phasefield-accelerator-benchmarks

pre-alpha

Generated by Doxygen 1.8.13

Contents

1 Module Index

1.1 Modules

Here is a list of all modules:

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

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

4 Module Documentation

4.1 Benchmarks using CPU hardware

Files

- file cpu/main.c

  Implementation of semi-infinite diffusion equation.
- file cpu/mesh.c

  Implemenatation of mesh handling functions.
- file cpu/mesh.h

  Declaration of mesh function prototypes for CPU benchmarks.
- file cpu/output.c

  Implementation of file output functions.
- file cpu/output.h

  Declaration of output function prototypes for CPU benchmarks.
- file cpu/timer.c

  High-resolution cross-platform machine time reader.
- file cpu/timer.h

  Declaration of timer function prototypes for CPU benchmarks.
- file cpu/type.h

  Definition of scalar data type and CPU-related Doxygen groups.

Typedefs

- typedef double fp_t

Functions

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

  Run simulation using input parameters specified on the command line.
- 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.
- 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.
- void StartTimer ()

  Set CPU frequency and begin timing.
- double GetTimer ()

  Return elapsed time in seconds.

Variables

- struct timeval timerStart

4.1.1 Detailed Description

4.1.2 Typedef Documentation

4.1.2.1 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.

Definition at line 42 of file cpu/type.h.

4.1.3 Function Documentation

4.1.3.1 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 51 of file cpu/main.c.

4.1.3.2 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 41 of file cpu/mesh.c.

4.1.3.3 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 49 of file cpu/output.c.

4.1.3.4 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 95 of file cpu/mesh.c.

4.1.4 Variable Documentation

4.1.4.1 timerStart

```
struct timeval timerStart
```

Platform-dependent data type of hardware time value

Definition at line 49 of file cpu/timer.c.

4.2 Analytical solution

Files

- file cpu/analytic/discretization.c

  Implementation of analytical solution functions.
- file cpu/analytic/discretization.h

  Declaration of analytical solution prototypes.
- file cpu/analytic/main.c

  Analytical solution to semi-infinite diffusion equation.

Functions

- 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 *c)

  Analytical solution of the diffusion equation for a carburizing process.
- 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.
- int main (int argc, char *argv[])

  Run simulation using input parameters specified on the command line.

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 analytical_value()

```
void analytical_value (
            fp_t x,
            fp_t t,
            fp_t D,
            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 $c_\infty = 0$ and $c_0 = 1$, to

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

Definition at line 78 of file cpu/analytic/discretization.c.

4.2.2.2 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 55 of file cpu/analytic/discretization.c.

4.2.2.3 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 50 of file cpu/analytic/main.c.

4.3 Serial implementation

Files

- file cpu/serial/boundaries.c

  Implementation of boundary condition functions without threading.
- file cpu/serial/boundaries.h

  Declaration of boundary condition function prototypes for CPU benchmarks.
- file cpu/serial/discretization.c

  Implementation of boundary condition functions without threading.
- file cpu/serial/discretization.h

  Declaration of discretized mathematical function prototypes for CPU benchmarks.

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.
- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 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 200 of file cpu/serial/discretization.c.

4.3.2.2 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 cpu/serial/boundaries.c.

4.3.2.3 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 209 of file cpu/serial/discretization.c.

4.3.2.4 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 120 of file cpu/serial/discretization.c.

4.3.2.5 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 177 of file cpu/serial/discretization.c.

4.3.2.6 five_point_Laplacian_stencil()

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

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 47 of file cpu/serial/discretization.c.

4.3.2.7 nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 61 of file cpu/serial/discretization.c.

4.3.2.8 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 cpu/serial/boundaries.c.

4.3.2.9 set_threads()

```
void set_threads (
              int n )
```

Set number of OpenMP threads to use in parallel code sections.

Warning

Serial code contains no parallel sections: this setting has no effect.

Definition at line 37 of file cpu/serial/discretization.c.

4.3.2.10 slow_nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

4×4 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 85 of file cpu/serial/discretization.c.

4.4 OpenMP implementation

Files

- file cpu/openmp/boundaries.c

  Implementation of boundary condition functions with OpenMP threading.
- file cpu/openmp/boundaries.h

  Declaration of boundary condition function prototypes for OpenMP benchmarks.
- file cpu/openmp/discretization.c

  Implementation of boundary condition functions with OpenMP threading.
- file cpu/openmp/discretization.h

  Declaration of discretized mathematical function prototypes for OpenMP benchmarks.

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.
- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

4.4.1 Detailed Description

4.4.2 Function Documentation

4.4.2.1 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 206 of file cpu/openmp/discretization.c.

4.4.2.2 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 56 of file cpu/openmp/boundaries.c.

4.4.2.3 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 216 of file cpu/openmp/discretization.c.

4.4.2.4 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 118 of file cpu/openmp/discretization.c.

4.4.2.5 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 183 of file cpu/openmp/discretization.c.

4.4.2.6 five_point_Laplacian_stencil()

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

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 46 of file cpu/openmp/discretization.c.

4.4.2.7 nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 60 of file cpu/openmp/discretization.c.

4.4.2.8 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 38 of file cpu/openmp/boundaries.c.

4.4.2.9 slow_nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

4×4 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 84 of file cpu/openmp/discretization.c.

4.5 Threading Building Blocks implementation

Files

- file boundaries.cpp

  Implementation of boundary condition functions with TBB threading.
- file cpu/tbb/boundaries.h

  Declaration of boundary condition function prototypes for TBB benchmarks.
- file discretization.cpp

  Implementation of boundary condition functions with TBB threading.
- file cpu/tbb/discretization.h

  Declaration of discretized mathematical function prototypes for TBB benchmarks.

Classes

- class ResidualSumOfSquares2D

  Comparison algorithm for execution on the block of threads.

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.
- void set_threads (int n)

  Requested number of TBB threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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 chi, fp_t *c)

  Analytical solution of the diffusion equation for a carburizing process.
- 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.
- 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.

4.5.1 Detailed Description

4.5.2 Function Documentation

4.5.2.1 analytical_value() [1/2]

