Matrix Test with Strasson Algorithm

Generated by Doxygen 1.7.6.1

Wed Sep 16 2015 14:03:07

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 matrix Struct Reference

```
#include <matrix.h>
```

Data Fields

- double ** a

3.1.1 Field Documentation

3.1.1.1 double** matrix::a

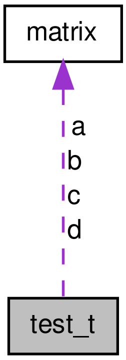
The documentation for this struct was generated from the following file:

- src/matrix.h

3.2 test_t Struct Reference

```
#include <test.h>
```

Collaboration diagram for test_t:



Data Fields

- int n
- int rn
- matrix ∗ a
- matrix ∗ b
- matrix ∗ c
- matrix ∗ d

3.2.1 Field Documentation

3.2.1.1 matrix∗ test_t::a

3.2.1.2 matrix∗ test_t::b

3.2.1.3 matrix∗ test_t::c

3.2.1.4 matrix∗ test_t::d

3.2.1.5 int test_t::n

3.2.1.6 int test_t::rn

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

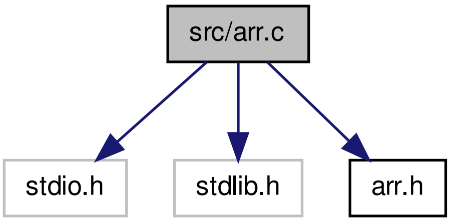
- src/test.h

Chapter 4

File Documentation

4.1 src/arr.c File Reference

`#include <stdio.h> #include <stdlib.h> #include "arr.h"`×
Include dependency graph for arr.c:



Functions

- int round_up_power_of_two (int n)

  round up a number to power of two.
- void arr_mul (double ∗∗a, double ∗∗b, double ∗∗c, int n)

  perform matrices multiply, C = A mul B.
- void arr_add (double ∗∗a, double ∗∗b, double ∗∗c, int n)

  perform matrices addition, C = A + B.
- void arr_sub (double ∗∗a, double ∗∗b, double ∗∗c, int n)

  perform matrices subtraction, C = A + B.

- double ∗∗ new_arr (int m, int n)

  create a new array with dimension m by n.
- void del_arr (double ∗∗r, int m)

  release the resource of an array.
- void dump_arr (double ∗∗r, int m, int n)

  dump the content of an array
- void copy_arr (double ∗∗dest, double ∗∗src, int m, int n)

  copy the content of an array to another one.
- void rand_arr (double ∗∗r, int m, int n)

  assign random data to an array.
- void set_ones_arr (double ∗∗r, int m, int n)

  assign all 1's data to an array.
- void set_seqs_arr (double ∗∗r, int m, int n)

  assign sequential data to an array, i.e.
- int comp_arr (double ∗∗a, double ∗∗b, int m, int n)

  compare the content of two arrays.

4.1.1 Function Documentation

4.1.1.1 void arr_add (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *n*)

perform matrices addition, C = A + B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices operation.
n	the dimension of input/output matrices

4.1.1.2 void arr_mul (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *n*)

perform matrices multiply, C = A mul B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices multiply operation.
n	the dimension of input/output matrices

4.1.1.3 void arr_sub (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *n*)

perform matrices subtraction, C = A + B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices operation.
n	the dimension of input/output matrices

4.1.1.4 int comp_arr (double ∗∗ *a*, double ∗∗ *b*, int *m*, int *n*)

compare the content of two arrays.

Parameters

a	the pointer to array A.
b	the pointer to array B.
m	the row size of the arrays.
n	the column size of the arrays.

Returns

0 indicates A and B are the same. 1 inticates A and B are different.

4.1.1.5 void copy_arr (double ∗∗ *dest*, double ∗∗ *src*, int *m*, int *n*)

copy the content of an array to another one.

Parameters

dest	the pointer to the destination array.
src	the pointer to the source array.
m	the row size of the arrays.
n	the column size of the arrays.

4.1.1.6 void del_arr (double ∗∗ *r*, int *m*)

release the resource of an array.

