> T* McCRS< T >::ds [protected]

Array containing the actual nnz non-zeros.

Referenced by McCRS< T >::load(), McCRS< T >::McCRS(), McCRS< T >::random_access(), McCRS< T >::values(), McCRS< T >::zax(), and McCRS< T >::~McCRS().

5.29.4.3 template<typename T> ULI* McCRS< T >::row_start [protected]

Array keeping track of individual row starting indices.

Referenced by McCRS< T >::getFirstIndexPair(), McCRS< T >::load(), McCRS< T >::McCRS(), McCRS< T >::random_access(), McCRS< T >::rowJump(), McCRS< T >::zax(), and McCRS< T >::~McCRS().

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

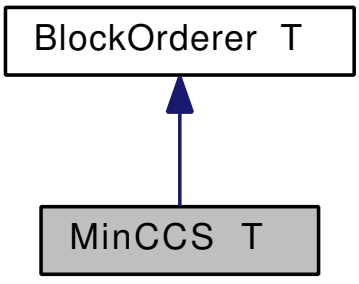
- McCRS.hpp

5.30 MinCCS< T > Class Template Reference

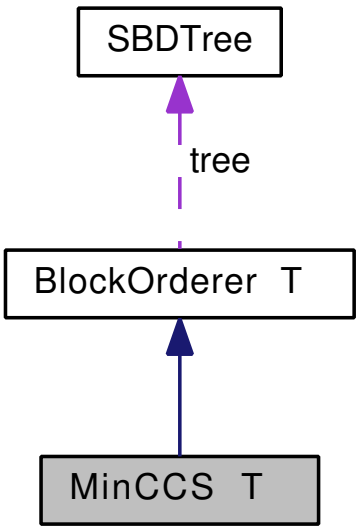
Codes the Minimal CCS block order.

`#include <MinCCS.hpp>`

Inheritance diagram for MinCCS< T >:



Collaboration diagram for MinCCS< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)
  Prefix operations during SBD tree traversal.
- virtual void **in_readout** (const unsigned long int index)
  Prefix operations during SBD tree traversal.
- virtual void **post_readout** (const unsigned long int index)
  Postfix operations during SBD tree traversal.

Additional Inherited Members

5.30.1 Detailed Description

template<typename T>class MinCCS< T >

Codes the Minimal CCS block order.

5.30.2 Member Function Documentation

5.30.2.1 template<typename T > virtual void MinCCS< T >::in_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

5.30.2.2 template<typename T > virtual void MinCCS< T >::post_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Postfix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::tree.

5.30.2.3 template<typename T > virtual void MinCCS< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::output, and BlockOrderer< T >::tree.

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

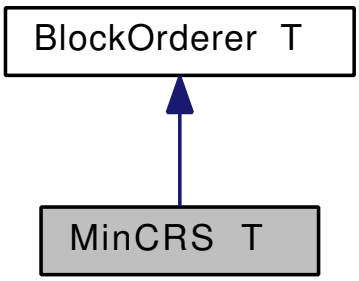
- MinCCS.hpp

5.31 MinCRS< T > Class Template Reference

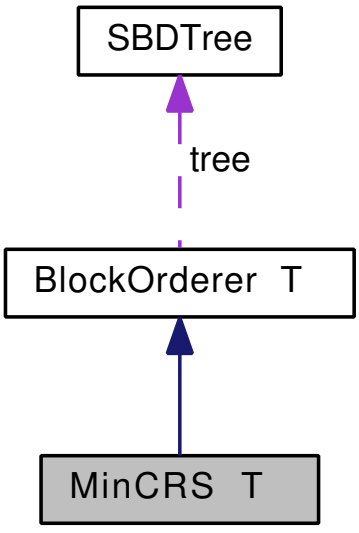
Codes the Minimal **CRS** (p. **??**) block order.

`#include <MinCRS.hpp>`

Inheritance diagram for MinCRS< T >:



Collaboration diagram for MinCRS< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)

  Prefix operations during SBD tree traversal.
- virtual void **in_readout** (const unsigned long int index)

  Infix operations during SBD tree traversal.
- virtual void **post_readout** (const unsigned long int index)

  Postfix operations during SBD tree traversal.

Additional Inherited Members

5.31.1 Detailed Description

template<typename T>class MinCRS< T >

Codes the Minimal **CRS** (p. **??**) block order.

5.31.2 Member Function Documentation

5.31.2.1 template<typename T > virtual void MinCRS< T >::in_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Infix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::tree.

5.31.2.2 template<typename T > virtual void MinCRS< T >::post_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Postfix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::output, and BlockOrderer< T >::tree.

5.31.2.3 template<typename T > virtual void MinCRS< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

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

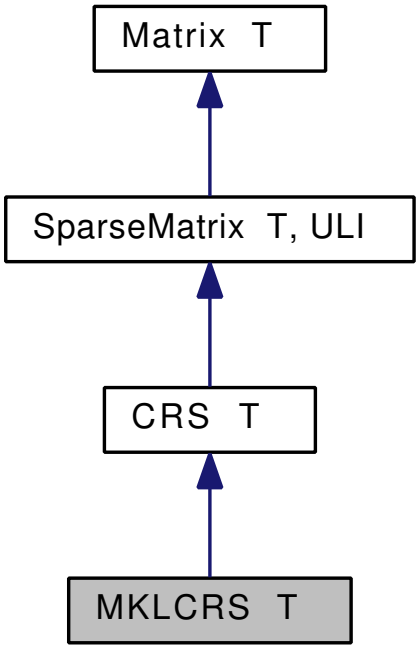
- MinCRS.hpp

5.32 MKLCRS< T > Class Template Reference

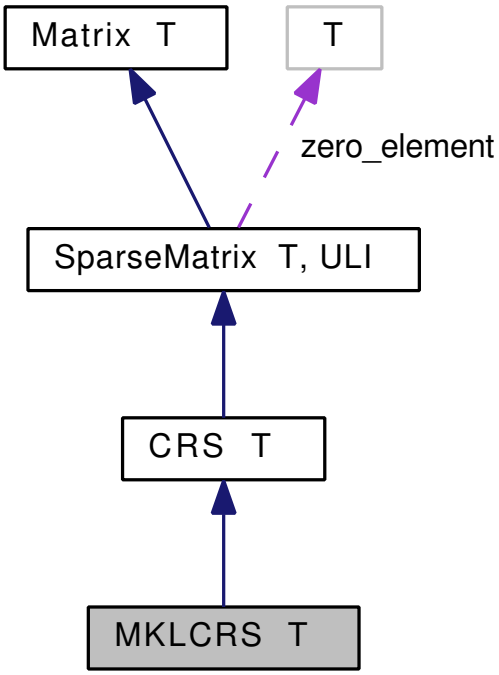
The compressed row storage sparse matrix data structure.

`#include <MKLCRS.hpp>`

Inheritance diagram for MKLCRS< T >:



Collaboration diagram for MKLCRS< T >:



Public Member Functions

- **MKLCRS** ()

  Base constructor.
- **MKLCRS** (std::string file, T zero=0)

  Base constructor.
- **MKLCRS** (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.
- **MKLCRS** (**CRS**< T > &toCopy)

  Copy constructor.
- **MKLCRS** (std::vector< **Triplet**< T > > input, ULI **m**, ULI **n**, T zero)

  *Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.*
- virtual T * **mv** (const T *x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=xA function.
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=Ax function.
- virtual ~**MKLCRS** ()

  Base deconstructor.

Protected Member Functions

- void **prepare** ()

  *Does the required post-processing of Sparse Library's **CRS** (p. **??**) representation to one compatible with Intel MKL.*

Protected Attributes

- double **_one**

  Required for call to MKL; a factor alpha=beta of 1.
- char **trans**

  Required for call to MKL; (no) transposition.
- char * **descr**

  Required for call to MKL; matrix descriptor.
- int **_m**

  Required for call to MKL; matrix row-size.
- int **_n**

  Required for call to MKL; matrix column-size.
- int * **_col_ind**

  Required for call to MKL; a plain-int version of col_ind.
- int * **_row_start**

  Required for call to MKL; a plain-int version of row_start.

Additional Inherited Members

5.32.1 Detailed Description

template<typename T>class MKLCRS< T >

The compressed row storage sparse matrix data structure.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 template<typename T> MKLCRS< T >::MKLCRS ( ) [inline]

Base constructor.

5.32.2.2 template<typename T> MKLCRS< T >::MKLCRS (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile(), and MKLCRS< T >::prepare().

5.32.2.3 template<typename T> MKLCRS< T >::MKLCRS (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** (p. **??**) 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 to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns of the matrix.
zero	The element considered to be zero.

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, MKLCRS< T >::prepare(), CRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.32.2.4 template<typename T> MKLCRS< T >::MKLCRS (CRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **CRS** (p. **??**) datastructure to copy.

References CRS< T >::col_ind, CRS< T >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::nor, MKLCRS< T >::prepare(), CRS< T >::row_start, and SparseMatrix< T, ULI >::zero_element.

5.32.2.5 template<typename T> MKLCRS< T >::MKLCRS (std::vector< Triplet< T > > *input*, ULI *m*, ULI *n*, T *zero*) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	The element considered to be zero.

References CRS< T >::load(), and MKLCRS< T >::prepare().

5.32.2.6 template<typename T> virtual MKLCRS< T >::~MKLCRS ( ) [inline], [virtual]

Base deconstructor.

References MKLCRS< T >::_col_ind, MKLCRS< T >::_row_start, and MKLCRS< T >::descr.

5.32.3 Member Function Documentation

5.32.3.1 template<typename T> virtual T* MKLCRS< T >::mv (const T * x) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, MKLCRS< T >::zax(), and SparseMatrix< T, ULI >::zero_element.

5.32.3.2 template<typename T> virtual void MKLCRS< T >::zax (const T *__restrict__ x, T *__restrict__ z) [inline], [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Reimplemented from **CRS< T >** (p. **??**).

References MKLCRS< T >::_col_ind, MKLCRS< T >::_m, MKLCRS< T >::_n, MKLCRS< T >::_one, MKLCRS< T >::_row_start, MKLCRS< T >::descr, CRS< T >::ds, and MKLCRS< T >::trans.

Referenced by MKLCRS< T >::mv().

5.32.4 Member Data Documentation

5.32.4.1 template<typename T> int* MKLCRS< T >::_col_ind [protected]

Required for call to MKL; a plain-int version of col_ind.

Referenced by MKLCRS< T >::prepare(), MKLCRS< T >::zax(), and MKLCRS< T >::~MKLCRS().

5.32.4.2 template<typename T> int MKLCRS< T >::_m [protected]

Required for call to MKL; matrix row-size.

Referenced by MKLCRS< T >::prepare(), and MKLCRS< T >::zax().

5.32.4.3 template<typename T> int MKLCRS< T >::_n [protected]

Required for call to MKL; matrix column-size.

Referenced by MKLCRS< T >::prepare(), and MKLCRS< T >::zax().

5.32.4.4 template<typename T> double MKLCRS< T >::_one [protected]

Required for call to MKL; a factor alpha=beta of 1.

Referenced by MKLCRS< T >::prepare(), and MKLCRS< T >::zax().

5.32.4.5 template<typename T> int* MKLCRS< T >::_row_start [protected]

Required for call to MKL; a plain-int version of row_start.

Referenced by MKLCRS< T >::prepare(), MKLCRS< T >::zax(), and MKLCRS< T >::~MKLCRS().

5.32.4.6 template<typename T> char* MKLCRS< T >::descr [protected]

Required for call to MKL; matrix descriptor.

Referenced by MKLCRS< T >::prepare(), MKLCRS< T >::zax(), and MKLCRS< T >::~MKLCRS().

5.32.4.7 template<typename T> char MKLCRS< T >::trans [protected]

Required for call to MKL; (no) transposition.

Referenced by MKLCRS< T >::prepare(), and MKLCRS< T >::zax().

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

- MKLCRS.hpp

5.33 RDB_shared_data< T > Class Template Reference

Shared data for **RDBHilbert** (p. **??**) threads.

`#include <RDBHilbert.hpp>`

Public Member Functions

- **RDB_shared_data** ()

  In case of ZXa or ZaX, a pointer to a collection of global input vectors.
- **RDB_shared_data** (size_t _id, size_t _P, std::vector< **Triplet**< double > > *_original, size_t *_nzb, pthread_mutex_t *_mutex, pthread_cond_t *_cond, pthread_mutex_t *_end_mutex, pthread_cond_t *_end_cond, size_t *_sync, size_t *_end_sync, size_t _ovsize, size_t _ovoffset, const T **const _in, T **const _out)

  Recommended constructor.

Public Attributes

- size_t **id**

  Process ID.
- size_t **P**

  Number of SPMD processes.
- unsigned char **mode**

  0 undef, 1 init, 2 zax, 3 zxa, 4 exit, 5 reset, 6 ZaX, 7 ZXa.

- size_t **repeat**

  How many times to repeat the operation set in 'mode' (above, only for 2, 3, 6 and 7)
- std::vector< **Triplet**< T > > * **original**

  How many SpMVs in a multiple-RHS computation (ZXa or ZaX).
- size_t * **nzb**

  Will store rowsums.
- double **time**

  Will store local timing.
- size_t **bytes**

  Will store the local amount of bytes used.
- size_t **fillIn**

  Will store the total fillIn at this thread.
- pthread_mutex_t * **mutex**

  Pointer to the sync mutex.
- pthread_cond_t * **cond**

  Pointer to the sync condition.
- pthread_mutex_t * **end_mutex**

  Pointer to the end mutex.
- pthread_cond_t * **end_cond**

  Pointer to the end condition.
- size_t * **sync**

  Sync counter.
- size_t * **end_sync**

  End counter.
- size_t **output_vector_size**

  Output vector length.
- size_t **output_vector_offset**

  Output vector offset (w.r.t.
- T * **local_y**

  Pointer to the local output vector.
- const T ** **input**

  Pointer to a global input vector.
- T ** **output**

  Pointer to a global output vector.

5.33.1 Detailed Description

template<typename T>class RDB_shared_data< T >

Shared data for **RDBHilbert** (p. **??**) threads.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 template<typename T> RDB_shared_data< T >::RDB_shared_data ( ) [inline]

In case of ZXa or ZaX, a pointer to a collection of global input vectors.

In case of ZXa or ZaX, a pointer to a collection of output vectors. Pointer to a local output vector set (in case of ZXa or ZaX). Base constructor. Will initialise all to invalid values or NULL.

5.33.3 Member Data Documentation

5.33.3.1 template<typename T> pthread_cond_t* RDB_shared_data< T >::cond

Pointer to the sync condition.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.2 template<typename T> pthread_cond_t* RDB_shared_data< T >::end_cond

Pointer to the end condition.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.3 template<typename T> pthread_mutex_t* RDB_shared_data< T >::end_mutex

Pointer to the end mutex.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.4 template<typename T> size_t* RDB_shared_data< T >::end_sync

End counter.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.5 template<typename T> size_t RDB_shared_data< T >::id

Process ID.

Referenced by RDBHilbert< T, MatrixType >::collectY(), and RDBHilbert< T, MatrixType >::thread().

5.33.3.6 template<typename T> const T** RDB_shared_data< T >::input

Pointer to a global input vector.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.7 template<typename T> T* RDB_shared_data< T >::local_y

Pointer to the local output vector.

Referenced by RDBHilbert< T, MatrixType >::collectY(), and RDBHilbert< T, MatrixType >::thread().

5.33.3.8 template<typename T> unsigned char RDB_shared_data< T >::mode

0 undef, 1 init, 2 zax, 3 zxa, 4 exit, 5 reset, 6 ZaX, 7 ZXa.

Note that currently modes 6 and 7 are disabled!

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.9 template<typename T> pthread_mutex_t* RDB_shared_data< T >::mutex

Pointer to the sync mutex.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.10 template<typename T> std::vector< Triplet< T > >* RDB_shared_data< T >::original

How many SpMVs in a multiple-RHS computation (ZXa or ZaX).

Pointer to the original input (for distributed input parsing).

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.11 template<typename T> T** RDB_shared_data< T >::output

Pointer to a global output vector.

Referenced by RDBHilbert< T, MatrixType >::collectY(), and RDBHilbert< T, MatrixType >::thread().

5.33.3.12 template<typename T> size_t RDB_shared_data< T >::output_vector_offset

Output vector offset (w.r.t.

global indices).

Referenced by RDBHilbert< T, MatrixType >::collectY(), and RDBHilbert< T, MatrixType >::thread().

5.33.3.13 template<typename T> size_t RDB_shared_data< T >::output_vector_size

Output vector length.

Referenced by RDBHilbert< T, MatrixType >::collectY(), and RDBHilbert< T, MatrixType >::thread().

5.33.3.14 template<typename T> size_t RDB_shared_data< T >::P

Number of SPMD processes.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.33.3.15 template<typename T> size_t* RDB_shared_data< T >::sync

Sync counter.

Referenced by RDBHilbert< T, MatrixType >::thread().

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

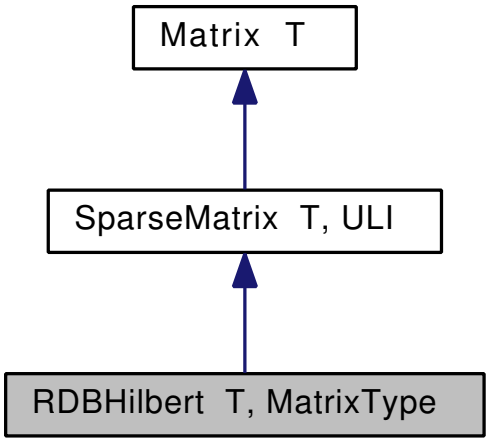
- RDBHilbert.hpp

5.34 RDBHilbert< T, MatrixType > Class Template Reference

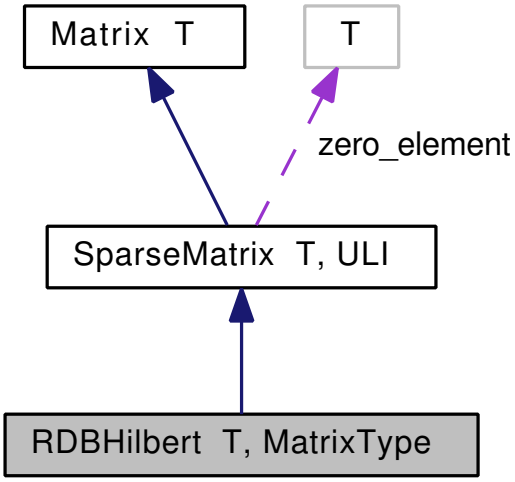
The Beta **Hilbert** (p. **??**) triplet scheme.

`#include <RDBHilbert.hpp>`

Inheritance diagram for RDBHilbert< T, MatrixType >:



Collaboration diagram for RDBHilbert< T, MatrixType >:



Public Member Functions

- **RDBHilbert** (const std::string file, T zero=0, std::vector< size_t > *_p_translate=NULL)

  Default file-based constructor.
- **RDBHilbert** (std::vector< **Triplet**< T > > &**input**, ULI **m**, ULI **n**, T zero=0, std::vector< size_t > *_p_translate=NULL)

  Default triplet-based constructor.
- virtual ~**RDBHilbert** ()

  Base deconstructor.
- void **wait** ()
- virtual void **load** (std::vector< **Triplet**< T > > &**input**, const ULI **m**, const ULI **n**, const T zero)
- virtual T * **mv** (const T *x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zxa** (const T *x, T *z)
- virtual void **zxa** (const T *x, T *z, const size_t repeat)
- void **reset** ()
- virtual void **zax** (const T *x, T *z)

- virtual void **zax** (const T ∗x, T ∗z, const size_t repeat, const clockid_t clock_id, double ∗elapsed_time)
- virtual void **getFirstIndexPair** (ULI &i, ULI &j)
- virtual size_t **bytesUsed** ()

Static Public Member Functions

- static void **end** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, size_t ∗**sync**, const size_t **P**)
- static void **synchronise** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, size_t ∗**sync**, const size_t **P**)
- static void ∗ **thread** (void ∗data)
- static void **collectY** (**RDB_shared_data**< T > ∗shared)

Protected Member Functions

- void **set_p_translate** (std::vector< size_t > ∗_p_translate)

  Sets p_translate to 0..P-1 by default, or equal to the optionally supplied vector.

Protected Attributes

- std::vector< size_t > **p_translate**

  Which processors to pin threads to.
- const T ∗ **input**

  Input vector.
- T ∗ **output**

  Output vector.
- pthread_t ∗ **threads**

  Input vectors for multiple-RHS operations.
- **RDB_shared_data**< T > ∗ **thread_data**

  array of initial thread data
- pthread_mutex_t **mutex**

  Stop/continue mechanism: mutex.
- pthread_cond_t **cond**

  Stop/continue mechanism: condition.
- pthread_mutex_t **end_mutex**

  Wait for end mechanism: mutex.
- pthread_cond_t **end_cond**

  Wait for end mechanism: condition.
- size_t **sync**

  Used for synchronising threads.
- size_t **end_sync**

  Used for construction end signal.

Static Protected Attributes

- static size_t **P** = 0

  Number of threads to fire up.
- static clockid_t **global_clock_id** = 0

  Clock type used for thread-local timing.
- static const ULI **max_n** = **FBICRS**< T >::beta_n

  *Given **FBICRS** (p. **??**), the maximum value for columnwise matrix size.*
- static const ULI **max_m** = **FBICRS**< T >::beta_m

  *Given **FBICRS** (p. **??**), the maximum value for the rowwise matrix size, assuming short ints on **ICRS** (p. **??**) at the lower level.*

Additional Inherited Members

5.34.1 Detailed Description

template<typename T, class MatrixType = FBICRS< T >>class RDBHilbert< T, MatrixType >

The Beta **Hilbert** (p. **??**) triplet scheme.

Full parallel SpMV, based on Blocked **Hilbert** (p. **??**) and PThreads. Inspired by Aydin & Gilbert's CSB and comments by Patrick Amestoy on the **BICRS** (p. **??**) **Hilbert** (p. **??**) scheme (both pointed towards sparse blocking, for different reasons that this scheme combines).

5.34.2 Constructor & Destructor Documentation

5.34.2.1 template<typename T, class MatrixType = FBICRS< T >> RDBHilbert< T, MatrixType >::RDBHilbert (const std::string *file*, T *zero* = 0, std::vector< size_t > * *_p_translate* = NULL) [inline]

Default file-based constructor.

References SparseMatrix< T, ULI >::loadFromFile(), and RDBHilbert< T, MatrixType >::set_p_translate().

5.34.2.2 template<typename T, class MatrixType = FBICRS< T >> RDBHilbert< T, MatrixType >::RDBHilbert (std::vector< Triplet< T > > & *input*, ULI *m*, ULI *n*, T *zero* = 0, std::vector< size_t > * *_p_translate* = NULL) [inline]

Default triplet-based constructor.

References RDBHilbert< T, MatrixType >::input, RDBHilbert< T, MatrixType >::load(), and RDBHilbert< T, MatrixType >::set_p_translate().

5.34.2.3 template<typename T, class MatrixType = FBICRS< T >> virtual RDBHilbert< T, MatrixType >::~RDBHilbert ( ) [inline], [virtual]

Base deconstructor.

References RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::mutex, RDBHilbert< T, MatrixType >::P, RDBHilbert< T, MatrixType >::thread_data, and RDBHilbert< T, MatrixType >::threads.

5.34.3 Member Function Documentation

5.34.3.1 template<typename T, class MatrixType = FBICRS< T >> virtual size_t RDBHilbert< T, MatrixType >::bytesUsed ( ) [inline], [virtual]

See Also

Matrix::bytesUsed (p. **??**)

Implements **Matrix< T >** (p. **??**).

References RDBHilbert< T, MatrixType >::P, and RDBHilbert< T, MatrixType >::thread_data.

5.34.3.2 template<typename T, class MatrixType = FBICRS< T >> static void RDBHilbert< T, MatrixType >::collectY (RDB_shared_data< T > * *shared*) [inline], [static]

See Also

RDScheme::collectY (p. **??**)

References RDB_shared_data< T >::id, RDB_shared_data< T >::local_y, RDB_shared_data< T >::output, RDB_shared_data< T >::output_vector_offset, and RDB_shared_data< T >::output_vector_size.

Referenced by RDBHilbert< T, MatrixType >::thread().

