# HiPerC

pre-alpha

Generated by Doxygen 1.8.13

# Contents

# 1 Class Index

## 1.1 Class List

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

# 2 File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# 3 Class Documentation

## 3.1 CudaData Struct Reference

Container for GPU array pointers and parameters.

```
#include <cuda_data.h>
```

#### Public Attributes

- fp_t * conc_old
- fp_t * conc_new
- fp_t * conc_lap

### 3.1.1 Detailed Description

Container for GPU array pointers and parameters.

Definition at line 35 of file cuda_data.h.

### 3.1.2 Member Data Documentation

#### 3.1.2.1 conc_lap

```
fp_t* CudaData::conc_lap
```

Definition at line 38 of file cuda_data.h.

#### 3.1.2.2 conc_new

```
fp_t* CudaData::conc_new
```

Definition at line 37 of file cuda_data.h.

#### 3.1.2.3 conc_old

```
fp_t* CudaData::conc_old
```

Definition at line 36 of file cuda_data.h.

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

- cuda_data.h

## 3.2 OpenCLData Struct Reference

Container for GPU array pointers and parameters.

```
#include <opencl_data.h>
```

### Public Attributes

- cl_mem conc_old
- cl_mem conc_new
- cl_mem conc_lap
- cl_mem mask
- cl_mem bc
- cl_program boundary_program
- cl_program convolution_program
- cl_program diffusion_program
- cl_kernel boundary_kernel
- cl_kernel convolution_kernel
- cl_kernel diffusion_kernel
- cl_context context
- cl_command_queue commandQueue

### 3.2.1 Detailed Description

Container for GPU array pointers and parameters.

Definition at line 47 of file opencl_data.h.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 bc

```
cl_mem OpenCLData::bc
```

Definition at line 54 of file opencl_data.h.

#### 3.2.2.2 boundary_kernel

`cl_kernel OpenCLData::boundary_kernel`

Definition at line 62 of file opencl_data.h.

#### 3.2.2.3 boundary_program

`cl_program OpenCLData::boundary_program`

Definition at line 57 of file opencl_data.h.

#### 3.2.2.4 commandQueue

`cl_command_queue OpenCLData::commandQueue`

Definition at line 68 of file opencl_data.h.

#### 3.2.2.5 conc_lap

`cl_mem OpenCLData::conc_lap`

Definition at line 51 of file opencl_data.h.

#### 3.2.2.6 conc_new

`cl_mem OpenCLData::conc_new`

Definition at line 50 of file opencl_data.h.

#### 3.2.2.7 conc_old

`cl_mem OpenCLData::conc_old`

Definition at line 49 of file opencl_data.h.

#### 3.2.2.8 context

`cl_context OpenCLData::context`

Definition at line 67 of file opencl_data.h.

#### 3.2.2.9 convolution_kernel

```
cl_kernel OpenCLData::convolution_kernel
```

Definition at line 63 of file opencl_data.h.

#### 3.2.2.10 convolution_program

```
cl_program OpenCLData::convolution_program
```

Definition at line 58 of file opencl_data.h.

#### 3.2.2.11 diffusion_kernel

```
cl_kernel OpenCLData::diffusion_kernel
```

Definition at line 64 of file opencl_data.h.

#### 3.2.2.12 diffusion_program

```
cl_program OpenCLData::diffusion_program
```

Definition at line 59 of file opencl_data.h.

#### 3.2.2.13 mask

```
cl_mem OpenCLData::mask
```

Definition at line 53 of file opencl_data.h.

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

- opencl_data.h

## 3.3 Stopwatch Struct Reference

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

### Public Attributes

- double conv
- double step
- double file
- double soln

### 3.3.1 Detailed Description

Container for timing data

Definition at line 41 of file type.h.

### 3.3.2 Member Data Documentation

#### 3.3.2.1 conv

`double Stopwatch::conv`

Cumulative time executing compute_convolution()

Definition at line 45 of file type.h.

#### 3.3.2.2 file

`double Stopwatch::file`

Cumulative time executing write_csv() and write_png()

Definition at line 55 of file type.h.

#### 3.3.2.3 soln

`double Stopwatch::soln`

Cumulative time executing check_solution()

Definition at line 60 of file type.h.

#### 3.3.2.4 step

`double Stopwatch::step`

Cumulative time executing solve_diffusion_equation()

Definition at line 50 of file type.h.

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

- type.h

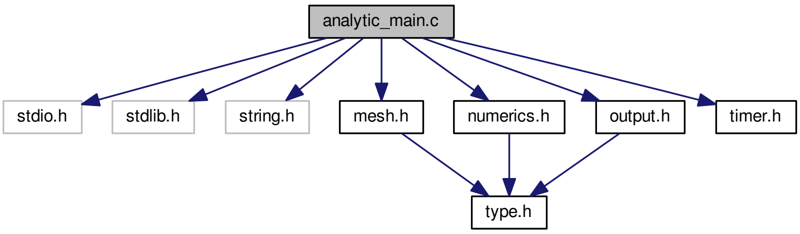
# 4 File Documentation

## 4.1 analytic_main.c File Reference

Analytical solution to semi-infinite diffusion equation.

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

Include dependency graph for analytic_main.c:




**Functions**

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

  *Update the scalar composition field using analytical solution.*
- int main (int argc, char *argv[ ])

  *Find analytical solution at intervals specified in the parameters file.*

### 4.1.1 Detailed Description

Analytical solution to semi-infinite diffusion equation.

### 4.1.2 Function Documentation

#### 4.1.2.1 main()

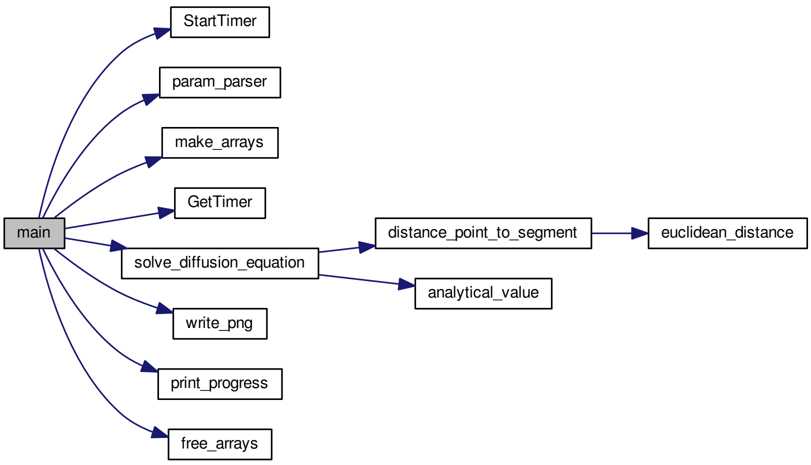
```
int main (
            int argc,
            char * argv[] )
```

Find analytical solution at intervals specified in the parameters file.

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

Definition at line 69 of file analytic_main.c.

Here is the call graph for this function:



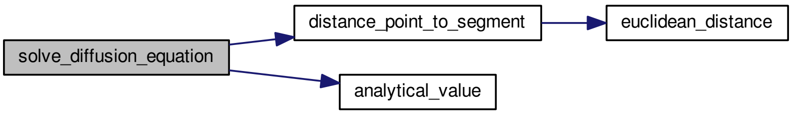

#### 4.1.2.2 solve_diffusion_equation()

```
void solve_diffusion_equation (
            fp_t ** conc,
            int nx,
            int ny,
            int nm,
            fp_t dx,
            fp_t dy,
            fp_t D,
            fp_t dt,
            fp_t elapsed )
```

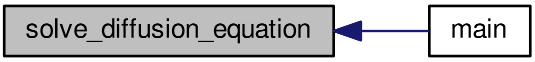
Update the scalar composition field using analytical solution.