Parameters

r	the pointer to an array.
m	the row size of an array.

4.1.1.7 void dump_arr (double ∗∗ *r*, int *m*, int *n*)

dump the content of an array

Parameters

r	the pointer to the array
m	the row size of the array
n	the column size of the array

4.1.1.8 double∗∗ new_arr (int *m*, int *n*)

create a new array with dimension m by n.

Parameters

m	the row size of an array.
n	the column size of an array.

Returns

an array with dimension m by n.

4.1.1.9 void rand_arr (double ∗∗ *r*, int *m*, int *n*)

assign random data to an array.

Parameters

r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.1.1.10 int round_up_power_of_two (int *n*)

round up a number to power of two.

it's used to expand the array with its dimension to be power-of-two.

Parameters

n	the number to be rounded-up

Returns

the rounded-up number

4.1.1.11 void set_ones_arr (double ∗∗ *r*, int *m*, int *n*)

assign all 1's data to an array.

Parameters

r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.1.1.12 void set_seqs_arr (double ∗∗ *r,* int *m,* int *n*)

assign sequential data to an array, i.e.

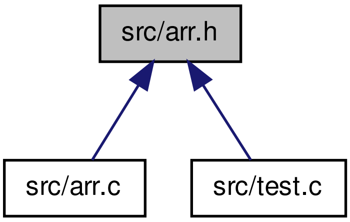
all rows contains data {0, 1, 2, .. , n - 1}.

Parameters

r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.2 src/arr.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define MAX_DIM 10000
- #define MAX_DATA_VALUE 46340
- #define MIN_DATA_VALUE -46340

Functions

- int round_up_power_of_two (int n)

round up a number to power of two.

- void arr_mul (double **a, double **b, double **c, int n)

  perform matrices multiply, C = A mul B.
- void arr_add (double **a, double **b, double **c, int n)

  perform matrices addition, C = A + B.
- void arr_sub (double **a, double **b, double **c, int n)

  perform matrices subtraction, C = A + B.
- double ** new_arr (int m, int n)

  create a new array with dimension m by n.
- void del_arr (double **r, int m)

  release the resource of an array.
- double ** new_arr_p (int m)
- void del_arr_p (double **r)
- void dump_arr (double **r, int m, int n)

  dump the content of an array
- void copy_arr (double **dest, double **src, int m, int n)

  copy the content of an array to another one.
- void rand_arr (double **r, int m, int n)

  assign random data to an array.
- void set_ones_arr (double **r, int m, int n)

  assign all 1's data to an array.
- void set_seqs_arr (double **r, int m, int n)

  assign sequential data to an array, i.e.
- void sub_arr (double **a, double **b, double **c, int m, int n)
- int comp_arr (double **a, double **b, int m, int n)

  compare the content of two arrays.

4.2.1 Define Documentation

4.2.1.1 #define **MAX_DATA_VALUE** 46340

4.2.1.2 #define **MAX_DIM** 10000

4.2.1.3 #define **MIN_DATA_VALUE** -46340

4.2.2 Function Documentation

4.2.2.1 void arr_add (double ** *a*, double ** *b*, double ** *c*, int *n*)

perform matrices addition, C = A + B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices operation.
n	the dimension of input/output matrices

4.2.2.2 void **arr_mul** (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *n*)

perform matrices multiply, C = A mul B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices multiply operation.
n	the dimension of input/output matrices

4.2.2.3 void **arr_sub** (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *n*)

perform matrices subtraction, C = A + B.

Parameters

a	the input matrix A
b	the input matrix B
c	the result of matrices operation.
n	the dimension of input/output matrices

4.2.2.4 int **comp_arr** (double ∗∗ *a,* double ∗∗ *b,* int *m,* int *n*)

compare the content of two arrays.

Parameters

a	the pointer to array A.
b	the pointer to array B.
m	the row size of the arrays.
n	the column size of the arrays.

Returns

0 indicates A and B are the same. 1 inticates A and B are different.

