phasefield-accelerator-benchmarks

pre-alpha

Generated by Doxygen 1.8.13

Contents

1 Class Index 2

1.1 Class List 2

2 File Index 2

2.1 File List 2

3 Class Documentation 3

3.1 ResidualSumOfSquares2D Class Reference 3

3.1.1 Detailed Description 4

3.2 Stopwatch Struct Reference 4

3.2.1 Detailed Description 4

4 File Documentation 4

4.1 boundaries.c File Reference 4

4.1.1 Detailed Description 5

4.1.2 Function Documentation 5

4.2 boundaries.c File Reference 6

4.2.1 Detailed Description 6

4.2.2 Function Documentation 6

4.3 boundaries.c File Reference 7

4.3.1 Detailed Description 7

4.3.2 Function Documentation 7

4.4 boundaries.c File Reference 8

4.4.1 Detailed Description 9

4.4.2 Function Documentation 9

4.5 boundaries.cpp File Reference 10

4.5.1 Detailed Description 10

4.5.2 Function Documentation 10

4.6 boundaries.h File Reference 11

4.6.1 Detailed Description 12

4.6.2 Function Documentation 12
4.7 boundaries.h File Reference 13
4.7.1 Detailed Description 13
4.7.2 Function Documentation 13
4.8 discretization.c File Reference 14
4.8.1 Detailed Description 15
4.9 discretization.c File Reference 15
4.9.1 Detailed Description 15
4.9.2 Function Documentation 16
4.10 discretization.c File Reference 16
4.10.1 Detailed Description 17
4.10.2 Function Documentation 17
4.11 discretization.c File Reference 18
4.11.1 Detailed Description 19
4.11.2 Function Documentation 19
4.12 discretization.cl File Reference 20
4.12.1 Detailed Description 20
4.12.2 Function Documentation 21
4.13 discretization.cpp File Reference 22
4.13.1 Detailed Description 22
4.13.2 Function Documentation 22
4.14 discretization.cu File Reference 23
4.14.1 Detailed Description 24
4.14.2 Function Documentation 25
4.15 discretization.h File Reference 26
4.15.1 Detailed Description 26
4.15.2 Function Documentation 27
4.16 discretization.h File Reference 28
4.16.1 Detailed Description 28
4.17 discretization.h File Reference 29

4.17.1 Detailed Description 29
4.17.2 Function Documentation 29
4.18 main.c File Reference 31
4.18.1 Detailed Description 31
4.18.2 Function Documentation 31
4.19 main.c File Reference 32
4.19.1 Detailed Description 32
4.19.2 Function Documentation 32
4.20 mesh.c File Reference 33
4.20.1 Detailed Description 33
4.20.2 Function Documentation 33
4.21 mesh.h File Reference 34
4.21.1 Detailed Description 35
4.21.2 Function Documentation 35
4.22 numerics.c File Reference 36
4.22.1 Detailed Description 37
4.22.2 Function Documentation 37
4.23 numerics.h File Reference 39
4.23.1 Detailed Description 40
4.23.2 Function Documentation 40
4.24 output.c File Reference 42
4.24.1 Detailed Description 43
4.24.2 Function Documentation 43
4.25 output.h File Reference 44
4.25.1 Detailed Description 44
4.25.2 Function Documentation 45
4.26 timer.c File Reference 45
4.26.1 Detailed Description 46
4.26.2 Variable Documentation 46
4.27 timer.h File Reference 46
4.27.1 Detailed Description 47
4.28 type.h File Reference 47
4.28.1 Detailed Description 47
4.28.2 Typedef Documentation 47

Index 49

1 Class Index

1.1 Class List

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

ResidualSumOfSquares2D
Comparison algorithm for execution on the block of threads 3

Stopwatch 4

2 File Index

2.1 File List

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

cpu-openmp-diffusion/boundaries.c
Implementation of boundary condition functions with OpenMP threading 4

cpu-serial-diffusion/boundaries.c
Implementation of boundary condition functions without threading 6

gpu-cuda-diffusion/boundaries.c
Implementation of boundary condition functions with OpenMP threading 7

gpu-openacc-diffusion/boundaries.c
Implementation of boundary condition functions with OpenMP threading 8

gpu-opencl-diffusion/boundaries.c ??