Definition at line 37 of file analytic_main.c.

Here is the call graph for this function:



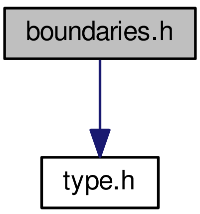

Here is the caller graph for this function:



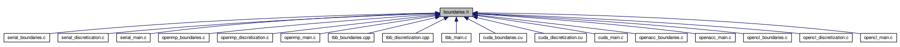

## 4.2 boundaries.h File Reference

Declaration of boundary condition function prototypes.

`#include "type.h"`
Include dependency graph for boundaries.h:




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



#### Functions

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

  *Set values to be used along the simulation domain boundaries.*
- void apply_initial_conditions (fp_t **conc_old, int nx, int ny, int nm, fp_t bc[2][2])

  *Initialize flat composition field with fixed boundary conditions.*
- void apply_boundary_conditions (fp_t **conc_old, int nx, int ny, int nm, fp_t bc[2][2])

  *Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.*

### 4.2.1 Detailed Description

Declaration of boundary condition function prototypes.

### 4.2.2 Function Documentation

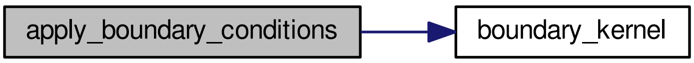
#### 4.2.2.1 apply_boundary_conditions()

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

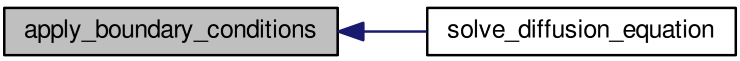
Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.

Definition at line 54 of file serial_boundaries.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 4.2.2.2 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 37 of file serial_boundaries.c.

Here is the caller graph for this function:


#### 4.2.2.3 set_boundaries()

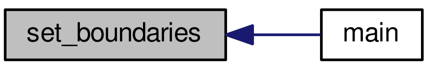
```
void set_boundaries (
            fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

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

Definition at line 28 of file serial_boundaries.c.

Here is the caller graph for this function:

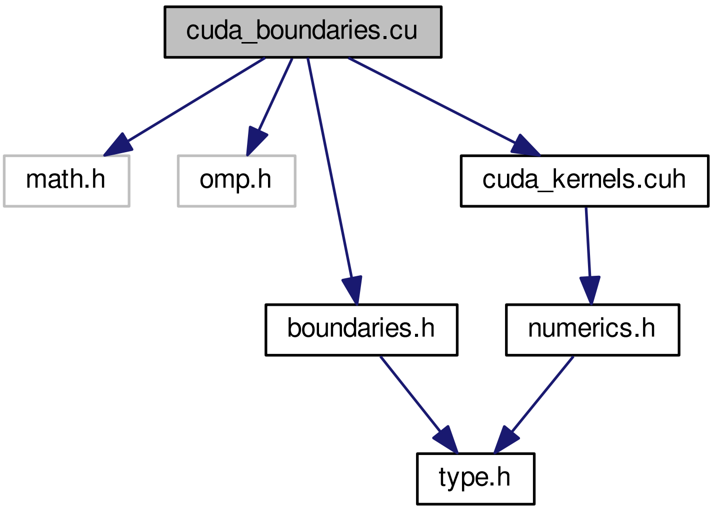


## 4.3 cuda_boundaries.cu File Reference

Implementation of boundary condition functions with OpenMP threading.

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

Include dependency graph for cuda_boundaries.cu:




### 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 boundary_kernel (fp_t ∗d_conc, const int nx, const int ny, const int nm)

  *Boundary condition kernel for execution on the GPU.*

### Variables

- fp_t d_bc [2][2]

  *Boundary condition array on the GPU, allocated in protected memory.*

### 4.3.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

### 4.3.2 Function Documentation

#### 4.3.2.1 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 45 of file cuda_boundaries.cu.

#### 4.3.2.2 boundary_kernel()

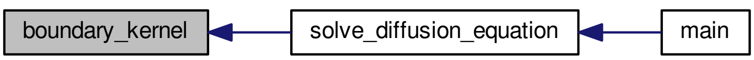
```
void boundary_kernel (
            fp_t * conc,
            const int nx,
            const int ny,
            const int nm )
```

Boundary condition kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

Definition at line 68 of file cuda_boundaries.cu.

Here is the caller graph for this function:



#### 4.3.2.3 set_boundaries()

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

Set values to be used along the simulation domain boundaries.

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

Definition at line 36 of file cuda_boundaries.cu.

### 4.3.3 Variable Documentation

#### 4.3.3.1 d_bc

```
fp_t d_bc[2][2]
```

Boundary condition array on the GPU, allocated in protected memory.

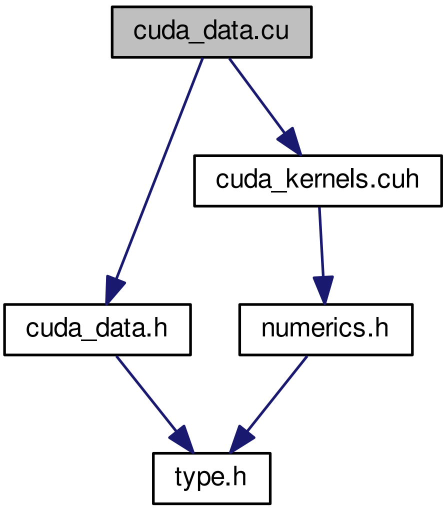
Definition at line 34 of file cuda_boundaries.cu.

## 4.4 cuda_data.cu File Reference

Implementation of functions to create and destroy CudaData struct.

```
#include "cuda_data.h"
#include "cuda_kernels.cuh"
```

Include dependency graph for cuda_data.cu:



### Functions

- void init_cuda (fp_t **conc_old, fp_t **mask_lap, fp_t bc[2][2], int nx, int ny, int nm, struct CudaData *dev)

  *Initialize CUDA device memory before marching.*
- void free_cuda (struct CudaData *dev)

  *Free CUDA device memory after marching.*

### 4.4.1 Detailed Description

Implementation of functions to create and destroy CudaData struct.

### 4.4.2 Function Documentation

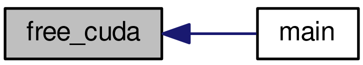
#### 4.4.2.1 free_cuda()

```
void free_cuda (
            struct CudaData * dev )
```

Free CUDA device memory after marching.

Definition at line 48 of file cuda_data.cu.

Here is the caller graph for this function:



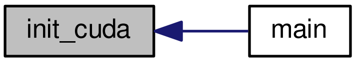

#### 4.4.2.2 init_cuda()

```
void init_cuda (
            fp_t ** conc_old,
            fp_t ** mask_lap,
            fp_t bc[2][2],
            int nx,
            int ny,
            int nm,
            struct CudaData * dev )
```

Initialize CUDA device memory before marching.

Definition at line 31 of file cuda_data.cu.

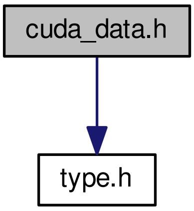
Here is the caller graph for this function:



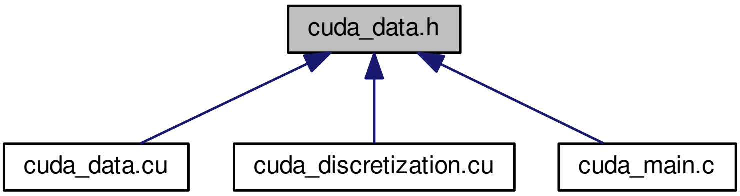

### 4.5 cuda_data.h File Reference

Declaration of CUDA data container.