4.2.2.5 void **copy_arr** (double ∗∗ *dest,* double ∗∗ *src,* int *m,* int *n*)

copy the content of an array to another one.

Parameters

dest	the pointer to the destination array.
src	the pointer to the source array.
m	the row size of the arrays.
n	the column size of the arrays.

4.2.2.6 void del_arr (double $**$ *r*, int *m*)

release the resource of an array.

Parameters

r	the pointer to an array.
m	the row size of an array.

4.2.2.7 void del_arr_p (double $**$ *r*)

4.2.2.8 void dump_arr (double $**$ *r*, int *m*, int *n*)

dump the content of an array

Parameters

r	the pointer to the array
m	the row size of the array
n	the column size of the array

4.2.2.9 double$**$ new_arr (int *m*, int *n*)

create a new array with dimension m by n.

Parameters

m	the row size of an array.
n	the column size of an array.

Returns

an array with dimension m by n.

4.2.2.10 double$**$ new_arr_p (int *m*)

4.2.2.11 void rand_arr (double $**$ *r*, int *m*, int *n*)

assign random data to an array.

Parameters

r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.2.2.12 int round_up_power_of_two (int *n*)

round up a number to power of two.

it’s used to expand the array with its dimension to be power-of-two.

Parameters

n	the number to be rounded-up

Returns

the rounded-up number

4.2.2.13 void set_ones_arr (double ∗∗ *r,* int *m,* int *n*)

assign all 1’s data to an array.

Parameters

r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.2.2.14 void set_seqs_arr (double ∗∗ *r,* int *m,* int *n*)

assign sequential data to an array, i.e.

all rows contains data {0, 1, 2, .. , n - 1}.

Parameters

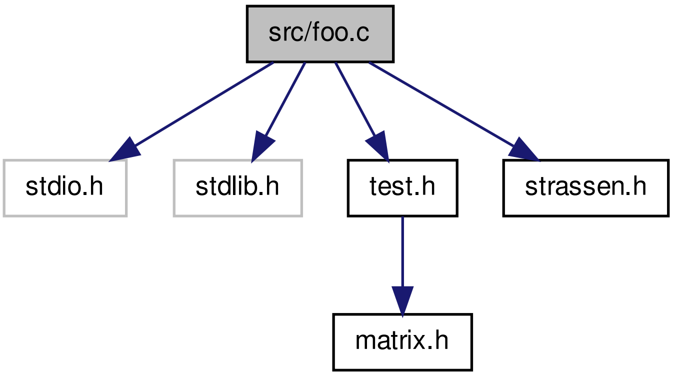
r	the pointer to an array.
m	the row size of the array.
n	the column size of the array.

4.2.2.15 void sub_arr (double ∗∗ *a,* double ∗∗ *b,* double ∗∗ *c,* int *m,* int *n*)

4.3 src/foo.c File Reference

```
#include <stdio.h> #include <stdlib.h> #include "test.h" ×
```

`#include "strassen.h"` Include dependency graph for foo.c:



Functions

- int main (int argc, const char ∗argv[])

  main the entry of this project.

4.3.1 Function Documentation

4.3.1.1 int **main** (int *argc*, const char ∗ *argv[]*)

main the entry of this project.

Parameters

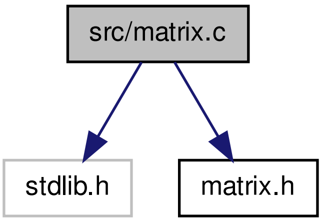
argc	the number of string parameters.
argv[]	the content of input string parameters. argv[0] the command name argv[1] the dimension of the test square matrices. argv[2] the operations of the test. argv[3] the number of g_break used in strassen algorithm. argv[4] the patterns generated for test others reserved.

Returns

int 0 as program finished well.

4.4 src/matrix.c File Reference

`#include <stdlib.h>` `#include "matrix.h"` Include dependency graph for matrix.c:



Functions

- matrix ∗ new_matrix (double ∗∗a)

  create a new matrix structure.
- void del_matrix (matrix ∗a)

  release the resource of a matrix

4.4.1 Function Documentation

4.4.1.1 void del_matrix (matrix ∗ *a*)

release the resource of a matrix

Parameters

a	the pointer to the matrix.

