

phasefield-accelerator-benchmarks
pre-alpha

Generated by Doxygen 1.8.13

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	2
2.1	File List	2
3	Class Documentation	3
3.1	Stopwatch Struct Reference	3
3.1.1	Detailed Description	3
3.1.2	Member Data Documentation	3
4	File Documentation	4
4.1	analytic_main.c File Reference	4
4.1.1	Detailed Description	5
4.1.2	Function Documentation	5
4.2	boundaries.h File Reference	5
4.2.1	Detailed Description	6
4.2.2	Function Documentation	6
4.3	cuda_boundaries.c File Reference	7
4.3.1	Detailed Description	7
4.3.2	Function Documentation	7
4.4	cuda_discretization.cu File Reference	8
4.4.1	Detailed Description	9
4.4.2	Function Documentation	9
4.5	discretization.h File Reference	11
4.5.1	Detailed Description	12
4.5.2	Function Documentation	12
4.6	main.c File Reference	13
4.6.1	Detailed Description	13
4.6.2	Function Documentation	14

4.7	mesh.c File Reference	14
4.7.1	Detailed Description	15
4.7.2	Function Documentation	15
4.8	mesh.h File Reference	16
4.8.1	Detailed Description	16
4.8.2	Function Documentation	17
4.9	numerics.c File Reference	17
4.9.1	Detailed Description	18
4.9.2	Function Documentation	18
4.10	numerics.h File Reference	20
4.10.1	Detailed Description	22
4.10.2	Function Documentation	22
4.11	openacc_boundaries.c File Reference	24
4.11.1	Detailed Description	25
4.11.2	Function Documentation	25
4.12	openacc_discretization.c File Reference	25
4.12.1	Detailed Description	26
4.12.2	Function Documentation	26
4.13	openmp_boundaries.c File Reference	27
4.13.1	Detailed Description	28
4.13.2	Function Documentation	28
4.14	openmp_discretization.c File Reference	29
4.14.1	Detailed Description	29
4.14.2	Function Documentation	29
4.15	output.c File Reference	31
4.15.1	Detailed Description	31
4.15.2	Function Documentation	31
4.16	output.h File Reference	32
4.16.1	Detailed Description	33
4.16.2	Function Documentation	33

4.17	serial_boundaries.c File Reference	34
4.17.1	Detailed Description	34
4.17.2	Function Documentation	34
4.18	serial_discretization.c File Reference	35
4.18.1	Detailed Description	36
4.18.2	Function Documentation	36
4.19	tbb_boundaries.cpp File Reference	37
4.19.1	Detailed Description	37
4.19.2	Function Documentation	38
4.20	tbb_discretization.cpp File Reference	38
4.20.1	Detailed Description	39
4.20.2	Function Documentation	39
4.21	timer.c File Reference	40
4.21.1	Detailed Description	41
4.21.2	Variable Documentation	41
4.22	timer.h File Reference	41
4.22.1	Detailed Description	41
4.23	type.h File Reference	42
4.23.1	Detailed Description	42
4.23.2	Typedef Documentation	42
	Index	43

1 Class Index

1.1 Class List

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

[Stopwatch](#)

3

2 File Index

2.1 File List

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

analytic_main.c	Analytical solution to semi-infinite diffusion equation	4
boundaries.h	Declaration of boundary condition function prototypes	5
cuda_boundaries.c	Implementation of boundary condition functions with OpenMP threading	7
cuda_discretization.cu	Implementation of boundary condition functions with CUDA acceleration	8
discretization.h	Declaration of discretized mathematical function prototypes	11
main.c	Implementation of semi-infinite diffusion equation	13
mesh.c	Implementation of mesh handling functions for diffusion benchmarks	14
mesh.h	Declaration of mesh function prototypes for diffusion benchmarks	16
numerics.c	Implementation of Laplacian operator and analytical solution functions	17
numerics.h	Declaration of Laplacian operator and analytical solution functions	20
openacc_boundaries.c	Implementation of boundary condition functions with OpenACC threading	24
openacc_discretization.c	Implementation of boundary condition functions with OpenACC threading	25
openmp_boundaries.c	Implementation of boundary condition functions with OpenMP threading	27
openmp_discretization.c	Implementation of boundary condition functions with OpenMP threading	29
output.c	Implementation of file output functions for diffusion benchmarks	31
output.h	Declaration of output function prototypes for diffusion benchmarks	32
serial_boundaries.c	Implementation of boundary condition functions without threading	34
serial_discretization.c	Implementation of boundary condition functions without threading	35

tbb_boundaries.cpp	Implementation of boundary condition functions with TBB threading	37
tbb_discretization.cpp	Implementation of boundary condition functions with TBB threading	38
timer.c	High-resolution cross-platform machine time reader	40
timer.h	Declaration of timer function prototypes for diffusion benchmarks	41
type.h	Definition of scalar data type and Doxygen diffusion group	42

3 Class Documentation

3.1 Stopwatch Struct Reference

```
#include <type.h>
```

Public Attributes

- double [conv](#)
- double [step](#)
- double [file](#)
- double [soln](#)

3.1.1 Detailed Description

Container for timing data

Definition at line 39 of file type.h.

3.1.2 Member Data Documentation

3.1.2.1 conv

```
double Stopwatch::conv
```

Cumulative time executing [compute_convolution\(\)](#)

Definition at line 43 of file type.h.

3.1.2.2 file

```
double Stopwatch::file
```

Cumulative time executing [write_csv\(\)](#) and [write_png\(\)](#)

Definition at line 53 of file type.h.

3.1.2.3 soln

```
double Stopwatch::soln
```

Cumulative time executing [check_solution\(\)](#)

Definition at line 58 of file type.h.

3.1.2.4 step

```
double Stopwatch::step
```

Cumulative time executing [solve_diffusion_equation\(\)](#)

Definition at line 48 of file type.h.

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

- [type.h](#)

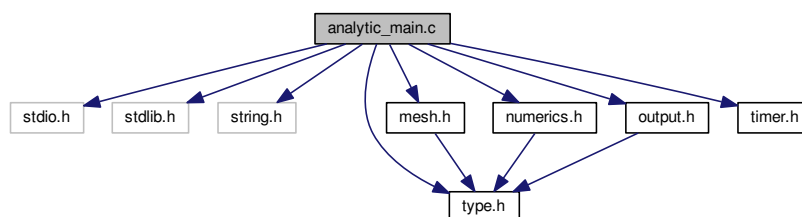
4 File Documentation

4.1 analytic_main.c File Reference

Analytical solution to semi-infinite diffusion equation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "type.h"
#include "mesh.h"
#include "numerics.h"
#include "output.h"
#include "timer.h"
```

Include dependency graph for analytic_main.c:



Functions

- void `solve_diffusion_equation` (`fp_t` **conc, int nx, int ny, int nm, `fp_t` dx, `fp_t` dy, `fp_t` D, `fp_t` dt, `fp_t` elapsed)
Update the scalar composition field using analytical solution.
- int `main` (int argc, char *argv[])
Find analytical solution at intervals specified in the parameters file.

4.1.1 Detailed Description

Analytical solution to semi-infinite diffusion equation.

4.1.2 Function Documentation

4.1.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Find analytical solution at intervals specified in the parameters file.

Program will write a series of PNG image files to visualize the scalar composition field, useful for qualitative verification of numerical results.

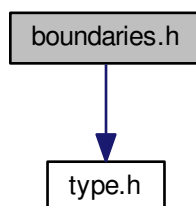
Definition at line 70 of file `analytic_main.c`.

4.2 boundaries.h File Reference

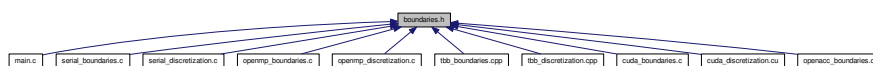
Declaration of boundary condition function prototypes.