5.34.3.3 template<typename T, class MatrixType = FBICRS< T >> static void RDBHilbert< T, MatrixType >::end (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

See Also

RDScheme::end (p. **??**)

Referenced by RDBHilbert< T, MatrixType >::thread().

5.34.3.4 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::getFirstIndexPair (ULI & *i,* ULI & *j*) [inline], [virtual]

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

5.34.3.5 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::end_cond, RDBHilbert< T, MatrixType >::end_mutex, RDBHilbert< T, MatrixType >::end_sync, RDBHilbert< T, MatrixType >::input, SparseMatrix< T, ULI >::m(), RDBHilbert< T, MatrixType >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, RDBHilbert< T, MatrixType >::output, RDBHilbert< T, MatrixType >::P, RDBHilbert< T, MatrixType >::p_translate, RDBHilbert< T, MatrixType >::sync, RDBHilbert< T, MatrixType >::thread(), RDBHilbert< T, MatrixType >::thread_data, RDBHilbert< T, MatrixType >::threads, RDBHilbert< T, MatrixType >::wait(), and SparseMatrix< T, ULI >::zero_element.

Referenced by RDBHilbert< T, MatrixType >::RDBHilbert().

5.34.3.6 template<typename T, class MatrixType = FBICRS< T >> virtual T* RDBHilbert< T, MatrixType >::mv (const T * *x*) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, RDBHilbert< T, MatrixType >::reset(), RDBHilbert< T, MatrixType >::zax(), and SparseMatrix< T, ULI >::zero_element.

5.34.3.7 template<typename T, class MatrixType = FBICRS< T >> void RDBHilbert< T, MatrixType >::reset ( ) [inline]

See Also

RDScheme::reset

References RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::end_mutex, RDBHilbert< T, MatrixType >::mutex, RDBHilbert< T, MatrixType >::P, RDBHilbert< T, MatrixType >::thread_data, and RDBHilbert< T, MatrixType >::wait().

Referenced by RDBHilbert< T, MatrixType >::mv().

5.34.3.8 template<typename T, class MatrixType = FBICRS< T >> void RDBHilbert< T, MatrixType >::set_p_translate (std::vector< size_t > * *_p_translate*) [inline], [protected]

Sets p_translate to 0..P-1 by default, or equal to the optionally supplied vector.

References MachineInfo::cores(), MachineInfo::getInstance(), RDBHilbert< T, MatrixType >::P, and RDBHilbert< T, MatrixType >::p_translate.

Referenced by RDBHilbert< T, MatrixType >::RDBHilbert().

5.34.3.9 template<typename T, class MatrixType = FBICRS< T >> static void RDBHilbert< T, MatrixType >::synchronise (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

See Also

RDScheme::synchronise (p. **??**)

Referenced by RDBHilbert< T, MatrixType >::thread().

5.34.3.10 template<typename T, class MatrixType = FBICRS< T >> static void* RDBHilbert< T, MatrixType >::thread (void * *data*) [inline], [static]

See Also

RDScheme::thread (p. **??**)

References RDB_shared_data< T >::bytes, RDBHilbert< T, MatrixType >::collectY(), RDB_shared_data< T >::cond, RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::end(), RDB_shared_data< T >::end_cond, RDB_shared_data< T >::end_mutex, RDB_shared_data< T >::end_sync, RDB_shared_data< T >::fillIn, RDBHilbert< T, MatrixType >::global_clock_id, RDB_shared_data< T >::id, RDB_shared_data< T >::input, Matrix2HilbertCoordinates::IntegerToHilbert(), RDB_shared_data< T >::local_y, SparseMatrix< T, ULI >::m(), RDBHilbert< T, MatrixType >::max_m, RDBHilbert< T, MatrixType >::max_n, RDB_shared_data< T >::mode, RDB_shared_data< T >::mutex, RDBHilbert< T, MatrixType >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, RDB_shared_data< T >::nzb, RDB_shared_data< T >::original, RDB_shared_data< T >::output, RDB_shared_data< T >::output_vector_offset, RDB_shared_data< T >::output_vector_size, RDB_shared_data< T >::P, RDBHilbert< T, MatrixType >::P, RDB_shared_data< T >::repeat, RDB_shared_data< T >::sync, RDBHilbert< T, MatrixType >::synchronise(), and RDB_shared_data< T >::time.

Referenced by RDBHilbert< T, MatrixType >::load().

5.34.3.11 template<typename T, class MatrixType = FBICRS< T >> void RDBHilbert< T, MatrixType >::wait ( ) [inline]

See Also

RDScheme::wait (p. **??**)

References RDBHilbert< T, MatrixType >::end_cond, and RDBHilbert< T, MatrixType >::end_mutex.

Referenced by RDBHilbert< T, MatrixType >::load(), RDBHilbert< T, MatrixType >::reset(), RDBHilbert< T, MatrixType >::zax(), and RDBHilbert< T, MatrixType >::zxa().

5.34.3.12 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::zax (const T * x, T * z) [inline], [virtual]

See Also

Matrix::zax (p. **??**)

Referenced by RDBHilbert< T, MatrixType >::mv().

5.34.3.13 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::zax (const T * x, T * z, const size_t *repeat*, const clockid_t *clock_id*, double * *elapsed_time*) [inline], [virtual]

See Also

RDScheme::zax (p. **??**)

References RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::end_mutex, RDBHilbert< T, MatrixType >::global_clock_id, RDBHilbert< T, MatrixType >::input, RDBHilbert< T, MatrixType >::mutex, RDBHilbert< T, MatrixType >::output, RDBHilbert< T, MatrixType >::P, RDBHilbert< T, MatrixType >::thread_data, and RDBHilbert< T, MatrixType >::wait().

5.34.3.14 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::zxa (const T * x, T * z) [inline], [virtual]

See Also

Matrix::zxa (p. **??**)

5.34.3.15 template<typename T, class MatrixType = FBICRS< T >> virtual void RDBHilbert< T, MatrixType >::zxa (const T * x, T * z, const size_t *repeat*) [inline], [virtual]

See Also

RDScheme::zxa (p. **??**)

References RDBHilbert< T, MatrixType >::cond, RDBHilbert< T, MatrixType >::end_mutex, RDBHilbert< T, MatrixType >::input, RDBHilbert< T, MatrixType >::mutex, RDBHilbert< T, MatrixType >::output, RDBHilbert< T, MatrixType >::P, RDBHilbert< T, MatrixType >::thread_data, and RDBHilbert< T, MatrixType >::wait().

5.34.4 Member Data Documentation

5.34.4.1 template<typename T, class MatrixType = FBICRS< T >> pthread_t* RDBHilbert< T, MatrixType >::threads [protected]

Input vectors for multiple-RHS operations.

Output vectors for multiple-RHS operations. p_threads associated to this data strcuture

Referenced by RDBHilbert< T, MatrixType >::load(), and RDBHilbert< T, MatrixType >::~RDBHilbert().

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

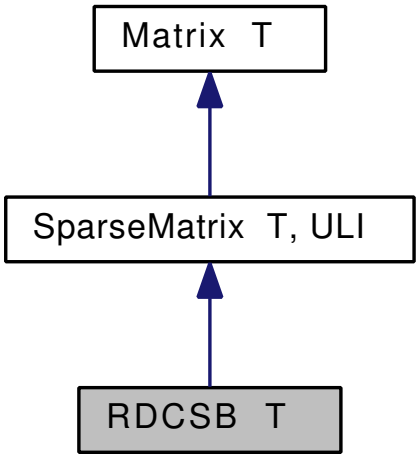
- RDBHilbert.hpp

5.35 RDCSB< T > Class Template Reference

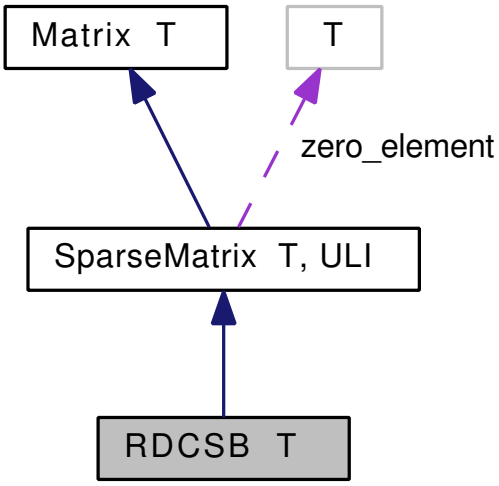
Full parallel row-distributed SpMV, based on CSB (**BlockCRS** (p. **??**) + Morton curve + Cilk) and PThreads.

`#include <RDCSB.hpp>`

Inheritance diagram for RDCSB< T >:



Collaboration diagram for RDCSB< T >:



Public Member Functions

- **RDCSB** (const std::string file, T zero)
- **RDCSB** (std::vector< **Triplet**< T > > &**input**, ULI **m**, ULI **n**, T zero)
- void **wait** ()
- virtual void **load** (std::vector< **Triplet**< T > > &**input**, const ULI **m**, const ULI **n**, const T zero)

  Function reading in from std::vector< Triplet< T > > format.
- virtual T * **mv** (const T *x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zxa** (const T *x, T *z)
- virtual void **zxa** (const T *x, T *z, const unsigned long int repeat)
- virtual void **zax** (const T *x, T *z)

- virtual void **zax** (const T ∗x, T ∗z, const unsigned long int repeat, const clockid_t clock_id, double ∗elapsed_time)
- virtual void **getFirstIndexPair** (ULI &i, ULI &j)

  Returns the first nonzero index, per reference.

Static Public Member Functions

- static void **end** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, unsigned long int ∗**sync**, const unsigned long int **P**)
- static void **synchronise** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, unsigned long int ∗**sync**, const unsigned long int **P**)
- static void ∗ **thread** (void ∗data)
- static void **collectY** (**RDCSB_shared_data**< T > ∗shared)

Protected Attributes

- pthread_t ∗ **threads**

  p_threads associated to this data strcuture
- **RDCSB_shared_data**< T > ∗ **thread_data**

  array of initial thread data
- pthread_mutex_t **mutex**

  Stop/continue mechanism: mutex.
- pthread_cond_t **cond**

  Stop/continue mechanism: condition.
- pthread_mutex_t **end_mutex**

  Wait for end mechanism: mutex.
- pthread_cond_t **end_cond**

  Wait for end mechanism: condition.
- unsigned long int **sync**

  Used for synchronising threads.
- unsigned long int **end_sync**

  Used for construction end signal.

Static Protected Attributes

- static unsigned long int **P** = 0

  Number of threads to fire up.
- static const T ∗ **input** = NULL

  Input vector.
- static T ∗ **output** = NULL

  Output vector.
- static clockid_t **global_clock_id** = 0

  Clock type used for thread-local timing.

Additional Inherited Members

5.35.1 Detailed Description

template<typename T>class RDCSB< T >

Full parallel row-distributed SpMV, based on CSB (**BlockCRS** (p. **??**) + Morton curve + Cilk) and PThreads.

Inspired by Aydin & Gilbert's CSB, and comments by Patrick Amestoy on the **BICRS** (p. **??**) **Hilbert** (p. **??**) scheme. May not compile due to PThreads/Cilk clashes.

5.35.2 Member Function Documentation

5.35.2.1 template<typename T> virtual void RDCSB< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

5.35.2.2 template<typename T> virtual void RDCSB< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

Function reading in from std::vector< Triplet< T > > format.

Parameters

input	The input matrix in triplet format.
m	The number of rows of the input matrix.
n	The number of columns of the input matrix.
zero	Which element is to be considered zero.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References RDCSB< T >::cond, MachineInfo::cores(), RDCSB< T >::end_cond, RDCSB< T >::end_mutex, RDCSB< T >::end_sync, MachineInfo::getInstance(), RDCSB< T >::input, SparseMatrix< T, ULI >::m(), RDCSB< T >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, RDCSB< T >::P, RDCSB< T >::sync, RDCSB< T >::thread_data, RDCSB< T >::threads, and SparseMatrix< T, ULI >::zero_element.

5.35.2.3 template<typename T> virtual T* RDCSB< T >::mv (const T * *x*) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, and SparseMatrix< T, ULI >::zero_element.

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

- RDCSB.hpp

5.36 RDCSB_shared_data< T > Class Template Reference

Shared data for **RDCSB** (p. **??**) threads.

`#include <RDCSB.hpp>`

Public Member Functions

- **RDCSB_shared_data** ()

  Base constructor.

- **RDCSB_shared_data** (unsigned long int _id, unsigned long int _P, std::vector< **Triplet**< double > > *_original, unsigned long int *_nzb, pthread_mutex_t *_mutex, pthread_cond_t *_cond, pthread_mutex_t *_end_mutex, pthread_cond_t *_end_cond, unsigned long int *_sync, unsigned long int *_end_sync, unsigned long int _ovsize, unsigned long int _ovoffset)

  Recommended constructor.

Public Attributes

- unsigned long int **id**
- unsigned long int **P**
- unsigned char **mode**

  0 undef, 1 init, 2 zax, 3 zxa, 4 exit

- unsigned long int **repeat**

  how many times to repeat the operation set in 'mode' (above, only for 2 and 3)

- std::vector< **Triplet**< T > > * **original**
- unsigned long int * **nzb**

  Will store rowsums.

- double **time**

  Will store local timing.

- pthread_mutex_t * **mutex**
- pthread_cond_t * **cond**
- pthread_mutex_t * **end_mutex**
- pthread_cond_t * **end_cond**
- unsigned long int * **sync**
- unsigned long int * **end_sync**
- unsigned long int **output_vector_size**
- unsigned long int **output_vector_offset**
- T * **local_y**

5.36.1 Detailed Description

template<typename T>class RDCSB_shared_data< T >

Shared data for **RDCSB** (p. **??**) threads.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 template<typename T> RDCSB_shared_data< T >::RDCSB_shared_data ( ) [inline]

Base constructor.

Will initialise all to invalid values or NULL.

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

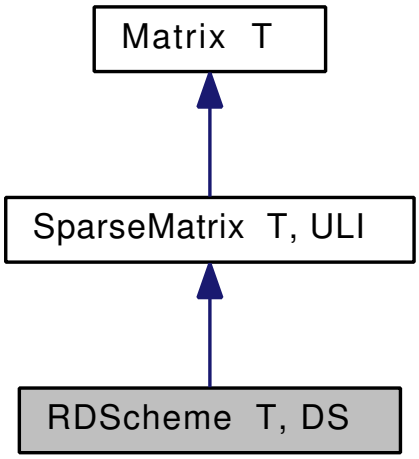
- RDCSB.hpp

5.37 RDScheme< T, DS > Class Template Reference

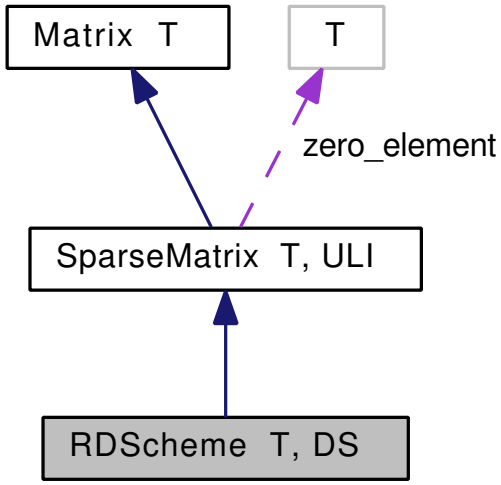
Full parallel row-distributed SpMV, based on CSB (Morton curve + Cilk) and PThreads.

`#include <RDScheme.hpp>`

Inheritance diagram for RDScheme< T, DS >:



Collaboration diagram for RDScheme< T, DS >:



Public Member Functions

- **RDScheme** (const std::string file, T zero)
  Base constructor.
- **RDScheme** (std::vector< **Triplet**< T > > &**input**, ULI **m**, ULI **n**, T zero)
  Base constructor.
- virtual ~**RDScheme** ()
  Base deconstructor.
- void **wait** ()
  Lets the calling thread wait for the end of the SpMV multiply.
- virtual void **load** (std::vector< **Triplet**< T > > &**input**, const ULI **m**, const ULI **n**, const T zero)
  Loads a sparse matrix from an input set of triplets.

- virtual T ∗ **mv** (const T ∗x)

  Overloaded mv call; allocates output vector using numa_interleaved.
- virtual void **zxa** (const T ∗x, T ∗z)
- virtual void **zxa** (const T ∗x, T ∗z, const unsigned long int repeat)
- virtual void **zax** (const T ∗x, T ∗z)

  *See **SparseMatrix::zax** (p. **??**).*
- virtual void **zax** (const T ∗x, T ∗z, const unsigned long int repeat, const clockid_t clock_id, double ∗elapsed-_time)

  *See **SparseMatrix::zax** (p. **??**).*
- virtual size_t **bytesUsed** ()
- virtual void **getFirstIndexPair** (ULI &i, ULI &j)

  Function disabled for parallel schemes!

Static Public Member Functions

- static void **end** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, size_t ∗**sync**, const size_t **P**)

  End synchronisation code.
- static void **synchronise** (pthread_mutex_t ∗**mutex**, pthread_cond_t ∗**cond**, size_t ∗**sync**, const size_t **P**)

  Synchronises all threads.
- static void ∗ **thread** (void ∗data)

  SPMD code for each thread involved with parallel SpMV multiplication.
- static void **collectY** (**RDScheme_shared_data**< T > ∗shared)

  Reduces a distributed output vector set into a single contiguous output vector at process 0.

Protected Attributes

- pthread_t ∗ **threads**

  p_threads associated to this data strcuture
- **RDScheme_shared_data**< T > ∗ **thread_data**

  array of initial thread data
- pthread_mutex_t **mutex**

  Stop/continue mechanism: mutex.
- pthread_cond_t **cond**

  Stop/continue mechanism: condition.
- pthread_mutex_t **end_mutex**

  Wait for end mechanism: mutex.
- pthread_cond_t **end_cond**

  Wait for end mechanism: condition.
- size_t **sync**

  Used for synchronising threads.
- size_t **end_sync**

  Used for construction end signal.

Static Protected Attributes

- static size_t **P** = 0

  Number of threads to fire up.
- static const T ∗ **input** = NULL

  Input vector.
- static T ∗ **output** = NULL

  Output vector.
- static clockid_t **global_clock_id** = 0

  Clock type used for thread-local timing.

Additional Inherited Members

5.37.1 Detailed Description

template<typename T, typename DS>class RDScheme< T, DS >

Full parallel row-distributed SpMV, based on CSB (Morton curve + Cilk) and PThreads.

Inspired by Aydin & Gilbert's CSB, and comments by Patrick Amestoy on the **BICRS** (p. **??**) **Hilbert** (p. **??**) scheme.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 template<typename T, typename DS> RDScheme< T, DS >::RDScheme (const std::string *file,* T *zero*) [inline]

Base constructor.

Reads input from file.

References SparseMatrix< T, ULI >::loadFromFile().

5.37.2.2 template<typename T, typename DS> RDScheme< T, DS >::RDScheme (std::vector< Triplet< T > > & *input,* ULI *m,* ULI *n,* T *zero*) [inline]

Base constructor.

Reads input from a set of triplets.

References RDScheme< T, DS >::input, and RDScheme< T, DS >::load().

5.37.2.3 template<typename T, typename DS> virtual RDScheme< T, DS >::~RDScheme ( ) [inline], [virtual]

Base deconstructor.

References RDScheme< T, DS >::cond, RDScheme< T, DS >::mutex, RDScheme< T, DS >::P, RDScheme< T, DS >::thread_data, and RDScheme< T, DS >::threads.

5.37.3 Member Function Documentation

5.37.3.1 template<typename T, typename DS> virtual size_t RDScheme< T, DS >::bytesUsed ( ) [inline], [virtual]

Returns

The memory usage of this data structure (summed over all threads).

Implements **Matrix< T >** (p. **??**).

References RDScheme< T, DS >::P, and RDScheme< T, DS >::thread_data.

5.37.3.2 template<typename T, typename DS> static void RDScheme< T, DS >::collectY (RDScheme_shared_data< T > * *shared*) [inline], [static]

Reduces a distributed output vector set into a single contiguous output vector at process 0.

Parameters

shared	Which set of output vectors to reduce.

References RDScheme_shared_data< T >::id, RDScheme_shared_data< T >::local_y, RDScheme_shared_data< T >::output_vector_offset, and RDScheme_shared_data< T >::output_vector_size.

Referenced by RDScheme< T, DS >::thread().

5.37.3.3 template<typename T, typename DS> static void RDScheme< T, DS >::end (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

End synchronisation code.

Referenced by RDScheme< T, DS >::thread().

5.37.3.4 template<typename T, typename DS> virtual void RDScheme< T, DS >::getFirstIndexPair (ULI & *i,* ULI & *j*) [inline], [virtual]

Function disabled for parallel schemes!

See Also

SparseMatrix::getFirstIndexPair (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

5.37.3.5 template<typename T, typename DS> virtual void RDScheme< T, DS >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

Loads a sparse matrix from an input set of triplets.

Parameters

input	The input set of triplets.
m	The number of rows.
n	The number of columns.
zero	What constitutes a zero in this sparse matrix instance.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References RDScheme< T, DS >::cond, MachineInfo::cores(), RDScheme< T, DS >::end_cond, RDScheme< T, DS >::end_mutex, RDScheme< T, DS >::end_sync, MachineInfo::getInstance(), RDScheme< T, DS >::input, SparseMatrix< T, ULI >::m(), RDScheme< T, DS >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, RDScheme< T, DS >::P, RDScheme< T, DS >::sync, RDScheme< T, DS >::thread(), RDScheme< T, DS >::thread_data, RDScheme< T, DS >::threads, RDScheme< T, DS >::wait(), and SparseMatrix< T, ULI >::zero_element.

Referenced by RDScheme< T, DS >::RDScheme().

5.37.3.6 template<typename T, typename DS> virtual T* RDScheme< T, DS >::mv (const T * *x*) [inline], [virtual]

Overloaded mv call; allocates output vector using numa_interleaved.

Reimplemented from **SparseMatrix< T, ULI >** (p. **??**).

References SparseMatrix< T, ULI >::nor, RDScheme< T, DS >::zax(), and SparseMatrix< T, ULI >::zero_element.

5.37.3.7 template<typename T, typename DS> static void RDScheme< T, DS >::synchronise (pthread_mutex_t * *mutex,* pthread_cond_t * *cond,* size_t * *sync,* const size_t *P*) [inline], [static]

Synchronises all threads.

Referenced by RDScheme< T, DS >::thread().

5.37.3.8 template<typename T, typename DS> static void* RDScheme< T, DS >::thread (void * *data*) [inline], [static]

SPMD code for each thread involved with parallel SpMV multiplication.

References RDScheme_shared_data< T >::bytes, RDScheme< T, DS >::collectY(), RDScheme_shared_data< T >::cond, RDScheme< T, DS >::cond, RDScheme< T, DS >::end(), RDScheme_shared_data< T >::end_cond, RDScheme_shared_data< T >::end_mutex, RDScheme_shared_data< T >::end_sync, RDScheme< T, DS >::global_clock_id, RDScheme_shared_data< T >::id, RDScheme_shared_data< T >::local_y, SparseMatrix< T, ULI >::m(), RDScheme_shared_data< T >::mode, RDScheme_shared_data< T >::mutex, RDScheme< T, DS >::mutex, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, RDScheme_shared_data< T >::nzb, RDScheme_shared_data< T >::original, RDScheme< T, DS >::output, RDScheme_shared_data< T >::output_vector_offset, RDScheme_shared_data< T >::output_vector_size, RDScheme_shared_data< T >::P, RDScheme< T, DS >::P, RDScheme_shared_data< T >::repeat, RDScheme_shared_data< T >::sync, RDScheme< T, DS >::synchronise(), and RDScheme_shared_data< T >::time.

Referenced by RDScheme< T, DS >::load().

5.37.3.9 template<typename T, typename DS> virtual void RDScheme< T, DS >::zxa (const T * *x,* T * *z*) [inline], [virtual]

See Also

SparseMatrix::zxa (p. **??**)

5.37.3.10 template<typename T, typename DS> virtual void RDScheme< T, DS >::zxa (const T * *x,* T * *z,* const unsigned long int *repeat*) [inline], [virtual]

See Also

SparseMatrix::zxa (p. **??**)

References RDScheme< T, DS >::cond, RDScheme< T, DS >::end_mutex, RDScheme< T, DS >::mutex, RDScheme< T, DS >::P, RDScheme< T, DS >::thread_data, and RDScheme< T, DS >::wait().

