phasefield-accelerator-benchmarks

pre-alpha

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

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

3 Class Documentation

3.1 ResidualSumOfSquares2D Class Reference

Comparison algorithm for execution on the block of threads.

3.1.1 Detailed Description

Comparison algorithm for execution on the block of threads.

Definition at line 99 of file tbb_discretization.cpp.

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

- tbb_discretization.cpp

3.2 Stopwatch Struct Reference

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

3.2.1 Detailed Description

Container for timing data

Definition at line 39 of file type.h.

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

- type.h

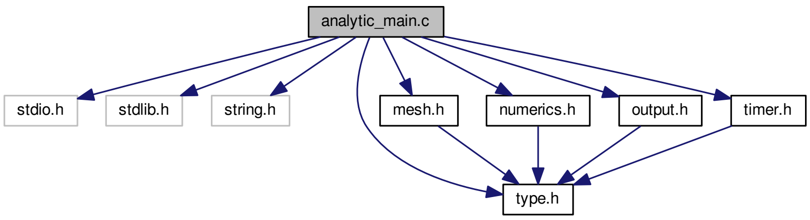
4 File Documentation

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[ ])

    Run simulation using input parameters specified on the command line.

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[ ] )
```

Run simulation using input parameters specified on the command line.

Program will write a series of PNG image files to visualize the scalar composition field, plus `delta.png` showing the difference between the analytical result and the image stored in ../common-diffusion/diffusion.10000.png.

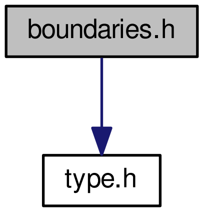
Definition at line 72 of file analytic_main.c.

4.2 boundaries.h File Reference

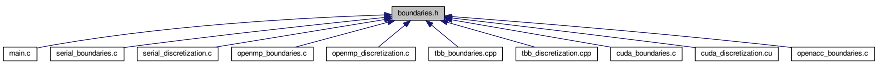
Declaration of boundary condition function prototypes.

`#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,
            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 51 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

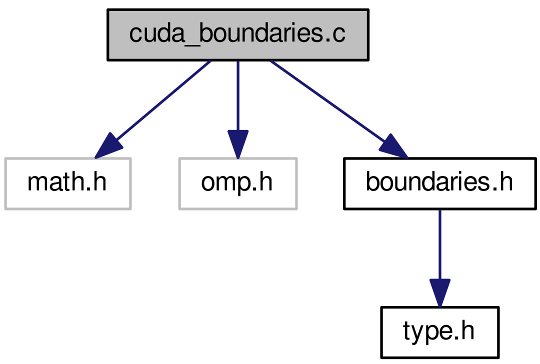
Definition at line 33 of file serial_boundaries.c.

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,
            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 52 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

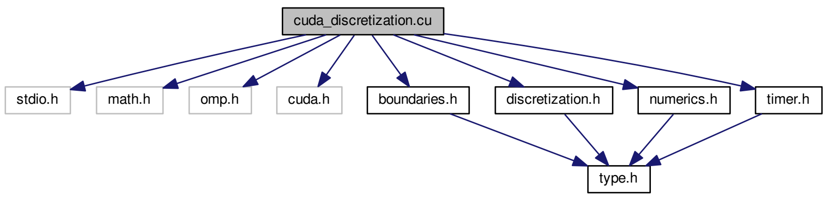
Definition at line 34 of file cuda_boundaries.c.

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.
- 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, 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]

  Allocate constant memory on the GPU for the convolution mask.

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

Definition at line 223 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.

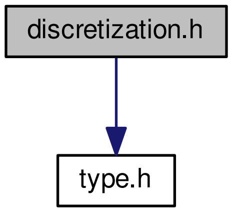
Definition at line 135 of file cuda_discretization.cu.

4.5 discretization.h File Reference

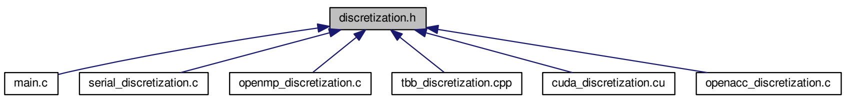
Declaration of discretized mathematical function prototypes.

```
#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, 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,
            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.

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

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

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

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

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

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

Definition at line 89 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.

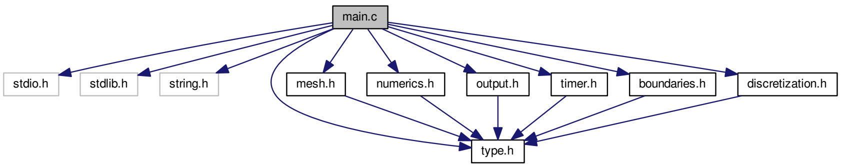
Definition at line 40 of file serial_discretization.c.