```
#include "type.h"
```
Include dependency graph for cuda_data.h:




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




#### Classes

- struct CudaData

  *Container for GPU array pointers and parameters.*

#### Functions

- void init_cuda (fp_t **conc_old, fp_t **mask_lap, fp_t bc[2][2], int nx, int ny, int nm, struct CudaData *dev)

  *Initialize CUDA device memory before marching.*
- void cuda_diffusion_solver (struct CudaData *dev, fp_t **conc_new, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t *elapsed, struct Stopwatch *sw)

  *Specialization of solve_diffusion_equation() using CUDA.*
- void free_cuda (struct CudaData *dev)

  *Free CUDA device memory after marching.*

### 4.5.1 Detailed Description

Declaration of CUDA data container.

### 4.5.2 Function Documentation

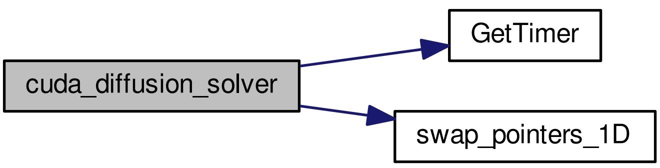
#### 4.5.2.1 cuda_diffusion_solver()

```
void cuda_diffusion_solver (
            struct CudaData * dev,
            fp_t ** conc_new,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

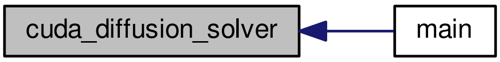
Specialization of solve_diffusion_equation() using CUDA.

Definition at line 132 of file cuda_discretization.cu.

Here is the call graph for this function:



Here is the caller graph for this function:



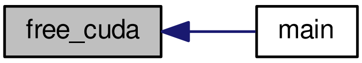

#### 4.5.2.2 free_cuda()

```
void free_cuda (
            struct CudaData * dev )
```

Free CUDA device memory after marching.

Definition at line 48 of file cuda_data.cu.

Here is the caller graph for this function:



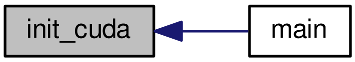

#### 4.5.2.3 init_cuda()

```
void init_cuda (
            fp_t ** conc_old,
            fp_t ** mask_lap,
            fp_t bc[2][2],
            int nx,
            int ny,
            int nm,
            struct CudaData * dev )
```

Initialize CUDA device memory before marching.

Definition at line 31 of file cuda_data.cu.

Here is the caller graph for this function:

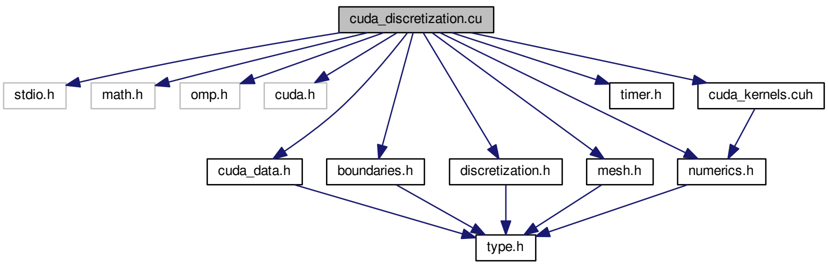



## 4.6 cuda_discretization.cu File Reference

Implementation of boundary condition functions with CUDA acceleration.

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

Include dependency graph for cuda_discretization.cu:




### Functions

- void convolution_kernel (fp_t ∗d_conc_old, fp_t ∗d_conc_lap, const int nx, const int ny, const int nm)

    *Tiled convolution algorithm for execution on the GPU.*
- void diffusion_kernel (fp_t ∗d_conc_old, fp_t ∗d_conc_new, fp_t ∗d_conc_lap, const int nx, const int ny, const int nm, const fp_t D, const fp_t dt)

    *Vector addition algorithm for execution on the GPU.*
- void cuda_diffusion_solver (struct CudaData ∗dev, fp_t ∗∗conc_new, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t ∗elapsed, struct Stopwatch ∗sw)

    *Specialization of solve_diffusion_equation() using CUDA.*
- void check_solution (fp_t ∗∗conc_new, fp_t ∗∗conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t ∗rss)

    *Compare numerical and analytical solutions of the diffusion equation.*
- 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.*

#### Variables

- [fp_t](#) d_mask [MAX_MASK_W *MAX_MASK_H]

  *Convolution mask array on the GPU, allocated in protected memory.*

### 4.6.1 Detailed Description

Implementation of boundary condition functions with CUDA acceleration.

### 4.6.2 Function Documentation

#### 4.6.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

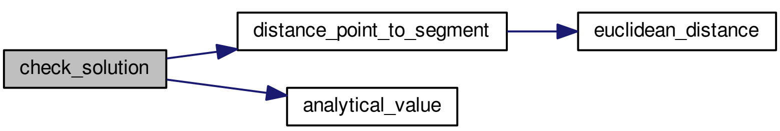
**Returns**

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

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

Definition at line 177 of file cuda_discretization.cu.

Here is the call graph for this function:



#### 4.6.2.2 compute_convolution()

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

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

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

Definition at line 225 of file cuda_discretization.cu.

#### 4.6.2.3 convolution_kernel()

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

Tiled convolution algorithm for execution on the GPU.

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

Note:

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

Definition at line 43 of file cuda_discretization.cu.

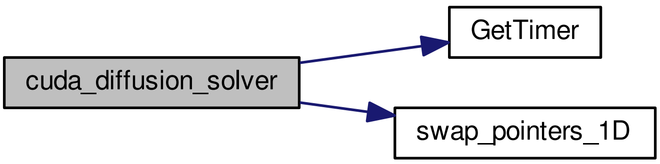
#### 4.6.2.4 cuda_diffusion_solver()

```
void cuda_diffusion_solver (
            struct CudaData * dev,
            fp_t ** conc_new,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

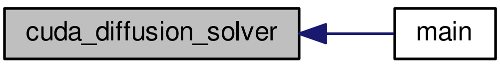
Specialization of solve_diffusion_equation() using CUDA.

Definition at line 132 of file cuda_discretization.cu.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 4.6.2.5 diffusion_kernel()

```
void diffusion_kernel (
            fp_t * conc_old,
            fp_t * conc_new,
            fp_t * conc_lap,
            const int nx,
```

```
const int ny,
const int nm,
const fp_t D,
const fp_t dt )
```

Vector addition algorithm for execution on the GPU.

Diffusion equation kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

Definition at line 104 of file cuda_discretization.cu.

### 4.6.3 Variable Documentation

#### 4.6.3.1 d_mask