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

- RDScheme.hpp

5.38 RDScheme_shared_data< T > Class Template Reference

Shared data for **RDScheme** (p. **??**) threads.

`#include <RDScheme.hpp>`

Public Member Functions

- **RDScheme_shared_data** ()

  Base constructor.

- **RDScheme_shared_data** (size_t _id, size_t _P, std::vector< **Triplet**< double > > *_original, size_t *_nzb, pthread_mutex_t *_mutex, pthread_cond_t *_cond, pthread_mutex_t *_end_mutex, pthread_cond_t *_end-_cond, size_t *_sync, size_t *_end_sync, size_t _ovsize, size_t _ovoffset)

  Default constructor.

Public Attributes

- size_t **id**

  Thread ID.
- size_t **P**

  Total number of processors.
- unsigned char **mode**

  0 undef, 1 init, 2 zax, 3 zxa, 4 exit
- unsigned long int **repeat**

  how many times to repeat the operation set in 'mode' (above, only for 2 and 3)
- std::vector< **Triplet**< T > > * **original**

  Array of vectors of thread-local nonzeroes.
- size_t * **nzb**

  Will store rowsums.
- double **time**

  Will store local timing.
- size_t **bytes**

  Will store memory use.
- pthread_mutex_t * **mutex**

  Mutex used for synchronisation.
- pthread_cond_t * **cond**

  Condition used for synchronisation.
- pthread_mutex_t * **end_mutex**

  Mutex used for end sync.
- pthread_cond_t * **end_cond**

  Condition used for end sync.
- size_t * **sync**

  Counter used for synchronisation.
- size_t * **end_sync**

  Counter used for end sync.
- size_t **output_vector_size**

  Length of the local output vector.
- size_t **output_vector_offset**

  Offset of the local output vector compared to global indices.
- T * **local_y**

  Pointer to the local output vector.

5.38.1 Detailed Description

template<typename T>class RDScheme_shared_data< T >

Shared data for **RDScheme** (p. **??**) threads.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 template<typename T> RDScheme_shared_data< T >::RDScheme_shared_data ( ) [inline]

Base constructor.

Will initialise all to invalid values or NULL.

5.38.2.2 template<typename T> RDScheme_shared_data< T >::RDScheme_shared_data (size_t *_id*, size_t *_P*, std::vector< Triplet< double > > * *_original*, size_t * *_nzb*, pthread_mutex_t * *_mutex*, pthread_cond_t * *_cond*, pthread_mutex_t * *_end_mutex*, pthread_cond_t * *_end_cond*, size_t * *_sync*, size_t * *_end_sync*, size_t *_ovsize*, size_t *_ovoffset*) [inline]

Default constructor.

Parameters

_id	Thread ID.
_P	Number of SPMD processes.
_original	Original set of nonzeroes, split by blocks.
_nzb	Number of sparse blocks.
_mutex	Sync mutex.
_cond	Sync condition.
_end_mutex	End sync mutex.
_end_cond	End sync condition.
_sync	Sync counter.
_end_sync	End sync counter.
_ovsize	Output vector size.
_ovoffset	Output vector start position (global view).

5.38.3 Member Data Documentation

5.38.3.1 template<typename T> size_t RDScheme_shared_data< T >::bytes

Will store memory use.

Referenced by RDScheme< T, DS >::thread().

5.38.3.2 template<typename T> pthread_cond_t* RDScheme_shared_data< T >::cond

Condition used for synchronisation.

Referenced by RDScheme< T, DS >::thread().

5.38.3.3 template<typename T> pthread_cond_t* RDScheme_shared_data< T >::end_cond

Condition used for end sync.

Referenced by RDScheme< T, DS >::thread().

5.38.3.4 template<typename T> pthread_mutex_t* RDScheme_shared_data< T >::end_mutex

Mutex used for end sync.

Referenced by RDScheme< T, DS >::thread().

5.38.3.5 template<typename T> size_t* RDScheme_shared_data< T >::end_sync

Counter used for end sync.

Referenced by RDScheme< T, DS >::thread().

5.38.3.6 template<typename T> T* RDScheme_shared_data< T >::local_y

Pointer to the local output vector.

Referenced by RDScheme< T, DS >::collectY(), and RDScheme< T, DS >::thread().

5.38.3.7 template<typename T> pthread_mutex_t* RDScheme_shared_data< T >::mutex

Mutex used for synchronisation.

Referenced by RDScheme< T, DS >::thread().

5.38.3.8 template<typename T> std::vector< Triplet< T > >* RDScheme_shared_data< T >::original

Array of vectors of thread-local nonzeroes.

Referenced by RDScheme< T, DS >::thread().

5.38.3.9 template<typename T> size_t RDScheme_shared_data< T >::output_vector_offset

Offset of the local output vector compared to global indices.

Referenced by RDScheme< T, DS >::collectY(), and RDScheme< T, DS >::thread().

5.38.3.10 template<typename T> size_t RDScheme_shared_data< T >::output_vector_size

Length of the local output vector.

Referenced by RDScheme< T, DS >::collectY(), and RDScheme< T, DS >::thread().

5.38.3.11 template<typename T> size_t* RDScheme_shared_data< T >::sync

Counter used for synchronisation.

Referenced by RDScheme< T, DS >::thread().

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

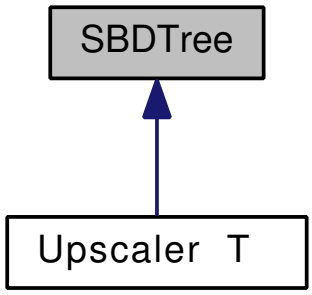
- RDScheme.hpp

5.39 SBDTree Class Reference

Models a Separated Block Diagonal tree structure.

`#include <SBDTree.hpp>`

Inheritance diagram for SBDTree:



Public Member Functions

- **SBDTree** (std::vector< unsigned long int > &r_hierarchy, std::vector< unsigned long int > &c_hierarchy, std::vector< unsigned long int > &r_bounds, std::vector< unsigned long int > &c_bounds)

  Base constructor.
- **SBDTree** (std::vector< unsigned long int > &hierarchy, std::vector< unsigned long int > &r_bounds, std::vector< unsigned long int > &c_bounds)

  Base constructor.
- ~**SBDTree** ()

  Base deconstructor.
- void **getSeparatorBB** (const unsigned long int index, unsigned long int &**r_lo**, unsigned long int &**r_hi**, unsigned long int &**c_lo**, unsigned long int &**c_hi**)

  Gets, from a separator node, the bounding box of the nonzeroes contained in the separator.
- unsigned long int **up** (const unsigned long int index)

  Returns the parent of a given node.
- unsigned long int **left** (const unsigned long int index)

  Returns the left child of a given node.
- unsigned long int **right** (const unsigned long int index)

  Returns the right child of a given node.
- void **rowBounds** (const unsigned long int index, unsigned long int &**r_lo**, unsigned long int &**r_hi**)

  Returns the row bounds corresponding to a given node.
- void **columnBounds** (const unsigned long int index, unsigned long int &**c_lo**, unsigned long int &**c_hi**)

  Returns the column bounds corresponding to a given node.
- char **isLeaf** (const unsigned long int index)

  Whether the given node is a leaf node.
- unsigned long int **size** ()

  Gets the number of nodes.
- unsigned long int **getRoot** ()

  Gets the root index.

Protected Member Functions

- void **build** (std::vector< unsigned long int > &hierarchy, std::vector< unsigned long int > &r_bounds, std::vector< unsigned long int > &c_bounds)

  Builds the SBD tree using three input vectors.

Protected Attributes

- unsigned long int * **parent**

  Array s.t.
- unsigned long int * **left_child**

  Array s.t.
- unsigned long int * **right_child**

  Array s.t.
- unsigned long int * **r_lo**

  Array s.t.
- unsigned long int * **r_hi**

  Array s.t.
- unsigned long int * **c_lo**

  Array s.t.
- unsigned long int * **c_hi**

  Array s.t.
- unsigned long int **root**

  Which node ID corresponds to the root.
- unsigned long int **nodes**

  The total number of tree nodes.
- char **root_is_set**

  Whether the root node is set.

Static Protected Attributes

- static const unsigned long int **NO_SUCH_NODE** = ULONG_MAX

  Integer corresponding to non-existing nodes.

5.39.1 Detailed Description

Models a Separated Block Diagonal tree structure.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 SBDTree::SBDTree (std::vector< unsigned long int > & *hierarchy,* std::vector< unsigned long int > & *r_bounds,* std::vector< unsigned long int > & *c_bounds*)

Base constructor.

Warning: avoids some assertions!

References build().

5.39.2.2 SBDTree::~SBDTree ( )

Base deconstructor.

References c_hi, c_lo, left_child, parent, r_hi, r_lo, and right_child.

5.39.3 Member Function Documentation

5.39.3.1 void SBDTree::build (std::vector< unsigned long int > & *hierarchy*, std::vector< unsigned long int > & *r_bounds*, std::vector< unsigned long int > & *c_bounds*) [protected]

Builds the SBD tree using three input vectors.

Parameters

hierarchy	The SBD hierarchy vector.
r_bounds	The row bounds of each SBD block.
c_bounds	The column bounds of each SBD block.

References c_hi, c_lo, left_child, NO_SUCH_NODE, nodes, parent, r_hi, r_lo, right_child, root, and root_is_set.

Referenced by SBDTree().

5.39.3.2 void SBDTree::columnBounds (const unsigned long int *index,* unsigned long int & *c_lo,* unsigned long int & *c_hi*)

Returns the column bounds corresponding to a given node.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::upper_vertical().

5.39.3.3 void SBDTree::getSeparatorBB (const unsigned long int *index,* unsigned long int & *r_lo,* unsigned long int & *r_hi,* unsigned long int & *c_lo,* unsigned long int & *c_hi*)

Gets, from a separator node, the bounding box of the nonzeroes contained in the separator.

Note that this is *not* the r_lo/hi,c_lo/hi of the node itself; those are the bounds of the row-wise and column-wise separators.

This is a logarithmic operation.

Parameters

index	The separator node ID.
r_lo	Where to store the lower row bound.
r_hi	Where to store the upper row bound.
c_lo	Where to store the lower column bound.
c_hi	Where to store the upper column bound.

Note that this is *not* the r_lo/hi,c_lo/hi of the node itself; those are the bounds of the row-wise and column-wise separators.

This is a logarithmic operation.

References left_child, NO_SUCH_NODE, and right_child.

Referenced by BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::upper_vertical().

5.39.3.4 unsigned long int SBDTree::left (const unsigned long int *index*)

Returns the left child of a given node.

References left_child, and NO_SUCH_NODE.

Referenced by BlockOrderer< T >::traverse().

5.39.3.5 unsigned long int SBDTree::right (const unsigned long int *index*)

Returns the right child of a given node.

References NO_SUCH_NODE, and right_child.

Referenced by BlockOrderer< T >::traverse().

5.39.3.6 void SBDTree::rowBounds (const unsigned long int *index,* unsigned long int & *r_lo,* unsigned long int & *r_hi*)

Returns the row bounds corresponding to a given node.

Referenced by BlockOrderer< T >::induce(), BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::upper_vertical().

5.39.3.7 unsigned long int SBDTree::up (const unsigned long int *index*)

Returns the parent of a given node.

References NO_SUCH_NODE, and parent.

Referenced by BlockOrderer< T >::traverse().

5.39.4 Member Data Documentation

5.39.4.1 unsigned long int* SBDTree::c_hi [protected]

Array s.t.

c_hi[i] returns the upper column bound of block i.

Referenced by build(), Upscaler< T >::getSubTree(), Upscaler< T >::Upscaler(), and ~SBDTree().

5.39.4.2 unsigned long int* SBDTree::c_lo [protected]

Array s.t.

c_lo[i] returns the lower column bound of block i.

Referenced by build(), Upscaler< T >::getSubTree(), Upscaler< T >::Upscaler(), and ~SBDTree().

5.39.4.3 unsigned long int* SBDTree::left_child [protected]

Array s.t.

left_child[i] returns the left child ID of block i.

Referenced by build(), getSeparatorBB(), Upscaler< T >::getSubTree(), isLeaf(), left(), Upscaler< T >::readout(), and ~SBDTree().

5.39.4.4 const unsigned long int SBDTree::NO_SUCH_NODE = ULONG_MAX [static], [protected]

Integer corresponding to non-existing nodes.

Referenced by build(), Upscaler< T >::determineMinMax(), getSeparatorBB(), Upscaler< T >::getSubTree(), isLeaf(), left(), Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::next(), Upscaler< T >::treePostOrderIterator::next(), Upscaler< T >::readout(), right(), and up().

5.39.4.5 unsigned long int SBDTree::nodes [protected]

The total number of tree nodes.

Referenced by build(), and size().

5.39.4.6 unsigned long int* SBDTree::parent [protected]

Array s.t.

parent[i] returns the parent ID of block i.

Referenced by build(), Upscaler< T >::determineMinMax(), Upscaler< T >::getSubTree(), Upscaler< T >::readout(), up(), Upscaler< T >::Upscaler(), and ~SBDTree().

5.39.4.7 unsigned long int* SBDTree::r_hi [protected]

Array s.t.

r_hi[i] returns the upper row bound of block i.

Referenced by build(), Upscaler< T >::getSubTree(), Upscaler< T >::Upscaler(), and ~SBDTree().

5.39.4.8 unsigned long int* SBDTree::r_lo [protected]

Array s.t.

r_lo[i] returns the lower row bound of block i.

Referenced by build(), Upscaler< T >::getSubTree(), Upscaler< T >::Upscaler(), and ~SBDTree().

5.39.4.9 unsigned long int* SBDTree::right_child [protected]

Array s.t.

right_child[i] returns the right child ID of block i.

Referenced by build(), getSeparatorBB(), Upscaler< T >::getSubTree(), isLeaf(), Upscaler< T >::readout(), right(), and ~SBDTree().

5.39.4.10 unsigned long int SBDTree::root [protected]

Which node ID corresponds to the root.

Referenced by build(), getRoot(), Upscaler< T >::getSubTree(), and Upscaler< T >::Upscaler().

5.39.4.11 char SBDTree::root_is_set [protected]

Whether the root node is set.

Referenced by build().

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

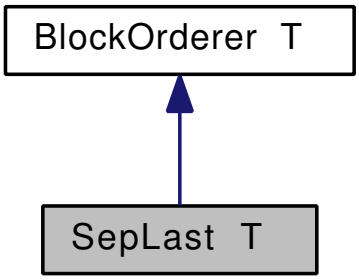
- SBDTree.hpp
- SBDTree.cpp

5.40 SepLast< T > Class Template Reference

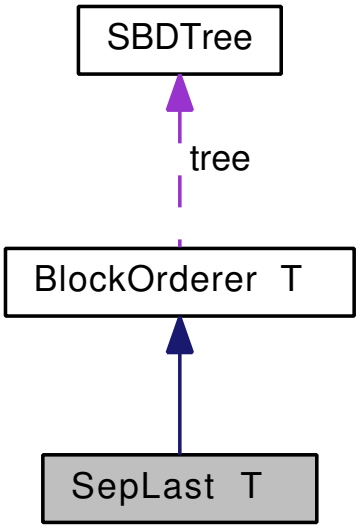
Codes the Minimal CCS block order.

`#include <SepLast.hpp>`

Inheritance diagram for SepLast< T >:



Collaboration diagram for SepLast< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)

  Prefix traversal code.
- virtual void **in_readout** (const unsigned long int index)

  Infix traversal code.
- virtual void **post_readout** (const unsigned long int index)

  Postfix traversal code.

Additional Inherited Members

5.40.1 Detailed Description

template<typename T>class SepLast< T >

Codes the Minimal CCS block order.

5.40.2 Member Function Documentation

5.40.2.1 template<typename T > virtual void SepLast< T >::in_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Infix traversal code.

Implements **BlockOrderer**< **T** > (p. **??**).

5.40.2.2 template<typename T > virtual void SepLast< T >::post_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Postfix traversal code.

Implements **BlockOrderer**< **T** > (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

5.40.2.3 template<typename T > virtual void SepLast< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix traversal code.

Implements **BlockOrderer**< **T** > (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, and BlockOrderer< T >::tree.

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

- SepLast.hpp

5.41 shared_data< T > Class Template Reference

Shared data for **BetaHilbert** (p. **??**) threads.

`#include <BetaHilbert.hpp>`

Public Member Functions

- **shared_data** ()

  Base constructor.
- **shared_data** (size_t _id, size_t _P, std::vector< **Triplet**< double > > *_original, size_t *_nzb, size_t **_nzc, pthread_mutex_t *_mutex, pthread_cond_t *_cond, pthread_mutex_t *_end_mutex, pthread_cond_t *_end_cond, size_t *_sync, size_t *_end_sync, size_t _m, const T **_in, T **_out)

  Default constructor.

Public Attributes

- size_t **id**

  Thread ID.
- size_t **P**

  Total number of processors.
- unsigned char **mode**

  0 undef, 1 init, 2 zax, 3 zxa, 4 exit, 5 reset

- size_t **repeat**

  how many times to repeat the operation set in 'mode' (above, only for 2 and 3)
- std::vector< **Triplet**< T > > * **original**

  Array of local sparse blocks.
- size_t * **nzb**

  Will cache block numbers of nonzeroes.
- size_t ** **nzc**

  Will contain the nonzero counts of separate blocks.
- double **time**

  Will store local timing.
- size_t **bytes**

  Local memory use.
- pthread_mutex_t * **mutex**

  Mutex used for synchronisation.
- pthread_cond_t * **cond**

  Condition used for synchronisation.
- pthread_mutex_t * **end_mutex**

  Mutex used for end sync.
- pthread_cond_t * **end_cond**

  Condition used for end sync.
- size_t * **sync**

  Counter used for synchronisation.
- size_t * **end_sync**

  Counter used for end sync.
- size_t **output_vector_size**

  Length of the local output vector.
- size_t **output_vector_offset**

  Offset of the local output vector compared to global indices.
- T * **local_y**

  Pointer to the local output vector.
- const T ** **input**

  Array of all local input vectors of all SPMD processes.
- T ** **output**

  Array of all output vectors local to all SPMD processes.

5.41.1 Detailed Description

template<typename T>class shared_data< T >

Shared data for **BetaHilbert** (p. **??**) threads.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 template<typename T> shared_data< T >::shared_data ( ) [inline]

Base constructor.

Initialises with invalid default arguments.

5.41.2.2 template<typename T> shared_data< T >::shared_data (size_t *_id*, size_t *_P*, std::vector< Triplet< double > > * *_original*, size_t * *_nzb*, size_t ** *_nzc*, pthread_mutex_t * *_mutex*, pthread_cond_t * *_cond*, pthread_mutex_t * *_end_mutex*, pthread_cond_t * *_end_cond*, size_t * *_sync*, size_t * *_end_sync*, size_t *_m*, const T ** *_in*, T ** *_out*) [inline]

Default constructor.

Parameters

_id	Thread ID.
_P	Number of SPMD processes.
_original	Original set of nonzeroes.
_nzb	Number of sparse blocks.
_nzc	Number of nonzeroes in each sparse block,
_mutex	Sync mutex.
_cond	Sync condition.
_end_mutex	End sync mutex.
_end_cond	End sync condition.
_sync	Sync counter.
_end_sync	End sync counter.
_m	Number of matrix rows.
_in	Input vectors of all SPMD processes.
_out	Output vectors of all SPMD processes.

5.41.3 Member Data Documentation

5.41.3.1 template<typename T> pthread_cond_t* shared_data< T >::cond

Condition used for synchronisation.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.41.3.2 template<typename T> pthread_cond_t* shared_data< T >::end_cond

Condition used for end sync.

Referenced by BetaHilbert< T >::thread().

5.41.3.3 template<typename T> pthread_mutex_t* shared_data< T >::end_mutex

Mutex used for end sync.

Referenced by BetaHilbert< T >::thread().

5.41.3.4 template<typename T> size_t* shared_data< T >::end_sync

Counter used for end sync.

Referenced by BetaHilbert< T >::thread().

5.41.3.5 template<typename T> const T** shared_data< T >::input

Array of all local input vectors of all SPMD processes.

Referenced by BetaHilbert< T >::thread().

5.41.3.6 template<typename T> T* shared_data< T >::local_y

Pointer to the local output vector.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.41.3.7 template<typename T> pthread_mutex_t* shared_data< T >::mutex

Mutex used for synchronisation.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.41.3.8 template<typename T> std::vector< Triplet< T > >* shared_data< T >::original

Array of local sparse blocks.

Referenced by BetaHilbert< T >::thread().

5.41.3.9 template<typename T> T** shared_data< T >::output

Array of all output vectors local to all SPMD processes.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.41.3.10 template<typename T> size_t shared_data< T >::output_vector_offset

Offset of the local output vector compared to global indices.

5.41.3.11 template<typename T> size_t shared_data< T >::output_vector_size

Length of the local output vector.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

5.41.3.12 template<typename T> size_t* shared_data< T >::sync

Counter used for synchronisation.

Referenced by BetaHilbert< T >::collectY(), and BetaHilbert< T >::thread().

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

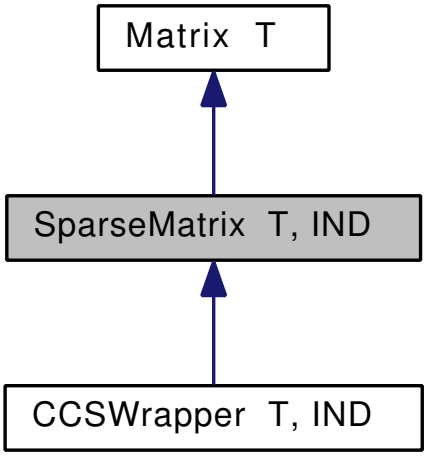
- BetaHilbert.hpp

5.42 SparseMatrix< T, IND > Class Template Reference

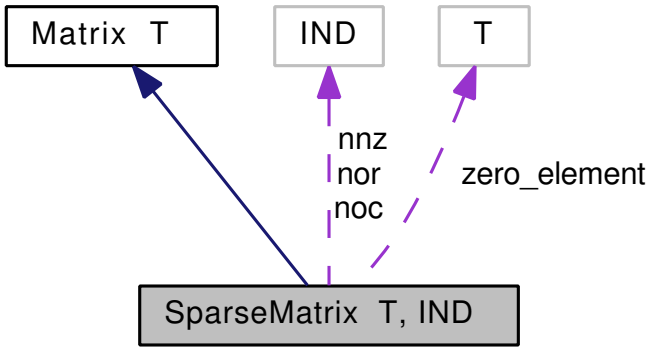
Interface common to all sparse matrix storage schemes.

`#include <SparseMatrix.hpp>`

Inheritance diagram for SparseMatrix< T, IND >:



Collaboration diagram for SparseMatrix< T, IND >:



Public Member Functions

- **SparseMatrix** ()

  Base constructor.
- **SparseMatrix** (const IND **nzs**, const IND nr, const IND nc, const T zero)

  Base constructor.
- virtual ~**SparseMatrix** ()

  Base deconstructor.
- virtual void **load** (std::vector< **Triplet**< T > > &input, const IND **m**, const IND **n**, const T zero)=0

  Function reading in from std::vector< Triplet< T > > format.
- void **loadFromFile** (const std::string file, const T zero=0)

  Function which loads a matrix from a matrix market file.
- virtual unsigned long int **m** ()

  Queries the number of rows this matrix contains.
- virtual unsigned long int **n** ()

  Queries the number of columns this matrix contains.
- virtual unsigned long int **nzs** ()

  Queries the number of nonzeroes stored in this matrix.
- virtual void **getFirstIndexPair** (IND &row, IND &col)=0

  Returns the first nonzero index, per reference.

- virtual T * **mv** (const T *x)

  Calculates and returns z=Ax.
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)=0

  In-place z=Ax function.
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)=0

  In-place z=xA function.

Public Attributes

- T **zero_element**

  The element considered to be zero.

Protected Attributes

- IND **nor**

  Number of rows.
- IND **noc**

  Number of columns.
- IND **nnz**

  Number of non-zeros.