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

- 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 input parameters from file specified on the command line.

- 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).

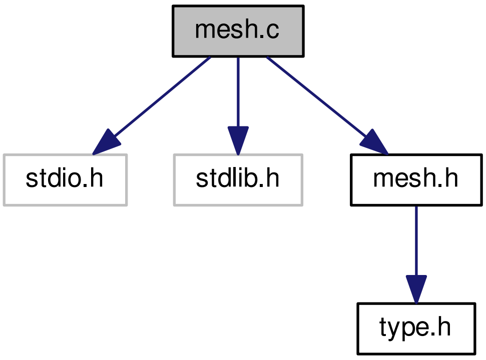
Definition at line 53 of file main.c.

4.7 mesh.c File Reference

Implemenatation 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

Implemenatation 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 37 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.

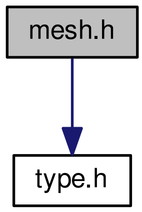
Rather than copy data from conc_old into conc_new, an expensive operation, simply trade the top-most pointers. New becomes old with no data lost and in almost no time.

Definition at line 91 of file mesh.c.

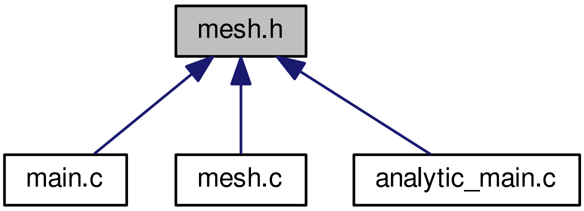
4.8 mesh.h File Reference

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 37 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.

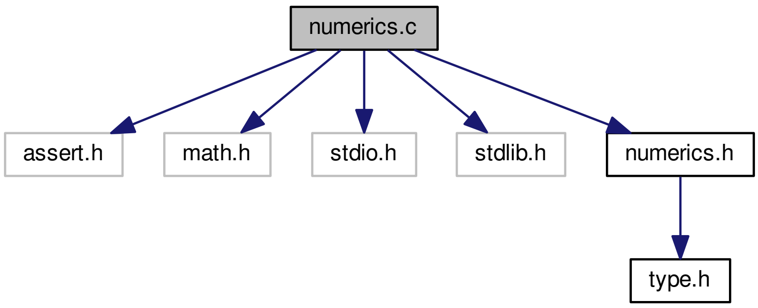
Definition at line 91 of file mesh.c.

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

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

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

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

Definition at line 179 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 156 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 71 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 87 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.* five-point Laplacian is 53, nine-point is 93.

To add your own mask (stencil), define its prototype in `numerics.h`, implement it in `numerics.c`, add a case to this function with your chosen numerical encoding, then specify that code in `params.txt`. Note that, for a Laplacian stencil, the sum of the coefficients must equal zero.

If your stencil is larger than 5×5, you must increase the values defined by `MAX_MASK_W` and `MAX_MASK_H` in `numerics.h`.

Definition at line 46 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 $\mathcal{O}(\Delta x^4)$

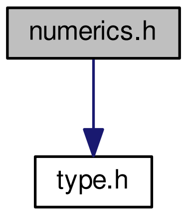
Provided for testing and demonstration of scalability, only: as the name indicates, this 9-point stencil is computationally more expensive than the 3×3 version. If your code requires $\mathcal{O}(\Delta x^4)$ accuracy, please use nine_point_Laplacian_stencil().

Definition at line 114 of file numerics.c.

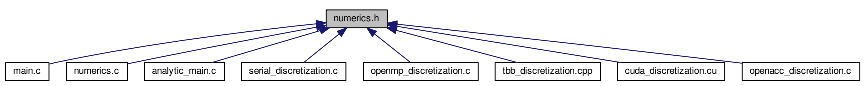
4.10 numerics.h File Reference

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

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

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

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

Definition at line 179 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 156 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 71 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 87 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.* five-point Laplacian is 53, nine-point is 93.

To add your own mask (stencil), define its prototype in `numerics.h`, implement it in `numerics.c`, add a case to this function with your chosen numerical encoding, then specify that code in `params.txt`. Note that, for a Laplacian stencil, the sum of the coefficients must equal zero.

If your stencil is larger than 5×5, you must increase the values defined by `MAX_MASK_W` and `MAX_MASK_H` in `numerics.h`.

Definition at line 46 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 114 of file numerics.c.