```
#include "type.h"
```

Include dependency graph for `boundaries.h`:



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



Functions

- void `set_boundaries` (`fp_t` `bc[2][2]`)
Set values to be used along the simulation domain boundaries.
- void `apply_initial_conditions` (`fp_t` `**conc_old`, `int` `nx`, `int` `ny`, `int` `nm`, `fp_t` `bc[2][2]`)
Initialize flat composition field with fixed boundary conditions.
- void `apply_boundary_conditions` (`fp_t` `**conc_old`, `int` `nx`, `int` `ny`, `int` `nm`, `fp_t` `bc[2][2]`)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.2.1 Detailed Description

Declaration of boundary condition function prototypes.

4.2.2 Function Documentation

4.2.2.1 `apply_initial_conditions()`

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 37 of file `serial_boundaries.c`.

4.2.2.2 `set_boundaries()`

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

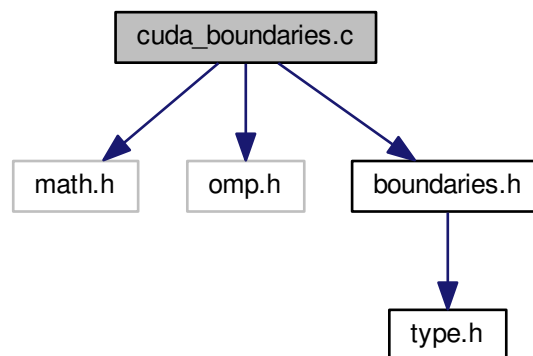
Definition at line 28 of file `serial_boundaries.c`.

4.3 cuda_boundaries.c File Reference

Implementation of boundary condition functions with OpenMP threading.

```
#include <math.h>
#include <omp.h>
#include "boundaries.h"
```

Include dependency graph for cuda_boundaries.c:



Functions

- void [set_boundaries](#) ([fp_t](#) bc[2][2])
Set values to be used along the simulation domain boundaries.
- void [apply_initial_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.3.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.3.2 Function Documentation

4.3.2.1 apply_initial_conditions()

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 38 of file cuda_boundaries.c.

4.3.2.2 set_boundaries()

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

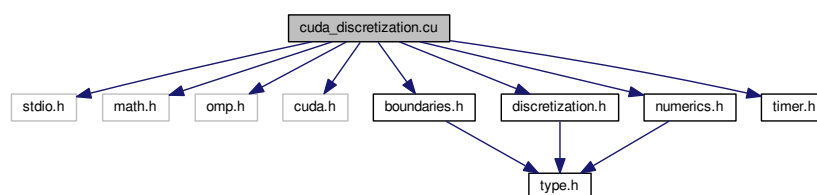
Definition at line 29 of file cuda_boundaries.c.

4.4 cuda_discretization.cu File Reference

Implementation of boundary condition functions with CUDA acceleration.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <cuda.h>
#include "boundaries.h"
#include "discretization.h"
#include "numerics.h"
#include "timer.h"
```

Include dependency graph for cuda_discretization.cu:



Macros

- `#define MAX_TILE_W 32`
Maximum width of an input tile, including halo cells, for GPU memory allocation.
- `#define MAX_TILE_H 32`
Maximum height of an input tile, including halo cells, for GPU memory allocation.

Functions

- `__global__ void convolution_kernel (fp_t *conc_old, fp_t *conc_lap, int nx, int ny, int nm)`
Tiled convolution algorithm for execution on the GPU
This function accesses 1D data rather than the 2D array representation of the scalar composition field, mapping into 2D tiles on the GPU with halo cells before computing the convolution. Note:
- `void compute_convolution (fp_t **conc_old, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm)`
Perform the convolution of the mask matrix with the composition matrix.
- `__global__ void diffusion_kernel (fp_t *conc_old, fp_t *conc_new, fp_t *conc_lap, int nx, int ny, int nm, fp_t D, fp_t dt)`
Vector addition algorithm for execution on the GPU
This function accesses 1D data rather than the 2D array representation of the scalar composition field.
- `void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, fp_t *elapsed, struct Stopwatch *sw)`
Update the scalar composition field using old and Laplacian values.
- `void check_solution (fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t *rss)`
Compare numerical and analytical solutions of the diffusion equation.

Variables

- `__constant__ fp_t Mc [MAX_MASK_W * MAX_MASK_H]`
Convolution mask array on the GPU, allocated in protected memory.

4.4.1 Detailed Description

Implementation of boundary condition functions with CUDA acceleration.

4.4.2 Function Documentation

4.4.2.1 check_solution()

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 203 of file *cuda_discretization.cu*.

4.4.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions *nx*, *ny*, and *nm* are properly specified, the convolution will be correctly computed.

Definition at line 123 of file *cuda_discretization.cu*.

4.4.2.3 convolution_kernel()

```
void convolution_kernel (
    fp_t * conc_old,
    fp_t * conc_lap,
    int nx,
    int ny,
    int nm )
```

Tiled convolution algorithm for execution on the GPU

This function accesses 1D data rather than the 2D array representation of the scalar composition field, mapping into 2D tiles on the GPU with halo cells before computing the convolution. Note:

- The source matrix (*conc_old*) and destination matrix (*conc_lap*) must be identical in size
- One CUDA core operates on one array index: there is no nested loop over matrix elements
- The halo ($nm/2$ perimeter cells) in *conc_lap* are unallocated garbage
- The same cells in *conc_old* are boundary values, and contribute to the convolution
- *conc_tile* is the shared tile of input data, accessible by all threads in this block

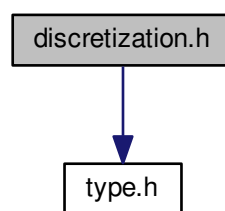
Definition at line 66 of file `cuda_discretization.cu`.