5.42.1 Detailed Description

template<typename T, typename IND>class SparseMatrix< T, IND >

Interface common to all sparse matrix storage schemes.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 template<typename T, typename IND> SparseMatrix< T, IND >::SparseMatrix ( ) [inline]

Base constructor.

5.42.2.2 template<typename T, typename IND> SparseMatrix< T, IND >::SparseMatrix (const IND *nzs,* const IND *nr,* const IND *nc,* const T *zero*) [inline]

Base constructor.

5.42.2.3 template<typename T, typename IND> virtual SparseMatrix< T, IND >::~SparseMatrix ( ) [inline], [virtual]

Base deconstructor.

5.42.3 Member Function Documentation

5.42.3.1 template<typename T, typename IND> virtual void SparseMatrix< T, IND >::getFirstIndexPair (IND & *row,* IND & *col*) [pure virtual]

Returns the first nonzero index, per reference.

Implemented in **BetaHilbert< T >** (p. **??**), **RDBHilbert< T, MatrixType >** (p. **??**), **RDScheme< T, DS >** (p. **??**), **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **RDCSB< T >** (p. **??**), **BlockHilbert< T >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **McCRS< T >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **SVM< T >** (p. **??**), **CRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **DD_MATRIX< T, number_of_diagonals, diagonal_offsets >** (p. **??**), **ZZ_CRS< T, _i_value >** (p. **??**), **HTS< T >** (p. **??**), **Hilbert< T >** (p. **??**), **CuHyb** (p. **??**), **CompressedHilbert< T >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), and **TS< T >** (p. **??**).

5.42.3.2 template<typename T, typename IND> virtual void SparseMatrix< T, IND >::load (std::vector< Triplet< T > > & *input,* const IND *m,* const IND *n,* const T *zero*) [pure virtual]

Function reading in from std::vector< Triplet< T > > format.

Parameters

input	The input matrix in triplet format.
m	The number of rows of the input matrix.
n	The number of columns of the input matrix.
zero	Which element is to be considered zero.

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **BlockHilbert< T >** (p. **??**), **RDBHilbert< T, MatrixType >** (p. **??**), **BetaHilbert< T >** (p. **??**), **RDScheme< T, DS >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **RDCSB< T >** (p. **??**), **McCRS< T >** (p. **??**), **SVM< T >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **CRS< T >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **ZZ_CRS< T, _i_value >** (p. **??**), **HTS< T >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **DD_MATRIX< T, number_of_diagonals, diagonal_offsets >** (p. **??**), **Hilbert< T >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), **CompressedHilbert< T >** (p. **??**), and **TS< T >** (p. **??**).

Referenced by SparseMatrix< double, size_t >::loadFromFile().

5.42.3.3 template<typename T, typename IND> void SparseMatrix< T, IND >::loadFromFile (const std::string *file,* const T *zero* = 0) [inline]

Function which loads a matrix from a matrix market file.

Parameters

file	Filename (including path) to the matrix market file.
zero	Which element is to be considered zero.

5.42.3.4 template<typename T, typename IND> virtual unsigned long int SparseMatrix< T, IND >::m ( ) [inline], [virtual]

Queries the number of rows this matrix contains.

Returns

The number of rows.

Implements **Matrix< T >** (p. **??**).

Reimplemented in **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

Referenced by SparseMatrix< double, size_t >::loadFromFile().

5.42.3.5 template<typename T, typename IND> virtual T* SparseMatrix< T, IND >::mv (const T * x) [inline], [virtual]

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 length equal to the number of rows in this function.

Warning The output vector is aligned to 64-byte boundaries. Free the assigned memory using _mm_free!

Parameters

x	The input (dense) x-vector.

Returns

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

See Also

mv (p. **??**)

Implements **Matrix< T >** (p. **??**).

Reimplemented in **BetaHilbert< T >** (p. **??**), **RDBHilbert< T, MatrixType >** (p. **??**), **RDScheme< T, DS >** (p. **??**), **RDCSB< T >** (p. **??**), **McCRS< T >** (p. **??**), **MKLCRS< T >** (p. **??**), and **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

5.42.3.6 template<typename T, typename IND> virtual unsigned long int SparseMatrix< T, IND >::n ( ) [inline], [virtual]

Queries the number of columns this matrix contains.

Returns

The number of columns.

Implements **Matrix< T >** (p. **??**).

Reimplemented in **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

Referenced by SparseMatrix< double, size_t >::loadFromFile().

5.42.3.7 template<typename T, typename IND> virtual unsigned long int SparseMatrix< T, IND >::nzs ( ) [inline], [virtual]

Queries the number of nonzeroes stored in this matrix.

Returns

The number of nonzeros.

Reimplemented from **Matrix< T >** (p. **??**).

Reimplemented in **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**).

5.42.3.8 template<typename T, typename IND> virtual void SparseMatrix< T, IND >::zax (const T *__restrict__ x, T *__restrict__ z) [pure virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **Matrix< T >** (p. **??**).

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **McCRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **SVM< T >** (p. **??**), **CRS< T >** (p. **??**), **MKLCRS< T >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), and **CuHyb** (p. **??**).

Referenced by SparseMatrix< double, size_t >::mv().

5.42.3.9 template<typename T, typename IND> virtual void SparseMatrix< T, IND >::zxa (const T *__restrict__ x, T *__restrict__ z) [pure virtual]

In-place z=xA function.

Parameters

x	The x vector to apply in left-multiplication to A
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **Matrix< T >** (p. **??**).

Implemented in **vecBICRS< T, block_length_row, block_length_column, _i_value >** (p. **??**), **CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >** (p. **??**), **FBICRS< _t_value, _i_value, _sub_ds, logBeta >** (p. **??**), **ICRS< T, _i_value >** (p. **??**), **HBICRS< _t_value >** (p. **??**), **HBICRS< T >** (p. **??**), **McCRS< T >** (p. **??**), **ZZ_ICRS< T >** (p. **??**), **SVM< T >** (p. **??**), **CRS< T >** (p. **??**), **BICRS< _t_value, _i_value >** (p. **??**), **MKLCRS< T >** (p. **??**), **CCSWrapper< T, SparseMatrixType, IND >** (p. **??**), and **CuHyb** (p. **??**).

5.42.4 Member Data Documentation

5.42.4.1 template<typename T, typename IND> IND SparseMatrix< T, IND >::nnz [protected]

Number of non-zeros.

Referenced by SparseMatrix< double, size_t >::nzs().

5.42.4.2 template<typename T, typename IND> IND SparseMatrix< T, IND >::nor [protected]

Number of rows.

Referenced by SparseMatrix< double, size_t >::m(), and SparseMatrix< double, size_t >::mv().

5.42.4.3 template<typename T, typename IND> T SparseMatrix< T, IND >::zero_element

The element considered to be zero.

Referenced by SparseMatrix< double, size_t >::mv().

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

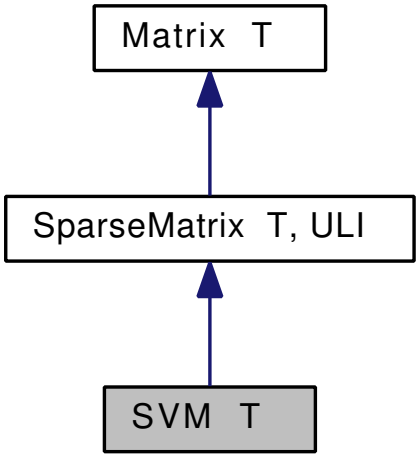
- SparseMatrix.hpp

5.43 SVM< T > Class Template Reference

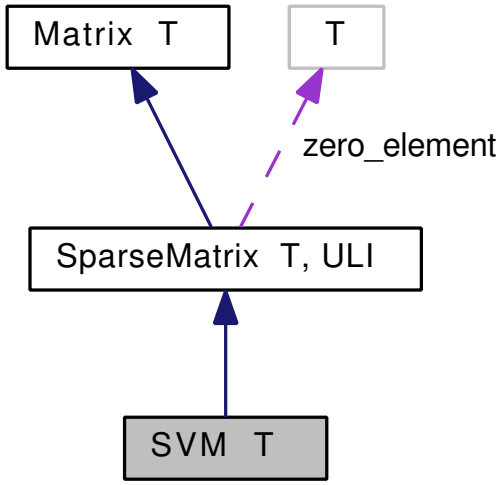
The sparse vector matrix format.

`#include <SVM.hpp>`

Inheritance diagram for SVM< T >:



Collaboration diagram for SVM< T >:



Public Member Functions

- **SVM** ()

  Base constructor.
- **SVM** (std::string file, T zero=0)

  Base constructor.
- **SVM** (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.
- **SVM** (**SVM**< T > &toCopy)

  Copy constructor.
- **SVM** (const std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero, const char direction=0)

*Constructor which transforms a collection of input triplets to **SVM** (p. **??**) format.*

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

  *This will default to row-major **SVM** (p. **??**) format.*
- void **load** (const std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero, const char direction)

  *Will load a collection of triplets into **SVM** (p. **??**) format.*
- std::vector< std::vector < **Triplet**< double > > > & **getData** ()
- T & **random_access** (ULI i, ULI j)

  Method which provides random matrix access to the stored sparse matrix.
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=xA function.
- virtual void **zax** (const T *__restrict__ x, T *__restrict__ z)

  In-place z=Ax function.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.
- ~**SVM** ()

  Base deconstructor.

Protected Member Functions

- bool **find** (const ULI col_index, const ULI row_index, **Triplet**< double > &ret)

  Helper function which finds a value with a given index.

Static Protected Member Functions

- static int **compareTripletsR** (const void *left, const void *right)

  Sorts 1D columnwise.
- static int **compareTripletsC** (const void *left, const void *right)

  Sorts 1D rowwise.

Protected Attributes

- char **major**

  Row major (0), column major (1)
- std::vector< std::vector < **Triplet**< T > > > **ds**

  ***SVM** (p. **??**) structure.*

Additional Inherited Members

5.43.1 Detailed Description

template<typename T>class SVM< T >

The sparse vector matrix format.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 template<typename T> SVM< T >::SVM ( ) [inline]

Base constructor.

5.43.2.2 template<typename T> SVM< T >::SVM (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.43.2.3 template<typename T> SVM< T >::SVM (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.

Parameters

number_of_nonzeros	The number of non-zeros to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns to be stored.
zero	Which element is considered to be the zero element.

5.43.2.4 template<typename T> SVM< T >::SVM (SVM< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **CRS** (p. **??**) datastructure to copy.

References SVM< T >::ds, SVM< T >::major, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and SparseMatrix< T, ULI >::zero_element.

5.43.2.5 template<typename T> SVM< T >::SVM (const std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero,* const char *direction* = 0) [inline]

Constructor which transforms a collection of input triplets to **SVM** (p. **??**) 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	Total number of rows.

n	Total number of columns.
zero	Which element is considered zero.
direction	0 for row-major, 1 for column major.

References SVM< T >::load().

5.43.2.6 template<typename T> SVM< T >::~SVM ( ) [inline]

Base deconstructor.

5.43.3 Member Function Documentation

5.43.3.1 template<typename T> virtual size_t SVM< T >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References SparseMatrix< T, ULI >::nnz, and SparseMatrix< T, ULI >::nor.

5.43.3.2 template<typename T> bool SVM< T >::find (const ULI *col_index,* const ULI *row_index,* Triplet< double > & *ret*) [inline], [protected]

Helper function which finds a value with a given index.

Parameters

col_index	The given column index.
row_index	The given row index.
ret	Reference to the variable where the return triplet is stored (if found).

Returns

Whether or not a non-zero value was found.

References SVM< T >::ds, Triplet< T >::i(), Triplet< T >::j(), and SVM< T >::major.

Referenced by SVM< T >::random_access().

5.43.3.3 template<typename T> std::vector< std::vector< Triplet< double > > >& SVM< T >::getData ( ) [inline]

Returns

Direct access to the **SVM** (p. **??**) datastructure.

References SVM< T >::ds.

5.43.3.4 template<typename T> virtual void SVM< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References SVM< T >::ds.

5.43.3.5 template<typename T> virtual void SVM< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

This will default to row-major **SVM** (p. **??**) format.

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

Referenced by SVM< T >::SVM().

5.43.3.6 template<typename T> void SVM< T >::load (const std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero,* const char *direction*) [inline]

Will load a collection of triplets into **SVM** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	Which element is considered zero.
direction	0 for row-major, 1 for column major.

References SVM< T >::compareTripletsC(), SVM< T >::compareTripletsR(), SVM< T >::ds, Triplet< T >::i(), Triplet< T >::j(), SparseMatrix< T, ULI >::m(), SVM< T >::major, SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, Triplet< T >::value, and SparseMatrix< T, ULI >::zero_element.

5.43.3.7 template<typename T> T& SVM< 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 (p. **??**) valuei at (i,j).

References SVM< T >::find(), Triplet< T >::value, and SparseMatrix< T, ULI >::zero_element.

5.43.3.8 template<typename T> virtual void SVM< T >::zax (const T *__restrict__ *x,* T *__restrict__ *z*) [inline], [virtual]

In-place z=Ax function.

Parameters

x	The x vector to multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References SVM< T >::ds, Triplet< T >::i(), Triplet< T >::j(), SVM< T >::major, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and Triplet< T >::value.

5.43.3.9 template<typename T> virtual void SVM< T >::zxa (const T *__restrict__ x, T *__restrict__ z) [inline], [virtual]

In-place z=xA function.

Parameters

x	The x vector to left-multiply current matrix with.
z	The result vector. Must be pre-allocated and its elements should be initialised to zero.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References SVM< T >::ds, Triplet< T >::i(), Triplet< T >::j(), SVM< T >::major, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and Triplet< T >::value.

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

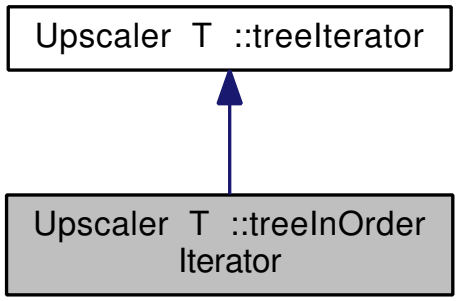
- SVM.hpp

5.44 Upscaler< T >::treeInOrderIterator Class Reference

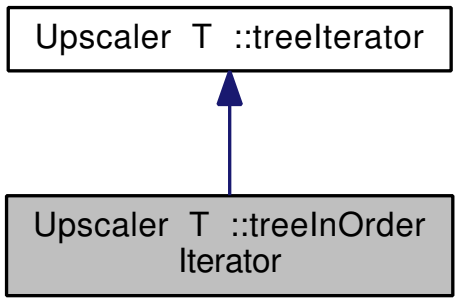
Same as **treeIterator** (p. **??**), but does in-order traversal instead of pre-order.

`#include <Upscaler.hpp>`

Inheritance diagram for Upscaler< T >::treeInOrderIterator:



Collaboration diagram for Upscaler< T >::treeInOrderIterator:



Public Member Functions

- **treeInOrderIterator** (**Upscaler**< T > *_p, const unsigned long int initial, std::vector< bool > *_pp=NULL)

  Base constructor.
- virtual bool **next** ()

  Moves iterator to its next position.

Additional Inherited Members

5.44.1 Detailed Description

template<typename T>class Upscaler< T >::treeInOrderIterator

Same as **treeIterator** (p. **??**), but does in-order traversal instead of pre-order.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 template<typename T> Upscaler< T >::treeInOrderIterator::treeInOrderIterator (Upscaler< T > * *_p*, const unsigned long int *initial*, std::vector< bool > * *_pp* = NULL) [inline]

Base constructor.

See Also

treeIterator::treeIterator (p. **??**).

Parameters

_p	Iterates on this object.
initial	Initial position.
_pp	Buffer where the visited subtrees may be stored.

References Upscaler< T >::treeIterator::ID, Upscaler< T >::treeInOrderIterator::next(), and Upscaler< T >::treeIterator::p_processed.

5.44.3 Member Function Documentation

5.44.3.1 template<typename T> virtual bool Upscaler< T >::treeInOrderIterator::next ( ) [inline], [virtual]

Moves iterator to its next position.

Returns

Whether there was a next position available.

Reimplemented from **Upscaler< T >::treeIterator** (p. **??**).

References SBDTree::NO_SUCH_NODE, Upscaler< T >::treeIterator::p, Upscaler< T >::treeIterator::p_processed, and Upscaler< T >::treeIterator::walk.

Referenced by Upscaler< T >::getSubTree(), and Upscaler< T >::treeInOrderIterator::treeInOrderIterator().

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

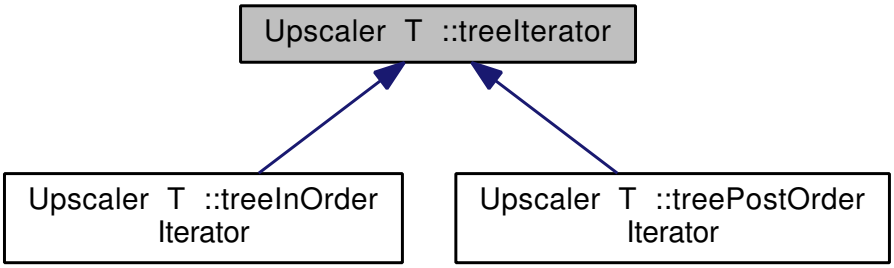
- Upscaler.hpp

5.45 Upscaler< T >::treeIterator Class Reference

Pre-order tree iterator.

`#include <Upscaler.hpp>`

Inheritance diagram for Upscaler< T >::treeIterator:



Public Member Functions

- **treeIterator** (**Upscaler**< T > *_p, const unsigned long int initial, std::vector< bool > *_pp=NULL)
  Base constructor.
- unsigned long int **position** ()
- virtual bool **next** ()

Protected Attributes

- unsigned long int **walk**
  Current position of the iterator.
- unsigned long int **ID**
  Starting position of the iterator.
- std::vector< bool > **processed**
  Processed[i] is true when this iterator has visited its i-th child.
- std::vector< bool > * **p_processed**
  Pointer to the 'processed'-vector actually used.
- **Upscaler**< T > * **p**
  Class this iterator works on.

5.45.1 Detailed Description

template<typename T>class Upscaler< T >::treeIterator

Pre-order tree iterator.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 template<typename T> Upscaler< T >::treeIterator::treeIterator (Upscaler< T > * *_p*, const unsigned long int *initial*, std::vector< bool > * *_pp* = NULL) [inline]

Base constructor.

Parameters

_p	Pointer to the base class
initial	Initial position in the tree.
_pp	Pointer to a vector where the visited subtrees may be stored.

References Upscaler< T >::treeIterator::ID, Upscaler< T >::treeIterator::p, Upscaler< T >::treeIterator::p_processed, and Upscaler< T >::treeIterator::processed.

5.45.3 Member Function Documentation

5.45.3.1 template<typename T> virtual bool Upscaler< T >::treeIterator::next ( ) [inline], [virtual]

Returns

Returns false if there is no next element.

Reimplemented in **Upscaler< T >::treePostOrderIterator** (p. **??**), and **Upscaler< T >::treeInOrderIterator** (p. **??**).

References Upscaler< T >::treeIterator::ID, SBDTree::NO_SUCH_NODE, Upscaler< T >::treeIterator::p, Upscaler< T >::treeIterator::p_processed, and Upscaler< T >::treeIterator::walk.

Referenced by Upscaler< T >::determineMinMax(), and Upscaler< T >::getSubTree().

5.45.3.2 template<typename T> unsigned long int Upscaler< T >::treeIterator::position ( ) [inline]

Returns

Returns the current position.

References Upscaler< T >::treeIterator::walk.

Referenced by Upscaler< T >::determineMinMax(), and Upscaler< T >::getSubTree().

5.45.4 Member Data Documentation

5.45.4.1 template<typename T> unsigned long int Upscaler< T >::treeIterator::ID [protected]

Starting position of the iterator.

Referenced by Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::treeInOrderIterator(), Upscaler< T >::treeIterator::treeIterator(), and Upscaler< T >::treePostOrderIterator::treePostOrderIterator().

5.45.4.2 template<typename T> Upscaler< T >* Upscaler< T >::treeIterator::p [protected]

Class this iterator works on.

Referenced by Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::next(), Upscaler< T >::treePostOrderIterator::next(), and Upscaler< T >::treeIterator::treeIterator().

5.45.4.3 template<typename T> std::vector< bool >* Upscaler< T >::treeIterator::p_processed [protected]

Pointer to the 'processed'-vector actually used.

Can be a pointer to **treeIterator::processed** (p. **??**), or a user-defined vector instead.

Referenced by Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::next(), Upscaler< T >::treePostOrderIterator::next(), Upscaler< T >::treeInOrderIterator::treeInOrderIterator(), Upscaler< T >::treeIterator::treeIterator(), and Upscaler< T >::treePostOrderIterator::treePostOrderIterator().

5.45.4.4 template<typename T> std::vector< bool > Upscaler< T >::treeIterator::processed [protected]

Processed[i] is true when this iterator has visited its i-th child.

May be unused if a pre-allocated vector is given to the iterator instead.

Referenced by Upscaler< T >::treeIterator::treeIterator().

5.45.4.5 template<typename T> unsigned long int Upscaler< T >::treeIterator::walk [protected]

Current position of the iterator.

Referenced by Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::next(), Upscaler< T >::treePostOrderIterator::next(), and Upscaler< T >::treeIterator::position().

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

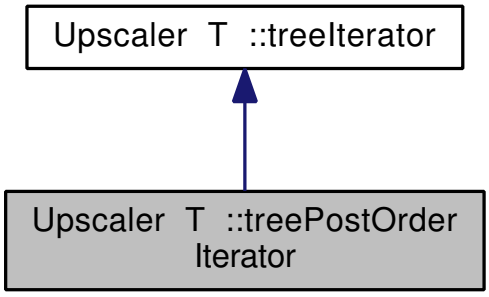
- Upscaler.hpp

5.46 Upscaler< T >::treePostOrderIterator Class Reference

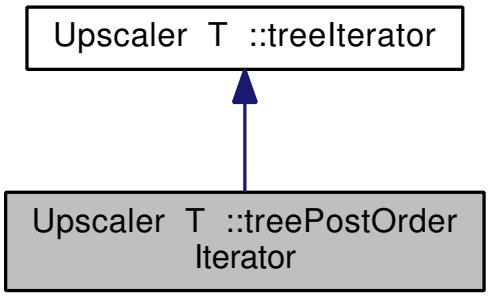
Same as **treeIterator** (p. **??**), but does post-order traversal instead of pre-order.

`#include <Upscaler.hpp>`

Inheritance diagram for Upscaler< T >::treePostOrderIterator:



Collaboration diagram for Upscaler< T >::treePostOrderIterator:



Public Member Functions

- **treePostOrderIterator** (**Upscaler**< T > *_p, const unsigned long int initial, std::vector< bool > *_pp=NULL)

  Base constructor.

- virtual bool **next** ()

  Moves iterator to next position.

Additional Inherited Members

5.46.1 Detailed Description

template<typename T>class Upscaler< T >::treePostOrderIterator

Same as **treeIterator** (p. **??**), but does post-order traversal instead of pre-order.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 template<typename T> Upscaler< T >::treePostOrderIterator::treePostOrderIterator (Upscaler< T > * *_p*, const unsigned long int *initial*, std::vector< bool > * *_pp* = NULL) [inline]

Base constructor.

See Also

treeIterator::treeIterator (p. **??**).

Parameters

_p	Iterates on this object.
initial	Initial position.
_pp	Keeps track of which children are already visited.

References Upscaler< T >::treeIterator::ID, Upscaler< T >::treePostOrderIterator::next(), and Upscaler< T >::treeIterator::p_processed.

5.46.3 Member Function Documentation

5.46.3.1 template<typename T> virtual bool Upscaler< T >::treePostOrderIterator::next ( ) [inline], [virtual]

Moves iterator to next position.

Returns

Returns false if there is no next element.

Reimplemented from **Upscaler< T >::treeIterator** (p. **??**).

References SBDTree::NO_SUCH_NODE, Upscaler< T >::treeIterator::p, Upscaler< T >::treeIterator::p_processed, and Upscaler< T >::treeIterator::walk.

Referenced by Upscaler< T >::getSubTree(), and Upscaler< T >::treePostOrderIterator::treePostOrderIterator().