```
fp_t d_mask[MAX_MASK_W *MAX_MASK_H]
```

Convolution mask array on the GPU, allocated in protected memory.

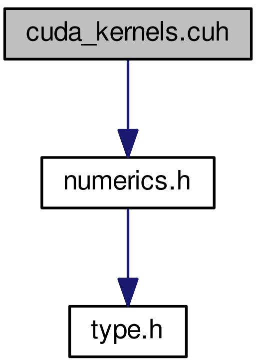
Definition at line 41 of file cuda_discretization.cu.

## 4.7 cuda_kernels.cuh File Reference

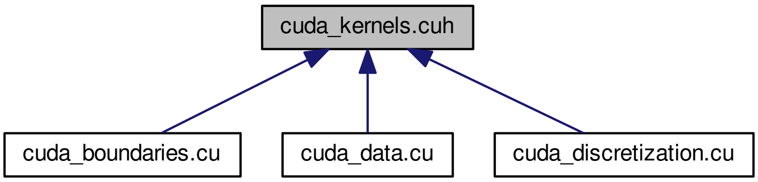
Declaration of functions to execute on the GPU (CUDA kernels)

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

Include dependency graph for cuda_kernels.cuh:



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




### Macros

- #define TILE_W 32

  *Width of an input tile, including halo cells, for GPU memory allocation.*
- #define TILE_H 32

  *Height of an input tile, including halo cells, for GPU memory allocation.*

### Functions

- void boundary_kernel (fp_t ∗conc, const int nx, const int ny, const int nm)

  *Boundary condition kernel for execution on the GPU.*
- void convolution_kernel (fp_t ∗conc_old, fp_t ∗conc_lap, const int nx, const int ny, const int nm)

  *Tiled convolution algorithm for execution on the GPU.*
- void diffusion_kernel (fp_t ∗conc_old, fp_t ∗conc_new, fp_t ∗conc_lap, const int nx, const int ny, const int nm, const fp_t D, const fp_t dt)

  *Vector addition algorithm for execution on the GPU.*

### Variables

- fp_t d_mask [MAX_MASK_W ∗MAX_MASK_H]

  *Convolution mask array on the GPU, allocated in protected memory.*
- fp_t d_bc [2][2]

  *Boundary condition array on the GPU, allocated in protected memory.*

### 4.7.1 Detailed Description

Declaration of functions to execute on the GPU (CUDA kernels)

### 4.7.2 Macro Definition Documentation

#### 4.7.2.1 TILE_H

```
#define TILE_H 32
```

Height of an input tile, including halo cells, for GPU memory allocation.

Definition at line 42 of file cuda_kernels.cuh.

#### 4.7.2.2 TILE_W

```
#define TILE_W 32
```

Width of an input tile, including halo cells, for GPU memory allocation.

Definition at line 37 of file cuda_kernels.cuh.

### 4.7.3 Function Documentation

#### 4.7.3.1 boundary_kernel()

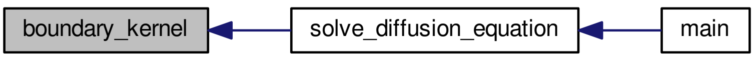
```
void boundary_kernel (
            fp_t * conc,
            const int nx,
            const int ny,
            const int nm )
```

Boundary condition kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

Definition at line 68 of file cuda_boundaries.cu.

Here is the caller graph for this function:



#### 4.7.3.2 convolution_kernel()

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

Tiled convolution algorithm for execution on the GPU.

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

Note:

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

Definition at line 43 of file cuda_discretization.cu.

#### 4.7.3.3 diffusion_kernel()

```
void diffusion_kernel (
            fp_t * conc_old,
            fp_t * conc_new,
            fp_t * conc_lap,
            const int nx,
            const int ny,
            const int nm,
            const fp_t D,
            const 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

Definition at line 104 of file cuda_discretization.cu.

### 4.7.4 Variable Documentation

#### 4.7.4.1 d_bc

```
fp_t d_bc[2][2]
```

Boundary condition array on the GPU, allocated in protected memory.

Definition at line 34 of file cuda_boundaries.cu.

#### 4.7.4.2 d_mask

```
fp_t d_mask[MAX_MASK_W *MAX_MASK_H]
```

Convolution mask array on the GPU, allocated in protected memory.

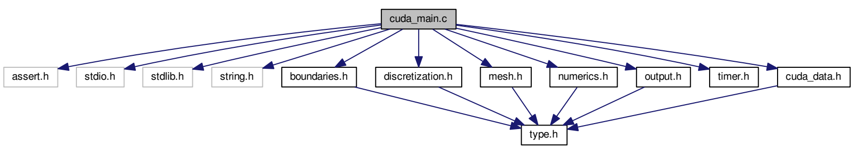
Definition at line 41 of file cuda_discretization.cu.

## 4.8 cuda_main.c File Reference

CUDA implementation of semi-infinite diffusion equation.

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

Include dependency graph for cuda_main.c:




### Functions

- int main (int argc, char *argv[ ])
  *Run simulation using input parameters specified on the command line.*

### 4.8.1 Detailed Description

CUDA implementation of semi-infinite diffusion equation.

### 4.8.2 Function Documentation

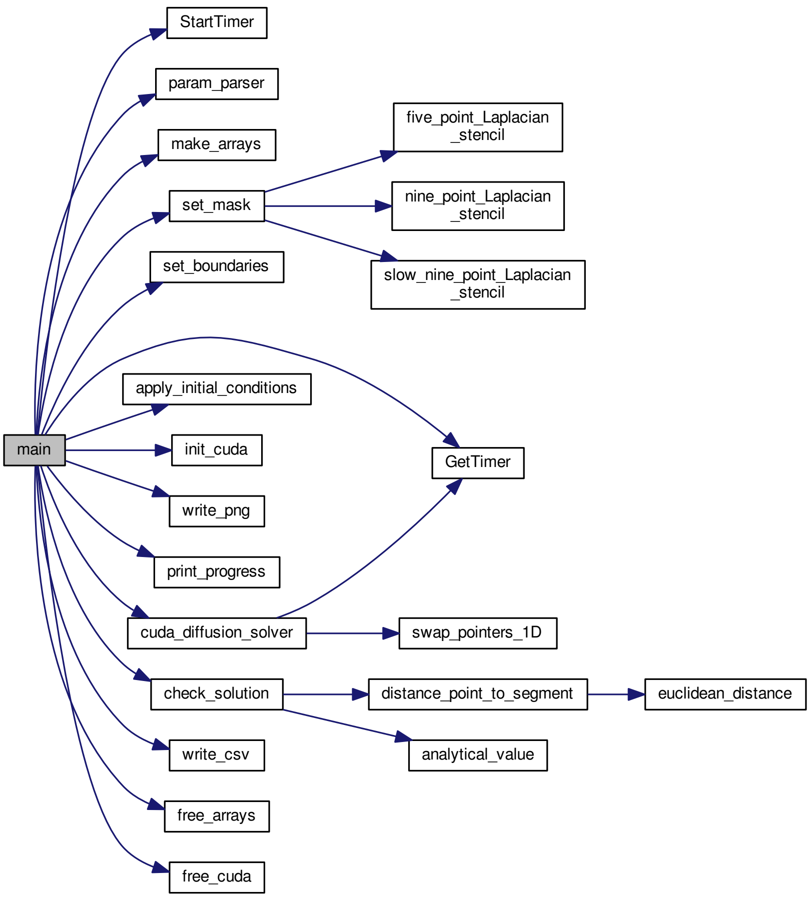
#### 4.8.2.1 main()

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

Run simulation using input parameters specified on the command line.

Definition at line 45 of file cuda_main.c.

Here is the call graph for this function:

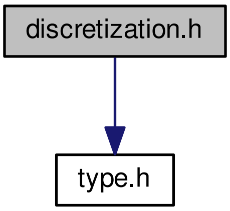



## 4.9 discretization.h File Reference

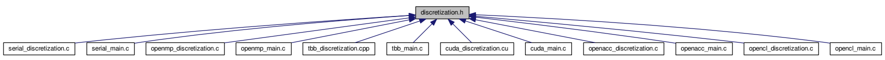
Declaration of discretized mathematical function prototypes.

`#include "type.h"`

Include dependency graph for discretization.h:




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




### Functions

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

  *Perform the convolution of the mask matrix with the composition matrix.*
- void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t *elapsed, struct Stopwatch *sw)

  *Update the scalar composition field using old and Laplacian values.*
- void check_solution (fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t *rss)

  *Compare numerical and analytical solutions of the diffusion equation.*

### 4.9.1 Detailed Description

Declaration of discretized mathematical function prototypes.