4.5 discretization.h File Reference

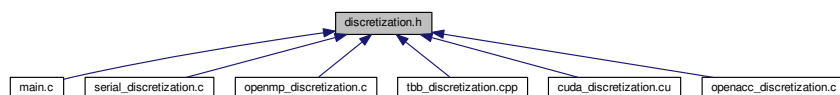
Declaration of discretized mathematical function prototypes.

```
#include "type.h"
```

Include dependency graph for `discretization.h`:



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



Functions

- void `compute_convolution` (`fp_t` **conc_old, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void `solve_diffusion_equation` (`fp_t` **conc_old, `fp_t` **conc_new, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm, `fp_t` bc[2][2], `fp_t` D, `fp_t` dt, `fp_t` *elapsed, struct `Stopwatch` *sw)
Update the scalar composition field using old and Laplacian values.
- void `check_solution` (`fp_t` **conc_new, `fp_t` **conc_lap, int nx, int ny, `fp_t` dx, `fp_t` dy, int nm, `fp_t` elapsed, `fp_t` D, `fp_t` bc[2][2], `fp_t` *rss)
Compare numerical and analytical solutions of the diffusion equation.

4.5.1 Detailed Description

Declaration of discretized mathematical function prototypes.

4.5.2 Function Documentation

4.5.2.1 `check_solution()`

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites `conc_lap`, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 73 of file `serial_discretization.c`.

4.5.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions *nx*, *ny*, and *nm* are properly specified, the convolution will be correctly computed.

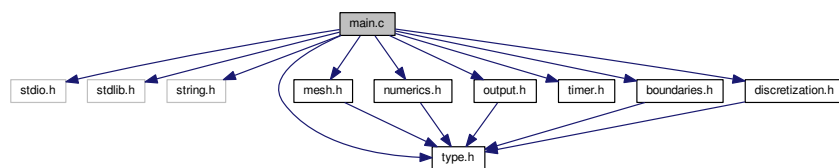
Definition at line 31 of file serial_discretization.c.

4.6 main.c File Reference

Implementation of semi-infinite diffusion equation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "type.h"
#include "mesh.h"
#include "numerics.h"
#include "output.h"
#include "timer.h"
#include "boundaries.h"
#include "discretization.h"
```

Include dependency graph for main.c:



Functions

- int [main](#) (int argc, char *argv[])

Run simulation using input parameters specified on the command line.

4.6.1 Detailed Description

Implementation of semi-infinite diffusion equation.

4.6.2 Function Documentation

4.6.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Run simulation using input parameters specified on the command line.

Program will write a series of PNG image files to visualize scalar composition field, plus a final CSV raw data file and CSV runtime log tabulating the iteration counter (*iter*), elapsed simulation time (*sim_time*), system free energy (*energy*), error relative to analytical solution (*wrss*), time spent performing convolution (*conv_time*), time spent updating fields (*step_time*), time spent writing to disk (*IO_time*), time spent generating analytical values (*soln_time*), and total elapsed (*run_time*).

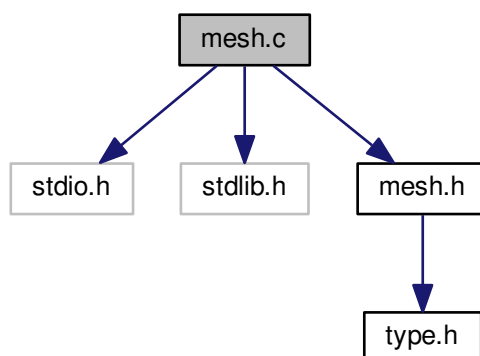
Definition at line 48 of file main.c.

4.7 mesh.c File Reference

Implementation of mesh handling functions for diffusion benchmarks.

```
#include <stdio.h>
#include <stdlib.h>
#include "mesh.h"
```

Include dependency graph for mesh.c:



Functions

- void `make_arrays` (`fp_t ***conc_old`, `fp_t ***conc_new`, `fp_t ***conc_lap`, `fp_t ***mask_lap`, `int nx`, `int ny`, `int nm`)
Allocate 2D arrays to store scalar composition values.
- void `free_arrays` (`fp_t **conc_old`, `fp_t **conc_new`, `fp_t **conc_lap`, `fp_t **mask_lap`)
Free dynamically allocated memory.
- void `swap_pointers` (`fp_t ***conc_old`, `fp_t ***conc_new`)
Swap pointers to data underlying two arrays.

4.7.1 Detailed Description

Implementation of mesh handling functions for diffusion benchmarks.

4.7.2 Function Documentation

4.7.2.1 make_arrays()

```
void make_arrays (
    fp_t *** conc_old,
    fp_t *** conc_new,
    fp_t *** conc_lap,
    fp_t *** mask_lap,
    int nx,
    int ny,
    int nm )
```

Allocate 2D arrays to store scalar composition values.

Arrays are allocated as 1D arrays, then 2D pointer arrays are mapped over the top. This facilitates use of either 1D or 2D data access, depending on whether the task is spatially dependent or not.

Definition at line 29 of file mesh.c.

4.7.2.2 swap_pointers()

```
void swap_pointers (
    fp_t *** conc_old,
    fp_t *** conc_new )
```

Swap pointers to data underlying two arrays.

Rather than copy data from *conc_old* into *conc_new*, an expensive operation, simply trade the top-most pointers. New becomes old with no data lost and in almost no time.

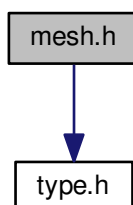
Definition at line 73 of file mesh.c.

4.8 mesh.h File Reference

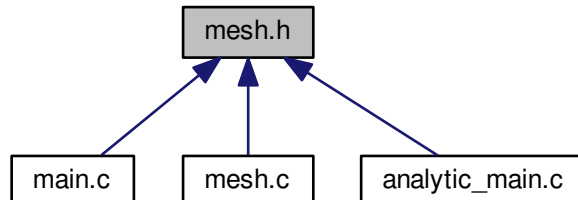
Declaration of mesh function prototypes for diffusion benchmarks.

```
#include "type.h"
```

Include dependency graph for mesh.h:



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



Functions

- void [make_arrays](#) ([fp_t](#) ***conc_old, [fp_t](#) ***conc_new, [fp_t](#) ***conc_lap, [fp_t](#) ***mask_lap, int nx, int ny, int nm)
Allocate 2D arrays to store scalar composition values.
- void [free_arrays](#) ([fp_t](#) **conc_old, [fp_t](#) **conc_new, [fp_t](#) **conc_lap, [fp_t](#) **mask_lap)
Free dynamically allocated memory.
- void [swap_pointers](#) ([fp_t](#) ***conc_old, [fp_t](#) ***conc_new)
Swap pointers to data underlying two arrays.

4.8.1 Detailed Description

Declaration of mesh function prototypes for diffusion benchmarks.

4.8.2 Function Documentation

4.8.2.1 make_arrays()

```
void make_arrays (
    fp_t *** conc_old,
    fp_t *** conc_new,
    fp_t *** conc_lap,
    fp_t *** mask_lap,
    int nx,
    int ny,
    int nm )
```

Allocate 2D arrays to store scalar composition values.

Arrays are allocated as 1D arrays, then 2D pointer arrays are mapped over the top. This facilitates use of either 1D or 2D data access, depending on whether the task is spatially dependent or not.

Definition at line 29 of file mesh.c.

4.8.2.2 swap_pointers()