boundaries.cpp
Implementation of boundary condition functions with TBB threading 10

common-diffusion/boundaries.h
Declaration of boundary condition function prototypes 11

gpu-opencl-diffusion/boundaries.h
Declaration of boundary condition function prototypes for OpenCL benchmarks 13

cpu-analytic-diffusion/discretization.c
Implementation of analytical solution functions 14

cpu-openmp-diffusion/discretization.c
Implementation of boundary condition functions with OpenMP threading 15

cpu-serial-diffusion/discretization.c
Implementation of boundary condition functions without threading 16

gpu-openacc-diffusion/discretization.c
Implementation of boundary condition functions with OpenACC threading 18

discretization.cl
Implementation of boundary condition functions with OpenCL acceleration 20

discretization.cpp
Implementation of boundary condition functions with TBB threading 22

discretization.cu
Implementation of boundary condition functions with CUDA acceleration 23

common-diffusion/discretization.h
Declaration of discretized mathematical function prototypes 26

cpu-analytic-diffusion/discretization.h
Declaration of analytical solution prototypes 28

gpu-opencl-diffusion/discretization.h
Declaration of discretized mathematical function prototypes for OpenCL benchmarks 29

common-diffusion/main.c
Implementation of semi-infinite diffusion equation 31

cpu-analytic-diffusion/main.c
Analytical solution to semi-infinite diffusion equation 32

mesh.c
Implemenatation of mesh handling functions for diffusion benchmarks 33

mesh.h
Declaration of mesh function prototypes for diffusion benchmarks 34

numerics.c
Implementation of boundary condition functions with OpenMP threading 36

numerics.h
Declaration of Laplacian operator and analytical solution functions 39

output.c
Implementation of file output functions for diffusion benchmarks 42

output.h
Declaration of output function prototypes for diffusion benchmarks 44

timer.c
High-resolution cross-platform machine time reader 45

timer.h
Declaration of timer function prototypes for diffusion benchmarks 46

type.h
Definition of scalar data type and Doxygen diffusion group 47

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 discretization.cpp.

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

- 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

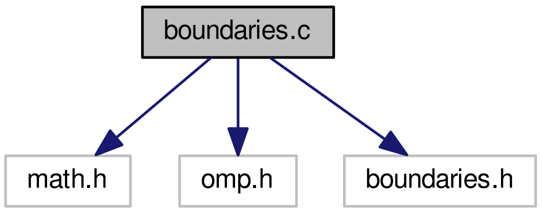
4 File Documentation

4.1 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 cpu-openmp-diffusion/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.

4.1.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.1.2 Function Documentation

4.1.2.1 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 cpu-openmp-diffusion/boundaries.c.

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

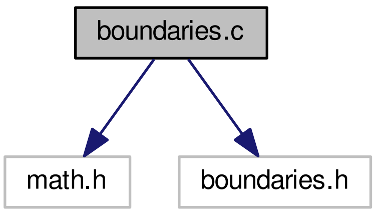
Definition at line 35 of file cpu-openmp-diffusion/boundaries.c.

4.2 boundaries.c File Reference

Implementation of boundary condition functions without threading.

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

Include dependency graph for cpu-serial-diffusion/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.

4.2.1 Detailed Description

Implementation of boundary condition functions without threading.

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 cpu-serial-diffusion/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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

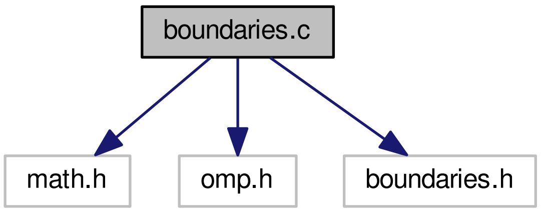
Definition at line 33 of file cpu-serial-diffusion/boundaries.c.

4.3 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 gpu-cuda-diffusion/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 gpu-cuda-diffusion/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 $\mathrm{bc} = [[y_{lo}, y_{hi}], [x_{lo}, x_{hi}]]$.

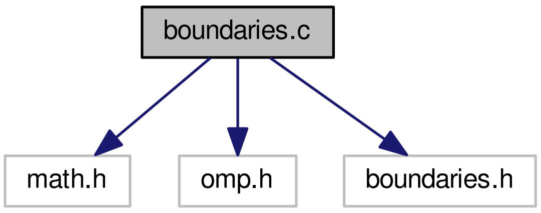
Definition at line 34 of file gpu-cuda-diffusion/boundaries.c.

4.4 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 gpu-openacc-diffusion/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.

4.4.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.4.2 Function Documentation

4.4.2.1 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 gpu-openacc-diffusion/boundaries.c.