4.4.1.2 matrix∗ new_matrix (double ∗∗ *a*)

create a new matrix structure.

Parameters

a	the pointer to a matrix with type 'double' elements.

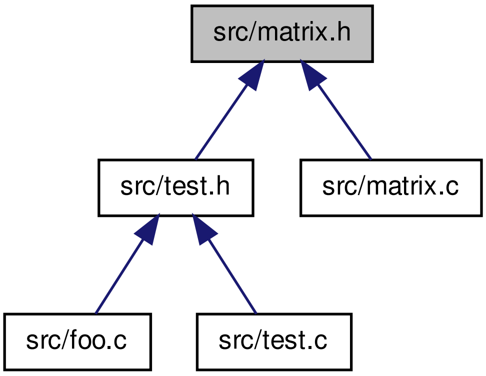
Returns

Note

the matrix structure can be expanded as necessary.

4.5 src/matrix.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct matrix

Functions

- matrix ∗ new_matrix (double ∗∗a)

  create a new matrix structure.
- void del_matrix (matrix ∗a)

  release the resource of a matrix

4.5.1 Function Documentation

4.5.1.1 void **del_matrix** (**matrix** ∗ *a*)

release the resource of a matrix

Parameters

a	the pointer to the matrix.

4.5.1.2 **matrix**∗ **new_matrix** (double ∗∗ *a*)

create a new matrix structure.

Parameters

a	the pointer to a matrix with type 'double' elements.

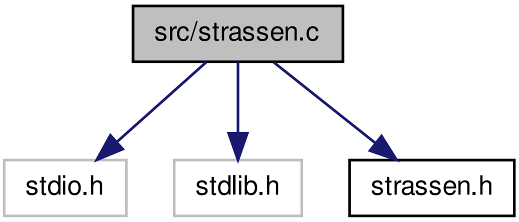
Returns

Note

the matrix structure can be expanded as necessary.

4.6 src/strassen.c File Reference

`#include <stdio.h>` `#include <stdlib.h>` `#include "strassen.h"` Include dependency graph for strassen.c:



Functions

- double *** arr_qsplit (double **a, int n)

  split a given array into 4 equal sub-arrays.
- void arr_qsplit_recall (double ***a)

  release the resources allocated for array split.
- void s_mul (int n, double **a, double **b, double **c, double **d)

  perform the strassen multiply for matrices.
- void s_add (int n, double **a, double **b, double **c)

  perform the addition operation of matrices with matrices dividing.
- void s_sub (int n, double **a, double **b, double **c)

  perform the substraction operation of matrices with matrices dividing.

Variables

- int g_break = 16

4.6.1 Function Documentation

4.6.1.1 double*** arr_qsplit (double ** *a,* int *n*)

split a given array into 4 equal sub-arrays.

Parameters

a	an input {2n x 2n} array
n	the dimension of new arrays, i.e. {n x n}.

Returns

a pointer to 4 arrays.

| AAAABBBB | | AAAABBBB | | AAAABBBB | | AAAA | | BBBB | | CCCC | | DDDD | | AAAABBBB | -> | AAAA | , | BBBB | , | CCCC | , | DDDD | | CCCCDDDD | | AAAA | | BBBB | | CCCC | | DDDD | | CCCCDDDD | | AAAA | | BBBB | | CCCC | | DDDD | | CCCCDDDD | | CCCCDDDD |

4.6.1.2 void arr_qsplit_recall (double *** *a*)

release the resources allocated for array split.

Parameters

a	a pointer to 4 arrays.

4.6.1.3 void s_add (int *n*, double ∗∗ *a*, double ∗∗ *b*, double ∗∗ *c*)

perform the addition operation of matrices with matrices dividing.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a adds matrix b.

if the dimension of input matrices are less than g_break, perform a normal matrix substraction operation. else, divide the matrices and calculate the result of each sub matrices.