### 4.9.2 Function Documentation

#### 4.9.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

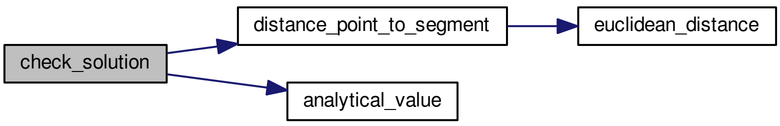
**Returns**

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

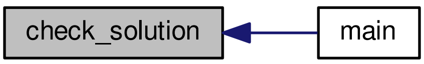
Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 78 of file serial_discretization.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 4.9.2.2 compute_convolution()

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

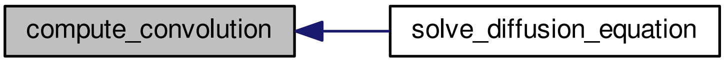
Perform the convolution of the mask matrix with the composition matrix.

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

Definition at line 32 of file serial_discretization.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 4.9.2.3 solve_diffusion_equation()

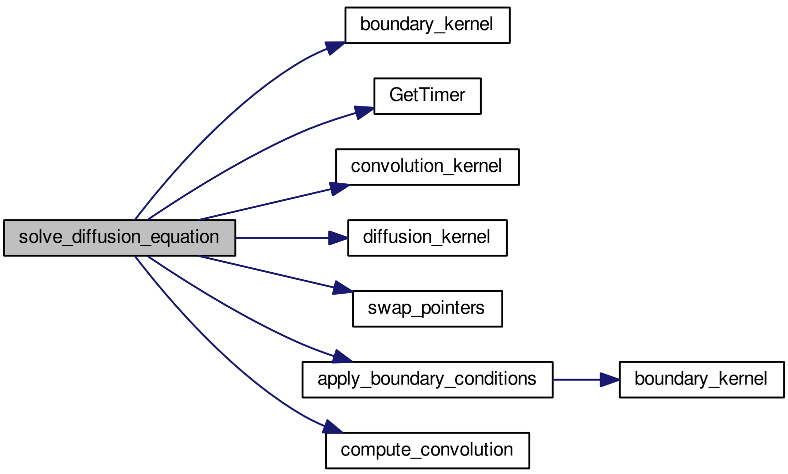
```
void solve_diffusion_equation (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
```

```
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

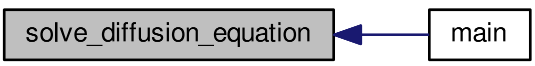
Update the scalar composition field using old and Laplacian values.

Definition at line 51 of file serial_discretization.c.

Here is the call graph for this function:




Here is the caller graph for this function:




## 4.10 kernel_boundary.cl File Reference

### Functions

- __kernel void boundary_kernel (__global fp_t *d_conc, __global fp_t d_bc[2][2], int nx, int ny, int nm)

### 4.10.1 Function Documentation

#### 4.10.1.1 boundary_kernel()

```
__kernel void boundary_kernel (
            __global fp_t * d_conc,
            __global fp_t d_bc[2][2],
            int nx,
            int ny,
            int nm )
```

Definition at line 27 of file kernel_boundary.cl.

## 4.11 kernel_convolution.cl File Reference

### Functions

- __kernel void convolution_kernel (__global fp_t ∗d_conc_old, __global fp_t ∗d_conc_lap, __constant fp_t ∗∗d_mask, int nx, int ny, int nm)

  *Tiled convolution algorithm for execution on the GPU.*

### 4.11.1 Function Documentation

#### 4.11.1.1 convolution_kernel()

```
__kernel void convolution_kernel (
            __global fp_t * d_conc_old,
            __global fp_t * d_conc_lap,
            __constant fp_t ** d_mask,
            int nx,
            int ny,
            int nm )
```

Tiled convolution algorithm for execution on the GPU.

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

Note:

- The source matrix (*d_conc_old*) and destination matrix (*d_conc_lap*) must be identical in size
- One OpenCL worker operates on one array index: there is no nested loop over matrix elements
- The halo (*nm/2* perimeter cells) in *d_conc_lap* are unallocated garbage
- The same cells in *d_conc_old* are boundary values, and contribute to the convolution
- *d_conc_tile* is the shared tile of input data, accessible by all threads in this block
- The *__local* specifier allocates the small *d_conc_tile* array in cache
- The *__constant* specifier allocates the small *d_mask* array in cache

Definition at line 40 of file kernel_convolution.cl.

## 4.12 kernel_diffusion.cl File Reference

### Functions

- __kernel void diffusion_kernel (__global fp_t *d_conc_old, __global fp_t *d_conc_new, __global fp_t *d_↩
conc_lap, int nx, int ny, int nm, fp_t D, fp_t dt)
  *Diffusion equation kernel for execution on the GPU.*

### 4.12.1 Function Documentation

#### 4.12.1.1 diffusion_kernel()

```
__kernel void diffusion_kernel (
            __global fp_t * d_conc_old,
            __global fp_t * d_conc_new,
            __global fp_t * d_conc_lap,
            int nx,
            int ny,
            int nm,
            fp_t D,
            fp_t dt )
```

Diffusion equation kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

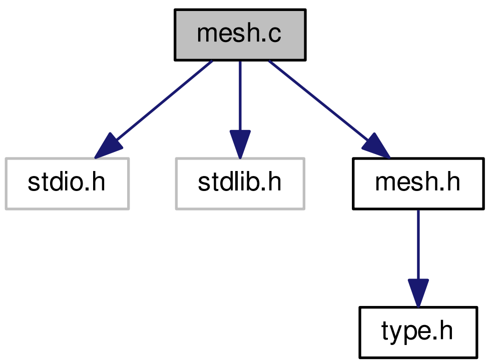
Definition at line 26 of file kernel_diffusion.cl.

## 4.13 mesh.c File Reference

Implemenatation of mesh handling functions for diffusion benchmarks.

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

Include dependency graph for mesh.c:



#### Functions

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

  *Allocate 2D arrays to store scalar composition values.*
- void free_arrays (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap)

  *Free dynamically allocated memory.*
- void swap_pointers (fp_t ***conc_old, fp_t ***conc_new)

  *Swap pointers to 2D arrays.*
- void swap_pointers_1D (fp_t **conc_old, fp_t **conc_new)

  *Swap pointers to data underlying 1D arrays.*

### 4.13.1 Detailed Description

Implemenatation of mesh handling functions for diffusion benchmarks.

### 4.13.2 Function Documentation

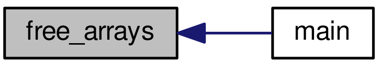
#### 4.13.2.1 free_arrays()