```
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 206 of file cpu/openmp/discretization.c.

4.5.2.2 analytical_value() [2/2]

```
void analytical_value (
            fp_t x,
            fp_t t,
            fp_t D,
            fp_t chi,
            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 209 of file discretization.cpp.

4.5.2.3 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 59 of file boundaries.cpp.

4.5.2.4 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 298 of file discretization.cpp.

4.5.2.5 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 124 of file discretization.cpp.

4.5.2.6 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 186 of file discretization.cpp.

4.5.2.7 five_point_Laplacian_stencil()

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

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 52 of file discretization.cpp.

4.5.2.8 nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 66 of file discretization.cpp.

4.5.2.9 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 41 of file boundaries.cpp.

4.5.2.10 set_threads()

```
void set_threads (
            int n )
```

Requested number of TBB threads to use in parallel code sections.

Warning

This setting does not appear to have any effect.

Definition at line 42 of file discretization.cpp.

4.5.2.11 slow_nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

4×4 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 90 of file discretization.cpp.

4.6 Benchmarks using GPU hardware

Files

- file gpu/main.c

  Implementation of semi-infinite diffusion equation.
- file gpu/mesh.c

  Implemenatation of mesh handling functions.
- file gpu/mesh.h

  Declaration of mesh function prototypes for GPU benchmarks.
- file gpu/output.c

  Implementation of file output functions.
- file gpu/output.h

  Declaration of output function prototypes for GPU benchmarks.
- file gpu/timer.c

  High-resolution cross-platform machine time reader.
- file gpu/timer.h

  Declaration of timer function prototypes for GPU benchmarks.
- file gpu/type.h

  Definition of scalar data type and GPU-related Doxygen groups.

Typedefs

- typedef double fp_t

Functions

- int main (int argc, char ∗argv[ ])

  Run simulation using input parameters specified on the command line.
- 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.
- 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.
- void StartTimer ()

  Set CPU frequency and begin timing.
- double GetTimer ()

  Return elapsed time in seconds.

Variables

- struct timeval timerStart

4.6.1 Detailed Description

4.6.2 Typedef Documentation

4.6.2.1 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.

Definition at line 40 of file gpu/type.h.

4.6.3 Function Documentation

4.6.3.1 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 50 of file gpu/main.c.

4.6.3.2 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 40 of file gpu/mesh.c.

4.6.3.3 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 48 of file gpu/output.c.

4.6.3.4 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 98 of file gpu/mesh.c.

4.6.4 Variable Documentation

4.6.4.1 timerStart