4.6.1.4 void s_mul (int *n*, double ∗∗ *a*, double ∗∗ *b*, double ∗∗ *c*, double ∗∗ *d*)

perform the strassen multiply for matrices.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a multiplies matrix b.
d	the scratchpad for calculation.

Note

> the algorithm is refer to wiki, https://en.wikipedia.org/wiki/-Strassen_algorithm

if the dimension of input matrices are less than g_break, perform a normal O(N^3) matrix multiply operation. else, divide the matrices and perform strassen algorithm.

4.6.1.5 void s_sub (int *n*, double ∗∗ *a*, double ∗∗ *b*, double ∗∗ *c*)

perform the substraction operation of matrices with matrices dividing.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a substracts matrix b.

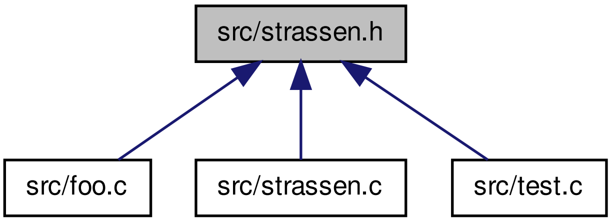
if the dimension of input matrices are less than g_break, perform a normal matrix addition operation. else, divide the matrices and calculate the result of each sub matrices.

4.6.2 Variable Documentation

4.6.2.1 int g_break = 16

4.7 src/strassen.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define a11 p[0]
- #define a12 p[1]
- #define a21 p[2]
- #define a22 p[3]
- #define b11 q[0]
- #define b12 q[1]
- #define b21 q[2]
- #define b22 q[3]
- #define c11 r[0]
- #define c12 r[1]
- #define c21 r[2]
- #define c22 r[3]
- #define d11 s[0]
- #define d12 s[1]
- #define d21 s[2]
- #define d22 s[3]

Functions

- double *** arr_qsplit (double **a, int n) *split a given array into 4 equal sub-arrays.*

- void arr_qsplit_recall (double ∗∗∗a)

  release the resources allocated for array split.

- void s_mul (int n, double ∗∗a, double ∗∗b, double ∗∗c, double ∗∗d)

  perform the strassen multiply for matrices.

- void s_add (int n, double ∗∗a, double ∗∗b, double ∗∗c)

  perform the addition operation of matrices with matrices dividing.

- void s_sub (int n, double ∗∗a, double ∗∗b, double ∗∗c)

  perform the substraction operation of matrices with matrices dividing.

Variables

- int g_break

4.7.1 Define Documentation

4.7.1.1 #define a11 p[0]

4.7.1.2 #define a12 p[1]

4.7.1.3 #define a21 p[2]

4.7.1.4 #define a22 p[3]

4.7.1.5 #define b11 q[0]

4.7.1.6 #define b12 q[1]

4.7.1.7 #define b21 q[2]

4.7.1.8 #define b22 q[3]

4.7.1.9 #define c11 r[0]

4.7.1.10 #define c12 r[1]

4.7.1.11 #define c21 r[2]

4.7.1.12 #define c22 r[3]

4.7.1.13 #define d11 s[0]

4.7.1.14 #define d12 s[1]

4.7.1.15 #define d21 s[2]

4.7.1.16 #define d22 s[3]

4.7.2 Function Documentation

4.7.2.1 double*** arr_qsplit (double ** *a*, int *n*)

split a given array into 4 equal sub-arrays.

Parameters

a	an input {2n x 2n} array
n	the dimension of new arrays, i.e. {n x n}.

Returns

a pointer to 4 arrays.

| AAAABBBB | | AAAABBBB | | AAAABBBB | | AAAA | | BBBB | | CCCC | | DDDD | | AAAABBBB | -> | AAAA | , | BBBB | , | CCCC | , | DDDD | | CCCCDDDD | | AAAA | | BBBB | | CCCC | | DDDD | | CCCCDDDD | | AAAA | | BBBB | | CCCC | | DDDD | | CCCCDDDD | | CCCCDDDD |

4.7.2.2 void arr_qsplit_recall (double *** *a*)

release the resources allocated for array split.