```
void swap_pointers (
    fp_t *** conc_old,
    fp_t *** conc_new )
```

Swap pointers to data underlying two arrays.

Rather than copy data from *conc_old* into *conc_new*, an expensive operation, simply trade the top-most pointers. New becomes old with no data lost and in almost no time.

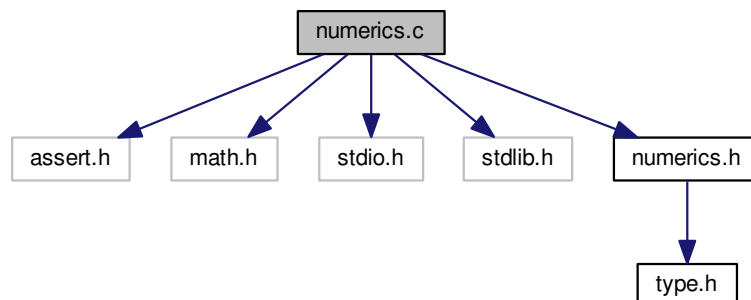
Definition at line 73 of file mesh.c.

4.9 numerics.c File Reference

Implementation of Laplacian operator and analytical solution functions.

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "numerics.h"
```

Include dependency graph for numerics.c:



Functions

- void `set_mask` (`fp_t` dx, `fp_t` dy, int code, `fp_t` **mask_lap, int nm)
Specify which stencil (mask) to use for the Laplacian (convolution)
- void `five_point_Laplacian_stencil` (`fp_t` dx, `fp_t` dy, `fp_t` **mask_lap, int nm)
Write 5-point Laplacian stencil into convolution mask.
- void `nine_point_Laplacian_stencil` (`fp_t` dx, `fp_t` dy, `fp_t` **mask_lap, int nm)
Write 9-point Laplacian stencil into convolution mask.
- void `slow_nine_point_Laplacian_stencil` (`fp_t` dx, `fp_t` dy, `fp_t` **mask_lap, int nm)
Write 9-point Laplacian stencil into convolution mask.
- `fp_t` `euclidean_distance` (`fp_t` ax, `fp_t` ay, `fp_t` bx, `fp_t` by)
Compute Euclidean distance between two points, a and b.
- `fp_t` `manhattan_distance` (`fp_t` ax, `fp_t` ay, `fp_t` bx, `fp_t` by)
Compute Manhattan distance between two points, a and b.
- `fp_t` `distance_point_to_segment` (`fp_t` ax, `fp_t` ay, `fp_t` bx, `fp_t` by, `fp_t` px, `fp_t` py)
Compute minimum distance from point p to a line segment bounded by points a and b.
- void `analytical_value` (`fp_t` x, `fp_t` t, `fp_t` D, `fp_t` bc[2][2], `fp_t` *c)
Analytical solution of the diffusion equation for a carburizing process.

4.9.1 Detailed Description

Implementation of Laplacian operator and analytical solution functions.

4.9.2 Function Documentation

4.9.2.1 `analytical_value()`

```
void analytical_value (
    fp_t x,
    fp_t t,
    fp_t D,
    fp_t bc[2][2],
    fp_t * c )
```

Analytical solution of the diffusion equation for a carburizing process.

For 1D diffusion through a semi-infinite domain with initial and far-field composition c_∞ and boundary value $c(x = 0, t) = c_0$ with constant diffusivity D , the solution to Fick's second law is

$$c(x, t) = c_0 - (c_0 - c_\infty) \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right)$$

which reduces, when $c_\infty = 0$, to

$$c(x, t) = c_0 \left[1 - \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right) \right].$$

Definition at line 121 of file numerics.c.

4.9.2.2 distance_point_to_segment()

```
fp_t distance_point_to_segment (
    fp_t ax,
    fp_t ay,
    fp_t bx,
    fp_t by,
    fp_t px,
    fp_t py )
```

Compute minimum distance from point p to a line segment bounded by points a and b .

This function computes the projection of p onto ab , limiting the projected range to $[0, 1]$ to handle projections that fall outside of ab . Implemented after Grumdrig on Stackoverflow, <https://stackoverflow.com/a/1501725>.

Definition at line 108 of file numerics.c.

4.9.2.3 five_point_Laplacian_stencil()

```
void five_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 5-point Laplacian stencil into convolution mask.

3×3 mask, 5 values, truncation error $\mathcal{O}(\Delta x^2)$

Definition at line 51 of file numerics.c.

4.9.2.4 nine_point_Laplacian_stencil()

```
void nine_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 9-point Laplacian stencil into convolution mask.

3×3 mask, 9 values, truncation error $\mathcal{O}(\Delta x^4)$

Definition at line 62 of file numerics.c.

4.9.2.5 set_mask()

```
void set_mask (
    fp_t dx,
    fp_t dy,
    int code,
    fp_t ** mask_lap,
    int nm )
```

Specify which stencil (mask) to use for the Laplacian (convolution)

The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, e.g. 53 specifies [five_point_Laplacian_stencil\(\)](#), while 93 specifies [nine_point_Laplacian_stencil\(\)](#).

To add your own mask (stencil), add a case to this function with your chosen numerical encoding, then specify that code in the input parameters file (params.txt by default). Note that, for a Laplacian stencil, the sum of the coefficients must equal zero and *nm* must be an odd integer.

If your stencil is larger than 5×5 , you must increase the values defined by [MAX_MASK_W](#) and [MAX_MASK_H](#).

Definition at line 31 of file numerics.c.

4.9.2.6 slow_nine_point_Laplacian_stencil()

```
void slow_nine_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 9-point Laplacian stencil into convolution mask.

5×5 mask, 9 values, truncation error $\mathcal{O}(\Delta x^4)$

Provided for testing and demonstration of scalability, only: as the name indicates, this 9-point stencil is computationally more expensive than the 3×3 version. If your code requires $\mathcal{O}(\Delta x^4)$ accuracy, please use [nine_point_Laplacian_stencil\(\)](#).

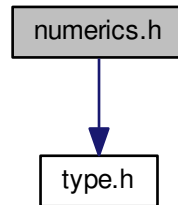
Definition at line 79 of file numerics.c.

4.10 numerics.h File Reference

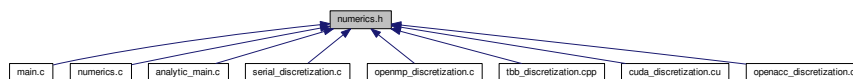
Declaration of Laplacian operator and analytical solution functions.