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

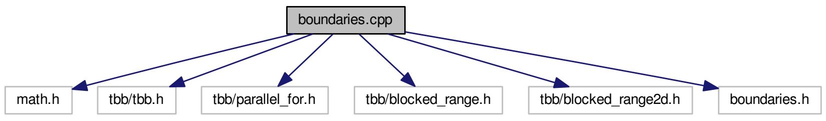
Definition at line 34 of file gpu-openacc-diffusion/boundaries.c.

4.5 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 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.

4.5.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

4.5.2 Function Documentation

4.5.2.1 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 boundaries.cpp.

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

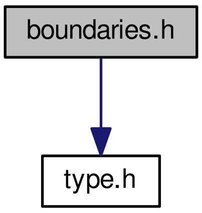
Definition at line 37 of file boundaries.cpp.

4.6 boundaries.h File Reference

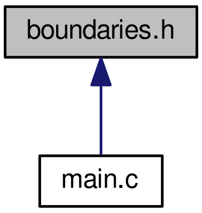
Declaration of boundary condition function prototypes.

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

Include dependency graph for common-diffusion/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.

4.6.1 Detailed Description

Declaration of boundary condition function prototypes.

4.6.2 Function Documentation

4.6.2.1 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 cpu-openmp-diffusion/boundaries.c.

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

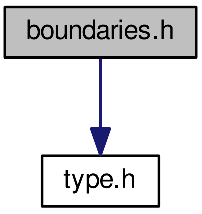
Definition at line 35 of file cpu-openmp-diffusion/boundaries.c.

4.7 boundaries.h File Reference

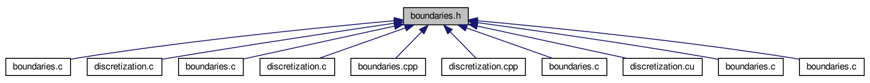
Declaration of boundary condition function prototypes for OpenCL benchmarks.

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

Include dependency graph for gpu-opencl-diffusion/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.

4.7.1 Detailed Description

Declaration of boundary condition function prototypes for OpenCL benchmarks.

4.7.2 Function Documentation

4.7.2.1 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 cpu-openmp-diffusion/boundaries.c.

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

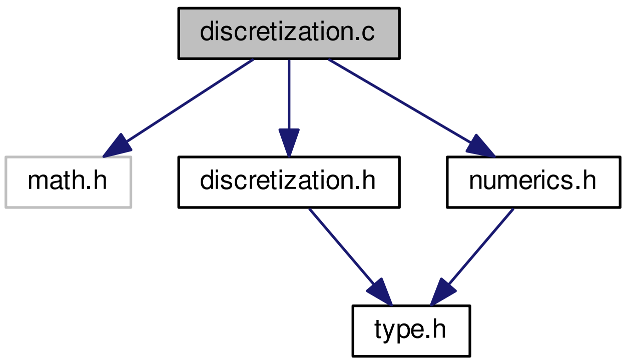
Definition at line 35 of file cpu-openmp-diffusion/boundaries.c.

4.8 discretization.c File Reference

Implementation of analytical solution functions.

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

Include dependency graph for cpu-analytic-diffusion/discretization.c:



Functions

- void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t D, fp_t dt, fp_t elapsed)

  Update the scalar composition field using analytical solution.

4.8.1 Detailed Description

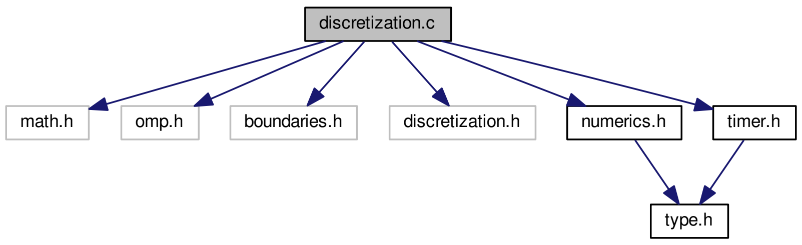
Implementation of analytical solution functions.

4.9 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 cpu-openmp-diffusion/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.

4.9.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.9.2 Function Documentation

4.9.2.1 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 cpu-openmp-diffusion/discretization.c.

4.9.2.2 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.

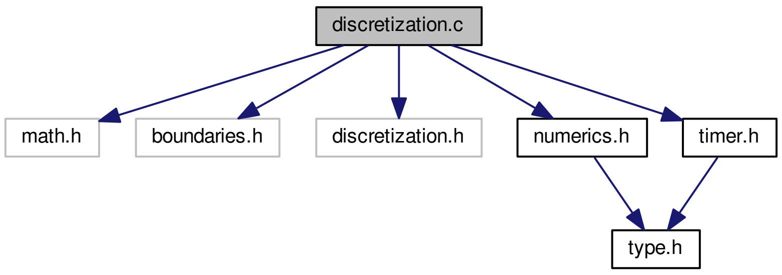
Definition at line 41 of file cpu-openmp-diffusion/discretization.c.