Parameters

a	a pointer to 4 arrays.

4.7.2.3 void s_add (int *n*, double ** *a*, double ** *b*, double ** *c*)

perform the addition operation of matrices with matrices dividing.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a adds matrix b.

if the dimension of input matrices are less than g_break, perform a normal matrix substraction operation. else, divide the matrices and calculate the result of each sub matrices.

4.7.2.4 void s_mul (int *n*, double ** *a*, double ** *b*, double ** *c*, double ** *d*)

perform the strassen multiply for matrices.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a multiplies matrix b.
d	the scratchpad for calculation.

Note

> the algorithm is refer to wiki, https://en.wikipedia.org/wiki/Strassen_algorithm

if the dimension of input matrices are less than g_break, perform a normal O(N^3) matrix multiply operation. else, divide the matrices and perform strassen algorithm.

4.7.2.5 void s_sub (int *n*, double ** *a*, double ** *b*, double ** *c*)

perform the substraction operation of matrices with matrices dividing.

Parameters

n	the dimension of the input and output matrices.
a	the first input matrix
b	the second input matrix
c	the result of matrix a substracts matrix b.

if the dimension of input matrices are less than g_break, perform a normal matrix addition operation. else, divide the matrices and calculate the result of each sub matrices.

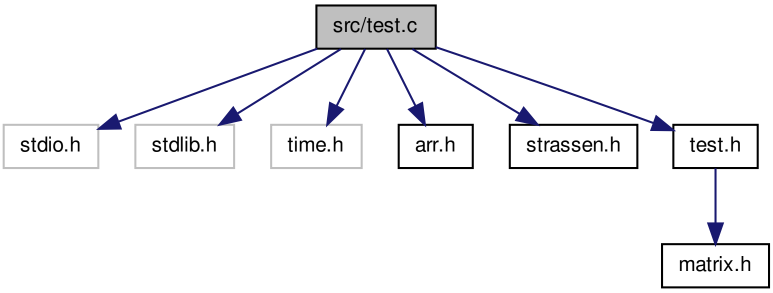
4.7.3 Variable Documentation

4.7.3.1 int g_break

4.8 src/test.c File Reference

```
#include <stdio.h> #include <stdlib.h> #include <time.h> #include "arr.h" #include "strassen.h" #include "test.-
```

h" Include dependency graph for test.c:



Functions

- test_t ∗ new_test (int n)

  create a new test object.
- void del_test (test_t ∗r)

  delete a test object and release its resources.
- void dump_result (test_t ∗t)

  dump test result
- void init_test_data (test_t ∗t, int pattern)

  set up data data for input marices.
- int test_strassen_multiply (test_t ∗t)

  perform C = A + S(A, B), while strassen algorithm is applied in S().
- int test_normal_multiply (test_t ∗t)

  perform C = A + M(A, B), while a normal matrix multiply function is applied in M().
- int test_check_result (test_t ∗t)

  check if the result of matrix c and d in test object are equal.
- int test_valid_data (test_t ∗t)

  validate each element test object input matrix is valid.
- int round_down_power_of_two (int n)

  round down a number to be power-of-two one.
- int test_tweak_breaks (int n)

  validate the value of g_break and tweak it.
- double test_case (int n, int ops)

  a test case for this matrix operation test.
- void test (int n, int ops)

  entry of test.

Variables

- int g_pattern = PATTERN_ONES

4.8.1 Function Documentation

4.8.1.1 void del_test (test_t ∗ *r*)

delete a test object and release its resources.

Parameters

r	a test object.

4.8.1.2 void dump_result (test_t ∗ *t*)

dump test result

Parameters

t	a test object

4.8.1.3 void init_test_data (test_t ∗ *t*, int *pattern*)

set up data data for input marices.

Parameters

t	a test object.
pattern	the pattern of given data.

PATTERN_RANDOM apply random data to the test object input. PATTERN_ONES apply all 1's to the test object input. PATTERN_SEQS apply sequential data, {0, 1, 2, .., n-1} for each row, to the object input.