```
#include "type.h"
```

Include dependency graph for numerics.h:



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



Macros

- `#define MAX_MASK_W 5`
Maximum width of the convolution mask (Laplacian stencil) array.
- `#define MAX_MASK_H 5`
Maximum height of the convolution mask (Laplacian stencil) array.

Functions

- `void set_mask (fp_t dx, fp_t dy, int code, fp_t **mask_lap, int nm)`
Specify which stencil (mask) to use for the Laplacian (convolution)
- `void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap, int nm)`
Write 5-point Laplacian stencil into convolution mask.
- `void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap, int nm)`
Write 9-point Laplacian stencil into convolution mask.
- `void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap, int nm)`
Write 9-point Laplacian stencil into convolution mask.
- `fp_t euclidean_distance (fp_t ax, fp_t ay, fp_t bx, fp_t by)`
Compute Euclidean distance between two points, a and b.
- `fp_t manhattan_distance (fp_t ax, fp_t ay, fp_t bx, fp_t by)`
Compute Manhattan distance between two points, a and b.
- `fp_t distance_point_to_segment (fp_t ax, fp_t ay, fp_t bx, fp_t by, fp_t px, fp_t py)`
Compute minimum distance from point p to a line segment bounded by points a and b.
- `void analytical_value (fp_t x, fp_t t, fp_t D, fp_t bc[2][2], fp_t *c)`
Analytical solution of the diffusion equation for a carburizing process.

4.10.1 Detailed Description

Declaration of Laplacian operator and analytical solution functions.

4.10.2 Function Documentation

4.10.2.1 analytical_value()

```
void analytical_value (
    fp_t x,
    fp_t t,
    fp_t D,
    fp_t bc[2][2],
    fp_t * c )
```

Analytical solution of the diffusion equation for a carburizing process.

For 1D diffusion through a semi-infinite domain with initial and far-field composition c_∞ and boundary value $c(x = 0, t) = c_0$ with constant diffusivity D , the solution to Fick's second law is

$$c(x, t) = c_0 - (c_0 - c_\infty) \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right)$$

which reduces, when $c_\infty = 0$, to

$$c(x, t) = c_0 \left[1 - \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right) \right].$$

Definition at line 121 of file numerics.c.

4.10.2.2 distance_point_to_segment()

```
fp_t distance_point_to_segment (
    fp_t ax,
    fp_t ay,
    fp_t bx,
    fp_t by,
    fp_t px,
    fp_t py )
```

Compute minimum distance from point p to a line segment bounded by points a and b .

This function computes the projection of p onto ab , limiting the projected range to $[0, 1]$ to handle projections that fall outside of ab . Implemented after Grumdrig on Stackoverflow, <https://stackoverflow.com/a/1501725>.

Definition at line 108 of file numerics.c.

4.10.2.3 five_point_Laplacian_stencil()

```
void five_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 5-point Laplacian stencil into convolution mask.

3×3 mask, 5 values, truncation error $\mathcal{O}(\Delta x^2)$

Definition at line 51 of file numerics.c.

4.10.2.4 nine_point_Laplacian_stencil()

```
void nine_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 9-point Laplacian stencil into convolution mask.

3×3 mask, 9 values, truncation error $\mathcal{O}(\Delta x^4)$

Definition at line 62 of file numerics.c.

4.10.2.5 set_mask()

```
void set_mask (
    fp_t dx,
    fp_t dy,
    int code,
    fp_t ** mask_lap,
    int nm )
```

Specify which stencil (mask) to use for the Laplacian (convolution)

The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, *e.g.* 53 specifies [five_point_Laplacian_stencil\(\)](#), while 93 specifies [nine_point_Laplacian_stencil\(\)](#).

To add your own mask (stencil), add a case to this function with your chosen numerical encoding, then specify that code in the input parameters file (params.txt by default). Note that, for a Laplacian stencil, the sum of the coefficients must equal zero and *nm* must be an odd integer.

If your stencil is larger than 5×5 , you must increase the values defined by [MAX_MASK_W](#) and [MAX_MASK_H](#).

Definition at line 31 of file numerics.c.

4.10.2.6 slow_nine_point_Laplacian_stencil()

```
void slow_nine_point_Laplacian_stencil (
    fp_t dx,
    fp_t dy,
    fp_t ** mask_lap,
    int nm )
```

Write 9-point Laplacian stencil into convolution mask.

5×5 mask, 9 values, truncation error $\mathcal{O}(\Delta x^4)$

Provided for testing and demonstration of scalability, only: as the name indicates, this 9-point stencil is computationally more expensive than the 3×3 version. If your code requires $\mathcal{O}(\Delta x^4)$ accuracy, please use [nine_point_Laplacian_stencil\(\)](#).

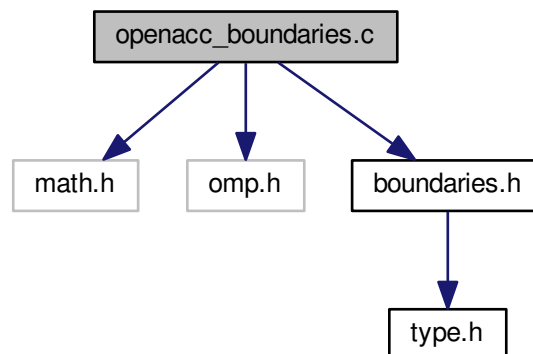
Definition at line 79 of file numerics.c.

4.11 openacc_boundaries.c File Reference

Implementation of boundary condition functions with OpenMP threading.

```
#include <math.h>
#include <omp.h>
#include "boundaries.h"
```

Include dependency graph for openacc_boundaries.c:



Functions

- void [set_boundaries](#) ([fp_t](#) bc[2][2])
Set values to be used along the simulation domain boundaries.
- void [apply_initial_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.11.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.11.2 Function Documentation

4.11.2.1 apply_initial_conditions()

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 38 of file openacc_boundaries.c.

4.11.2.2 set_boundaries()

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

Definition at line 29 of file openacc_boundaries.c.

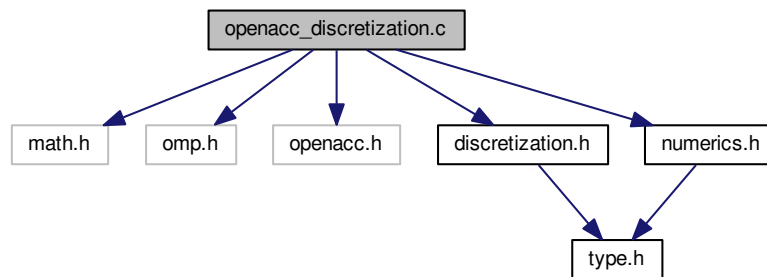
4.12 openacc_discretization.c File Reference

Implementation of boundary condition functions with OpenACC threading.