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

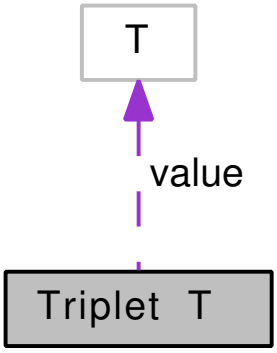
- Upscaler.hpp

5.47 Triplet< T > Class Template Reference

A single triplet value.

`#include <Triplet.hpp>`

Collaboration diagram for Triplet< T >:



Public Member Functions

- ULI **i** () const
- ULI **j** () const
- void **rowOffset** (ULI offset)

  Subtracts a given value from this nonzero row index.
- void **setPosition** (const unsigned long int **i**, const unsigned long int **j**)

  Set the coordinates of this nonzero.
- void **setRowPosition** (const unsigned long int **i**)

  Set the row coordinate of this nonzero.
- void **setColumnPosition** (const unsigned long int **j**)

  Set the column coordinate of this nonzero.
- **Triplet** (ULI ii, ULI ij, T val)

  Base constructor.
- **Triplet** ()

  Base constructor.
- **Triplet** (const **Triplet**< T > &toCopy)

  Copy constructor.
- void **transpose** ()

  Transposes this triplet, i.e., swapping the row and column value.

Static Public Member Functions

- static std::vector< **Triplet**< T > > **load** (const std::string fn, ULI &m, ULI &n)

  Loads an array of triplets from a binary file.
- static std::vector< **Triplet**< T > > **loadCRS** (const std::string fn, ULI &m, ULI &n)

  *Loads a **CRS** (p. **??**) text file and transforms it into a vector of Triplets.*
- static void **save** (std::string fn, **Triplet**< T > *toWrite, const ULI m, const ULI n, const ULI s)

  Saves an array of triplets to a file, in binary format.
- static void **save** (std::string fn, std::vector< **Triplet**< T > > &toWrite, const ULI m, const ULI n)

  Saves a std::vector of triplets to a file, in binary format.

Public Attributes

- T **value**

  Value stored at this triplet.

Protected Attributes

- ULI **row**

  The row coordinate of this triplet.
- ULI **column**

  The column coordinate of this triplet.

5.47.1 Detailed Description

template<typename T>class Triplet< T >

A single triplet value.

Stores, at minimum, a row position, a column position, and a value of the template type.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 template<typename T> Triplet< T >::Triplet (ULI *ii*, ULI *ij*, T *val*) [inline]

Base constructor.

Parameters

ii	row index.
ij	column index.
val	nonzero value.

5.47.2.2 template<typename T> Triplet< T >::Triplet ( ) [inline]

Base constructor.

Sets all values to zero.

5.47.2.3 template<typename T> Triplet< T >::Triplet (const Triplet< T > & *toCopy*) [inline]

Copy constructor.

5.47.3 Member Function Documentation

5.47.3.1 template<typename T> ULI Triplet< T >::i ( ) const [inline]

Returns

Row index of this triplet.

References Triplet< T >::row.

Referenced by ICRS< T, _i_value >::compareTriplets(), ZZ_ICRS< T >::compareTriplets(), vecBICRS< T, block_length_row, block_length_column, _i_value >::compareTriplets(), SVM< T >::compareTripletsC(), ZZ_CRS< T, _i_value >::compareTripletsLTR(), ZZ_CRS< T, _i_value >::compareTripletsRTL(), SVM< T >::find(), BlockOrderer< T >::left_horizontal(), Triplet< T >::load(), ZZ_CRS< T, _i_value >::load(), ZZ_ICRS< T >::load(), ICRS< T, _i_value >::load(), SVM< T >::load(), Triplet< T >::loadCRS(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::right_horizontal(), Triplet< T >::save(), Triplet< T >::setPosition(), BetaHilbert< T >::thread(), BlockOrderer< T >::upper_vertical(), SVM< T >::zax(), and SVM< T >::zxa().

5.47.3.2 template<typename T> ULI Triplet< T >::j ( ) const [inline]

Returns

Column index of this triplet.

References Triplet< T >::column.

Referenced by CuHyb::compareTriplets(), CRS< T >::compareTriplets(), McCRS< T >::compareTriplets(), ICRS< T, _i_value >::compareTriplets(), ZZ_ICRS< T >::compareTriplets(), vecBICRS< T, block_length_row, block_length_column, _i_value >::compareTriplets(), ZZ_CRS< T, _i_value >::compareTripletsLTR(), SVM< T >::compareTripletsR(), ZZ_CRS< T, _i_value >::compareTripletsRTL(), SVM< T >::find(), BlockOrderer< T >::left_horizontal(), CuHyb::load(), Triplet< T >::load(), ZZ_CRS< T, _i_value >::load(), ZZ_ICRS< T >::load(), CRS< T >::load(), ICRS< T, _i_value >::load(), McCRS< T >::load(), SVM< T >::load(), Triplet< T >::loadCRS(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), vecBICRS< T, block_length_row, block_length_column, _i_value >::pColumnSort(), vecBICRS< T, block_length_row, block_length_column, _i_value >::pRowSort(), BlockOrderer< T >::right_horizontal(), Triplet< T >::save(), Triplet< T >::setPosition(), BlockOrderer< T >::upper_vertical(), SVM< T >::zax(), and SVM< T >::zxa().

5.47.3.3 template<typename T> static std::vector< Triplet< T > > Triplet< T >::load (const std::string *fn,* ULI & *m,* ULI & *n*) [inline], [static]

Loads an array of triplets from a binary file.

Warning: there is a difference between 32 and 64 bits files!

Parameters

fn	Filename of the file to load from.
m	Reference to where the total number of rows is to-be stored.
n	Reference to where the total number of columns is to-be stored.

Returns

A std::vector containing the triplets in the order they were loaded in.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by Hilbert< T >::loadBinary(), HTS< T >::loadBinary(), and SparseMatrix< double, size_t >::loadFromFile().

5.47.3.4 template<typename T> static std::vector< Triplet< T > > Triplet< T >::loadCRS (const std::string *fn,* ULI & *m,* ULI & *n*) [inline], [static]

Loads a **CRS** (p. **??**) text file and transforms it into a vector of Triplets.

Parameters

fn	Filename of the file to load from.
m	Reference to where to store the number of rows of this matrix.
n	Reference to where to store the number of columns of this matrix.

Returns

A std::vector containing the read and converted triplets.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by SparseMatrix< double, size_t >::loadFromFile().

5.47.3.5 template<typename T> void Triplet< T >::rowOffset (ULI *offset*) [inline]

Subtracts a given value from this nonzero row index.

References Triplet< T >::row.

5.47.3.6 template<typename T> static void Triplet< T >::save (std::string *fn,* Triplet< T > * *toWrite,* const ULI *m,* const ULI *n,* const ULI *s*) [inline], [static]

Saves an array of triplets to a file, in binary format.

Parameters

fn	Filename to save to (overwrite mode).
toWrite	Array of triplets to write.
m	Total number of rows in the matrix.+
n	Total number of columns in the matrix.
s	Size of the array toWrite.

References Triplet< T >::i(), Triplet< T >::j(), and Triplet< T >::value.

Referenced by Triplet< T >::save().

5.47.3.7 template<typename T> static void Triplet< T >::save (std::string *fn,* std::vector< Triplet< T > > & *toWrite,* const ULI *m,* const ULI *n*) [inline], [static]

Saves a std::vector of triplets to a file, in binary format.

Parameters

fn	Filename to save to (overwrite mode).
toWrite	Vector of triplets to write.
m	Total number of rows in the matrix.
n	Total number of columns in the matrix.

References Triplet< T >::save().

5.47.3.8 template<typename T> void Triplet< T >::setColumnPosition (const unsigned long int *j*) [inline]

Set the column coordinate of this nonzero.

References Triplet< T >::row, and Triplet< T >::setPosition().

5.47.3.9 template<typename T> void Triplet< T >::setPosition (const unsigned long int *i,* const unsigned long int *j*) [inline]

Set the coordinates of this nonzero.

References Triplet< T >::column, Triplet< T >::i(), Triplet< T >::j(), and Triplet< T >::row.

Referenced by Triplet< T >::setColumnPosition(), and Triplet< T >::setRowPosition().

5.47.3.10 template<typename T> void Triplet< T >::setRowPosition (const unsigned long int *i*) [inline]

Set the row coordinate of this nonzero.

References Triplet< T >::column, and Triplet< T >::setPosition().

5.47.3.11 template<typename T> void Triplet< T >::transpose ( ) [inline]

Transposes this triplet, i.e., swapping the row and column value.

References Triplet< T >::column, and Triplet< T >::row.

5.47.4 Member Data Documentation

5.47.4.1 template<typename T> ULI Triplet< T >::column [protected]

The column coordinate of this triplet.

Referenced by Triplet< T >::j(), Triplet< T >::setPosition(), Triplet< T >::setRowPosition(), and Triplet< T >::transpose().

5.47.4.2 template<typename T> ULI Triplet< T >::row [protected]

The row coordinate of this triplet.

Referenced by Triplet< T >::i(), Triplet< T >::rowOffset(), Triplet< T >::setColumnPosition(), Triplet< T >::setPosition(), and Triplet< T >::transpose().

5.47.4.3 template<typename T> T Triplet< T >::value

Value stored at this triplet.

Referenced by CuHyb::load(), ZZ_CRS< T, _i_value >::load(), ZZ_ICRS< T >::load(), CRS< T >::load(), ICRS< T, _i_value >::load(), McCRS< T >::load(), SVM< T >::load(), SVM< T >::random_access(), Triplet< T >::save(), SVM< T >::zax(), and SVM< T >::zxa().

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

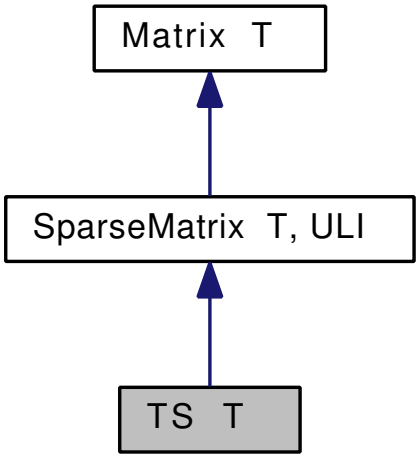
- Triplet.hpp

5.48 TS< T > Class Template Reference

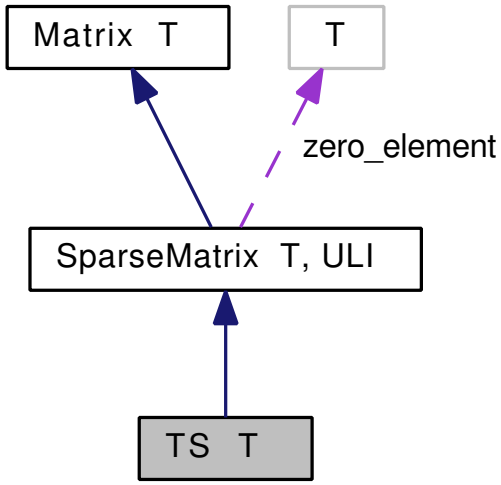
The triplet scheme; a storage scheme for sparse matrices using triplets.

`#include <TS.hpp>`

Inheritance diagram for TS< T >:



Collaboration diagram for TS< T >:



Public Member Functions

- **TS** ()

  Base constructor.
- **TS** (std::string file, T zero=0)

  Base constructor.
- **TS** (std::vector< **Triplet**< T > > &input, ULI **m**, ULI **n**, T zero=0)

  Base constructor.
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *x, T *z)

  In-place z=xA calculation algorithm.
- virtual void **zax** (const T *x, T *z)

  In-place z=Ax calculation algorithm.
- template<size_t k>
  void **ZaX** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- template<size_t k>
  void **ZXa** (const T *__restrict__ const *__restrict__ const X, T *__restrict__ const *__restrict__ const Z)
- virtual size_t **bytesUsed** ()
- ~**TS** ()

  Base destructor.

Protected Attributes

- ULI * **i**

  The row indices of the nonzeros.
- ULI * **j**

  The column indices of the nonzeros.
- T * **ds**

  The values of the nonzeros.

Additional Inherited Members

5.48.1 Detailed Description

template<typename T>class TS< T >

The triplet scheme; a storage scheme for sparse matrices using triplets.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 template<typename T> TS< T >::TS ( ) [inline]

Base constructor.

5.48.2.2 template<typename T> TS< T >::TS (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.48.2.3 template<typename T> TS< T >::TS (std::vector< Triplet< T > > & *input,* ULI *m,* ULI *n,* T *zero* = 0) [inline]

Base constructor.

Parameters

input	std::vector of triplets to be stored in this scheme.
m	total number of rows.
n	total number of columns.
zero	what is to be considered the zero element.

References TS< T >::load().

5.48.2.4 template<typename T> TS< T >::~TS ( ) [inline]

Base destructor.

References TS< T >::ds, TS< T >::i, and TS< T >::j.

5.48.3 Member Function Documentation

5.48.3.1 template<typename T> virtual size_t TS< T >::bytesUsed ( ) [inline], [virtual]

See Also

Matrix::bytesUsed (p. **??**)

Implements **Matrix**< **T** > (p. **??**).

References SparseMatrix< T, ULI >::nnz.

5.48.3.2 template<typename T> virtual void TS< T >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References TS< T >::i, and TS< T >::j.

5.48.3.3 template<typename T> virtual void TS< T >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References TS< T >::ds, TS< T >::i, TS< T >::j, SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and SparseMatrix< T, ULI >::zero_element.

Referenced by TS< T >::TS().

5.48.3.4 template<typename T> virtual void TS< T >::zax (const T * *x,* T * *z*) [inline], [virtual]

In-place z=Ax calculation algorithm.

Parameters

x	The vector x supplied for calculation of Ax.
z	The result vector z. Should be pre-allocated with entries set to zero.

References TS< T >::ds, TS< T >::i, TS< T >::j, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, and SparseMatrix< T, ULI >::nor.

5.48.3.5 template<typename T> template<size_t k> void TS< T >::ZaX (const T *__restrict__ const *__restrict__ const *X,* T *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZaX (p. **??**)

References TS< T >::ds, TS< T >::i, TS< T >::j, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, and SparseMatrix< T, ULI >::nor.

5.48.3.6 template<typename T> virtual void TS< T >::zxa (const T * *x,* T * *z*) [inline], [virtual]

In-place z=xA calculation algorithm.

Parameters

x	The vector x supplied for calculation of xA.
z	The result vector z. Should be pre-allocated with entries set to zero.

References TS< T >::ds, TS< T >::i, TS< T >::j, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, and SparseMatrix< T, ULI >::nor.

5.48.3.7 template<typename T> template<size_t k> void TS< T >::ZXa (const T *__restrict__ const *__restrict__ const *X,* T *__restrict__ const *__restrict__ const *Z*) [inline]

See Also

Matrix::ZXa (p. **??**)

References TS< T >::ds, TS< T >::i, TS< T >::j, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, and SparseMatrix< T, ULI >::nor.

5.48.4 Member Data Documentation

5.48.4.1 template<typename T> T* TS< T >::ds [protected]

The values of the nonzeros.

Referenced by TS< T >::load(), TS< T >::zax(), TS< T >::ZaX(), TS< T >::zxa(), TS< T >::ZXa(), and TS< T >::~TS().

5.48.4.2 template<typename T> ULI* TS< T >::i [protected]

The row indices of the nonzeros.

Referenced by TS< T >::getFirstIndexPair(), TS< T >::load(), TS< T >::zax(), TS< T >::ZaX(), TS< T >::zxa(), TS< T >::ZXa(), and TS< T >::~TS().

5.48.4.3 template<typename T> ULI* TS< T >::j [protected]

The column indices of the nonzeros.

Referenced by TS< T >::getFirstIndexPair(), TS< T >::load(), TS< T >::zax(), TS< T >::ZaX(), TS< T >::zxa(), TS< T >::ZXa(), and TS< T >::~TS().

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

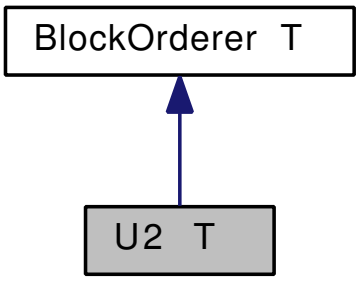
- TS.hpp

5.49 U2< T > Class Template Reference

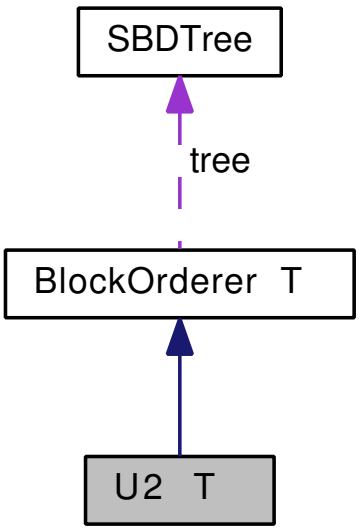
Codes the Minimal CCS block order.

`#include <U2.hpp>`

Inheritance diagram for U2< T >:



Collaboration diagram for U2< T >:



Protected Member Functions

- virtual void **pre_readout** (const unsigned long int index)
  Prefix operations during SBD tree traversal.
- virtual void **in_readout** (const unsigned long int index)
  Infix operations during SBD tree traversal.
- virtual void **post_readout** (const unsigned long int index)
  Postfix operations during SBD tree traversal.

Additional Inherited Members

5.49.1 Detailed Description

template<typename T>class U2< T >

Codes the Minimal CCS block order.

5.49.2 Member Function Documentation

5.49.2.1 template<typename T > virtual void U2< T >::in_readout (const unsigned long int *index*) `[inline], [protected], [virtual]`

Infix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::left_horizontal(), BlockOrderer< T >::lower_vertical(), BlockOrderer< T >::middle(), BlockOrderer< T >::output, BlockOrderer< T >::tree, and BlockOrderer< T >::upper_vertical().

5.49.2.2 template<typename T > virtual void U2< T >::post_readout (const unsigned long int *index*) `[inline], [protected], [virtual]`

Postfix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, BlockOrderer< T >::right_horizontal(), and BlockOrderer< T >::tree.

5.49.2.3 template<typename T > virtual void U2< T >::pre_readout (const unsigned long int *index*) [inline], [protected], [virtual]

Prefix operations during SBD tree traversal.

Implements **BlockOrderer< T >** (p. **??**).

References BlockOrderer< T >::datatype, SBDTree::isLeaf(), BlockOrderer< T >::items, BlockOrderer< T >::output, and BlockOrderer< T >::tree.

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

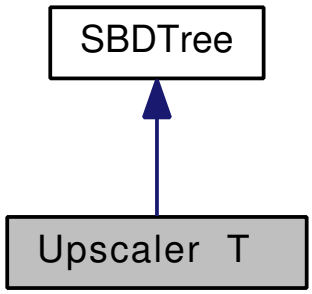
- U2.hpp

5.50 Upscaler< T > Class Template Reference

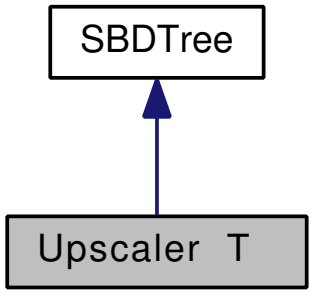
Transforms SBD-structures over q parts into an SBD structure over p parts, with q>p, and q,p both powers of two.

`#include <Upscaler.hpp>`

Inheritance diagram for Upscaler< T >:



Collaboration diagram for Upscaler< T >:



Classes

- class **treeInOrderIterator**

  *Same as **treeIterator** (p. **??**), but does in-order traversal instead of pre-order.*
- class **treeIterator**

Pre-order tree iterator.

- class **treePostOrderIterator**

  *Same as **treeIterator** (p. **??**), but does post-order traversal instead of pre-order.*

Public Member Functions

- **Upscaler** (const std::vector< **Triplet**< T > > &**nonzeroes**, const unsigned long int P, std::vector< unsigned long int > &r_hierarchy, std::vector< unsigned long int > &c_hierarchy, std::vector< unsigned long int > &r_bounds, std::vector< unsigned long int > &c_bounds, unsigned long int *row_perm=NULL, unsigned long int *col_perm=NULL, unsigned long int *proc2proc=NULL)

  Base constructor.

- ~**Upscaler** ()

  Base deconstructor.

- void **getSubTree** (const unsigned long int s, std::vector< **Triplet**< T > > &local_nonzeroes, std::vector< **Triplet**< T > > &remote_nonzeroes, std::vector< unsigned long int > &upscaled_hierarchy, std::vector< unsigned long int > &upscaled_row_bounds, std::vector< unsigned long int > &upscaled_column_bounds, std::vector< unsigned long int > &rowLocalToGlobal, std::vector< unsigned long int > &columnLocalToGlobal, std::map< unsigned long int, unsigned long int > &rowGlobalToLocal, std::map< unsigned long int, unsigned long int > &colGlobalToLocal, const unsigned long int P, const unsigned long int Pref)

  Reads out a subtree corresponding to only the nonzeroes owned by processor s, and returns the upscaled version.

- void **readout** ()

  Reads out SBD data and prints to std::cout.

Protected Member Functions

- void **determineEmpty** (const std::vector< **Triplet**< T > > &**nonzeroes**, const unsigned long int s, const unsigned long int min_i, const unsigned long int min_j, std::vector< bool > &emptyRows, std::vector< bool > &emptyCols, bool &empty)

  Determines, given a collection of nonzeroes, which rows and columns nonzeroes owned by s reside on, and flags these rows and columns used in the appropriate vectors.

- void **updateMinMax** (const unsigned long int walk, const unsigned long int s, unsigned long int &min_i, unsigned long int &max_i, unsigned long int &min_j, unsigned long int &max_j)

  Determine min/max of nonzeroes owned by processor s inside node walk.

- void **determineMinMax** (const unsigned long int ID, const unsigned long int s, unsigned long int &min_i, unsigned long int &max_i, unsigned long int &min_j, unsigned long int &max_j)

  Determine min/max of nonzeroes owned by processor s, contained in the subtree with root ID, as well as all separators on the path from ID to the true root.

- void **addNonzeroes** (std::vector< **Triplet**< T > > &into, const unsigned long int from, const unsigned long int s, const std::map< unsigned long int, unsigned long int > &rowGlobalToLocal, const std::map< unsigned long int, unsigned long int > &colGlobalToLocal)

  adds nonzeroes from a given node into a given vector

- void **getSubTree** (const unsigned long int ID, const unsigned long int s, std::vector< **Triplet**< T > > &local_nonzeroes, std::vector< **Triplet**< T > > &remote_nonzeroes, std::vector< unsigned long int > &upscaled_hierarchy, std::vector< unsigned long int > &upscaled_row_bounds, std::vector< unsigned long int > &upscaled_column_bounds, std::vector< unsigned long int > &rowLocalToGlobal, std::vector< unsigned long int > &columnLocalToGlobal, std::map< unsigned long int, unsigned long int > &rowGlobalToLocal, std::map< unsigned long int, unsigned long int > &colGlobalToLocal)

  Reads out a subtree and returns the upscaled version.

Protected Attributes

- std::vector< std::vector < **Triplet**< T > > > **nonzeroes**

All nonzeroes, stored block-by-block.

- std::vector< std::vector< bool > > **containsPID**

  Keeps track which processes are represented in which blocks.

Additional Inherited Members

5.50.1 Detailed Description

template<typename T>class Upscaler< T >

Transforms SBD-structures over q parts into an SBD structure over p parts, with q>p, and q,p both powers of two.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 template<typename T> Upscaler< T >::Upscaler (const std::vector< Triplet< T > > & *nonzeroes,* const unsigned long int *P,* std::vector< unsigned long int > & *r_hierarchy,* std::vector< unsigned long int > & *c_hierarchy,* std::vector< unsigned long int > & *r_bounds,* std::vector< unsigned long int > & *c_bounds,* unsigned long int * *row_perm* = NULL, unsigned long int * *col_perm* = NULL, unsigned long int * *proc2proc* = NULL) [inline]

Base constructor.

References SBDTree::c_hi, SBDTree::c_lo, Upscaler< T >::containsPID, Upscaler< T >::nonzeroes, SBDTree::parent, SBDTree::r_hi, SBDTree::r_lo, SBDTree::root, and SBDTree::size().