4.10 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 cpu-serial-diffusion/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.

4.10.1 Detailed Description

Implementation of boundary condition functions without threading.

4.10.2 Function Documentation

4.10.2.1 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 cpu-serial-diffusion/discretization.c.

4.10.2.2 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.

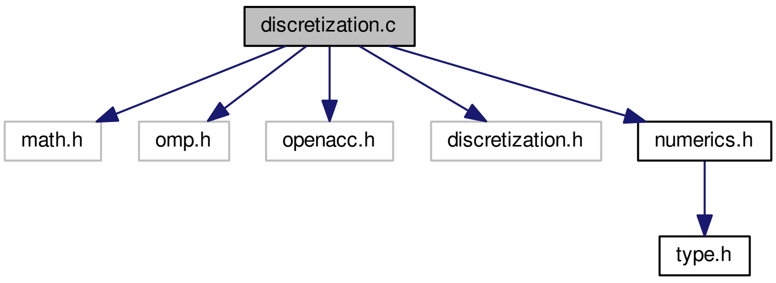
Definition at line 40 of file cpu-serial-diffusion/discretization.c.

4.11 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 gpu-openacc-diffusion/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.

4.11.1 Detailed Description

Implementation of boundary condition functions with OpenACC threading.

4.11.2 Function Documentation

4.11.2.1 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 gpu-openacc-diffusion/discretization.c.

4.11.2.2 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.

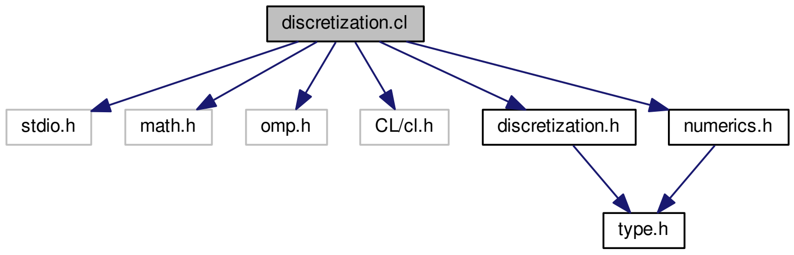
Definition at line 40 of file gpu-openacc-diffusion/discretization.c.

4.12 discretization.cl File Reference

Implementation of boundary condition functions with OpenCL acceleration.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <CL/cl.h>
#include "discretization.h"
#include "numerics.h"
```

Include dependency graph for discretization.cl:



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.
- #define MAX_MASK_W 3

  Maximum height and width of the mask array, for GPU memory allocation.

Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void compute_convolution (fp_t ∗∗conc_old, fp_t ∗∗conc_lap, fp_t ∗∗mask_lap, int nx, int ny, int nm, int bs)

  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, int nx, int ny, int nm, int bs, fp_t D, fp_t dt, fp_t ∗elapsed)

  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 OpenCL acceleration.

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.

Definition at line 246 of file discretization.cl.

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,
            int bs )
```

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.

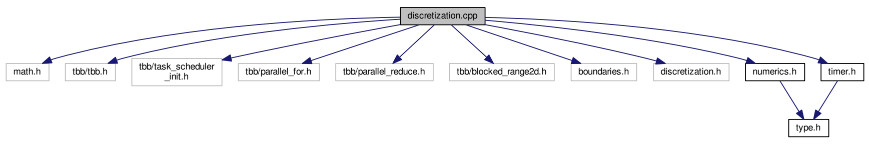
Definition at line 143 of file discretization.cl.

4.13 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 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.

4.13.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

4.13.2 Function Documentation

4.13.2.1 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 discretization.cpp.

4.13.2.2 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 discretization.cpp.

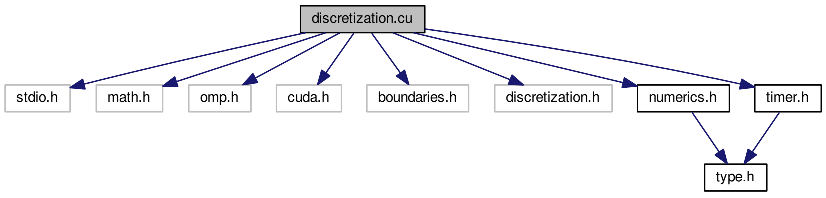
4.14 discretization.cu File Reference

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 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.