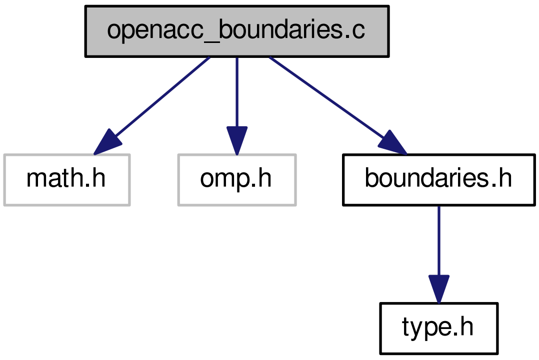
4.11 openacc_boundaries.c File Reference

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,
             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 52 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

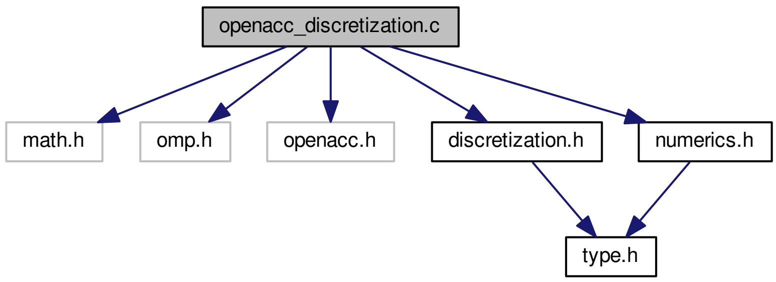
Definition at line 34 of file openacc_boundaries.c.

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

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

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

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

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

Definition at line 115 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.

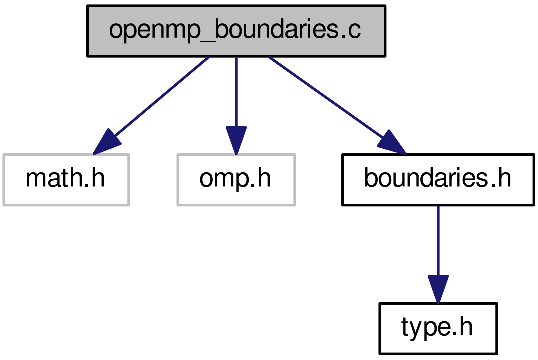
Definition at line 40 of file openacc_discretization.c.

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,
            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 53 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

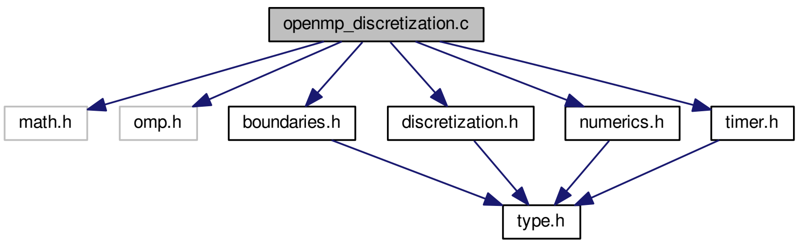
Definition at line 35 of file openmp_boundaries.c.

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

Definition at line 97 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.

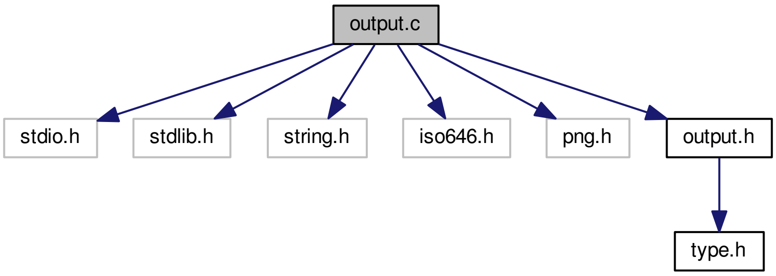
Definition at line 41 of file openmp_discretization.c.

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 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;
}
```

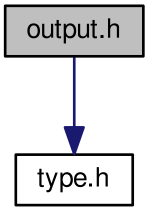
Definition at line 45 of file output.c.

4.16 output.h File Reference

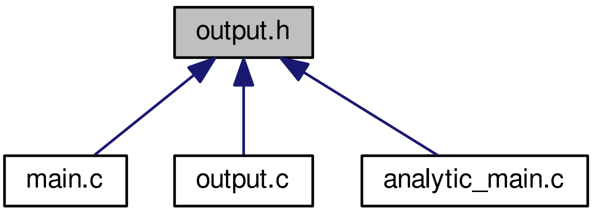
Declaration of output function prototypes for diffusion benchmarks.

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

Include dependency graph for output.h:



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



Functions

- void 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;
}
```

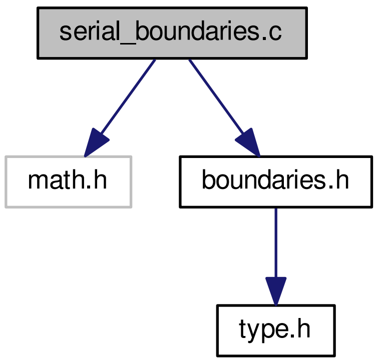
Definition at line 45 of file output.c.