```
void free_arrays (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap )
```

Free dynamically allocated memory.

Definition at line 58 of file mesh.c.

Here is the caller graph for this function:



#### 4.13.2.2 make_arrays()

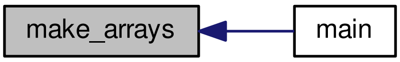
```
void make_arrays (
            fp_t *** conc_old,
            fp_t *** conc_new,
            fp_t *** conc_lap,
            fp_t *** mask_lap,
            int nx,
            int ny,
            int nm )
```

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 29 of file mesh.c.

Here is the caller graph for this function:




#### 4.13.2.3 swap_pointers()

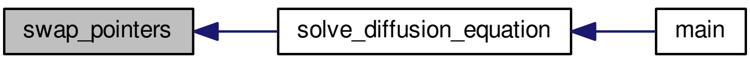
```
void swap_pointers (
            fp_t *** conc_old,
            fp_t *** conc_new )
```

Swap pointers to 2D arrays.

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

Definition at line 73 of file mesh.c.

Here is the caller graph for this function:



#### 4.13.2.4 swap_pointers_1D()

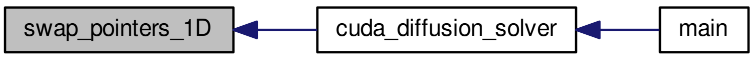
```
void swap_pointers_1D (
            fp_t ** conc_old,
            fp_t ** conc_new )
```

Swap pointers to data underlying 1D arrays.

Rather than copy data from `fp_t*` *conc_old*[0] into `fp_t*` *conc_new*[0], an expensive operation, simply trade the top-most pointers. New becomes old, old becomes new, with no data lost and in almost no time.

Definition at line 82 of file mesh.c.

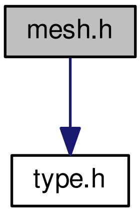
Here is the caller graph for this function:



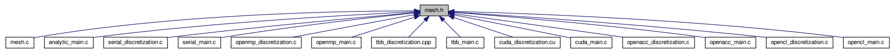

## 4.14 mesh.h File Reference

Declaration of mesh function prototypes for diffusion benchmarks.

`#include "type.h"`
Include dependency graph for mesh.h:




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



#### Functions

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

    *Allocate 2D arrays to store scalar composition values.*
- void free_arrays (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap)

    *Free dynamically allocated memory.*
- void swap_pointers (fp_t ***conc_old, fp_t ***conc_new)

    *Swap pointers to 2D arrays.*
- void swap_pointers_1D (fp_t **conc_old, fp_t **conc_new)

    *Swap pointers to data underlying 1D arrays.*

### 4.14.1 Detailed Description

Declaration of mesh function prototypes for diffusion benchmarks.

### 4.14.2 Function Documentation

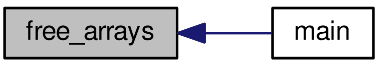
#### 4.14.2.1 free_arrays()

```
void free_arrays (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap )
```

Free dynamically allocated memory.

Definition at line 58 of file mesh.c.

Here is the caller graph for this function:



#### 4.14.2.2 make_arrays()

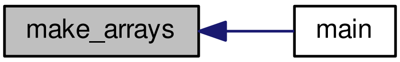
```
void make_arrays (
            fp_t *** conc_old,
            fp_t *** conc_new,
            fp_t *** conc_lap,
            fp_t *** mask_lap,
            int nx,
            int ny,
            int nm )
```

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 29 of file mesh.c.

Here is the caller graph for this function:




#### 4.14.2.3 swap_pointers()

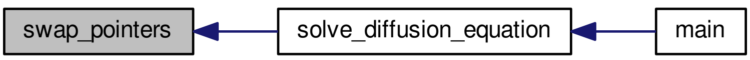
```
void swap_pointers (
            fp_t *** conc_old,
            fp_t *** conc_new )
```

Swap pointers to 2D arrays.

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

Definition at line 73 of file mesh.c.

Here is the caller graph for this function:



#### 4.14.2.4 swap_pointers_1D()

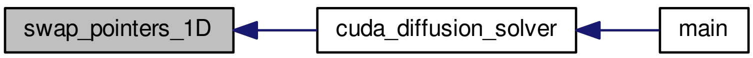
```
void swap_pointers_1D (
            fp_t ** conc_old,
            fp_t ** conc_new )
```

Swap pointers to data underlying 1D arrays.

Rather than copy data from `fp_t*` *conc_old*[0] into `fp_t*` *conc_new*[0], an expensive operation, simply trade the top-most pointers. New becomes old, old becomes new, with no data lost and in almost no time.

Definition at line 82 of file mesh.c.

Here is the caller graph for this function:

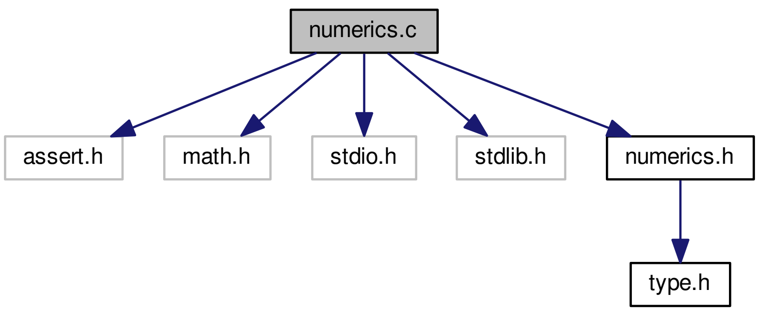



## 4.15 numerics.c File Reference

Implementation of Laplacian operator and analytical solution functions.

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

Include dependency graph for numerics.c:



### Functions

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

### 4.15.1 Detailed Description

Implementation of Laplacian operator and analytical solution functions.

### 4.15.2 Function Documentation

#### 4.15.2.1 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 $c_\infty$ and boundary value $c(x = 0, t) = c_0$ with constant diffusivity *D*, the solution to Fick's second law is

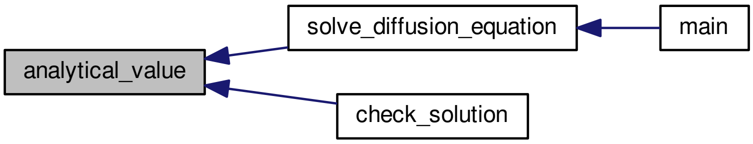
$$c(x,t) = c_0 - (c_0 - c_\infty)\mathrm{erf}\left(\frac{x}{\sqrt{4Dt}}\right)$$

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 121 of file numerics.c.

Here is the caller graph for this function:




#### 4.15.2.2 distance_point_to_segment()

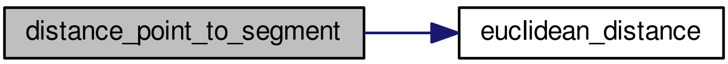
```
fp_t distance_point_to_segment (
            fp_t ax,
            fp_t ay,
            fp_t bx,
            fp_t by,
            fp_t px,
            fp_t py )
```

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

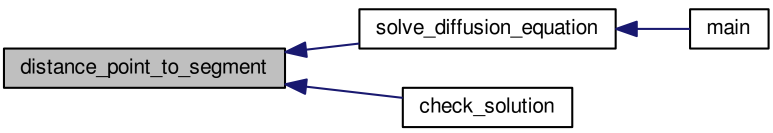
This function computes the projection of *p* onto *ab*, limiting the projected range to [0, 1] to handle projections that fall outside of *ab*. Implemented after Grumdrig on Stackoverflow, https://stackoverflow.com/a/1501725.

Definition at line 108 of file numerics.c.

Here is the call graph for this function:




Here is the caller graph for this function:



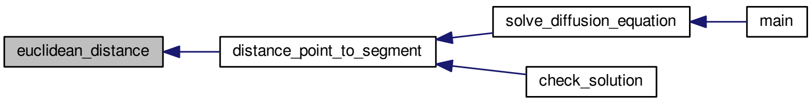
#### 4.15.2.3 euclidean_distance()

```
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*.

Definition at line 98 of file numerics.c.

Here is the caller graph for this function:




#### 4.15.2.4 five_point_Laplacian_stencil()

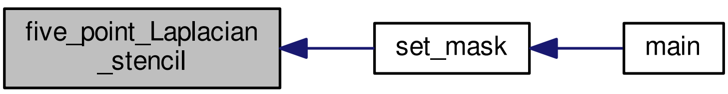
```
void five_point_Laplacian_stencil (
            fp_t dx,
            fp_t dy,
            fp_t ** mask_lap,
            int nm )
```

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 51 of file numerics.c.