```
struct timeval timerStart
```

Platform-dependent data type of hardware time value

Definition at line 49 of file gpu/timer.c.

4.7 CUDA implementation

Files

- file gpu/cuda/boundaries.c

  Implementation of boundary condition functions with OpenMP threading.
- file gpu/cuda/boundaries.h

  Declaration of boundary condition function prototypes for CUDA benchmarks.
- file discretization.cu

  Implementation of boundary condition functions with CUDA acceleration.
- file gpu/cuda/discretization.h

  Declaration of discretized mathematical function prototypes for CUDA benchmarks.

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_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.
- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- __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, int bs)

  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, 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.
- 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.
- 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_W]

  Allocate constant memory on the GPU for the convolution mask.

4.7.1 Detailed Description

4.7.2 Function Documentation

4.7.2.1 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 361 of file discretization.cu.

4.7.2.2 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 56 of file gpu/cuda/boundaries.c.

4.7.2.3 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 371 of file discretization.cu.

4.7.2.4 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 217 of file discretization.cu.

4.7.2.5 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 338 of file discretization.cu.

4.7.2.6 five_point_Laplacian_stencil()

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

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 72 of file discretization.cu.

4.7.2.7 nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 86 of file discretization.cu.

4.7.2.8 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 38 of file gpu/cuda/boundaries.c.

4.7.2.9 slow_nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

4×4 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 110 of file discretization.cu.

4.8 OpenAcc implementation

Files

- file gpu/openacc/boundaries.c

  Implementation of boundary condition functions with OpenMP threading.
- file gpu/openacc/boundaries.h

  Declaration of boundary condition function prototypes for OpenAcc benchmarks.
- file gpu/openacc/discretization.c

  Implementation of boundary condition functions with OpenACC threading.
- file gpu/openacc/discretization.h

  Declaration of discretized mathematical function prototypes for OpenAcc benchmarks.

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.
- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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.
- 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 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.
- 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.

4.8.1 Detailed Description

4.8.2 Function Documentation

4.8.2.1 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 206 of file cpu/openmp/discretization.c.

4.8.2.2 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 56 of file gpu/openacc/boundaries.c.

4.8.2.3 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 222 of file gpu/openacc/discretization.c.

4.8.2.4 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 119 of file gpu/openacc/discretization.c.

4.8.2.5 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 197 of file gpu/openacc/discretization.c.

4.8.2.6 five_point_Laplacian_stencil()

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

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 47 of file gpu/openacc/discretization.c.

4.8.2.7 nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 61 of file gpu/openacc/discretization.c.

4.8.2.8 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 38 of file gpu/openacc/boundaries.c.

4.8.2.9 slow_nine_point_Laplacian_stencil()

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

Write 9-point Laplacian stencil into convolution mask.

4×4 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 85 of file gpu/openacc/discretization.c.

5 Class Documentation

5.1 ResidualSumOfSquares2D Class Reference

Comparison algorithm for execution on the block of threads.

5.1.1 Detailed Description

Comparison algorithm for execution on the block of threads.

Definition at line 217 of file discretization.cpp.

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

- discretization.cpp

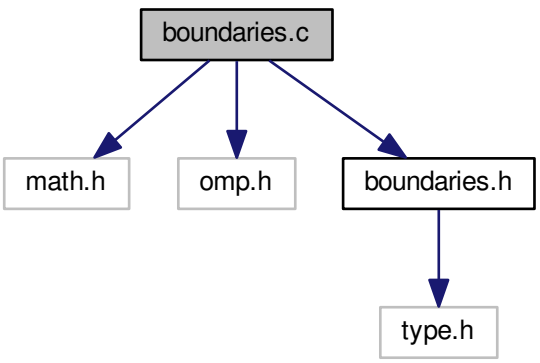
6 File Documentation

6.1 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/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.

6.1.1 Detailed Description

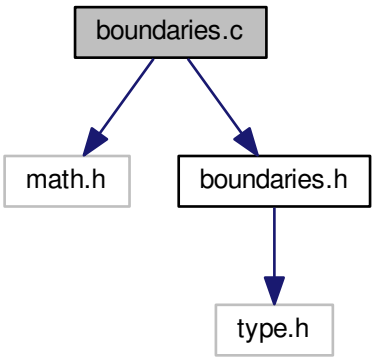
Implementation of boundary condition functions with OpenMP threading.

6.2 boundaries.c File Reference

Implementation of boundary condition functions without threading.

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

Include dependency graph for cpu/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.

6.2.1 Detailed Description