```
#include <math.h>
#include <omp.h>
#include <openacc.h>
#include "discretization.h"
```

```
#include "numerics.h"
```

Include dependency graph for openacc_discretization.c:



Functions

- void `compute_convolution` (`fp_t **conc_old`, `fp_t **conc_lap`, `fp_t **mask_lap`, `int nx`, `int ny`, `int nm`)
Perform the convolution of the mask matrix with the composition matrix.
- void `solve_diffusion_equation` (`fp_t **conc_old`, `fp_t **conc_new`, `fp_t **conc_lap`, `fp_t **mask_lap`, `int nx`, `int ny`, `int nm`, `fp_t bc[2][2]`, `fp_t D`, `fp_t dt`, `fp_t *elapsed`, `struct Stopwatch *sw`)
Update the scalar composition field using old and Laplacian values.
- void `check_solution` (`fp_t **conc_new`, `fp_t **conc_lap`, `int nx`, `int ny`, `fp_t dx`, `fp_t dy`, `int nm`, `fp_t elapsed`, `fp_t D`, `fp_t bc[2][2]`, `fp_t *rss`)
Compare numerical and analytical solutions of the diffusion equation.

4.12.1 Detailed Description

Implementation of boundary condition functions with OpenACC threading.

4.12.2 Function Documentation

4.12.2.1 `check_solution()`

```

void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )

```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 86 of file openacc_discretization.c.

4.12.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

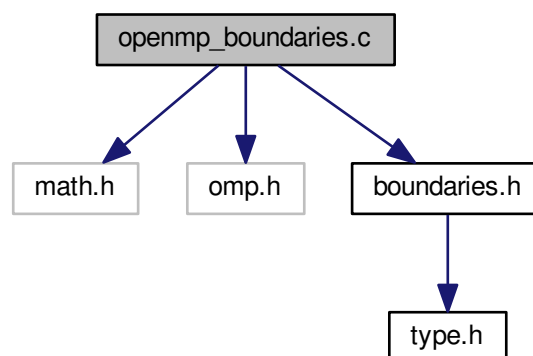
If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions *nx*, *ny*, and *nm* are properly specified, the convolution will be correctly computed.

Definition at line 31 of file openacc_discretization.c.

4.13 openmp_boundaries.c File Reference

Implementation of boundary condition functions with OpenMP threading.

```
#include <math.h>
#include <omp.h>
#include "boundaries.h"
Include dependency graph for openmp_boundaries.c:
```



Functions

- void `set_boundaries` (`fp_t` bc[2][2])
Set values to be used along the simulation domain boundaries.
- void `apply_initial_conditions` (`fp_t` **conc, int nx, int ny, int nm, `fp_t` bc[2][2])
Initialize flat composition field with fixed boundary conditions.
- void `apply_boundary_conditions` (`fp_t` **conc, int nx, int ny, int nm, `fp_t` bc[2][2])
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.13.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.13.2 Function Documentation

4.13.2.1 `apply_initial_conditions()`

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 39 of file `openmp_boundaries.c`.

4.13.2.2 `set_boundaries()`

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

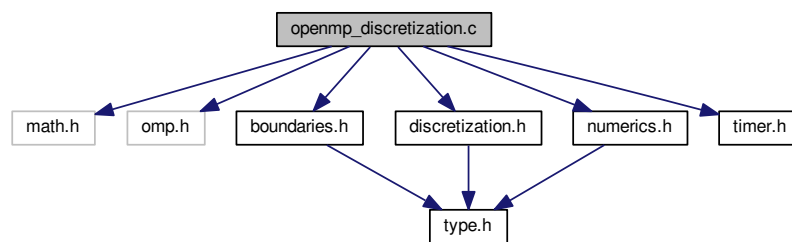
Definition at line 30 of file `openmp_boundaries.c`.

4.14 openmp_discretization.c File Reference

Implementation of boundary condition functions with OpenMP threading.

```
#include <math.h>
#include <omp.h>
#include "boundaries.h"
#include "discretization.h"
#include "numerics.h"
#include "timer.h"
```

Include dependency graph for openmp_discretization.c:



Functions

- void [compute_convolution](#) ([fp_t](#) **conc_old, [fp_t](#) **conc_lap, [fp_t](#) **mask_lap, int nx, int ny, int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void [solve_diffusion_equation](#) ([fp_t](#) **conc_old, [fp_t](#) **conc_new, [fp_t](#) **conc_lap, [fp_t](#) **mask_lap, int nx, int ny, int nm, [fp_t](#) bc[2][2], [fp_t](#) D, [fp_t](#) dt, [fp_t](#) *elapsed, struct [Stopwatch](#) *sw)
Update the scalar composition field using old and Laplacian values.
- void [check_solution](#) ([fp_t](#) **conc_new, [fp_t](#) **conc_lap, int nx, int ny, [fp_t](#) dx, [fp_t](#) dy, int nm, [fp_t](#) elapsed, [fp_t](#) D, [fp_t](#) bc[2][2], [fp_t](#) *rss)
Compare numerical and analytical solutions of the diffusion equation.

4.14.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.14.2 Function Documentation

4.14.2.1 check_solution()

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 80 of file *openmp_discretization.c*.

4.14.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

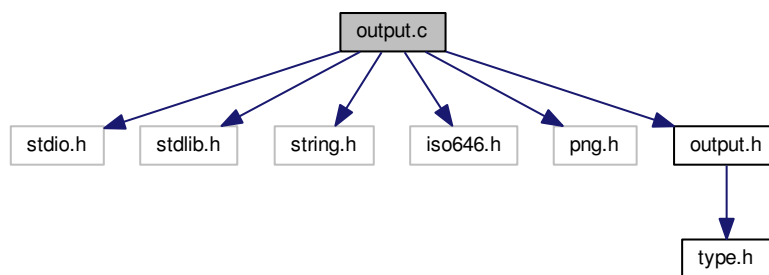
If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions *nx*, *ny*, and *nm* are properly specified, the convolution will be correctly computed.

Definition at line 32 of file *openmp_discretization.c*.

4.15 output.c File Reference

Implementation of file output functions for diffusion benchmarks.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iso646.h>
#include <png.h>
#include "output.h"
Include dependency graph for output.c:
```



Functions

- void [param_parser](#) (int argc, char *argv[], int *nx, int *ny, int *nm, int *code, [fp_t](#) *dx, [fp_t](#) *dy, [fp_t](#) *D, [fp_t](#) *linStab, int *steps, int *checks)
Read parameters from file specified on the command line.
- void [print_progress](#) (const int step, const int steps)
Prints timestamps and a 20-point progress bar to stdout.
- void [write_csv](#) ([fp_t](#) **conc, int nx, int ny, [fp_t](#) dx, [fp_t](#) dy, int step)
Writes scalar composition field to diffusion.???????.csv.
- void [write_png](#) ([fp_t](#) **conc, int nx, int ny, int step)
Writes scalar composition field to diffusion.???????.png.

4.15.1 Detailed Description

Implementation of file output functions for diffusion benchmarks.

4.15.2 Function Documentation

4.15.2.1 print_progress()

```
void print_progress (
    const int step,
    const int steps )
```

Prints timestamps and a 20-point progress bar to stdout.

Call inside the timestepping loop, near the top, e.g.

```
for (int step=0; step<steps; step++) {
    print_progress(step, steps);
    take_a_step();
    elapsed += dt;
}
```

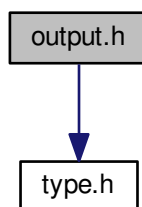
Definition at line 123 of file output.c.