4.14.1 Detailed Description

Implementation of boundary condition functions with CUDA acceleration.

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 223 of file discretization.cu.

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.

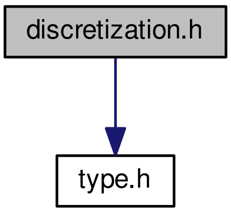
Definition at line 135 of file discretization.cu.

4.15 discretization.h File Reference

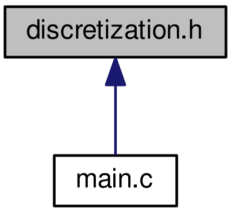
Declaration of discretized mathematical function prototypes.

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

Include dependency graph for common-diffusion/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.

4.15.1 Detailed Description

Declaration of discretized mathematical function prototypes.

4.15.2 Function Documentation

4.15.2.1 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.

Returns

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 97 of file cpu-openmp-diffusion/discretization.c.

4.15.2.2 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.

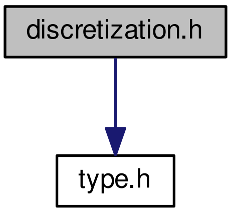
If the convolution mask is the Laplacian stencil, the convolution evaluates the discrete Laplacian of the composition field. Other masks are possible, for example the Sobel filters for edge detection. This function is general purpose: as long as the dimensions `nx`, `ny`, and `nm` are properly specified,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

Definition at line 41 of file cpu-openmp-diffusion/discretization.c.

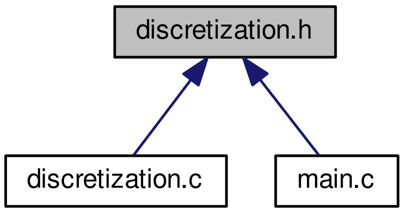
4.16 discretization.h File Reference

Declaration of analytical solution prototypes.

```
#include "type.h"
```
Include dependency graph for cpu-analytic-diffusion/discretization.h:



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



Functions

- void solve_diffusion_equation (fp_t ∗∗conc_old, fp_t ∗∗conc_new, fp_t ∗∗conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t D, fp_t dt, fp_t elapsed)

  Update the scalar composition field using analytical solution.

4.16.1 Detailed Description

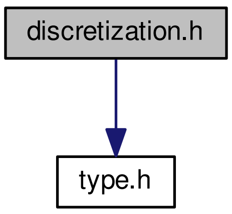
Declaration of analytical solution prototypes.

4.17 discretization.h File Reference

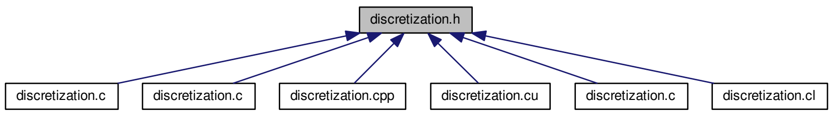
Declaration of discretized mathematical function prototypes for OpenCL benchmarks.

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

Include dependency graph for gpu-opencl-diffusion/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, int bs)

    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, int nx, int ny, int nm, int bs, fp_t D, fp_t dt, fp_t ∗elapsed)

    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.

4.17.1 Detailed Description

Declaration of discretized mathematical function prototypes for OpenCL benchmarks.

4.17.2 Function Documentation

4.17.2.1 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.

Returns

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 97 of file cpu-openmp-diffusion/discretization.c.

4.17.2.2 compute_convolution()

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

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.

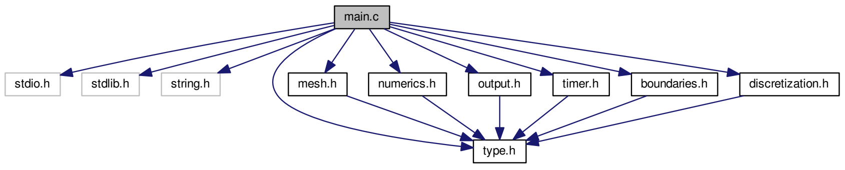
Definition at line 143 of file discretization.cl.

4.18 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 common-diffusion/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.

4.18.1 Detailed Description

Implementation of semi-infinite diffusion equation.

4.18.2 Function Documentation

4.18.2.1 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).