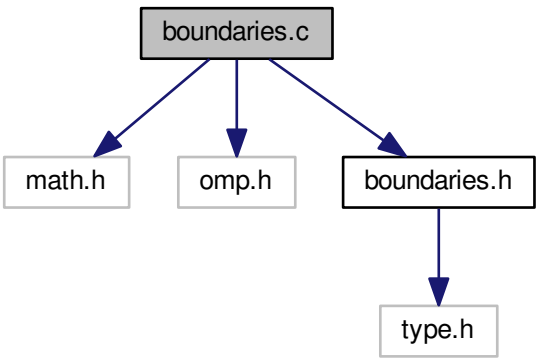
Implementation of boundary condition functions without threading.

6.3 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/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.

6.3.1 Detailed Description

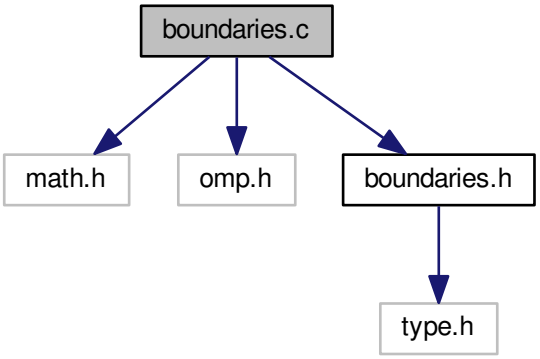
Implementation of boundary condition functions with OpenMP threading.

6.4 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/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.

6.4.1 Detailed Description

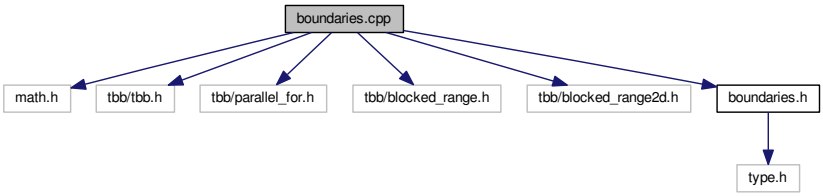
Implementation of boundary condition functions with OpenMP threading.

6.5 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.

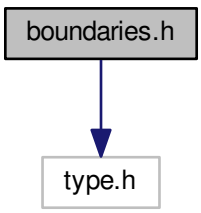
6.5.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

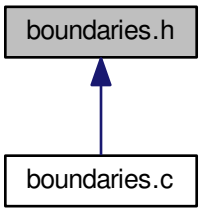
6.6 boundaries.h File Reference

Declaration of boundary condition function prototypes for OpenMP benchmarks.

`#include "type.h"`
Include dependency graph for cpu/openmp/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, 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.

6.6.1 Detailed Description

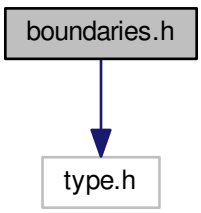
Declaration of boundary condition function prototypes for OpenMP benchmarks.

6.7 boundaries.h File Reference

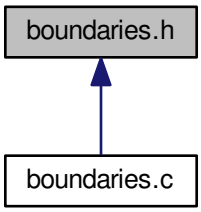
Declaration of boundary condition function prototypes for CPU benchmarks.

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

Include dependency graph for cpu/serial/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, 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.

6.7.1 Detailed Description

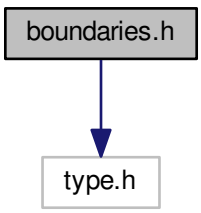
Declaration of boundary condition function prototypes for CPU benchmarks.

6.8 boundaries.h File Reference

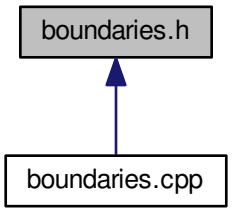
Declaration of boundary condition function prototypes for TBB benchmarks.

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

Include dependency graph for cpu/tbb/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, 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.

6.8.1 Detailed Description

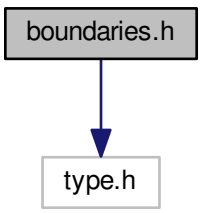
Declaration of boundary condition function prototypes for TBB benchmarks.

6.9 boundaries.h File Reference

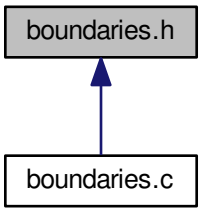
Declaration of boundary condition function prototypes for CUDA benchmarks.

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

Include dependency graph for gpu/cuda/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, 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.

6.9.1 Detailed Description

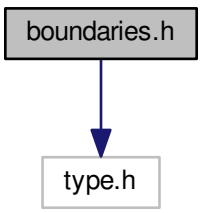
Declaration of boundary condition function prototypes for CUDA benchmarks.

6.10 boundaries.h File Reference

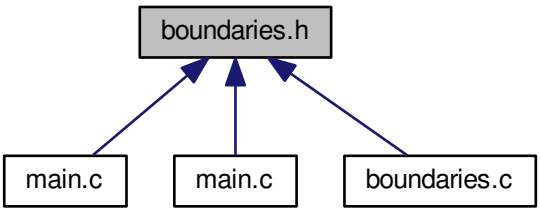
Declaration of boundary condition function prototypes for OpenAcc benchmarks.

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

Include dependency graph for gpu/openacc/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, 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.

6.10.1 Detailed Description

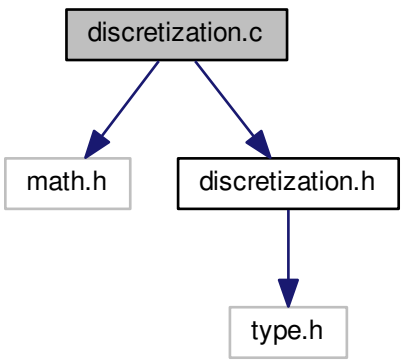
Declaration of boundary condition function prototypes for OpenAcc benchmarks.

6.11 discretization.c File Reference

Implementation of analytical solution functions.

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

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



Functions

- 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 *c)

  Analytical solution of the diffusion equation for a carburizing process.
- 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.

6.11.1 Detailed Description

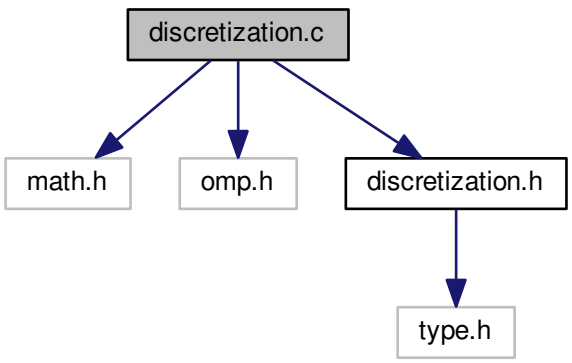
Implementation of analytical solution functions.

6.12 discretization.c File Reference

Implementation of boundary condition functions with OpenMP threading.

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

Include dependency graph for cpu/openmp/discretization.c:



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

6.12.1 Detailed Description

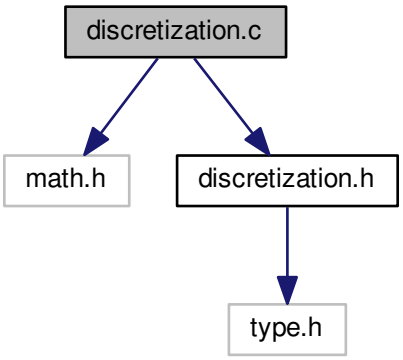
Implementation of boundary condition functions with OpenMP threading.

6.13 discretization.c File Reference

Implementation of boundary condition functions without threading.

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

Include dependency graph for cpu/serial/discretization.c:



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

6.13.1 Detailed Description

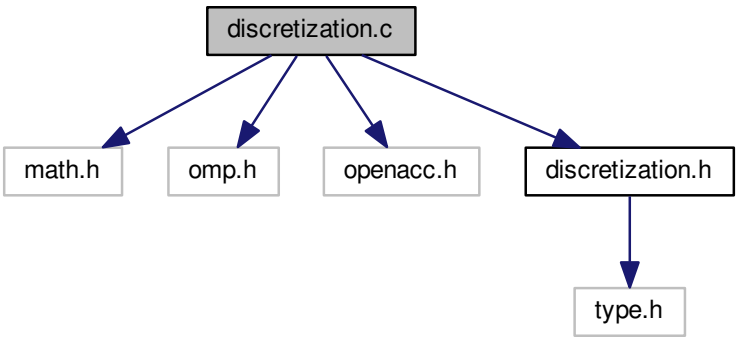
Implementation of boundary condition functions without threading.

6.14 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 dependency graph for gpu/openacc/discretization.c:



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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.
- 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 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.

6.14.1 Detailed Description

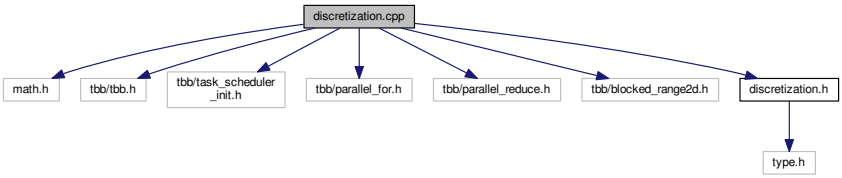
Implementation of boundary condition functions with OpenACC threading.

6.15 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 "discretization.h"
```