4.8.1.4 test_t∗ new_test (int *n*)

create a new test object.

Parameters

n	the dimension of a matrix.

Returns

a pointer to new test object.

4.8.1.5 int round_down_power_of_two (int *n*)

round down a number to be power-of-two one.

Parameters

n	the number to be rounded-down

Returns

the rounded-down number

4.8.1.6 void test (int *n,* int *ops*)

entry of test.

Parameters

n	the dimension of input matrices.
ops	the operation code of test

OP_STRASSEN_MULTIPLY perform C = A + S(A, B), where {A, B, C} are mapped to {a, b, c} in test object. OP_NORMAL_MULTIPLY perform D = A + M(A, B), where {A, B, D} are mapped to {a, b, d} in test object. OP_VERIFY_CORRECTNESS per form C = A + S(A, B) and D = A + M(A, B), where {A, B, C, D} are mapped to {a, b, c, d} in test object. And then, compare the matrix C and D to see if the results are the same.

4.8.1.7 double test_case (int *n,* int *ops*)

a test case for this matrix operation test.

Parameters

n	the dimension of input metrices.
ops	the operations for this test.

Returns

the elapsed time in mini-second.

Note

the users can modify the time elapse functions in their specific platform to calculate the performance of the matrix operation test.

4.8.1.8 int **test_check_result** (**test_t** ∗ *t*)

check if the result of matrix c and d in test object are equal.

Parameters

t	the test object.

Returns

0 as equal;, and -1 as inequal.

4.8.1.9 int **test_normal_multiply** (**test_t** ∗ *t*)

perform C = A + M(A, B), while a normal matrix multiply function is applied in M().

Parameters

t	a test object.

Returns

0 as passed.

4.8.1.10 int **test_strassen_multiply** (**test_t** ∗ *t*)

perform C = A + S(A, B), while strassen algorithm is applied in S().

Parameters

t	a test object.

Returns

0 as passed.

4.8.1.11 int **test_tweak_breaks** (int *n*)

validate the value of g_break and tweak it.

Parameters

n	the number to be validated and tweaked.

Returns

the number has been tweaked.

Note

MAX_DIM is a predefined number for max dimension of the matrices.

4.8.1.12 int test_valid_data (test_t ∗ *t*)

validate each element test object input matrix is valid.

Parameters

t	the test object

Returns

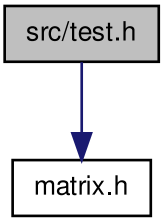
0 as all data are valid. -1 as input matrix a is invalid. -2 as input matrix b is invalid.

4.8.2 Variable Documentation

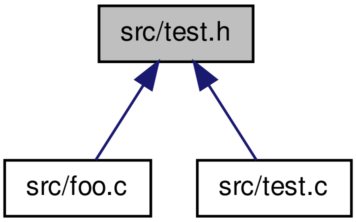
4.8.2.1 int g_pattern = PATTERN_ONES

4.9 src/test.h File Reference

`#include "matrix.h"` Include dependency graph for test.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct test_t

Enumerations

- enum PATTERN_MODE { PATTERN_RANDOM = 0, PATTERN_ONES, PATTERN_SEQS, NUMBER_OF_PATTERNS }
- enum OPS { OP_STRASSEN_MULTIPLY = 0, OP_NORMAL_MULTIPLY, OP_VERIFY_CORRECTNESS, OP_DUMP_RESULT, NUMBER_OF_OPS }

Functions

- test_t ∗ new_test (int n)

  create a new test object.
- void del_test (test_t ∗r)

  delete a test object and release its resources.
- void dump_result (test_t ∗t)

  dump test result
- void init_test_data (test_t ∗t, int pattern)

  set up data data for input marices.
- int test_strassen_multiply (test_t ∗t)

  perform C = A + S(A, B), while strassen algorithm is applied in S().
- int test_normal_multiply (test_t ∗t)

  perform C = A + M(A, B), while a normal matrix multiply function is applied in M().
- int test_check_result (test_t ∗t)

  check if the result of matrix c and d in test object are equal.