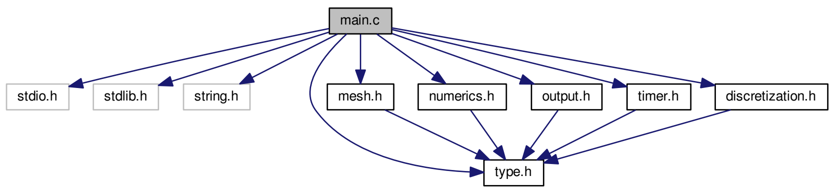
Definition at line 53 of file common-diffusion/main.c.

4.19 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 "discretization.h"
```

Include dependency graph for cpu-analytic-diffusion/main.c:



Functions

- int main (int argc, char ∗argv[ ])

  Run simulation using input parameters specified on the command line.

4.19.1 Detailed Description

Analytical solution to semi-infinite diffusion equation.

4.19.2 Function Documentation

4.19.2.1 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.

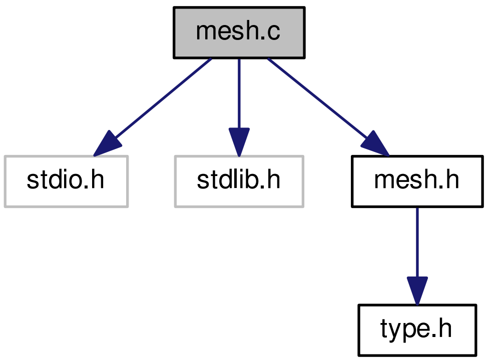
Definition at line 44 of file cpu-analytic-diffusion/main.c.

4.20 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.

4.20.1 Detailed Description

Implemenatation of mesh handling functions for diffusion benchmarks.

4.20.2 Function Documentation

4.20.2.1 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.

4.20.2.2 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.

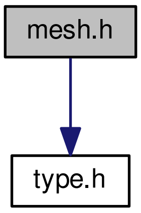
Definition at line 91 of file mesh.c.

4.21 mesh.h File Reference

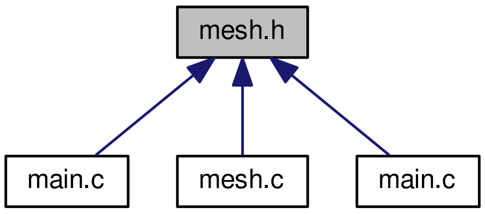
Declaration of mesh function prototypes for diffusion benchmarks.

```
#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.

4.21.1 Detailed Description

Declaration of mesh function prototypes for diffusion benchmarks.

4.21.2 Function Documentation

4.21.2.1 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.

4.21.2.2 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.

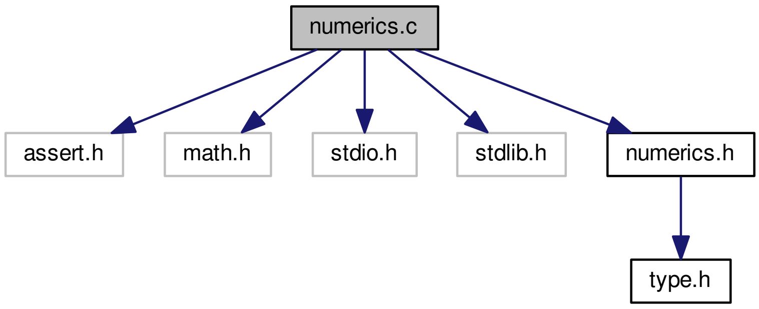
Definition at line 91 of file mesh.c.

4.22 numerics.c File Reference

Implementation of boundary condition functions with OpenMP threading.

```
#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.

4.22.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.22.2 Function Documentation

4.22.2.1 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.

4.22.2.2 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.

4.22.2.3 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.

4.22.2.4 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.

4.22.2.5 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.

4.22.2.6 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().

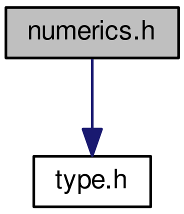
Definition at line 114 of file numerics.c.

4.23 numerics.h File Reference

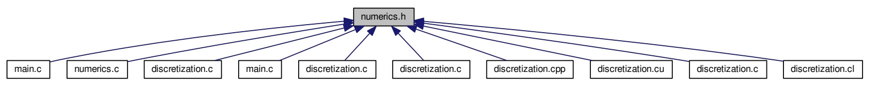
Declaration of Laplacian operator and analytical solution functions.

```
#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.

4.23.1 Detailed Description

Declaration of Laplacian operator and analytical solution functions.

4.23.2 Function Documentation

4.23.2.1 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.

4.23.2.2 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.

4.23.2.3 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.

4.23.2.4 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.

4.23.2.5 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.

4.23.2.6 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().