Include dependency graph for discretization.cpp:



Classes

- class ResidualSumOfSquares2D

  Comparison algorithm for execution on the block of threads.

Functions

- void set_threads (int n)

  Requested number of TBB threads to use in parallel code sections.
- void five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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 chi, fp_t ∗c)

  Analytical solution of the diffusion equation for a carburizing process.
- 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.

6.15.1 Detailed Description

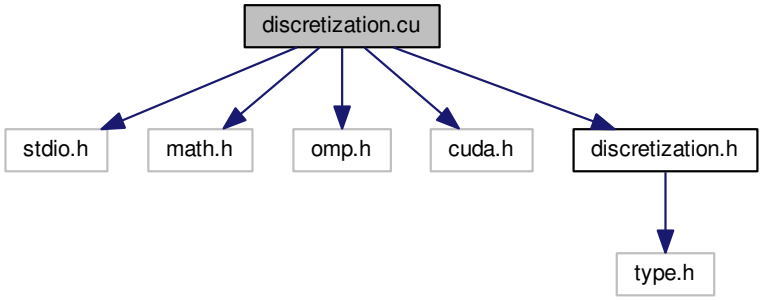
Implementation of boundary condition functions with TBB threading.

6.16 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 "discretization.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.
- #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 five_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 5-point Laplacian stencil into convolution mask.
- void nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void slow_nine_point_Laplacian_stencil (fp_t dx, fp_t dy, fp_t **mask_lap)

  Write 9-point Laplacian stencil into convolution mask.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- __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, int bs)

  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, 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.
- 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.
- 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_W]

  Allocate constant memory on the GPU for the convolution mask.

6.16.1 Detailed Description

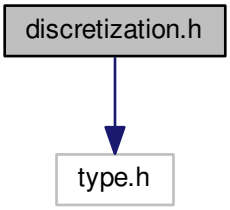
Implementation of boundary condition functions with CUDA acceleration.

6.17 discretization.h File Reference

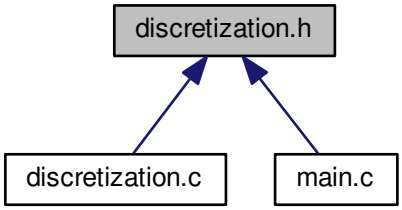
Declaration of analytical solution prototypes.

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

Include dependency graph for cpu/analytic/discretization.h:



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



Functions

- 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 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.
- void analytical_value (fp_t x, fp_t t, fp_t D, fp_t *c)

  Analytical solution of the diffusion equation for a carburizing process.

6.17.1 Detailed Description

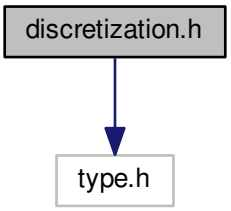
Declaration of analytical solution prototypes.

6.18 discretization.h File Reference

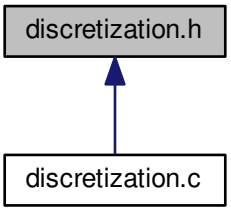
Declaration of discretized mathematical function prototypes for OpenMP benchmarks.

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

Include dependency graph for cpu/openmp/discretization.h:



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



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

Update the scalar composition field using old and Laplacian values.

- 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.
- 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.

6.18.1 Detailed Description

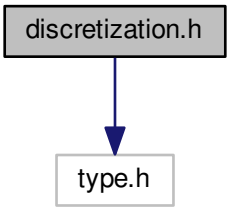
Declaration of discretized mathematical function prototypes for OpenMP benchmarks.

6.19 discretization.h File Reference

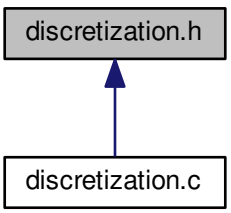
Declaration of discretized mathematical function prototypes for CPU benchmarks.

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

Include dependency graph for cpu/serial/discretization.h:



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



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in parallel code sections.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

6.19.1 Detailed Description

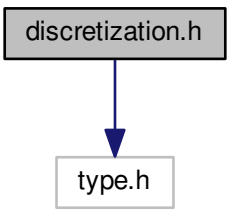
Declaration of discretized mathematical function prototypes for CPU benchmarks.