5.50.2.2 template<typename T> Upscaler< T >::~Upscaler ( ) [inline]

Base deconstructor.

5.50.3 Member Function Documentation

5.50.3.1 template<typename T> void Upscaler< T >::determineEmpty (const std::vector< Triplet< T > > & *nonzeroes,* const unsigned long int *s,* const unsigned long int *min_i,* const unsigned long int *min_j,* std::vector< bool > & *emptyRows,* std::vector< bool > & *emptyCols,* bool & *empty*) [inline],[protected]

Determines, given a collection of nonzeroes, which rows and columns nonzeroes owned by s reside on, and flags these rows and columns used in the appropriate vectors.

Parameters

nonzeroes	The collection of nonzeroes.
s	The process ID.
min_i	Minimum row index to consider (other nonzeroes are ignored).
min_j	Minimum column index to consider (other nonzeroes are ignored).
emptyRows	emptyRows[i] is true when no nonzeroes on row i are encountered.
emptyCols	emptyCols[j] is true when no nonzeroes on column j exist.
empty	Will be true when process s has no nonzeroes in this collection.

References Upscaler< T >::nonzeroes.

Referenced by Upscaler< T >::getSubTree().

5.50.3.2 template<typename T> void **Upscaler**< T >::getSubTree (const unsigned long int *ID,* const unsigned long int *s,* std::vector< **Triplet**< T > > & *local_nonzeroes,* std::vector< **Triplet**< T > > & *remote_nonzeroes,* std::vector< unsigned long int > & *upscaled_hierarchy,* std::vector< unsigned long int > & *upscaled_row_bounds,* std::vector< unsigned long int > & *upscaled_column_bounds,* std::vector< unsigned long int > & *rowLocalToGlobal,* std::vector< unsigned long int > & *columnLocalToGlobal,* std::map< unsigned long int, unsigned long int > & *rowGlobalToLocal,* std::map< unsigned long int, unsigned long int > & *colGlobalToLocal*) [inline], [protected]

Reads out a subtree and returns the upscaled version.

Does global to local index translation. The non-binary part of the tree is returned in remote_nonzeroes and are not part of the upscaled_hierarchy structures.

Parameters

ID	root of the subtree to read out.
s	only output nonzeroes distributed to processor s.
local_nonzeroes	nonzeroes corresponding to this new tree (flat vector).
remote_nonzeroes	nonzeroes belonging to s contained in the path from ID to the root.
upscaled_hierarchy	hierarchy corresponding to the upscaled nonzeroes.
upscaled_row_bounds	row-wise boundaries corresponding to the upscaled nonzeroes.
upscaled_column_bounds	column-wise boundaries corresponding to the upscaled nonzeroes.
rowLocalToGlobal	maps local row indices to global indices.
columnLocalToGlobal	maps local column indices to global indices.
rowGlobalToLocal	maps global row indices to local indices.
colGlobalToLocal	maps global column indices to local indices.

References Upscaler< T >::addNonzeroes(), SBDTree::c_hi, SBDTree::c_lo, Upscaler< T >::determineEmpty(), Upscaler< T >::determineMinMax(), Upscaler< T >::treeIterator::next(), Upscaler< T >::treeInOrderIterator::next(), SBDTree::NO_SUCH_NODE, Upscaler< T >::nonzeroes, SBDTree::parent, Upscaler< T >::treeIterator::position(), SBDTree::r_hi, SBDTree::r_lo, SBDTree::root, and SBDTree::size().

Referenced by Upscaler< T >::getSubTree().

5.50.3.3 template<typename T> void **Upscaler**< T >::getSubTree (const unsigned long int *s,* std::vector< **Triplet**< T > > & *local_nonzeroes,* std::vector< **Triplet**< T > > & *remote_nonzeroes,* std::vector< unsigned long int > & *upscaled_hierarchy,* std::vector< unsigned long int > & *upscaled_row_bounds,* std::vector< unsigned long int > & *upscaled_column_bounds,* std::vector< unsigned long int > & *rowLocalToGlobal,* std::vector< unsigned long int > & *columnLocalToGlobal,* std::map< unsigned long int, unsigned long int > & *rowGlobalToLocal,* std::map< unsigned long int, unsigned long int > & *colGlobalToLocal,* const unsigned long int *P,* const unsigned long int *Pref*) [inline]

Reads out a subtree corresponding to only the nonzeroes owned by processor s, and returns the upscaled version.

Does global to local index translation. The non-binary part of the tree is returned in remote_nonzeroes and are not part of the upscaled_hierarchy structures.

Parameters

s	only output nonzeroes distributed to processor s.
local_nonzeroes	nonzeroes corresponding to this new tree (flat vector).
remote_nonzeroes	nonzeroes belonging to s contained in the path from ID to the root.
upscaled_hierarchy	hierarchy corresponding to the upscaled nonzeroes.
upscaled_row_bounds	row-wise boundaries corresponding to the upscaled nonzeroes.
upscaled_column_bounds	column-wise boundaries corresponding to the upscaled nonzeroes.
rowLocalToGlobal	maps local row indices to global indices.
columnLocalToGlobal	maps local column indices to global indices.
rowGlobalToLocal	maps global row indices to local indices.
colGlobalToLocal	maps global column indices to local indices.
P	The total number of parts to reduce the SBD tree to.
Pref	The total number of parts (blocks) in this SBD tree.

References Upscaler< T >::getSubTree(), SBDTree::left_child, Upscaler< T >::treeIterator::next(), Upscaler< T >::treePostOrderIterator::next(), SBDTree::NO_SUCH_NODE, Upscaler< T >::nonzeroes, Upscaler< T >::treeIterator::position(), SBDTree::right_child, SBDTree::root, and SBDTree::size().

5.50.3.4 template<typename T> void Upscaler< T >::readout ( ) [inline]

Reads out SBD data and prints to std::cout.

Useful for debugging purposes.

References SBDTree::left_child, SBDTree::NO_SUCH_NODE, SBDTree::parent, SBDTree::right_child, and SBDTree::size().

5.50.3.5 template<typename T> void Upscaler< T >::updateMinMax (const unsigned long int *walk,* const unsigned long int *s,* unsigned long int & *min_i,* unsigned long int & *max_i,* unsigned long int & *min_j,* unsigned long int & *max_j*) [inline], [protected]

Determine min/max of nonzeroes owned by processor s inside node walk.

References Upscaler< T >::nonzeroes.

Referenced by Upscaler< T >::determineMinMax().

5.50.4 Member Data Documentation

5.50.4.1 template<typename T> std::vector< std::vector< bool > > Upscaler< T >::containsPID [protected]

Keeps track which processes are represented in which blocks.

Referenced by Upscaler< T >::Upscaler().

5.50.4.2 template<typename T> std::vector< std::vector< Triplet< T > > > Upscaler< T >::nonzeroes [protected]

All nonzeroes, stored block-by-block.

Referenced by Upscaler< T >::addNonzeroes(), Upscaler< T >::determineEmpty(), Upscaler< T >::getSubTree(), Upscaler< T >::updateMinMax(), and Upscaler< T >::Upscaler().

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

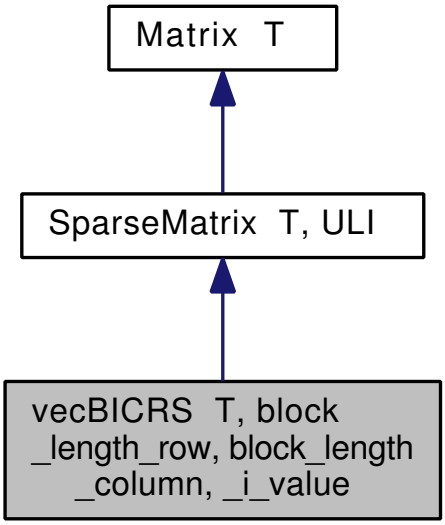
- Upscaler.hpp

5.51 vecBICRS< T, block_length_row, block_length_column, _i_value > Class Template Reference

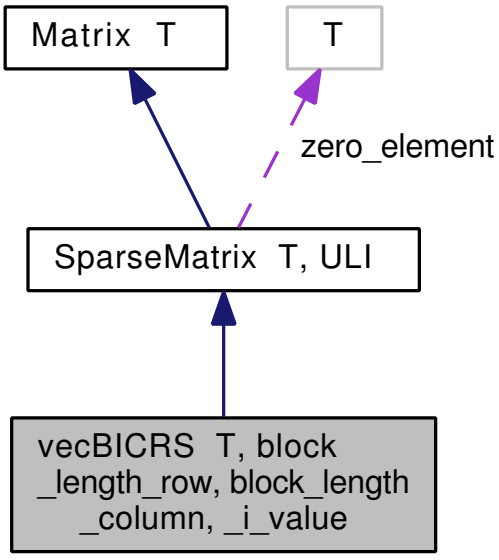
The *incremental* compressed row storage sparse matrix data structure.

`#include <vecBICRS.hpp>`

Inheritance diagram for vecBICRS< T, block_length_row, block_length_column, _i_value >:



Collaboration diagram for vecBICRS< T, block_length_row, block_length_column, _i_value >:



Public Member Functions

- **vecBICRS** ()

  Base constructor.
- **vecBICRS** (std::string file, T zero=0)

  Base constructor.
- **vecBICRS** (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.
- **vecBICRS** (**vecBICRS**< T > &toCopy)

  Copy constructor.
- **vecBICRS** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero=0)

  *Constructor which transforms a collection of input triplets to **CRS** (*p. **??***) format.*
- virtual void **load** (std::vector< **Triplet**< T > > &input, const ULI **m**, const ULI **n**, const T zero)
- virtual void **getFirstIndexPair** (ULI &row, ULI &col)

  Returns the first nonzero index, per reference.
- virtual void **zxa** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=xA function.
- virtual void **zax** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=Ax function.
- ~**vecBICRS** ()

  Base deconstructor.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.

Static Public Member Functions

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

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

  Comparison function used for sorting vectorised blocks; in-row column ordering.
- static int **pRowSort** (const void *left, const void *right)

  Comparison function used for sorting vectorised blocks; row ordering.

Public Attributes

- size_t **fillIn**

  Fill-in (explicitly added nonzeroes to enable vectorisation).

Protected Member Functions

- void **allocate** ()

  Utility function: allocate memory areas using the allocsize and jumpsize fields.
- void **deallocate** ()

  Utility function: free all memory areas.
- void **postProcessRowIncrements** (std::vector< _i_value > &**r_ind**)

  Makes suitable for vectorisation the row increment array.

Static Protected Member Functions

- static void **addPadding** (std::vector< _i_value > &row_increments, const std::map< size_t, size_t > &row2local, const size_t blockingSize, const size_t prev_row, const size_t **m**)

  *Helper function for **vecBICRS*** (p. **??**) *construction.*
- static size_t **prepareBlockIteration** (std::vector< size_t > *const row_indices, std::vector< _i_value > &row_increments, _i_value &prev_row, const std::vector< **Triplet**< double > > &input, const size_t k, const size_t **m**)

  Helper method for oBICRS constructor.

Protected Attributes

- T * **ds**

  Array containing the actual nnz non-zeros.
- _i_value * **c_ind**

  Array containing the column jumps.
- _i_value * **r_ind**

  Array containing the row jumps.
- size_t **bytes**

  Remembers the number of bytes allocated.
- size_t **allocsize**

  The number of nonzeroes allocated (may differ from the actual number of nonzeroes).
- size_t **jumpsize**

  The number of row jumps plus one; i.e., the length of r_ind.

Static Protected Attributes

- static const size_t **block_length** = block_length_row * block_length_column

  Combined blocking size.

5.51.1 Detailed Description

template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI>class vecBICRS< T, block_length_row, block_length_column, _i_value >

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

5.51.2 Constructor & Destructor Documentation

5.51.2.1 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS ( ) [inline]

Base constructor.

5.51.2.2 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS (std::string *file,* T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, ULI >::loadFromFile().

5.51.2.3 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS (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 **ICRS** (p. **??**) 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 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.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::jumpsize, and SparseMatrix< T, ULI >::nnz.

5.51.2.4 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS (vecBICRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	Reference to the **ICRS** (p. **??**) datastructure to copy.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, vecBICRS< T, block_length_row, block_length_column, _i_value >::ds, vecBICRS< T, block_length_row, block_length_column, _i_value >::jumpsize, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind, and SparseMatrix< T, ULI >::zero_element.

5.51.2.5 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero* = 0) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.

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

Parameters

input	The input collection of triplets (i,j,val).

m	The number of rows of the input matrix.
n	The number of columns of the input matrix.
zero	Which element is considered zero.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::load().

5.51.2.6 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> vecBICRS< T, block_length_row, block_length_column, _i_value >::~vecBICRS ( ) [inline]

Base deconstructor.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::deallocate().

5.51.3 Member Function Documentation

5.51.3.1 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> static void vecBICRS< T, block_length_row, block_length_column, _i_value >::addPadding (std::vector< _i_value > & *row_increments,* const std::map< size_t, size_t > & *row2local,* const size_t *blockingSize,* const size_t *prev_row,* const size_t *m*) [inline], [static], [protected]

Helper function for **vecBICRS** (p. **??**) construction.

Parameters

row_increments	The row increments array.
row2local	Maps global row indices to local indices.
blockingSize	Pads to this boundary (sensible values are block_length_row or block_length)
prev_row	Base row index.
m	Maximum row index.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length, and SparseMatrix< T, ULI >::m().

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::prepareBlockIteration().

5.51.3.2 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> void vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate ( ) [inline], [protected]

Utility function: allocate memory areas using the allocsize and jumpsize fields.

See Also

allocsize (p. **??**)
jumpsize (p. **??**)

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::bytes, vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, vecBICRS< T, block_length_row, block_length_column, _i_value >::ds, vecBICRS< T, block_length_row, block_length_column, _i_value >::jumpsize, and vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS().

5.51.3.3 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> virtual size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::bytes.

5.51.3.4 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> static int vecBICRS< T, block_length_row, block_length_column, _i_value >::compareTriplets (const void * *left,* const void * *right*) [inline], [static]

Comparison function used for sorting input data.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::load().

5.51.3.5 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> virtual void vecBICRS< T, block_length_row, block_length_column, _i_value >::getFirstIndexPair (ULI & *row,* ULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, and vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind.

5.51.3.6 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> virtual void vecBICRS< T, block_length_row, block_length_column, _i_value >::load (std::vector< Triplet< T > > & *input,* const ULI *m,* const ULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, ULI >** (p. **??**).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length, vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, vecBICRS< T, block_length_row, block_length_column, _i_value >::compareTriplets(), vecBICRS< T, block_length_row, block_length_column, _i_value >::ds, vecBICRS< T, block_length_row, block_length_column, _i_value >::fillIn, vecBICRS< T, block_length_row, block_length_column, _i_value >::jumpsize, SparseMatrix< T, ULI >::m(), SparseMatrix< T, ULI >::n(), SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, vecBICRS< T, block_length_row, block_length_column, _i_value >::postProcessRowIncrements(), vecBICRS< T, block_length_row, block_length_column, _i_value >::prepareBlockIteration(), vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind, and SparseMatrix< T, ULI >::zero_element.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS().

5.51.3.7 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> void vecBICRS< T, block_length_row, block_length_column, _i_value >::postProcessRowIncrements (std::vector< _i_value > & *r_ind*) [inline], [protected]

Makes suitable for vectorisation the row increment array.

Parameters

r_ind	The row increment array.

References vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::load().

5.51.3.8 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> static size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::prepareBlockIteration (std::vector< size_t > *const *row_indices, std::vector< _i_value > & *row_increments, _i_value & *prev_row, const std::vector< Triplet< double > > & *input, const size_t *k, const size_t *m*) [inline], [static], [protected]

Helper method for oBICRS constructor.

Takes an input array of nnz triplets. This function looks at all triplets from index k on, takes the first block_length_row different rows that are encountered, and adds the nonzeroes on those row with a maximum of eight per row.

If a new row index is encountered but block_length_rows were already added, this function stops iterating.

If a new column index on a given row is encountered, but that row already contains block_length_col elements, then this function also stops iterating.

The above two constraints ensure the nonzeroes are always processed in the intended order.

Parameters

row_indices	an array of block_length_row sets that will contain the nonzero indices that will be added to each block.
row_increments	Keep track of row increments.
prev_row	The global index of the last previously added row.
input	the input triplet array.
k	From which k on to start iterating.
m	The number of matrix rows.

Returns

The k at which iteration stopped (exclusive).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::addPadding(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::load().

5.51.3.9 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> virtual void vecBICRS< T, block_length_row, block_length_column, _i_value >::zax (const T *__restrict__ *pDataX*, T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=Ax function.

Adapted from the master thesis of Joris Koster, Utrecht University.

Parameters

pDataX	Pointer to array x to multiply by the current matrix (Ax).
pDataZ	Pointer to result array. Must be pre-allocated and its elements set to zero for correct results.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length, vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, vecBICRS< T, block_length_row, block_length_column, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind.

5.51.3.10 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> virtual void vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa (const T *__restrict__ *pDataX*, T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=xA function.

Adapted from the master thesis of Joris Koster, Utrecht University.

Parameters

pDataX	Pointer to array x to multiply by the current matrix (Ax).
pDataZ	Pointer to result array. Must be pre-allocated and its elements set to zero for correct results.

Implements **SparseMatrix< T, ULI >** (p. **??**).

References vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize, vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length, vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind, vecBICRS< T, block_length_row, block_length_column, _i_value >::ds, SparseMatrix< T, ULI >::nnz, SparseMatrix< T, ULI >::noc, SparseMatrix< T, ULI >::nor, and vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind.

5.51.4 Member Data Documentation

5.51.4.1 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::allocsize [protected]

The number of nonzeroes allocated (may differ from the actual number of nonzeroes).

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS(), vecBICRS< T, block_length_row, block_length_column, _i_value >::zax(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa().

5.51.4.2 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> const size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::block_length = block_length_row * block_length_column [static], [protected]

Combined blocking size.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::addPadding(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), vecBICRS< T, block_length_row, block_length_column, _i_value >::postProcessRowIncrements(), vecBICRS< T, block_length_row, block_length_column, _i_value >::prepareBlockIteration(), vecBICRS< T, block_length_row, block_length_column, _i_value >::zax(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa().

5.51.4.3 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::bytes [protected]

Remembers the number of bytes allocated.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::bytesUsed().

5.51.4.4 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> _i_value* vecBICRS< T, block_length_row, block_length_column, _i_value >::c_ind [protected]

Array containing the column jumps.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::deallocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::getFirstIndexPair(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS(), vecBICRS< T, block_length_row, block_length_column, _i_value >::zax(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa().

5.51.4.5 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> T* vecBICRS< T, block_length_row, block_length_column, _i_value >::ds [protected]

Array containing the actual nnz non-zeros.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::deallocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS(), vecBICRS< T, block_length_row, block_length_column, _i_value >::zax(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa().

5.51.4.6 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::fillIn

Fill-in (explicitly added nonzeroes to enable vectorisation).

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::load().

5.51.4.7 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> size_t vecBICRS< T, block_length_row, block_length_column, _i_value >::jumpsize [protected]

The number of row jumps plus one; i.e., the length of r_ind.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS().

5.51.4.8 template<typename T, size_t block_length_row = 1, size_t block_length_column = 8, typename _i_value = LI> _i_value* vecBICRS< T, block_length_row, block_length_column, _i_value >::r_ind [protected]

Array containing the row jumps.

Referenced by vecBICRS< T, block_length_row, block_length_column, _i_value >::allocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::deallocate(), vecBICRS< T, block_length_row, block_length_column, _i_value >::getFirstIndexPair(), vecBICRS< T, block_length_row, block_length_column, _i_value >::load(), vecBICRS< T, block_length_row, block_length_column, _i_value >::vecBICRS(), vecBICRS< T, block_length_row, block_length_column, _i_value >::zax(), and vecBICRS< T, block_length_row, block_length_column, _i_value >::zxa().

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

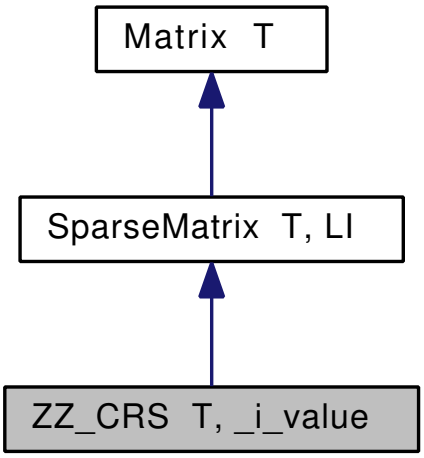
- vecBICRS.hpp

5.52 ZZ_CRS< T, _i_value > Class Template Reference

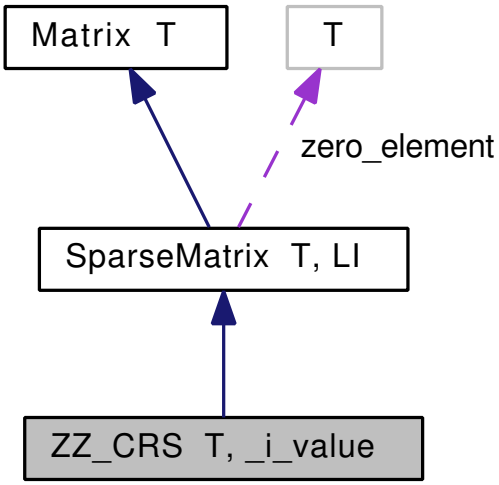
The zig-zag compressed row storage sparse matrix data structure.

`#include <ZZ_CRS.hpp>`

Inheritance diagram for ZZ_CRS< T, _i_value >:



Collaboration diagram for ZZ_CRS< T, _i_value >:



Public Member Functions

- **ZZ_CRS** ()

  Base constructor.
- **ZZ_CRS** (std::string file, T zero=0)

  Base constructor.
- **ZZ_CRS** (const long int number_of_nonzeros, const long int number_of_rows, const long int number_of_cols, T zero)

  Base constructor which only initialises the internal arrays.
- **ZZ_CRS** (**ZZ_CRS**< T > &toCopy)

  Copy constructor.
- **ZZ_CRS** (std::vector< **Triplet**< T > > &input, const LI **m**, const LI **n**, const T zero)

  *Constructor which transforms a collection of input triplets to **CRS** (*p. **??**) *format.*
- virtual void **load** (std::vector< **Triplet**< T > > &input, const LI **m**, const LI **n**, const T zero)
- virtual void **getFirstIndexPair** (LI &row, LI &col)

  Returns the first nonzero index, per reference.

- virtual void **zxa** (const T *x, T *z)

  In-place z=xA multiplication algorithm.
- virtual void **zax** (const T *__restrict__ x, T *__restrict z)

  In-place z=Ax multiplication algorithm.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.
- ~**ZZ_CRS** ()

  Base deconstructor.

Static Protected Member Functions

- static bool **compareTripletsLTR** (const **Triplet**< T > *one, const **Triplet**< T > *two)

  Comparison function used for sorting input data.
- static bool **compareTripletsRTL** (const **Triplet**< T > *one, const **Triplet**< T > *two)

  Comparison function used for sorting input data.

Protected Attributes

- T * **ds**

  Array containing the actual this->nnz non-zeros.
- LI * **col_ind**

  Array containing the column jumps.
- LI * **row_start**

  Array containing the row jumps.

Additional Inherited Members

5.52.1 Detailed Description

template<typename T, typename _i_value = LI>class ZZ_CRS< T, _i_value >

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

5.52.2 Constructor & Destructor Documentation

5.52.2.1 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::ZZ_CRS ( ) [inline]

Base constructor.

5.52.2.2 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::ZZ_CRS (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, LI >::loadFromFile().

5.52.2.3 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::ZZ_CRS (const long int *number_of_nonzeros,* const long int *number_of_rows,* const long int *number_of_cols,* T *zero*) [inline]

Base constructor which only initialises the internal arrays.

Note that to gain a valid **CRS** (p. **??**) 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 to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns to be stored.
zero	Which element is considered to be the zero element.

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::ds, SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::nor, and ZZ_CRS< T, _i_value >::row_start.

5.52.2.4 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::ZZ_CRS (ZZ_CRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **CRS** (p. **??**) datastructure to copy.

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::ds, SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, ZZ_CRS< T, _i_value >::row_start, and SparseMatrix< T, LI >::zero_element.