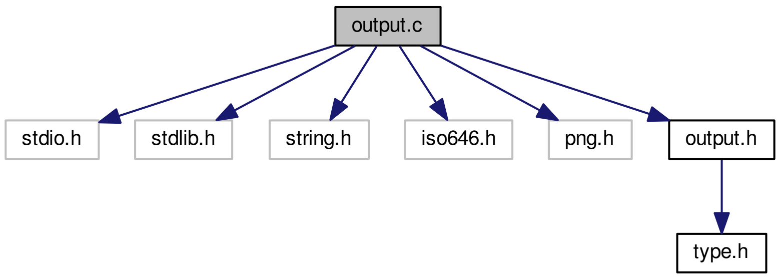
Definition at line 114 of file numerics.c.

4.24 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.

4.24.1 Detailed Description

Implementation of file output functions for diffusion benchmarks.

4.24.2 Function Documentation

4.24.2.1 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;
}
```

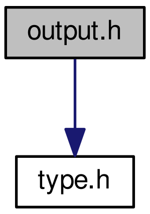
Definition at line 45 of file output.c.

4.25 output.h File Reference

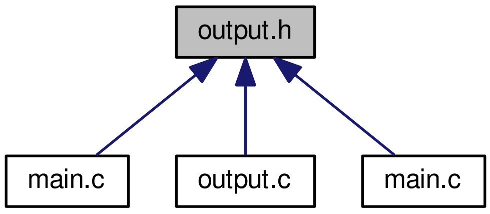
Declaration of output function prototypes for diffusion benchmarks.

```
#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.

4.25.1 Detailed Description

Declaration of output function prototypes for diffusion benchmarks.

4.25.2 Function Documentation

4.25.2.1 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;
}
```

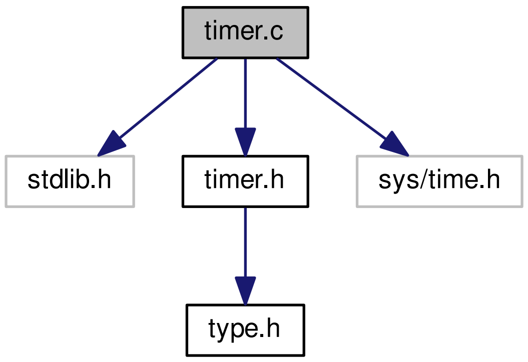
Definition at line 45 of file output.c.

4.26 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:


$$\begin{tikzcd}
 & \boxed{\text{timer.c}} \arrow[dl] \arrow[d] \arrow[dr] & \\
\boxed{\text{stdlib.h}} & \boxed{\text{timer.h}} \arrow[d] & \boxed{\text{sys/time.h}} \\
 & \boxed{\text{type.h}} &
\end{tikzcd}$$

Functions

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

Variables

- struct timeval timerStart

4.26.1 Detailed Description

High-resolution cross-platform machine time reader.

Author

NVIDIA

4.26.2 Variable Documentation

4.26.2.1 timerStart

```
struct timeval timerStart
```

Platform-dependent data type of hardware time value

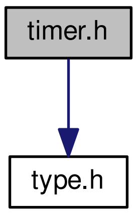
Definition at line 45 of file timer.c.

4.27 timer.h File Reference

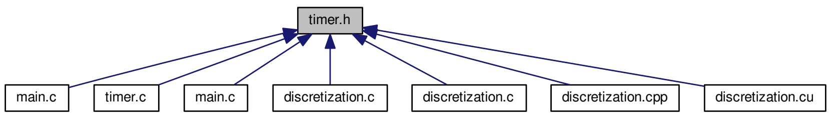
Declaration of timer function prototypes for diffusion benchmarks.

```
#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.

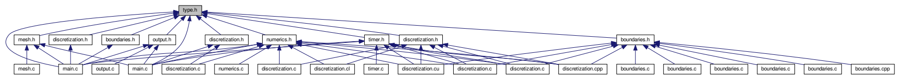
4.27.1 Detailed Description

Declaration of timer function prototypes for diffusion benchmarks.

4.28 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

4.28.1 Detailed Description

Definition of scalar data type and Doxygen diffusion group.