- int test_valid_data (test_t ∗t)

  validate each element test object input matrix is valid.
- int round_down_power_of_two (int n)

  round down a number to be power-of-two one.
- int test_tweak_breaks (int n)

  validate the value of g_break and tweak it.
- double test_case (int n, int ops)

  a test case for this matrix operation test.
- void test (int n, int ops)

  entry of test.

Variables

- int g_pattern

4.9.1 Enumeration Type Documentation

4.9.1.1 enum OPS

Enumerator:

OP_STRASSEN_MULTIPLY

OP_NORMAL_MULTIPLY

OP_VERIFY_CORRECTNESS

OP_DUMP_RESULT

NUMBER_OF_OPS

4.9.1.2 enum PATTERN_MODE

Enumerator:

PATTERN_RANDOM

PATTERN_ONES

PATTERN_SEQS

NUMBER_OF_PATTERNS

4.9.2 Function Documentation

4.9.2.1 void del_test (test_t ∗ *r*)

delete a test object and release its resources.

Parameters

r	a test object.

4.9.2.2 void **dump_result** (**test_t** ∗ *t*)

dump test result

Parameters

t	a test object

4.9.2.3 void **init_test_data** (**test_t** ∗ *t,* int *pattern*)

set up data data for input marices.

Parameters

t	a test object.
pattern	the pattern of given data.

PATTERN_RANDOM apply random data to the test object input. PATTERN_ONES apply all 1's to the test object input. PATTERN_SEQS apply sequential data, {0, 1, 2, .., n-1} for each row, to the object input.

4.9.2.4 **test_t**∗ **new_test** (int *n*)

create a new test object.

Parameters

n	the dimension of a matrix.

Returns

a pointer to new test object.

4.9.2.5 int **round_down_power_of_two** (int *n*)

round down a number to be power-of-two one.

Parameters

n	the number to be rounded-down

Returns

the rounded-down number

4.9.2.6 void test (int *n,* int *ops*)

entry of test.

Parameters

n	the dimension of input matrices.
ops	the operation code of test

OP_STRASSEN_MULTIPLY perform C = A + S(A, B), where {A, B, C} are mapped to {a, b, c} in test object. OP_NORMAL_MULTIPLY perform D = A + M(A, B), where {A, B, D} are mapped to {a, b, d} in test object. OP_VERIFY_CORRECTNESS per form C = A + S(A, B) and D = A + M(A, B), where {A, B, C, D} are mapped to {a, b, c, d} in test object. And then, compare the matrix C and D to see if the results are the same.

4.9.2.7 double test_case (int *n,* int *ops*)

a test case for this matrix operation test.

Parameters

n	the dimension of input metrices.
ops	the operations for this test.

Returns

the elapsed time in mini-second.

Note

the users can modify the time elapse functions in their specific platform to calculate the performance of the matrix operation test.

4.9.2.8 int test_check_result (test_t ∗ *t*)

check if the result of matrix c and d in test object are equal.

Parameters

t	the test object.

Returns

0 as equal;, and -1 as inequal.

4.9.2.9 int **test_normal_multiply** (**test_t** ∗ *t*)

perform C = A + M(A, B), while a normal matrix multiply function is applied in M().

Parameters

t	a test object.

Returns

0 as passed.

4.9.2.10 int **test_strassen_multiply** (**test_t** ∗ *t*)

perform C = A + S(A, B), while strassen algorithm is applied in S().

Parameters

t	a test object.

Returns

0 as passed.

4.9.2.11 int **test_tweak_breaks** (int *n*)

validate the value of g_break and tweak it.

Parameters

n	the number to be validated and tweaked.

Returns

the number has been tweaked.

Note

MAX_DIM is a predefined number for max dimension of the matrices.

4.9.2.12 int test_valid_data (test_t $*$ *t*)

validate each element test object input matrix is valid.

Parameters

t	the test object

Returns

0 as all data are valid. -1 as input matrix a is invalid. -2 as input matrix b is invalid.

4.9.3 Variable Documentation

4.9.3.1 int g_pattern