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,
            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 51 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

Definition at line 33 of file serial_boundaries.c.

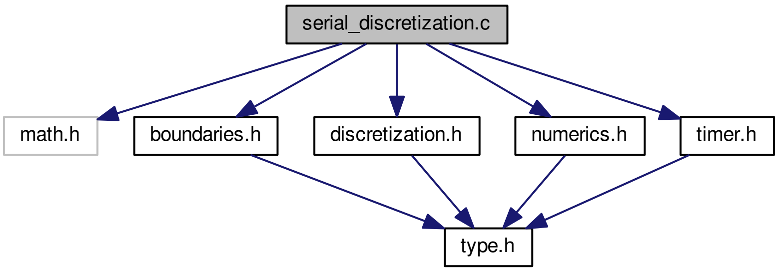
4.18 serial_discretization.c File Reference

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, 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,
            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.

Definition at line 89 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.

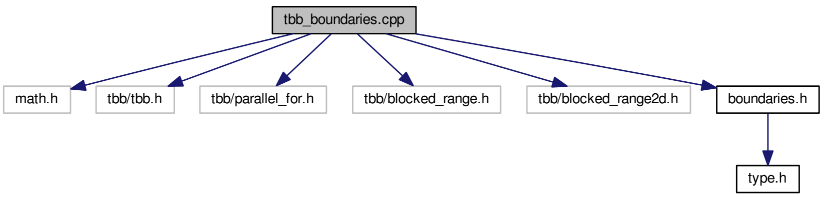
Definition at line 40 of file serial_discretization.c.

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,
            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 55 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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

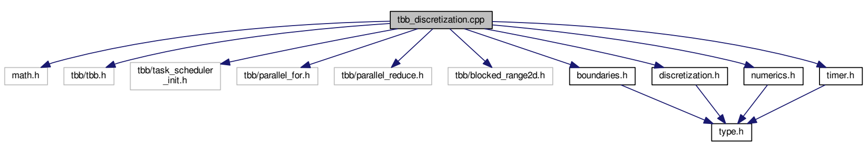
Definition at line 37 of file tbb_boundaries.cpp.

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:



Classes

- class ResidualSumOfSquares2D

  Comparison algorithm for execution on the block of threads.

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 **B, 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, 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,
            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 the residual sum of squares (RSS), normalized to the domain size.

Definition at line 181 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.

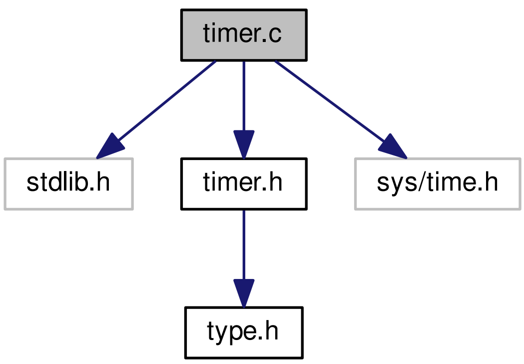
Definition at line 45 of file tbb_discretization.cpp.

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

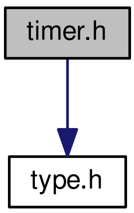
Definition at line 45 of file timer.c.

4.22 timer.h File Reference

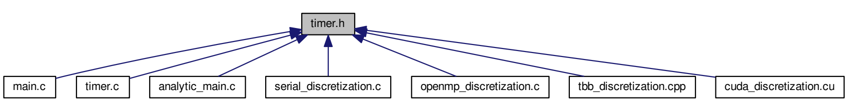
Declaration of timer function prototypes for diffusion benchmarks.

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

Include dependency graph for timer.h:



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.

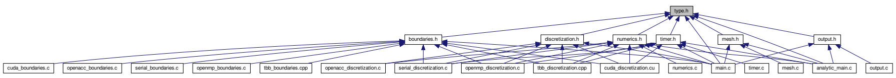
4.22.1 Detailed Description

Declaration of timer function prototypes for diffusion benchmarks.

4.23 type.h File Reference

Definition of scalar data type and Doxygen diffusion group.

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



Classes

- struct Stopwatch

Typedefs

- typedef double fp_t

4.23.1 Detailed Description

Definition of scalar data type and Doxygen diffusion group.

4.23.2 Typedef Documentation

4.23.2.1 fp_t

```
typedef double fp_t
```

Specify the basic data type to achieve the desired accuracy in floating-point arithmetic: float for single-precision, double for double-precision. This choice propagates throughout the code, and may significantly affect runtime on GPU hardware.

Definition at line 34 of file type.h.

Index