4.16 output.h File Reference

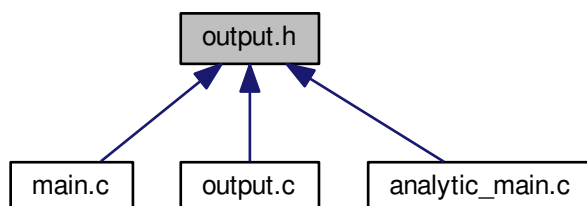
Declaration of output function prototypes for diffusion benchmarks.

```
#include "type.h"
```

Include dependency graph for output.h:



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



Functions

- void `param_parser` (int argc, char *argv[], int *nx, int *ny, int *nm, int *code, `fp_t` *dx, `fp_t` *dy, `fp_t` *D, `fp_t` *linStab, int *steps, int *checks)
Read parameters from file specified on the command line.
- void `print_progress` (const int step, const int steps)
Prints timestamps and a 20-point progress bar to stdout.
- void `write_csv` (`fp_t` **conc, int nx, int ny, `fp_t` dx, `fp_t` dy, int step)
Writes scalar composition field to diffusion.????????.csv.
- void `write_png` (`fp_t` **conc, int nx, int ny, int step)
Writes scalar composition field to diffusion.????????.png.

4.16.1 Detailed Description

Declaration of output function prototypes for diffusion benchmarks.

4.16.2 Function Documentation

4.16.2.1 `print_progress()`

```
void print_progress (
    const int step,
    const int steps )
```

Prints timestamps and a 20-point progress bar to stdout.

Call inside the timestepping loop, near the top, e.g.

```
for (int step=0; step<steps; step++) {
    print_progress(step, steps);
    take_a_step();
    elapsed += dt;
}
```

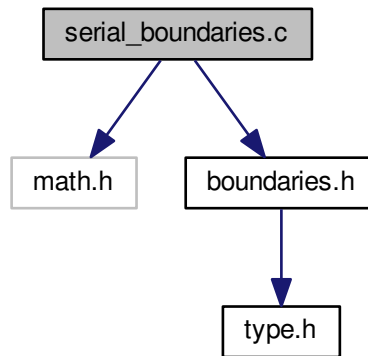
Definition at line 123 of file output.c.

4.17 serial_boundaries.c File Reference

Implementation of boundary condition functions without threading.

```
#include <math.h>
#include "boundaries.h"
```

Include dependency graph for serial_boundaries.c:



Functions

- void [set_boundaries](#) ([fp_t](#) bc[2][2])
Set values to be used along the simulation domain boundaries.
- void [apply_initial_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.17.1 Detailed Description

Implementation of boundary condition functions without threading.

4.17.2 Function Documentation

4.17.2.1 apply_initial_conditions()

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 37 of file serial_boundaries.c.

4.17.2.2 set_boundaries()

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

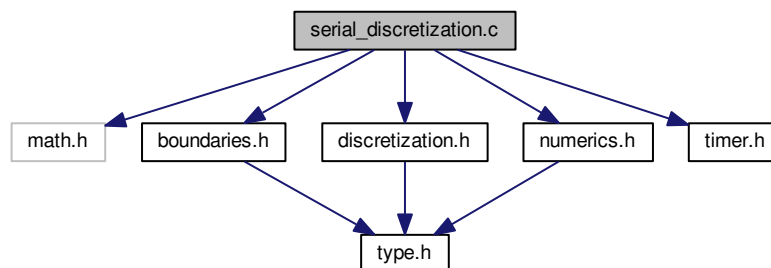
Definition at line 28 of file serial_boundaries.c.

4.18 serial_discretization.c File Reference

Implementation of boundary condition functions without threading.

```
#include <math.h>
#include "boundaries.h"
#include "discretization.h"
#include "numerics.h"
#include "timer.h"
```

Include dependency graph for serial_discretization.c:



Functions

- void `compute_convolution` (`fp_t` **conc_old, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void `solve_diffusion_equation` (`fp_t` **conc_old, `fp_t` **conc_new, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm, `fp_t` bc[2][2], `fp_t` D, `fp_t` dt, `fp_t` *elapsed, struct `Stopwatch` *sw)
Update the scalar composition field using old and Laplacian values.
- void `check_solution` (`fp_t` **conc_new, `fp_t` **conc_lap, int nx, int ny, `fp_t` dx, `fp_t` dy, int nm, `fp_t` elapsed, `fp_t` D, `fp_t` bc[2][2], `fp_t` *rss)
Compare numerical and analytical solutions of the diffusion equation.

4.18.1 Detailed Description

Implementation of boundary condition functions without threading.

4.18.2 Function Documentation

4.18.2.1 `check_solution()`

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites `conc_lap`, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 73 of file `serial_discretization.c`.

4.18.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions nx , ny , and nm are properly specified, the convolution will be correctly computed.

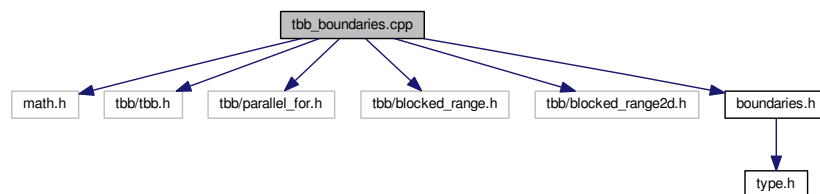
Definition at line 31 of file serial_discretization.c.

4.19 tbb_boundaries.cpp File Reference

Implementation of boundary condition functions with TBB threading.

```
#include <math.h>
#include <tbb/tbb.h>
#include <tbb/parallel_for.h>
#include <tbb/blocked_range.h>
#include <tbb/blocked_range2d.h>
#include "boundaries.h"
```

Include dependency graph for tbb_boundaries.cpp:



Functions

- void [set_boundaries](#) ([fp_t](#) bc[2][2])
Set values to be used along the simulation domain boundaries.
- void [apply_initial_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, int nx, int ny, int nm, [fp_t](#) bc[2][2])
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.19.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

4.19.2 Function Documentation

4.19.2.1 `apply_initial_conditions()`

```
void apply_initial_conditions (
    fp_t ** conc_old,
    int nx,
    int ny,
    int nm,
    fp_t bc[2][2] )
```

Initialize flat composition field with fixed boundary conditions.

The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce an inhomogeneous workload and highlight numerical errors at the boundaries.

Definition at line 41 of file `tbb_boundaries.cpp`.

4.19.2.2 `set_boundaries()`

```
void set_boundaries (
    fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

Indexing is row-major, i.e. $A[y][x]$, so $bc = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

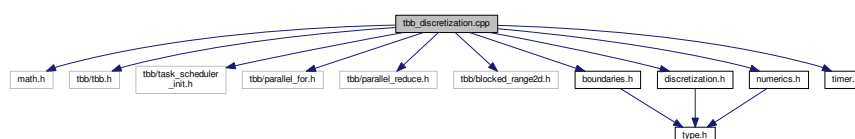
Definition at line 32 of file `tbb_boundaries.cpp`.

4.20 `tbb_discretization.cpp` File Reference

Implementation of boundary condition functions with TBB threading.