5.52.2.5 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::ZZ_CRS (std::vector< Triplet< T > > & *input,* const LI *m,* const LI *n,* const T *zero*) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	Which element is considered zero.

References ZZ_CRS< T, _i_value >::load().

5.52.2.6 template<typename T, typename _i_value = LI> ZZ_CRS< T, _i_value >::~ZZ_CRS ( ) [inline]

Base deconstructor.

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::ds, and ZZ_CRS< T, _i_value >::row_start.

5.52.3 Member Function Documentation

5.52.3.1 template<typename T, typename _i_value = LI> virtual size_t ZZ_CRS< T, _i_value >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References SparseMatrix< T, LI >::nnz, and SparseMatrix< T, LI >::nor.

5.52.3.2 template<typename T, typename _i_value = LI> static bool ZZ_CRS< T, _i_value >::compareTripletsLTR (const Triplet< T > * *one,* const Triplet< T > * *two*) [inline], [static], [protected]

Comparison function used for sorting input data.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by ZZ_CRS< T, _i_value >::load().

5.52.3.3 template<typename T, typename _i_value = LI> static bool ZZ_CRS< T, _i_value >::compareTripletsRTL (const Triplet< T > * *one,* const Triplet< T > * *two*) [inline], [static], [protected]

Comparison function used for sorting input data.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by ZZ_CRS< T, _i_value >::load().

5.52.3.4 template<typename T, typename _i_value = LI> virtual void ZZ_CRS< T, _i_value >::getFirstIndexPair (LI & *row,* LI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_CRS< T, _i_value >::col_ind, and ZZ_CRS< T, _i_value >::row_start.

5.52.3.5 template<typename T, typename _i_value = LI> virtual void ZZ_CRS< T, _i_value >::load (std::vector< Triplet< T > > & *input,* const LI *m,* const LI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::compareTripletsLTR(), ZZ_CRS< T, _i_value >::compareTripletsRTL(), ZZ_CRS< T, _i_value >::ds, Triplet< T >::i(), Triplet< T >::j(), SparseMatrix< T, LI >::m(), SparseMatrix< T, LI >::n(), SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, ZZ_CRS< T, _i_value >::row_start, Triplet< T >::value, and SparseMatrix< T, LI >::zero_element.

Referenced by ZZ_CRS< T, _i_value >::ZZ_CRS().

5.52.3.6 template<typename T, typename _i_value = LI> virtual void ZZ_CRS< T, _i_value >::zax (const T *__restrict__ *x,* T *__restrict *z*) [inline], [virtual]

In-place z=Ax multiplication algorithm.

Parameters

x	The vector x supplied for multiplication.
z	The pre-allocated result vector. All elements should be set to zero in advance.

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::ds, SparseMatrix< T, LI >::nor, and ZZ_CRS< T, _i_value >::row_start.

5.52.3.7 template<typename T, typename _i_value = LI> virtual void ZZ_CRS< T, _i_value >::zxa (const T * x, T * z) [inline], [virtual]

In-place z=xA multiplication algorithm.

Parameters

x	The vector x supplied for left-multiplication.
z	The pre-allocated result vector. All elements should be set to zero in advance.

References ZZ_CRS< T, _i_value >::col_ind, ZZ_CRS< T, _i_value >::ds, SparseMatrix< T, LI >::nor, and ZZ_CRS< T, _i_value >::row_start.

5.52.4 Member Data Documentation

5.52.4.1 template<typename T, typename _i_value = LI> LI* ZZ_CRS< T, _i_value >::col_ind [protected]

Array containing the column jumps.

Referenced by ZZ_CRS< T, _i_value >::getFirstIndexPair(), ZZ_CRS< T, _i_value >::load(), ZZ_CRS< T, _i_value >::zax(), ZZ_CRS< T, _i_value >::zxa(), ZZ_CRS< T, _i_value >::ZZ_CRS(), and ZZ_CRS< T, _i_value >::~ZZ_CRS().

5.52.4.2 template<typename T, typename _i_value = LI> T* ZZ_CRS< T, _i_value >::ds [protected]

Array containing the actual this->nnz non-zeros.

Referenced by ZZ_CRS< T, _i_value >::load(), ZZ_CRS< T, _i_value >::zax(), ZZ_CRS< T, _i_value >::zxa(), ZZ_CRS< T, _i_value >::ZZ_CRS(), and ZZ_CRS< T, _i_value >::~ZZ_CRS().

5.52.4.3 template<typename T, typename _i_value = LI> LI* ZZ_CRS< T, _i_value >::row_start [protected]

Array containing the row jumps.

Referenced by ZZ_CRS< T, _i_value >::getFirstIndexPair(), ZZ_CRS< T, _i_value >::load(), ZZ_CRS< T, _i_value >::zax(), ZZ_CRS< T, _i_value >::zxa(), ZZ_CRS< T, _i_value >::ZZ_CRS(), and ZZ_CRS< T, _i_value >::~ZZ_CRS().

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

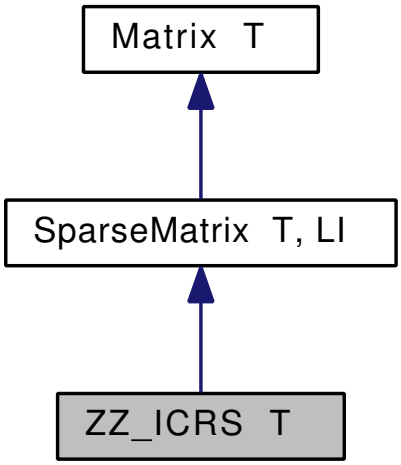
- ZZ_CRS.hpp

5.53 ZZ_ICRS< T > Class Template Reference

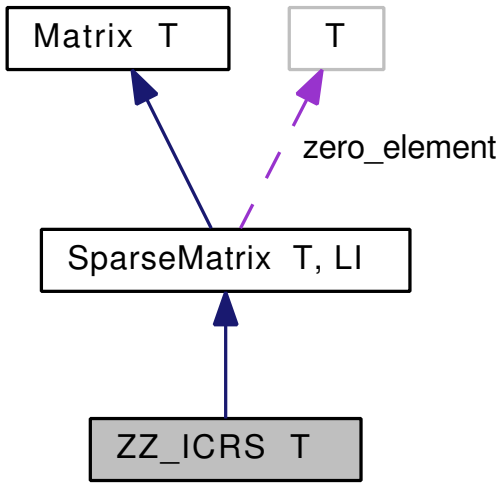
The zig-zag incremental compressed row storage sparse matrix data structure.

`#include <ZZ_ICRS.hpp>`

Inheritance diagram for ZZ_ICRS< T >:



Collaboration diagram for ZZ_ICRS< T >:



Public Member Functions

- **ZZ_ICRS** ()

  Base constructor.
- **ZZ_ICRS** (std::string file, T zero=0)

  Base constructor.
- **ZZ_ICRS** (const myULI number_of_nonzeros, const myULI number_of_rows, const myULI number_of_cols, T zero)

  Base constructor which only initialises the internal arrays.
- **ZZ_ICRS** (**ZZ_ICRS**< T > &toCopy)

  Copy constructor.
- **ZZ_ICRS** (std::vector< **Triplet**< T > > &input, const myULI **m**, const myULI **n**, const T zero)

  *Constructor which transforms a collection of input triplets to **CRS** (p. **??**) format.*
- virtual void **load** (std::vector< **Triplet**< T > > &input, const myULI **m**, const myULI **n**, const T zero)
- virtual void **getFirstIndexPair** (myULI &row, myULI &col)

  Returns the first nonzero index, per reference.

- virtual void **zxa** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=xA multiplication algorithm.
- virtual void **zax** (const T *__restrict__ pDataX, T *__restrict__ pDataZ)

  In-place z=Ax multiplication algorithm.
- virtual size_t **bytesUsed** ()

  Function to query the amount of storage required by this sparse matrix.
- ~**ZZ_ICRS** ()

  Base deconstructor.

Static Protected Member Functions

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

  Comparison function used for sorting input data.

Protected Attributes

- size_t **jumps**

  The number of row jumps.
- T * **ds**

  Array containing the actual this->nnz non-zeros.
- myULI * **c_ind**

  Array containing the column jumps.
- myULI * **r_ind**

  Array containing the row jumps.

Additional Inherited Members

5.53.1 Detailed Description

template<typename T>class ZZ_ICRS< T >

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

5.53.2 Constructor & Destructor Documentation

5.53.2.1 template<typename T> ZZ_ICRS< T >::ZZ_ICRS ( ) [inline]

Base constructor.

5.53.2.2 template<typename T> ZZ_ICRS< T >::ZZ_ICRS (std::string *file*, T *zero* = 0) [inline]

Base constructor.

Will read in from **Matrix** (p. **??**) Market file.

See Also

SparseMatrix::SparseMatrix(file, zero)

References SparseMatrix< T, LI >::loadFromFile().

5.53.2.3 template<typename T> ZZ_ICRS< T >::ZZ_ICRS (const myULI *number_of_nonzeros,* const myULI *number_of_rows,* const myULI *number_of_cols,* T *zero*) [inline]

Base constructor which only initialises the internal arrays.

Note that to gain a valid **CRS** (p. **??**) 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 to be stored.
number_of_rows	The number of rows to be stored.
number_of_cols	The number of columns to be stored.
zero	Which element is considered to be the zero element.

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::ds, SparseMatrix< T, LI >::nnz, and ZZ_ICRS< T >::r_ind.

5.53.2.4 template<typename T> ZZ_ICRS< T >::ZZ_ICRS (ZZ_ICRS< T > & *toCopy*) [inline]

Copy constructor.

Parameters

toCopy	reference to the **CRS** (p. **??**) datastructure to copy.

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::ds, SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, ZZ_ICRS< T >::r_ind, and SparseMatrix< T, LI >::zero_element.

5.53.2.5 template<typename T> ZZ_ICRS< T >::ZZ_ICRS (std::vector< Triplet< T > > & *input,* const myULI *m,* const myULI *n,* const T *zero*) [inline]

Constructor which transforms a collection of input triplets to **CRS** (p. **??**) 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	Total number of rows.
n	Total number of columns.
zero	Which element is considered zero.

References ZZ_ICRS< T >::load().

5.53.2.6 template<typename T> ZZ_ICRS< T >::~ZZ_ICRS ( ) [inline]

Base deconstructor.

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::ds, and ZZ_ICRS< T >::r_ind.

5.53.3 Member Function Documentation

5.53.3.1 template<typename T> virtual size_t ZZ_ICRS< T >::bytesUsed ( ) [inline], [virtual]

Function to query the amount of storage required by this sparse matrix.

Returns

The size of the sparse matrix in bytes.

Implements **Matrix< T >** (p. **??**).

References ZZ_ICRS< T >::jumps, and SparseMatrix< T, LI >::nnz.

5.53.3.2 template<typename T> static bool ZZ_ICRS< T >::compareTriplets (const Triplet< T > & *one,* const Triplet< T > & *two*) [inline], [static], [protected]

Comparison function used for sorting input data.

References Triplet< T >::i(), and Triplet< T >::j().

Referenced by ZZ_ICRS< T >::load().

5.53.3.3 template<typename T> virtual void ZZ_ICRS< T >::getFirstIndexPair (myULI & *row,* myULI & *col*) [inline], [virtual]

Returns the first nonzero index, per reference.

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_ICRS< T >::c_ind, and ZZ_ICRS< T >::r_ind.

5.53.3.4 template<typename T> virtual void ZZ_ICRS< T >::load (std::vector< Triplet< T > > & *input,* const myULI *m,* const myULI *n,* const T *zero*) [inline], [virtual]

See Also

SparseMatrix::load (p. **??**)

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::compareTriplets(), ZZ_ICRS< T >::ds, Triplet< T >::i(), Triplet< T >::j(), ZZ_ICRS< T >::jumps, SparseMatrix< T, LI >::m(), SparseMatrix< T, LI >::n(), SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, ZZ_ICRS< T >::r_ind, Triplet< T >::value, and SparseMatrix< T, LI >::zero_element.

Referenced by ZZ_ICRS< T >::ZZ_ICRS().

5.53.3.5 template<typename T> virtual void ZZ_ICRS< T >::zax (const T *__restrict__ *pDataX,* T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=Ax multiplication algorithm.

Parameters

pDataX	The vector x supplied for multiplication.
pDataZ	The pre-allocated result vector. All elements should be set to zero in advance.

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::ds, SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, and ZZ_ICRS< T >::r_ind.

5.53.3.6 template<typename T> virtual void ZZ_ICRS< T >::zxa (const T *__restrict__ *pDataX,* T *__restrict__ *pDataZ*) [inline], [virtual]

In-place z=xA multiplication algorithm.

Parameters

pDataX	The vector x supplied for left-multiplication.
pDataZ	The pre-allocated result vector. All elements should be set to zero in advance.

Implements **SparseMatrix< T, LI >** (p. **??**).

References ZZ_ICRS< T >::c_ind, ZZ_ICRS< T >::ds, SparseMatrix< T, LI >::nnz, SparseMatrix< T, LI >::noc, SparseMatrix< T, LI >::nor, and ZZ_ICRS< T >::r_ind.

5.53.4 Member Data Documentation

5.53.4.1 template<typename T> myULI* ZZ_ICRS< T >::c_ind [protected]

Array containing the column jumps.

Referenced by ZZ_ICRS< T >::getFirstIndexPair(), ZZ_ICRS< T >::load(), ZZ_ICRS< T >::zax(), ZZ_ICRS< T >::zxa(), ZZ_ICRS< T >::ZZ_ICRS(), and ZZ_ICRS< T >::~ZZ_ICRS().

5.53.4.2 template<typename T> T* ZZ_ICRS< T >::ds [protected]

Array containing the actual this->nnz non-zeros.

Referenced by ZZ_ICRS< T >::load(), ZZ_ICRS< T >::zax(), ZZ_ICRS< T >::zxa(), ZZ_ICRS< T >::ZZ_ICRS(), and ZZ_ICRS< T >::~ZZ_ICRS().

5.53.4.3 template<typename T> size_t ZZ_ICRS< T >::jumps [protected]

The number of row jumps.

Referenced by ZZ_ICRS< T >::bytesUsed(), and ZZ_ICRS< T >::load().

5.53.4.4 template<typename T> myULI* ZZ_ICRS< T >::r_ind [protected]

Array containing the row jumps.

Referenced by ZZ_ICRS< T >::getFirstIndexPair(), ZZ_ICRS< T >::load(), ZZ_ICRS< T >::zax(), ZZ_ICRS< T >::zxa(), ZZ_ICRS< T >::ZZ_ICRS(), and ZZ_ICRS< T >::~ZZ_ICRS().

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

- ZZ_ICRS.hpp

Chapter 6

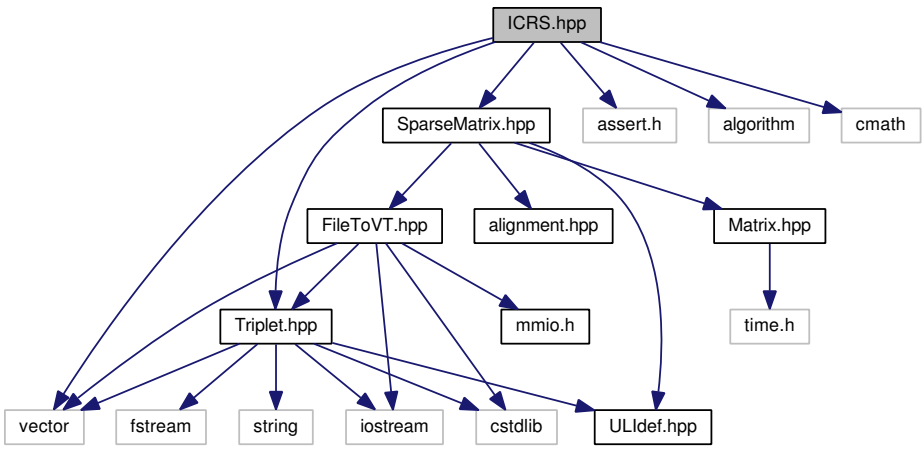
File Documentation

6.1 ICRS.hpp File Reference

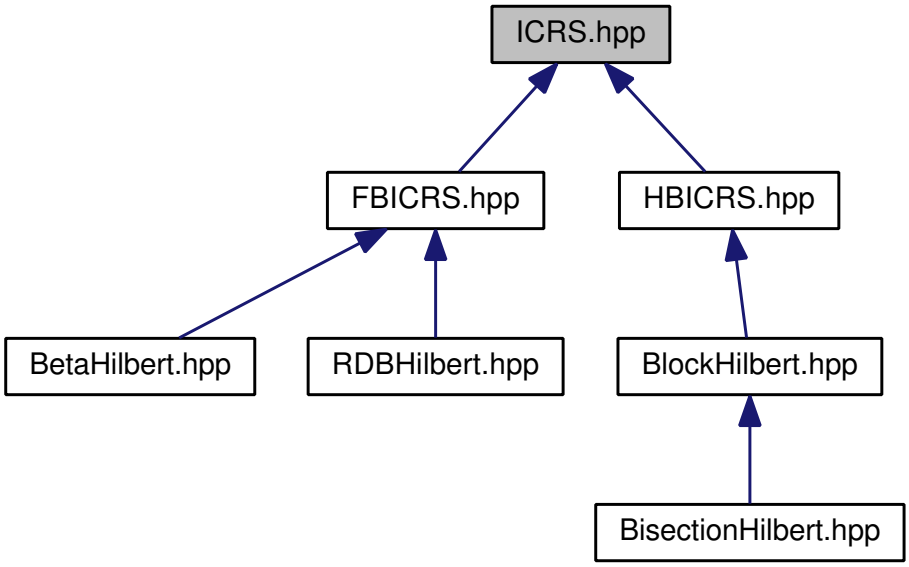
File created by: A.

```
#include "Triplet.hpp"
#include "SparseMatrix.hpp"
#include <assert.h>
#include <vector>
#include <algorithm>
#include <cmath>
```

Include dependency graph for ICRS.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **ICRS**< **T, _i_value** >

  The incremental compressed row storage sparse matrix data structure.

6.1.1 Detailed Description

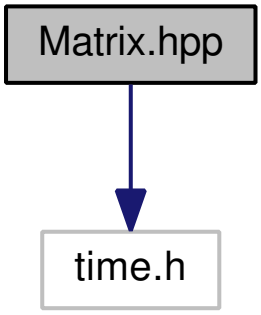
File created by: A. N. Yzelman, Dept. of Mathematics, Utrecht University, 2007.

6.2 Matrix.hpp File Reference

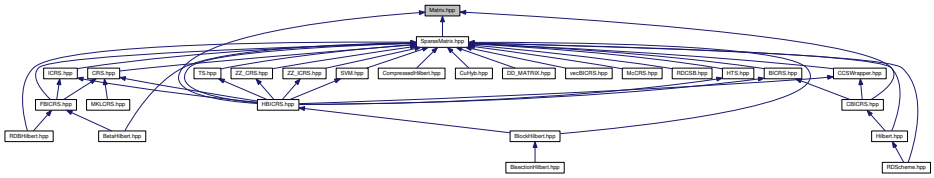
File created by: A.

```
#include <time.h>
```

Include dependency graph for Matrix.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **Matrix**< **T** >

  Defines operations common to all matrices, which are implemented in this library.

6.2.1 Detailed Description

File created by: A. N. Yzelman, Dept. of Mathematics, Utrecht University, 2009.

Index

~BICRS
    BICRS, 21
~BetaHilbert
    BetaHilbert, 14
~BisectionHilbert
    BisectionHilbert, 29
~BlockHilbert
    BlockHilbert, 34
~CBICRS
    CBICRS, 47
~CCSWrapper
    CCSWrapper, 58
~CRS
    CRS, 68
~CompressedHilbert
    CompressedHilbert, 63
~CuHyb
    CuHyb, 73
~DD_MATRIX
    DD_MATRIX, 78
~FBICRS
    FBICRS, 85
~HBICRS
    HBICRS, 94
~Hilbert
    Hilbert, 98
~HilbertArray
    HilbertArray, 103
~HilbertArrayInterface
    HilbertArrayInterface, 107
~ICRS
    ICRS, 121
~MKLCRS
    MKLCRS, 146
~Matrix
    Matrix, 126
~McCRS
    McCRS, 136
~RDBHilbert
    RDBHilbert, 153
~RDScheme
    RDScheme, 163
~SBDTree
    SBDTree, 170
~SVM
    SVM, 189
~SparseMatrix
    SparseMatrix, 182
~TS
    TS, 203
~Upscaler
    Upscaler, 209
~ZZ_CRS
    ZZ_CRS, 223
~ZZ_ICRS
    ZZ_ICRS, 228
~vecBICRS
    vecBICRS, 216
_col_ind
    MKLCRS, 146
_m
    MKLCRS, 146
_n
    MKLCRS, 146
_one
    MKLCRS, 147
_row_start
    MKLCRS, 147

addPadding
    vecBICRS, 216
ads
    Hilbert, 100
allocate
    vecBICRS, 216
allocsize
    vecBICRS, 219
array
    HilbertArray, 105

BICRS
    ~BICRS, 21
    BICRS, 21, 22
    BICRS, 21, 22
    bytesUsed, 22
    c_end, 24
    c_inc, 24
    c_start, 24
    getFirstIndexPair, 22
    jumps, 24
    load, 22
    ntt, 24
    r_end, 24
    r_inc, 24
    r_start, 25
    vals, 25
    zax, 23
    zax_fb, 23
    zxa, 23

BICRS< _t_value, _i_value >, 19
BITWIDTH
Matrix2HilbertCoordinates, 132
BetaHilbert
~BetaHilbert, 14
BetaHilbert, 14
BetaHilbert, 14
bytesUsed, 14
collectY, 14
end, 15
getFirstIndexPair, 15
load, 15
mv, 15
set_p_translate, 17
synchronise, 17
thread, 17
wait, 17
zax, 18
zxa, 18
BetaHilbert< T >, 11
BigInt, 25
BigInt, 26
BigInt, 26
low, 27
operator unsigned char, 26
operator unsigned int, 26
operator unsigned long int, 26
operator unsigned short int, 26
operator=, 26
bisect
BlockHilbert, 35
BisectionHilbert
~BisectionHilbert, 29
BisectionHilbert, 29
BisectionHilbert, 29
BisectionHilbert< T >, 27
block_length
vecBICRS, 219
BlockCRS
in_readout, 30
post_readout, 31
pre_readout, 31
BlockCRS< T >, 29
BlockHilbert
~BlockHilbert, 34
bisect, 35
BlockHilbert, 34, 35
BlockHilbert, 34, 35
buildBisectionBlocks, 37
bytesUsed, 37
cmp, 37
DS, 39
find, 37
getFirstIndexPair, 38
load, 38
MAX_DEPTH, 39
zax, 38
zxa, 38
BlockHilbert< T >, 31
BlockOrderer
cur_height, 43
datatype, 43
height, 43
in_height, 41
in_readout, 41
infix, 41
items, 43
leftpass, 43
output, 43
post_height, 41
post_readout, 42
postfix, 42
pre_height, 42
pre_readout, 42
prefix, 42
READOUT, 43
rightpass, 44
TRAVERSE_HEIGHT, 44
traverse, 42
traverse_mode, 44
tree, 44
BlockOrderer< T >, 39
build
SBDTree, 171
buildBisectionBlocks
BlockHilbert, 37
bytes
CBICRS, 51
DD_MATRIX, 80
HilbertArray, 106
ICRS, 123
RDScheme_shared_data, 167
vecBICRS, 219
bytesUsed
BetaHilbert, 14
BICRS, 22
BlockHilbert, 37
CBICRS, 48
CCSWrapper, 58
CompressedHilbert, 63
CRS, 68
CuHyb, 73
DD_MATRIX, 79
FBICRS, 86
HBICRS, 94
Hilbert, 99
HilbertArray, 103
HilbertArrayInterface, 108
HTS, 115
ICRS, 121
Matrix, 126
McCRS, 136
RDBHilbert, 153
RDScheme, 163
SVM, 189
TS, 203

vecBICRS, 216
ZZ_CRS, 223
ZZ_ICRS, 228

c_end
BICRS, 24
FBICRS, 90
c_hi
SBDTree, 173
c_inc
BICRS, 24
CBICRS, 51
FBICRS, 90
HBICRS, 96
c_ind
ICRS, 123
vecBICRS, 219
ZZ_ICRS, 230
c_lo
SBDTree, 173
c_start
BICRS, 24
CBICRS, 51
FBICRS, 90
CBICRS
∼CBICRS, 47
bytes, 51
bytesUsed, 48
c_inc, 51
c_start, 51
CBICRS, 47, 48
CBICRS, 47, 48
getFirstIndexPair, 48
getMemoryUsage, 49
getNumberOfOverflows, 49
load, 49, 50
mask1, 52
mask2, 52
memoryUsage, 50
ntt, 52
r_inc, 52
r_start, 52
vals, 52
zax, 50
zxa, 51
CBICRS< _t_value, _master_i_value, _master_j_value, _i_value, _j_value >, 44
CBICRS_factory
getCBICCS, 54
getCBICRS, 54
CBICRS_factory< _t_value >, 53
CCSWrapper
∼CCSWrapper, 58
bytesUsed, 58
CCSWrapper, 58
CCSWrapper, 58
ds, 61
getFirstIndexPair, 59
load, 59
loadFromFile, 59
m, 59
mv, 59
n, 60
nzs, 60
transposeVector, 60
zax, 60
zxa, 60
CCSWrapper< T, SparseMatrixType, IND >, 56
CRS
∼CRS, 68
bytesUsed, 68
CRS, 67
col_ind, 70
columnIndices, 68
CRS, 67
ds, 70
find, 68
getFirstIndexPair, 68
load, 69
random_access, 69
row_start, 70
rowJump, 69
values, 69
ZXa, 70
ZaX, 69
zax, 69
CRS< T >, 64
cmp
BlockHilbert, 37
HTS, 116
col_ind
CRS, 70
McCRS, 139
ZZ_CRS, 225
collectY
BetaHilbert, 14
RDBHilbert, 153
RDScheme, 163
column
HilbertTriplet, 112
Triplet, 201
columnBounds
SBDTree, 172
columnIndices
CRS, 68
McCRS, 136
compareTriplets
ICRS, 121
vecBICRS, 217
ZZ_ICRS, 229
compareTripletsLTR
ZZ_CRS, 224
compareTripletsRTL
ZZ_CRS, 224
CompressedHilbert
∼CompressedHilbert, 63
bytesUsed, 63