6.20 discretization.h File Reference

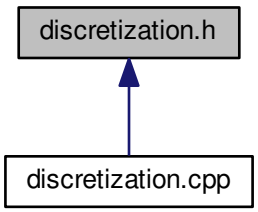
Declaration of discretized mathematical function prototypes for TBB benchmarks.

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

Include dependency graph for cpu/tbb/discretization.h:



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



Functions

- void set_threads (int n)

  Requested number of TBB threads to use in parallel code sections.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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, int nx, int ny, int nm, fp_t D, fp_t dt, fp_t *elapsed)

  Update the scalar composition field using old and Laplacian values.
- 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.
- 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.

6.20.1 Detailed Description

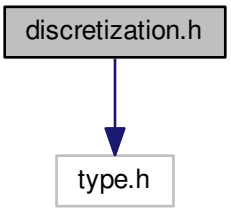
Declaration of discretized mathematical function prototypes for TBB benchmarks.

6.21 discretization.h File Reference

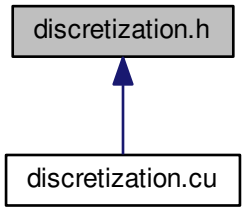
Declaration of discretized mathematical function prototypes for CUDA benchmarks.

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

Include dependency graph for gpu/cuda/discretization.h:



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



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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.
- 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.
- 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.

6.21.1 Detailed Description

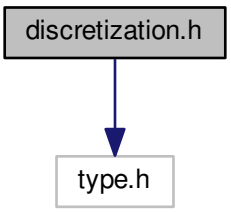
Declaration of discretized mathematical function prototypes for CUDA benchmarks.

6.22 discretization.h File Reference

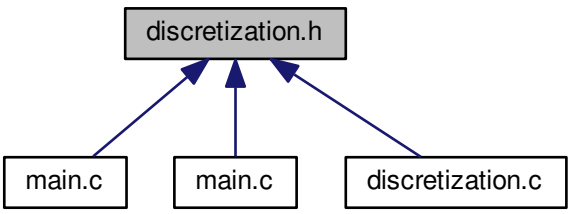
Declaration of discretized mathematical function prototypes for OpenAcc benchmarks.

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

Include dependency graph for gpu/openacc/discretization.h:



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



Functions

- void set_threads (int n)

  Set number of OpenMP threads to use in CPU code sections.
- void set_mask (fp_t dx, fp_t dy, int nm, fp_t **mask_lap)

  Specify which stencil to use for the Laplacian.
- 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.
- 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.
- 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.

6.22.1 Detailed Description

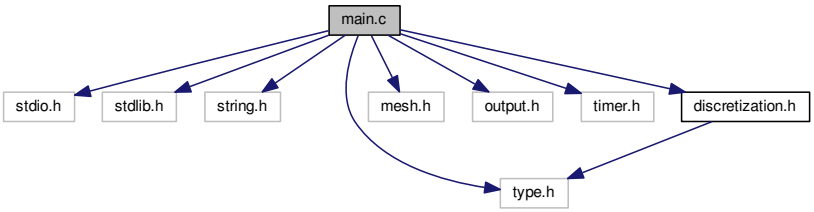
Declaration of discretized mathematical function prototypes for OpenAcc benchmarks.

6.23 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 "output.h"
#include "timer.h"
#include "discretization.h"
```

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



Functions

- int main (int argc, char ∗argv[ ])

  Run simulation using input parameters specified on the command line.

6.23.1 Detailed Description

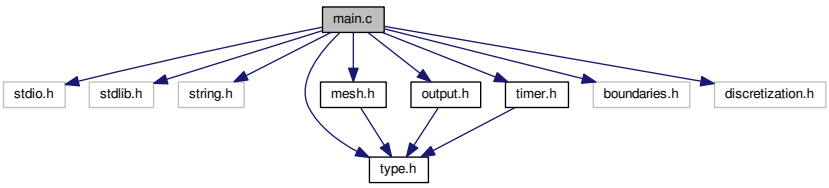
Analytical solution to semi-infinite diffusion equation.

6.24 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 "output.h"
#include "timer.h"
#include "boundaries.h"
#include "discretization.h"
```

Include dependency graph for cpu/main.c:



Functions

- int main (int argc, char ∗argv[ ])

  Run simulation using input parameters specified on the command line.

6.24.1 Detailed Description

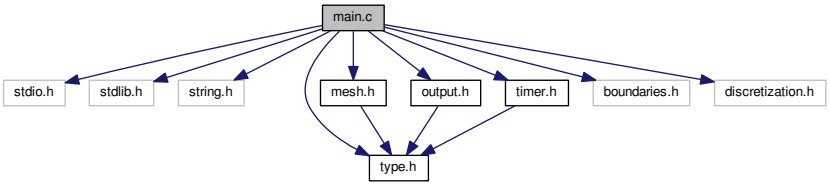
Implementation of semi-infinite diffusion equation.