4.28.2 Typedef Documentation

4.28.2.1 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

analytical_value
- numerics.c, 37
- numerics.h, 40

apply_initial_conditions
- boundaries.cpp, 10
- common-diffusion/boundaries.h, 12
- cpu-openmp-diffusion/boundaries.c, 5
- cpu-serial-diffusion/boundaries.c, 6
- gpu-cuda-diffusion/boundaries.c, 7
- gpu-openacc-diffusion/boundaries.c, 9
- gpu-opencl-diffusion/boundaries.h, 13

boundaries.c, 4, 6–8
boundaries.cpp, 10
- apply_initial_conditions, 10
- set_boundaries, 10

boundaries.h, 11, 13

check_solution
- common-diffusion/discretization.h, 27
- cpu-openmp-diffusion/discretization.c, 16
- cpu-serial-diffusion/discretization.c, 17
- discretization.cl, 21
- discretization.cpp, 22
- discretization.cu, 25
- gpu-openacc-diffusion/discretization.c, 19
- gpu-opencl-diffusion/discretization.h, 29

common-diffusion/boundaries.h
- apply_initial_conditions, 12
- set_boundaries, 12

common-diffusion/discretization.h
- check_solution, 27
- compute_convolution, 27

common-diffusion/main.c
- main, 31

compute_convolution
- common-diffusion/discretization.h, 27
- cpu-openmp-diffusion/discretization.c, 16
- cpu-serial-diffusion/discretization.c, 17
- discretization.cl, 21
- discretization.cpp, 23
- discretization.cu, 25
- gpu-openacc-diffusion/discretization.c, 19
- gpu-opencl-diffusion/discretization.h, 30

cpu-analytic-diffusion/main.c
- main, 32

cpu-openmp-diffusion/boundaries.c
- apply_initial_conditions, 5
- set_boundaries, 5

cpu-openmp-diffusion/discretization.c
- check_solution, 16
- compute_convolution, 16

cpu-serial-diffusion/boundaries.c
- apply_initial_conditions, 6
- set_boundaries, 6

cpu-serial-diffusion/discretization.c
- check_solution, 17
- compute_convolution, 17

discretization.c, 14–16, 18
discretization.cl, 20
- check_solution, 21
- compute_convolution, 21

discretization.cpp, 22
- check_solution, 22
- compute_convolution, 23

discretization.cu, 23
- check_solution, 25
- compute_convolution, 25

discretization.h, 26, 28, 29
distance_point_to_segment
- numerics.c, 37
- numerics.h, 40

five_point_Laplacian_stencil
- numerics.c, 37
- numerics.h, 41

fp_t
- type.h, 47

gpu-cuda-diffusion/boundaries.c
- apply_initial_conditions, 7
- set_boundaries, 8

gpu-openacc-diffusion/boundaries.c
- apply_initial_conditions, 9
- set_boundaries, 9

gpu-openacc-diffusion/discretization.c
- check_solution, 19
- compute_convolution, 19

gpu-opencl-diffusion/boundaries.h
- apply_initial_conditions, 13
- set_boundaries, 14

gpu-opencl-diffusion/discretization.h
- check_solution, 29
- compute_convolution, 30

main
- common-diffusion/main.c, 31
- cpu-analytic-diffusion/main.c, 32

main.c, 31, 32
make_arrays
- mesh.c, 33
- mesh.h, 35

mesh.c, 33
- make_arrays, 33
- swap_pointers, 34

mesh.h, 34
- make_arrays, 35
- swap_pointers, 35

nine_point_Laplacian_stencil
- numerics.c, 38

numerics.h, 41
numerics.c, 36
analytical_value, 37
distance_point_to_segment, 37
five_point_Laplacian_stencil, 37
nine_point_Laplacian_stencil, 38
set_mask, 38
slow_nine_point_Laplacian_stencil, 38
numerics.h, 39
analytical_value, 40
distance_point_to_segment, 40
five_point_Laplacian_stencil, 41
nine_point_Laplacian_stencil, 41
set_mask, 41
slow_nine_point_Laplacian_stencil, 42

output.c, 42
print_progress, 43
output.h, 44
print_progress, 45

print_progress
output.c, 43
output.h, 45

ResidualSumOfSquares2D, 3

set_boundaries
boundaries.cpp, 10
common-diffusion/boundaries.h, 12
cpu-openmp-diffusion/boundaries.c, 5
cpu-serial-diffusion/boundaries.c, 6
gpu-cuda-diffusion/boundaries.c, 8
gpu-openacc-diffusion/boundaries.c, 9
gpu-opencl-diffusion/boundaries.h, 14
set_mask
numerics.c, 38
numerics.h, 41
slow_nine_point_Laplacian_stencil
numerics.c, 38
numerics.h, 42
Stopwatch, 4
swap_pointers
mesh.c, 34
mesh.h, 35

timer.c, 45
timerStart, 46
timer.h, 46
timerStart
timer.c, 46
type.h, 47
fp_t, 47