CompressedHilbert, 63
CompressedHilbert, 63
getFirstIndexPair, 63
load, 63
zax, 64
zxa, 64
CompressedHilbert< T >, 61
cond
RDB_shared_data, 149
RDScheme_shared_data, 167
shared_data, 179
containsPID
Upscaler, 211
CuHyb, 70
∼CuHyb, 73
bytesUsed, 73
CuHyb, 72, 73
CuHyb, 72, 73
descrA, 74
GPUx, 74
GPUz, 74
getFirstIndexPair, 73
handle, 75
hybA, 75
i, 75
j, 75
load, 73
zax, 74
cur_height
BlockOrderer, 43
curcol
HilbertArray, 106
curcoor
HilbertArray, 106
curpos
HilbertArray, 106
currow
HilbertArray, 106

d
DD_MATRIX, 80
DD_MATRIX
∼DD_MATRIX, 78
bytes, 80
bytesUsed, 79
d, 80
DD_MATRIX, 77, 78
DD_MATRIX, 77, 78
full, 80
getFirstIndexPair, 79
load, 79
nzs, 80
SELF_ALLOCATED, 80
zax, 79
zxa, 79
DD_MATRIX< T, number_of_diagonals, diagonal_-
offsets >, 75
DS
BlockHilbert, 39
datatype
BlockOrderer, 43
decode
HilbertArray, 103
descr
MKLCRS, 147
descrA
CuHyb, 74
determineEmpty
Upscaler, 209
ds
CCSWrapper, 61
CRS, 70
Hilbert, 100
HTS, 117
ICRS, 123
McCRS, 139
TS, 205
vecBICRS, 220
ZZ_CRS, 225
ZZ_ICRS, 230
Duck
in_readout, 82
post_readout, 82
pre_readout, 82
Duck< T >, 81

end
BetaHilbert, 15
RDBHilbert, 154
RDScheme, 164
end_cond
RDB_shared_data, 149
RDScheme_shared_data, 167
shared_data, 179
end_mutex
RDB_shared_data, 149
RDScheme_shared_data, 167
shared_data, 179
end_sync
RDB_shared_data, 149
RDScheme_shared_data, 167
shared_data, 179

FBICRS
∼FBICRS, 85
bytesUsed, 86
c_end, 90
c_inc, 90
c_start, 90
FBICRS, 85, 86
FBICRS, 85, 86
fillIn, 90
getFirstIndexPair, 86
jumps, 90
load, 87
r_end, 90
r_inc, 90
r_start, 91

ZXa, 89
ZaX, 88
zax, 88
zax_fb, 88
zxa, 89
zxa_fb, 89
FBICRS< _t_value, _i_value, _sub_ds, logBeta >, 82
FileToVT, 91
fillIn
FBICRS, 90
ICRS, 123
vecBICRS, 220
find
BlockHilbert, 37
CRS, 68
HTS, 116
McCRS, 136
SVM, 189
full
DD_MATRIX, 80

GPUx
CuHyb, 74
GPUz
CuHyb, 74
getCBICCS
CBICRS_factory, 54
getCBICRS
CBICRS_factory, 54
getData
SVM, 189
getDataStructure
Hilbert, 99
getFirstColumnIndex
HilbertArray, 104
HilbertArrayInterface, 108
getFirstIndexPair
BetaHilbert, 15
BICRS, 22
BlockHilbert, 38
CBICRS, 48
CCSWrapper, 59
CompressedHilbert, 63
CRS, 68
CuHyb, 73
DD_MATRIX, 79
FBICRS, 86
HBICRS, 94
Hilbert, 99
HTS, 116
ICRS, 121
McCRS, 136
RDBHilbert, 154
RDCSB, 159
RDScheme, 164
SparseMatrix, 182
SVM, 189
TS, 203
vecBICRS, 217
ZZ_CRS, 224
ZZ_ICRS, 229
getFirstRowIndex
HilbertArray, 104
HilbertArrayInterface, 108
getHilbertCoordinate
HilbertTriplet, 110
getLeastSignificantHilbertBits
HilbertTriplet, 110
getMemoryUsage
CBICRS, 49
getMostSignificantHilbertBits
HilbertTriplet, 110
getNumberOfOverflows
CBICRS, 49
getSeparatorBB
SBDTree, 172
getSubTree
Upscaler, 209, 210

HBICRS
∼HBICRS, 94
bytesUsed, 94
c_inc, 96
getFirstIndexPair, 94
HBICRS, 94
HBICRS, 94
jumps, 96
load, 95
ntt, 96
r_inc, 96
vals, 96
zax, 95
zxa, 95
HBICRS< _t_value >, 92
HTS
bytesUsed, 115
cmp, 116
ds, 117
find, 116
getFirstIndexPair, 116
HTS, 115
HTS, 115
load, 116
saveBinary, 116
zax, 117
zxa, 117
HTS< T >, 113
handle
CuHyb, 75
height
BlockOrderer, 43
Hilbert
∼Hilbert, 98
ads, 100
bytesUsed, 99
ds, 100
getDataStructure, 99
getFirstIndexPair, 99

Hilbert, 98, 99
load, 99
saveBinary, 100
zax, 100
zxa, 100
Hilbert< T >, 96
hilbert1
HilbertTriplet, 112
hilbert2
HilbertTriplet, 112
HilbertArray
~HilbertArray, 103
array, 105
bytes, 106
bytesUsed, 103
curcol, 106
curcoor, 106
curpos, 106
currow, 106
decode, 103
getFirstColumnIndex, 104
getFirstRowIndex, 104
HilbertArray, 103
HilbertArray, 103
moveToNext, 104
moveToStart, 104
start_coor, 106
zax, 105
zxa, 105
HilbertArray< T, I, hI, mI >, 101
HilbertArrayInterface
~HilbertArrayInterface, 107
bytesUsed, 108
getFirstColumnIndex, 108
getFirstRowIndex, 108
moveToNext, 108
moveToStart, 108
zax, 108
zxa, 108
HilbertArrayInterface< T >, 107
HilbertTriplet
column, 112
getHilbertCoordinate, 110
getLeastSignificantHilbertBits, 110
getMostSignificantHilbertBits, 110
hilbert1, 112
hilbert2, 112
HilbertTriplet, 110
HilbertTriplet, 110
i, 111
j, 111
row, 112
save, 111
value, 112
HilbertTriplet< T >, 108
HilbertTripletCompare
operator(), 113
HilbertTripletCompare< T >, 112
hybA
CuHyb, 75

i
CuHyb, 75
HilbertTriplet, 111
Triplet, 198
TS, 205
ICRS
~ICRS, 121
bytes, 123
bytesUsed, 121
c_ind, 123
compareTriplets, 121
ds, 123
fillIn, 123
getFirstIndexPair, 121
ICRS, 120
ICRS, 120
load, 121
r_ind, 123
setStartingPos, 122
ZXa, 123
ZaX, 122
zax, 122
zxa, 122
ICRS< T, _i_value >, 117
ICRS.hpp, 231
ID
Upscaler::treeIterator, 194
id
RDB_shared_data, 149
in_height
BlockOrderer, 41
in_readout
BlockCRS, 30
BlockOrderer, 41
Duck, 82
MinCCS, 140
MinCRS, 142
SepLast, 175
U2, 206
infix
BlockOrderer, 41
input
RDB_shared_data, 149
shared_data, 179
IntegerToHilbert
Matrix2HilbertCoordinates, 132
items
BlockOrderer, 43

j
CuHyb, 75
HilbertTriplet, 111
Triplet, 198
TS, 205
jumps
BICRS, 24

FBICRS, 90
HBICRS, 96
ZZ_ICRS, 230
jumpsize
vecBICRS, 220

left
SBDTree, 172
left_child
SBDTree, 173
leftpass
BlockOrderer, 43
load
BetaHilbert, 15
BICRS, 22
BlockHilbert, 38
CBICRS, 49, 50
CCSWrapper, 59
CompressedHilbert, 63
CRS, 69
CuHyb, 73
DD_MATRIX, 79
FBICRS, 87
HBICRS, 95
Hilbert, 99
HTS, 116
ICRS, 121
McCRS, 137
RDBHilbert, 154
RDCSB, 159
RDScheme, 164
SparseMatrix, 183
SVM, 189, 190
Triplet, 199
TS, 204
vecBICRS, 217
ZZ_CRS, 224
ZZ_ICRS, 229
loadCRS
Triplet, 199
loadFromFile
CCSWrapper, 59
SparseMatrix, 183
local_y
RDB_shared_data, 149
RDScheme_shared_data, 168
shared_data, 179
low
BigInt, 27

m
CCSWrapper, 59
Matrix, 126
SparseMatrix, 183
MAX_DEPTH
BlockHilbert, 39
MKLCRS
∼MKLCRS, 146
_col_ind, 146
_m, 146
_n, 146
_one, 147
_row_start, 147
descr, 147
MKLCRS, 145
MKLCRS, 145
mv, 146
trans, 147
zax, 146
MKLCRS< T >, 143
MachineInfo, 124
mask1
CBICRS, 52
mask2
CBICRS, 52
Matrix
∼Matrix, 126
bytesUsed, 126
m, 126
Matrix, 126
mv, 127
n, 127
nzs, 127
ZXa, 131
ZaX, 129
zax, 127, 129
zxa, 129, 131
Matrix< T >, 124
Matrix.hpp, 232
Matrix2HilbertCoordinates, 131
BITWIDTH, 132
IntegerToHilbert, 132
McCRS
∼McCRS, 136
bytesUsed, 136
col_ind, 139
columnIndices, 136
ds, 139
find, 136
getFirstIndexPair, 136
load, 137
McCRS, 135
McCRS, 135
mv, 137
random_access, 137
row_start, 139
rowJump, 137
values, 137
zax, 137
McCRS< T >, 133
memoryUsage
CBICRS, 50
MinCCS
in_readout, 140
post_readout, 140
pre_readout, 141
MinCCS< T >, 139

MinCRS
in_readout, 142
post_readout, 142
pre_readout, 142
MinCRS< T >, 141
mode
RDB_shared_data, 149
moveToNext
HilbertArray, 104
HilbertArrayInterface, 108
moveToStart
HilbertArray, 104
HilbertArrayInterface, 108
mutex
RDB_shared_data, 149
RDScheme_shared_data, 168
shared_data, 179
mv
BetaHilbert, 15
CCSWrapper, 59
Matrix, 127
McCRS, 137
MKLCRS, 146
RDBHilbert, 154
RDCSB, 159
RDScheme, 164
SparseMatrix, 183

n
CCSWrapper, 60
Matrix, 127
SparseMatrix, 184
NO_SUCH_NODE
SBDTree, 173
next
Upscaler::treeInOrderIterator, 192
Upscaler::treeIterator, 194
Upscaler::treePostOrderIterator, 196
nnz
SparseMatrix, 185
nodes
SBDTree, 173
nonzeroes
Upscaler, 211
nor
SparseMatrix, 185
ntt
BICRS, 24
CBICRS, 52
HBICRS, 96
nzs
CCSWrapper, 60
DD_MATRIX, 80
Matrix, 127
SparseMatrix, 184

operator unsigned char
BigInt, 26
operator unsigned int
BigInt, 26
operator unsigned long int
BigInt, 26
operator unsigned short int
BigInt, 26
operator()
HilbertTripletCompare, 113
operator=
BigInt, 26
original
RDB_shared_data, 149
RDScheme_shared_data, 168
shared_data, 180
output
BlockOrderer, 43
RDB_shared_data, 150
shared_data, 180
output_vector_offset
RDB_shared_data, 150
RDScheme_shared_data, 168
shared_data, 180
output_vector_size
RDB_shared_data, 150
RDScheme_shared_data, 168
shared_data, 180

P
RDB_shared_data, 150
p
Upscaler::treeIterator, 194
p_processed
Upscaler::treeIterator, 194
parent
SBDTree, 173
position
Upscaler::treeIterator, 194
post_height
BlockOrderer, 41
post_readout
BlockCRS, 31
BlockOrderer, 42
Duck, 82
MinCCS, 140
MinCRS, 142
SepLast, 176
U2, 206
postProcessRowIncrements
vecBICRS, 217
postfix
BlockOrderer, 42
pre_height
BlockOrderer, 42
pre_readout
BlockCRS, 31
BlockOrderer, 42
Duck, 82
MinCCS, 141
MinCRS, 142
SepLast, 176

U2, 207
prefix
BlockOrderer, 42
prepareBlockIteration
vecBICRS, 218
processed
Upscaler::treeIterator, 194

r_end
BICRS, 24
FBICRS, 90
r_hi
SBDTree, 174
r_inc
BICRS, 24
CBICRS, 52
FBICRS, 90
HBICRS, 96
r_ind
ICRS, 123
vecBICRS, 220
ZZ_ICRS, 230
r_lo
SBDTree, 174
r_start
BICRS, 25
CBICRS, 52
FBICRS, 91
RDB_shared_data
cond, 149
end_cond, 149
end_mutex, 149
end_sync, 149
id, 149
input, 149
local_y, 149
mode, 149
mutex, 149
original, 149
output, 150
output_vector_offset, 150
output_vector_size, 150
P, 150
RDB_shared_data, 148
RDB_shared_data, 148
sync, 150
RDB_shared_data< T >, 147
RDBHilbert
∼RDBHilbert, 153
bytesUsed, 153
collectY, 153
end, 154
getFirstIndexPair, 154
load, 154
mv, 154
RDBHilbert, 153
RDBHilbert, 153
reset, 154
set_p_translate, 155
synchronise, 155
thread, 155
threads, 156
wait, 155
zax, 155, 156
zxa, 156
RDBHilbert< T, MatrixType >, 150
RDCSB
getFirstIndexPair, 159
load, 159
mv, 159
RDCSB< T >, 157
RDCSB_shared_data
RDCSB_shared_data, 160
RDCSB_shared_data, 160
RDCSB_shared_data< T >, 159
RDScheme
∼RDScheme, 163
bytesUsed, 163
collectY, 163
end, 164
getFirstIndexPair, 164
load, 164
mv, 164
RDScheme, 163
RDScheme, 163
synchronise, 164
thread, 165
zxa, 165
RDScheme< T, DS >, 161
RDScheme_shared_data
bytes, 167
cond, 167
end_cond, 167
end_mutex, 167
end_sync, 167
local_y, 168
mutex, 168
original, 168
output_vector_offset, 168
output_vector_size, 168
RDScheme_shared_data, 167
RDScheme_shared_data, 167
sync, 168
RDScheme_shared_data< T >, 165
READOUT
BlockOrderer, 43
random_access
CRS, 69
McCRS, 137
SVM, 190
readout
Upscaler, 211
reset
RDBHilbert, 154
right
SBDTree, 172
right_child

SBDTree, 174
rightpass
BlockOrderer, 44
root
SBDTree, 174
root_is_set
SBDTree, 174
row
HilbertTriplet, 112
Triplet, 201
row_start
CRS, 70
McCRS, 139
ZZ_CRS, 225
rowBounds
SBDTree, 172
rowJump
CRS, 69
McCRS, 137
rowOffset
Triplet, 199

SBDTree, 168
∼SBDTree, 170
build, 171
c_hi, 173
c_lo, 173
columnBounds, 172
getSeparatorBB, 172
left, 172
left_child, 173
NO_SUCH_NODE, 173
nodes, 173
parent, 173
r_hi, 174
r_lo, 174
right, 172
right_child, 174
root, 174
root_is_set, 174
rowBounds, 172
SBDTree, 170
SBDTree, 170
up, 173
SELF_ALLOCATED
DD_MATRIX, 80
SVM
∼SVM, 189
bytesUsed, 189
find, 189
getData, 189
getFirstIndexPair, 189
load, 189, 190
random_access, 190
SVM, 188
SVM, 188
zax, 190
zxa, 191
SVM< T >, 186
save
HilbertTriplet, 111
Triplet, 200
saveBinary
Hilbert, 100
HTS, 116
SepLast
in_readout, 175
post_readout, 176
pre_readout, 176
SepLast< T >, 174
set_p_translate
BetaHilbert, 17
RDBHilbert, 155
setColumnPosition
Triplet, 200
setPosition
Triplet, 200
setRowPosition
Triplet, 200
setStartingPos
ICRS, 122
shared_data
cond, 179
end_cond, 179
end_mutex, 179
end_sync, 179
input, 179
local_y, 179
mutex, 179
original, 180
output, 180
output_vector_offset, 180
output_vector_size, 180
shared_data, 177
shared_data, 177
sync, 180
shared_data< T >, 176
SparseMatrix
∼SparseMatrix, 182
getFirstIndexPair, 182
load, 183
loadFromFile, 183
m, 183
mv, 183
n, 184
nnz, 185
nor, 185
nzs, 184
SparseMatrix, 182
SparseMatrix, 182
zax, 184
zero_element, 185
zxa, 185
SparseMatrix< T, IND >, 180
start_coor
HilbertArray, 106
sync

RDB_shared_data, 150
RDScheme_shared_data, 168
shared_data, 180
synchronise
BetaHilbert, 17
RDBHilbert, 155
RDScheme, 164

TRAVERSE_HEIGHT
BlockOrderer, 44
TS
∼TS, 203
bytesUsed, 203
ds, 205
getFirstIndexPair, 203
i, 205
j, 205
load, 204
TS, 203
TS, 203
ZXa, 204
ZaX, 204
zax, 204
zxa, 204
TS< T >, 201
thread
BetaHilbert, 17
RDBHilbert, 155
RDScheme, 165
threads
RDBHilbert, 156
trans
MKLCRS, 147
transpose
Triplet, 200
transposeVector
CCSWrapper, 60
traverse
BlockOrderer, 42
traverse_mode
BlockOrderer, 44
tree
BlockOrderer, 44
treeInOrderIterator
Upscaler::treeInOrderIterator, 192
treeIterator
Upscaler::treeIterator, 193
treePostOrderIterator
Upscaler::treePostOrderIterator, 196
Triplet
column, 201
i, 198
j, 198
load, 199
loadCRS, 199
row, 201
rowOffset, 199
save, 200
setColumnPosition, 200
setPosition, 200
setRowPosition, 200
transpose, 200
Triplet, 198
value, 201
Triplet< T >, 196

U2
in_readout, 206
post_readout, 206
pre_readout, 207
U2< T >, 205
up
SBDTree, 173
updateMinMax
Upscaler, 211
Upscaler
∼Upscaler, 209
containsPID, 211
determineEmpty, 209
getSubTree, 209, 210
nonzeroes, 211
readout, 211
updateMinMax, 211
Upscaler, 209
Upscaler< T >, 207
Upscaler< T >::treeInOrderIterator, 191
Upscaler< T >::treeIterator, 193
Upscaler< T >::treePostOrderIterator, 195
Upscaler::treeInOrderIterator
next, 192
treeInOrderIterator, 192
Upscaler::treeIterator
ID, 194
next, 194
p, 194
p_processed, 194
position, 194
processed, 194
treeIterator, 193
walk, 195
Upscaler::treePostOrderIterator
next, 196
treePostOrderIterator, 196

vals
BICRS, 25
CBICRS, 52
HBICRS, 96
value
HilbertTriplet, 112
Triplet, 201
values
CRS, 69
McCRS, 137
vecBICRS
∼vecBICRS, 216
addPadding, 216
allocate, 216

allocsize, 219
block_length, 219
bytes, 219
bytesUsed, 216
c_ind, 219
compareTriplets, 217
ds, 220
fillIn, 220
getFirstIndexPair, 217
jumpsize, 220
load, 217
postProcessRowIncrements, 217
prepareBlockIteration, 218
r_ind, 220
vecBICRS, 214, 215
vecBICRS, 214, 215
zax, 218
zxa, 218
vecBICRS< T, block_length_row, block_length_column, _i_value >, 212

wait
BetaHilbert, 17
RDBHilbert, 155
walk
Upscaler::treeIterator, 195

ZXa
CRS, 70
FBICRS, 89
ICRS, 123
Matrix, 131
TS, 204
ZZ_CRS
∼ZZ_CRS, 223
bytesUsed, 223
col_ind, 225
compareTripletsLTR, 224
compareTripletsRTL, 224
ds, 225
getFirstIndexPair, 224
load, 224
row_start, 225
ZZ_CRS, 222, 223
zax, 224
zxa, 225
ZZ_CRS, 222, 223
ZZ_CRS< T, _i_value >, 220
ZZ_ICRS
∼ZZ_ICRS, 228
bytesUsed, 228
c_ind, 230
compareTriplets, 229
ds, 230
getFirstIndexPair, 229
jumps, 230
load, 229
r_ind, 230
ZZ_ICRS, 227, 228
zax, 229
zxa, 229
ZZ_ICRS, 227, 228
ZZ_ICRS< T >, 225
ZaX
CRS, 69
FBICRS, 88
ICRS, 122
Matrix, 129
TS, 204
zax
BetaHilbert, 18
BICRS, 23
BlockHilbert, 38
CBICRS, 50
CCSWrapper, 60
CompressedHilbert, 64
CRS, 69
CuHyb, 74
DD_MATRIX, 79
FBICRS, 88
HBICRS, 95
Hilbert, 100
HilbertArray, 105
HilbertArrayInterface, 108
HTS, 117
ICRS, 122
Matrix, 127, 129
McCRS, 137
MKLCRS, 146
RDBHilbert, 155, 156
SparseMatrix, 184
SVM, 190
TS, 204
vecBICRS, 218
ZZ_CRS, 224
ZZ_ICRS, 229
zax_fb
BICRS, 23
FBICRS, 88
zero_element
SparseMatrix, 185
zxa
BetaHilbert, 18
BICRS, 23
BlockHilbert, 38
CBICRS, 51
CCSWrapper, 60
CompressedHilbert, 64
DD_MATRIX, 79
FBICRS, 89
HBICRS, 95
Hilbert, 100
HilbertArray, 105
HilbertArrayInterface, 108
HTS, 117
ICRS, 122
Matrix, 129, 131

RDBHilbert, 156
RDScheme, 165
SparseMatrix, 185
SVM, 191
TS, 204
vecBICRS, 218
ZZ_CRS, 225
ZZ_ICRS, 229
zxa_fb
FBICRS, 89