6.25 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 "output.h"
#include "timer.h"
#include "boundaries.h"
#include "discretization.h"
```

Include dependency graph for gpu/main.c:



Functions

- int main (int argc, char ∗argv[ ])

  Run simulation using input parameters specified on the command line.

6.25.1 Detailed Description

Implementation of semi-infinite diffusion equation.

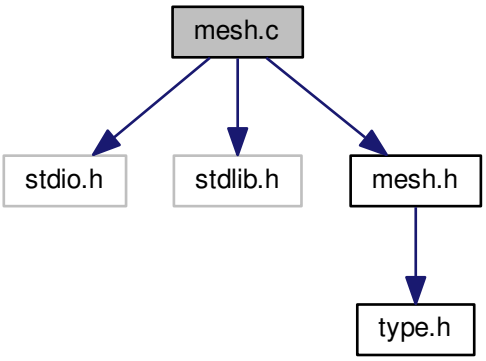
6.26 mesh.c File Reference

Implemenatation of mesh handling functions.

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

```
#include "mesh.h"
```

Include dependency graph for cpu/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.

6.26.1 Detailed Description

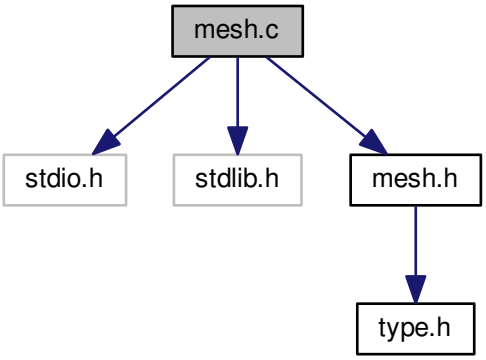
Implemenatation of mesh handling functions.

6.27 mesh.c File Reference

Implemenatation of mesh handling functions.

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

`#include "mesh.h"`
Include dependency graph for gpu/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.

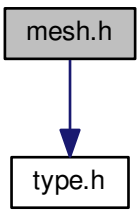
6.27.1 Detailed Description

Implemenatation of mesh handling functions.

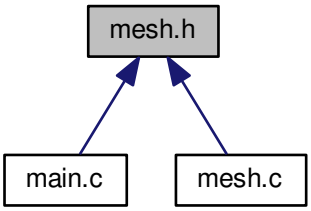
6.28 mesh.h File Reference

Declaration of mesh function prototypes for CPU benchmarks.

`#include "type.h"`
Include dependency graph for cpu/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.

6.28.1 Detailed Description

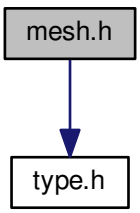
Declaration of mesh function prototypes for CPU benchmarks.

6.29 mesh.h File Reference

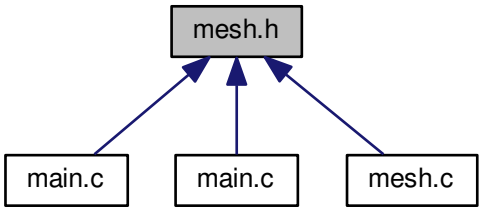
Declaration of mesh function prototypes for GPU benchmarks.

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

Include dependency graph for gpu/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.

6.29.1 Detailed Description

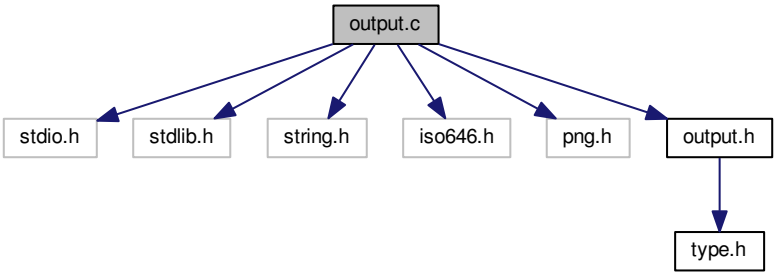
Declaration of mesh function prototypes for GPU benchmarks.

6.30 output.c File Reference

Implementation of file output functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iso646.h>
#include <png.h>
#include "output.h"
```

Include dependency graph for cpu/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.

6.30.1 Detailed Description

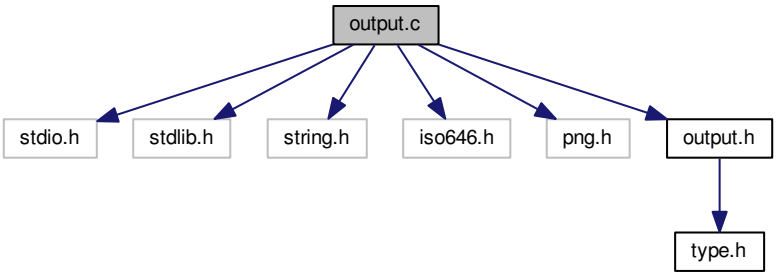
Implementation of file output functions.