```
#include <math.h>
#include <tbb/tbb.h>
#include <tbb/task_scheduler_init.h>
#include <tbb/parallel_for.h>
#include <tbb/parallel_reduce.h>
#include <tbb/blocked_range2d.h>
#include "boundaries.h"
#include "discretization.h"
#include "numerics.h"
#include "timer.h"
```

Include dependency graph for `tbb_discretization.cpp`:



Functions

- void `compute_convolution` (`fp_t` **conc_old, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void `solve_diffusion_equation` (`fp_t` **conc_old, `fp_t` **conc_new, `fp_t` **conc_lap, `fp_t` **mask_lap, int nx, int ny, int nm, `fp_t` bc[2][2], `fp_t` D, `fp_t` dt, `fp_t` *elapsed, struct `Stopwatch` *sw)
Update the scalar composition field using old and Laplacian values.
- void `check_solution` (`fp_t` **conc_new, `fp_t` **conc_lap, int nx, int ny, `fp_t` dx, `fp_t` dy, int nm, `fp_t` elapsed, `fp_t` D, `fp_t` bc[2][2], `fp_t` *rss)
Compare numerical and analytical solutions of the diffusion equation.

4.20.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

4.20.2 Function Documentation

4.20.2.1 `check_solution()`

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    int nx,
    int ny,
    fp_t dx,
    fp_t dy,
    int nm,
    fp_t elapsed,
    fp_t D,
    fp_t bc[2][2],
    fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

Returns

Residual sum of squares (RSS), normalized to the domain size.

Overwrites `conc_lap`, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 132 of file `tbb_discretization.cpp`.

4.20.2.2 compute_convolution()

```
void compute_convolution (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** mask_lap,
    int nx,
    int ny,
    int nm )
```

Perform the convolution of the mask matrix with the composition matrix.

If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions *nx*, *ny*, and *nm* are properly specified, the convolution will be correctly computed.

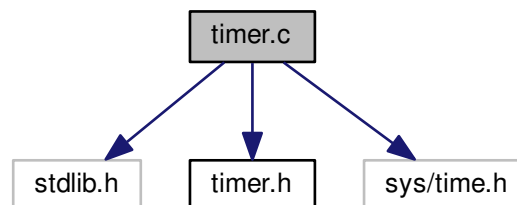
Definition at line 36 of file `tbb_discretization.cpp`.

4.21 timer.c File Reference

High-resolution cross-platform machine time reader.

```
#include <stdlib.h>
#include "timer.h"
#include <sys/time.h>
```

Include dependency graph for `timer.c`:



Functions

- void `StartTimer` ()
Set CPU frequency and begin timing.
- double `GetTimer` ()
Return elapsed time in seconds.

Variables

- struct timeval `timerStart`

4.21.1 Detailed Description

High-resolution cross-platform machine time reader.

Author

NVIDIA

4.21.2 Variable Documentation

4.21.2.1 timerStart

```
struct timeval timerStart
```

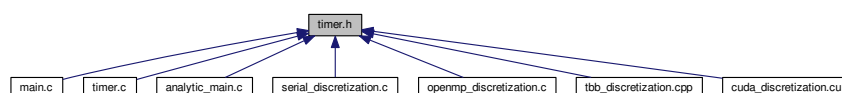
Platform-dependent data type of hardware time value

Definition at line 45 of file timer.c.

4.22 timer.h File Reference

Declaration of timer function prototypes for diffusion benchmarks.

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



Functions

- void [StartTimer](#) ()
Set CPU frequency and begin timing.
- double [GetTimer](#) ()
Return elapsed time in seconds.

4.22.1 Detailed Description

Declaration of timer function prototypes for diffusion benchmarks.

Index

- analytic_main.c, [4](#)
 - main, [5](#)
- analytical_value
 - numerics.c, [18](#)
 - numerics.h, [22](#)
- apply_initial_conditions
 - boundaries.h, [6](#)
 - cuda_boundaries.c, [7](#)
 - openacc_boundaries.c, [25](#)
 - openmp_boundaries.c, [28](#)
 - serial_boundaries.c, [34](#)
 - tbb_boundaries.cpp, [38](#)
- boundaries.h, [5](#)
 - apply_initial_conditions, [6](#)
 - set_boundaries, [6](#)
- check_solution
 - cuda_discretization.cu, [9](#)
 - discretization.h, [12](#)
 - openacc_discretization.c, [26](#)
 - openmp_discretization.c, [29](#)
 - serial_discretization.c, [36](#)
 - tbb_discretization.cpp, [39](#)
- compute_convolution
 - cuda_discretization.cu, [10](#)
 - discretization.h, [12](#)
 - openacc_discretization.c, [27](#)
 - openmp_discretization.c, [30](#)
 - serial_discretization.c, [36](#)
 - tbb_discretization.cpp, [39](#)
- conv
 - Stopwatch, [3](#)
- convolution_kernel
 - cuda_discretization.cu, [10](#)
- cuda_boundaries.c, [7](#)
 - apply_initial_conditions, [7](#)
 - set_boundaries, [8](#)
- cuda_discretization.cu, [8](#)
 - check_solution, [9](#)
 - compute_convolution, [10](#)
 - convolution_kernel, [10](#)
- discretization.h, [11](#)
 - check_solution, [12](#)
 - compute_convolution, [12](#)
- distance_point_to_segment
 - numerics.c, [18](#)
 - numerics.h, [22](#)
- file
 - Stopwatch, [3](#)
- five_point_Laplacian_stencil
 - numerics.c, [19](#)
 - numerics.h, [22](#)
- fp_t
 - type.h, [42](#)
- main
 - analytic_main.c, [5](#)
 - main.c, [14](#)
- main.c, [13](#)
 - main, [14](#)
- make_arrays
 - mesh.c, [15](#)
 - mesh.h, [17](#)
- mesh.c, [14](#)
 - make_arrays, [15](#)
 - swap_pointers, [15](#)
- mesh.h, [16](#)
 - make_arrays, [17](#)
 - swap_pointers, [17](#)
- nine_point_Laplacian_stencil
 - numerics.c, [19](#)
 - numerics.h, [23](#)
- numerics.c, [17](#)
 - analytical_value, [18](#)
 - distance_point_to_segment, [18](#)
 - five_point_Laplacian_stencil, [19](#)
 - nine_point_Laplacian_stencil, [19](#)
 - set_mask, [19](#)
 - slow_nine_point_Laplacian_stencil, [20](#)
- numerics.h, [20](#)
 - analytical_value, [22](#)
 - distance_point_to_segment, [22](#)
 - five_point_Laplacian_stencil, [22](#)
 - nine_point_Laplacian_stencil, [23](#)
 - set_mask, [23](#)
 - slow_nine_point_Laplacian_stencil, [23](#)
- openacc_boundaries.c, [24](#)
 - apply_initial_conditions, [25](#)
 - set_boundaries, [25](#)
- openacc_discretization.c, [25](#)
 - check_solution, [26](#)
 - compute_convolution, [27](#)
- openmp_boundaries.c, [27](#)
 - apply_initial_conditions, [28](#)
 - set_boundaries, [28](#)
- openmp_discretization.c, [29](#)
 - check_solution, [29](#)
 - compute_convolution, [30](#)
- output.c, [31](#)
 - print_progress, [31](#)
- output.h, [32](#)
 - print_progress, [33](#)
- print_progress
 - output.c, [31](#)
 - output.h, [33](#)
- serial_boundaries.c, [34](#)