6.31 output.c File Reference

Implementation of file output functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iso646.h>
#include <png.h>
#include "output.h"
```

Include dependency graph for gpu/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.

6.31.1 Detailed Description

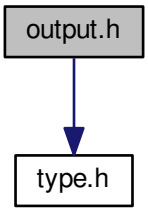
Implementation of file output functions.

6.32 output.h File Reference

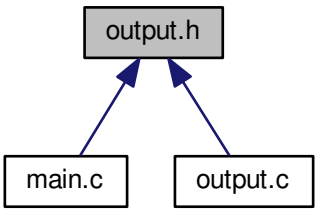
Declaration of output function prototypes for CPU benchmarks.

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

Include dependency graph for cpu/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.

6.32.1 Detailed Description

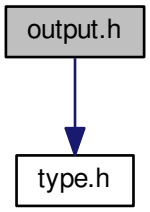
Declaration of output function prototypes for CPU benchmarks.

6.33 output.h File Reference

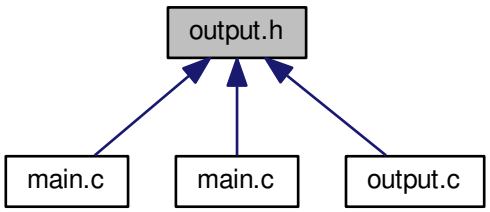
Declaration of output function prototypes for GPU benchmarks.

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

Include dependency graph for gpu/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.

6.33.1 Detailed Description

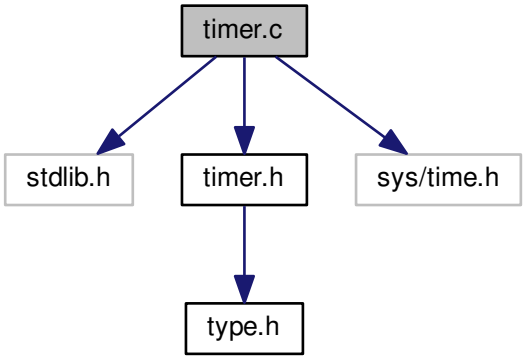
Declaration of output function prototypes for GPU benchmarks.

6.34 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 cpu/timer.c:



Functions

- void StartTimer ()

  Set CPU frequency and begin timing.
- double GetTimer ()

  Return elapsed time in seconds.

Variables

- struct timeval timerStart

6.34.1 Detailed Description

High-resolution cross-platform machine time reader.

Author

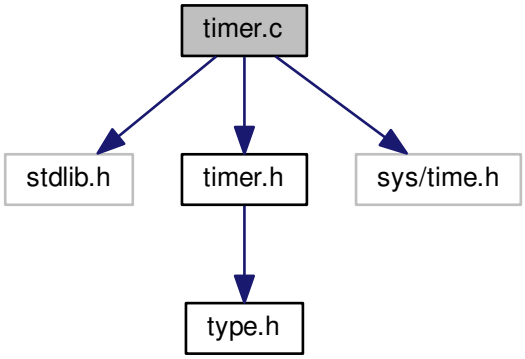
NVIDIA

6.35 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 gpu/timer.c:



Functions

- void StartTimer ()

  Set CPU frequency and begin timing.
- double GetTimer ()

  Return elapsed time in seconds.

Variables

- struct timeval timerStart

6.35.1 Detailed Description

High-resolution cross-platform machine time reader.

Author

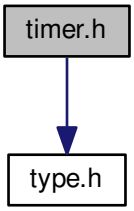
NVIDIA

6.36 timer.h File Reference

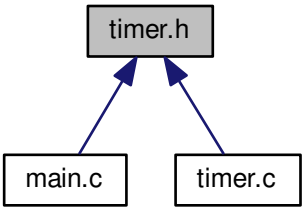
Declaration of timer function prototypes for CPU benchmarks.

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

Include dependency graph for cpu/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.

6.36.1 Detailed Description

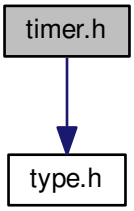
Declaration of timer function prototypes for CPU benchmarks.

6.37 timer.h File Reference

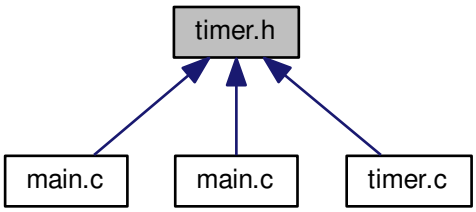
Declaration of timer function prototypes for GPU benchmarks.

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

Include dependency graph for gpu/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.

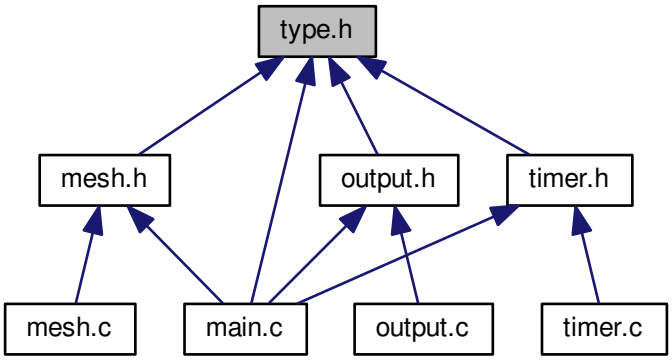
6.37.1 Detailed Description

Declaration of timer function prototypes for GPU benchmarks.

6.38 type.h File Reference

Definition of scalar data type and CPU-related Doxygen groups.

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



Typedefs

- typedef double fp_t

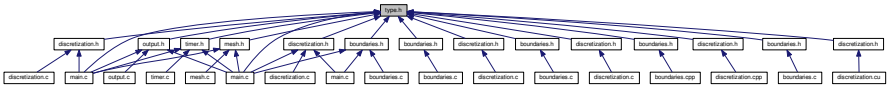
6.38.1 Detailed Description

Definition of scalar data type and CPU-related Doxygen groups.

6.39 type.h File Reference

Definition of scalar data type and GPU-related Doxygen groups.

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



Typedefs

- typedef double fp_t

6.39.1 Detailed Description

Definition of scalar data type and GPU-related Doxygen groups.

Index