Here is the caller graph for this function:



#### 4.15.2.5 manhattan_distance()

```
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*.

Definition at line 103 of file numerics.c.

#### 4.15.2.6 nine_point_Laplacian_stencil()

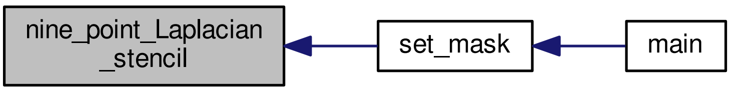
```
void nine_point_Laplacian_stencil (
            fp_t dx,
            fp_t dy,
            fp_t ** mask_lap,
            int nm )
```

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 62 of file numerics.c.

Here is the caller graph for this function:




#### 4.15.2.7 set_mask()

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

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

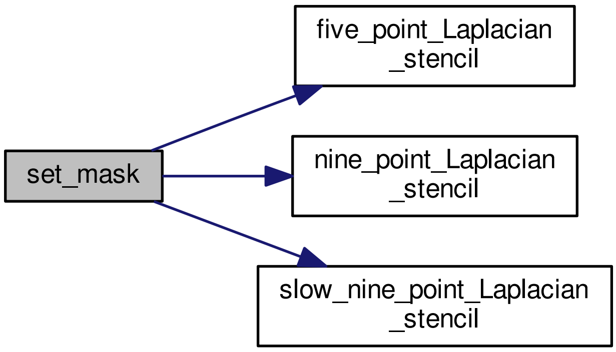
The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, *e.g.* 53 specifies five_point_Laplacian_stencil(), while 93 specifies nine_point_Laplacian_stencil().

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

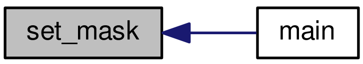
If your stencil is larger than $5 \times 5$, you must increase the values defined by MAX_MASK_W and MAX_MASK_H.

Definition at line 31 of file numerics.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 4.15.2.8 slow_nine_point_Laplacian_stencil()

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

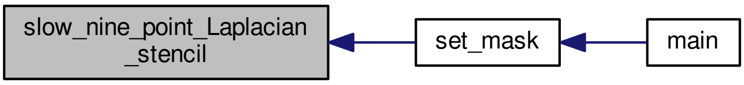
Write 9-point Laplacian stencil into convolution mask.

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

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

Definition at line 79 of file numerics.c.

Here is the caller graph for this function:

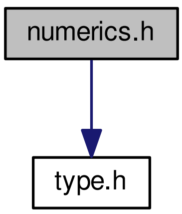



## 4.16 numerics.h File Reference

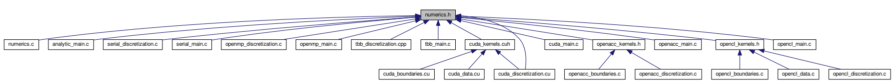
Declaration of Laplacian operator and analytical solution functions.

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

Include dependency graph for numerics.h:




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



### Macros

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

### Functions

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

### 4.16.1 Detailed Description

Declaration of Laplacian operator and analytical solution functions.

### 4.16.2 Macro Definition Documentation

#### 4.16.2.1 MAX_MASK_H

```
#define MAX_MASK_H 5
```

Maximum height of the convolution mask (Laplacian stencil) array.

Definition at line 40 of file numerics.h.

#### 4.16.2.2 MAX_MASK_W

```
#define MAX_MASK_W 5
```

Maximum width of the convolution mask (Laplacian stencil) array.

Definition at line 35 of file numerics.h.

### 4.16.3 Function Documentation

#### 4.16.3.1 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 $c_{\infty}$ and boundary value $c(x = 0, t) = c_0$ with constant diffusivity *D*, the solution to Fick's second law is

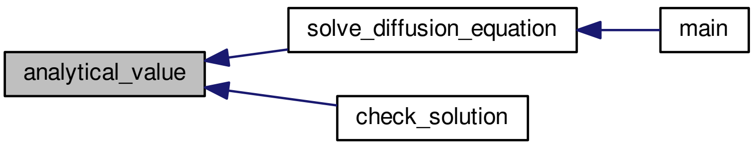
$$c(x,t) = c_0 - (c_0 - c_{\infty})\mathrm{erf}\left(\frac{x}{\sqrt{4Dt}}\right)$$

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 121 of file numerics.c.

Here is the caller graph for this function:




#### 4.16.3.2 distance_point_to_segment()

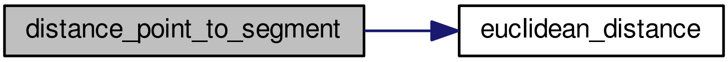
```
fp_t distance_point_to_segment (
            fp_t ax,
            fp_t ay,
            fp_t bx,
            fp_t by,
            fp_t px,
            fp_t py )
```

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

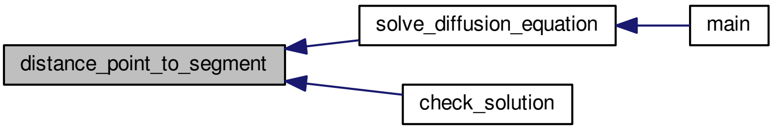
This function computes the projection of *p* onto *ab*, limiting the projected range to [0, 1] to handle projections that fall outside of *ab*. Implemented after Grumdrig on Stackoverflow, https://stackoverflow.com/a/1501725.

Definition at line 108 of file numerics.c.

Here is the call graph for this function:




Here is the caller graph for this function:



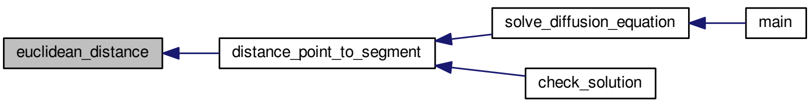

#### 4.16.3.3 euclidean_distance()

```
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*.

Definition at line 98 of file numerics.c.

Here is the caller graph for this function:



#### 4.16.3.4 five_point_Laplacian_stencil()

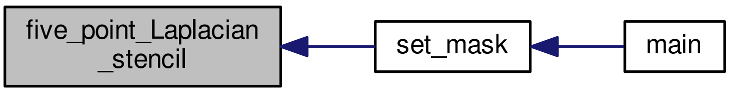
```
void five_point_Laplacian_stencil (
            fp_t dx,
            fp_t dy,
            fp_t ** mask_lap,
            int nm )
```

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 51 of file numerics.c.

Here is the caller graph for this function:




#### 4.16.3.5 manhattan_distance()

```
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*.

Definition at line 103 of file numerics.c.

#### 4.16.3.6 nine_point_Laplacian_stencil()

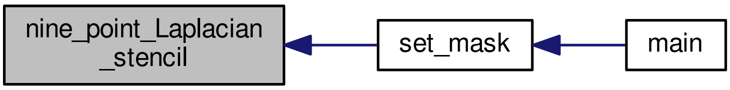
```
void nine_point_Laplacian_stencil (
            fp_t dx,
            fp_t dy,
            fp_t ** mask_lap,
            int nm )
```

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 62 of file numerics.c.

Here is the caller graph for this function:




#### 4.16.3.7 set_mask()

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

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

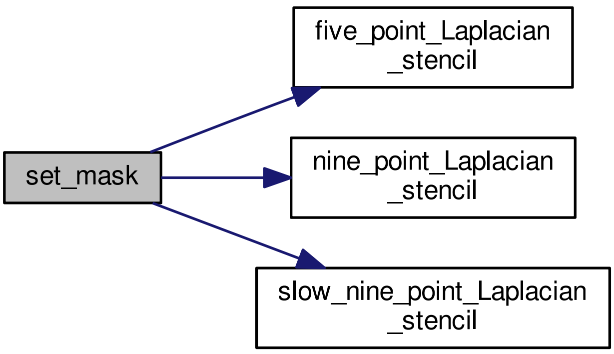
The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, *e.g.* 53 specifies five_point_Laplacian_stencil(), while 93 specifies nine_point_Laplacian_stencil().

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

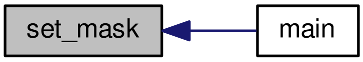
If your stencil is larger than $5 \times 5$, you must increase the values defined by MAX_MASK_W and MAX_MASK_H.

Definition at line 31 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 4.16.3.8 slow_nine_point_Laplacian_stencil()

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

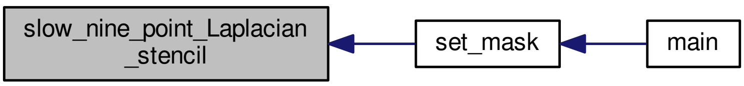
Write 9-point Laplacian stencil into convolution mask.

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

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

Definition at line 79 of file numerics.c.

Here is the caller graph for this function:



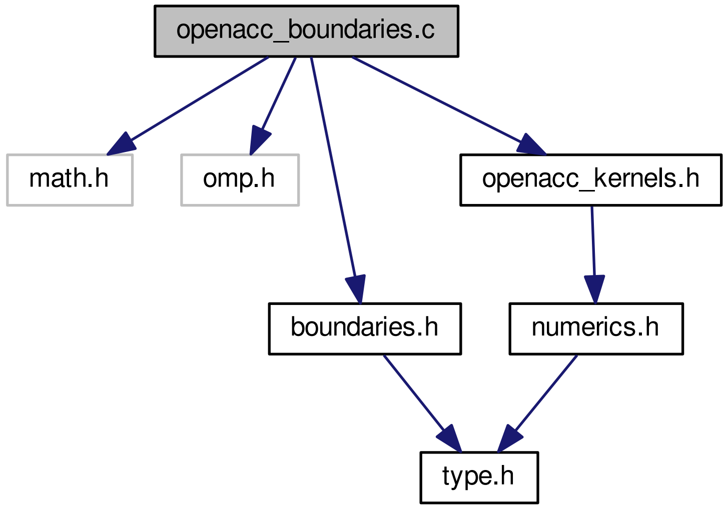

## 4.17 openacc_boundaries.c File Reference

Implementation of boundary condition functions with OpenMP threading.

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

```
#include "openacc_kernels.h"
```

Include dependency graph for openacc_boundaries.c:




### Functions

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

  *Set values to be used along the simulation domain boundaries.*

- void apply_initial_conditions (fp_t $**$conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Initialize flat composition field with fixed boundary conditions.*

- void boundary_kernel (fp_t $**$conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Boundary condition kernel for execution on the GPU.*

- void apply_boundary_conditions (fp_t $**$conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Set fixed value* $(c_{hi})$ *along left and bottom, zero-flux elsewhere.*

#### 4.17.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

#### 4.17.2 Function Documentation

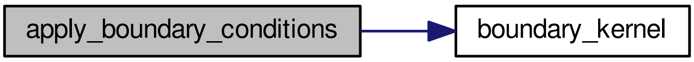
#### 4.17.2.1 apply_boundary_conditions()

```
void apply_boundary_conditions (
            fp_t ** conc,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2] )
```

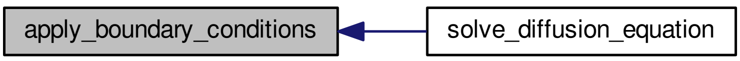
Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.

Definition at line 108 of file openacc_boundaries.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 4.17.2.2 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 39 of file openacc_boundaries.c.

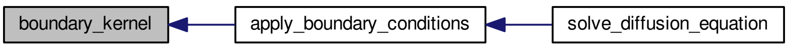
#### 4.17.2.3 boundary_kernel()

```
void boundary_kernel (
            fp_t ** conc,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2] )
```

Boundary condition kernel for execution on the GPU.

Definition at line 62 of file openacc_boundaries.c.

Here is the caller graph for this function:




#### 4.17.2.4 set_boundaries()

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

Set values to be used along the simulation domain boundaries.

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

Definition at line 30 of file openacc_boundaries.c.

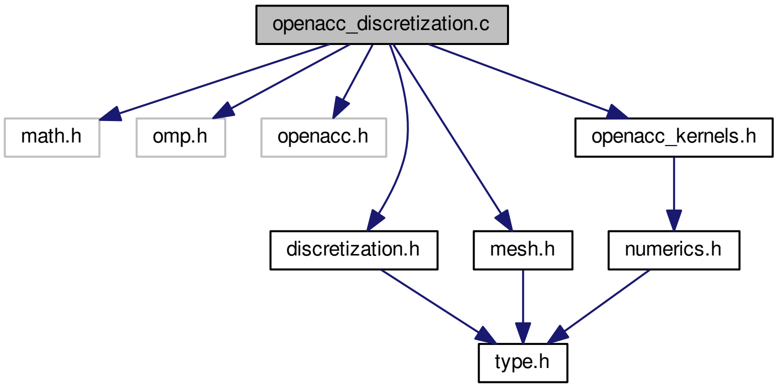
## 4.18 openacc_discretization.c File Reference

Implementation of boundary condition functions with OpenACC threading.

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

```
#include "openacc_kernels.h"
```

Include dependency graph for openacc_discretization.c:




### Functions

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

  *Tiled convolution algorithm for execution on the GPU.*
- 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.*
- void compute_convolution (fp_t **conc_old, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm)

  *Perform the convolution of the mask matrix with the composition matrix.*
- void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t *elapsed, struct Stopwatch *sw)

  *Update the scalar composition field using old and Laplacian values.*
- void check_solution (fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t *rss)

  *Compare numerical and analytical solutions of the diffusion equation.*

### 4.18.1 Detailed Description

Implementation of boundary condition functions with OpenACC threading.

### 4.18.2 Function Documentation

#### 4.18.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

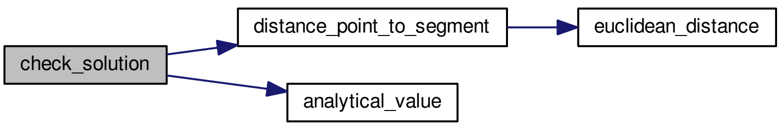
**Returns**

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

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

Definition at line 106 of file openacc_discretization.c.

Here is the call graph for this function:




#### 4.18.2.2 compute_convolution()

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

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

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

Definition at line 68 of file openacc_discretization.c.

Here is the call graph for this function:

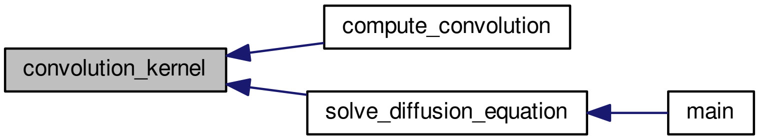

#### 4.18.2.3 convolution_kernel()

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

Tiled convolution algorithm for execution on the GPU.

Definition at line 32 of file openacc_discretization.c.

Here is the caller graph for this function:




#### 4.18.2.4 diffusion_kernel()

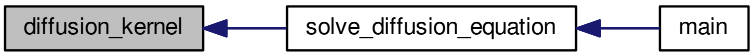
```
void diffusion_kernel (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            int nx,
            int ny,
            int nm,
```

```
            fp_t D,
            fp_t dt )
```

Vector addition algorithm for execution on the GPU.

Definition at line 52 of file openacc_discretization.c.

Here is the caller graph for this function:



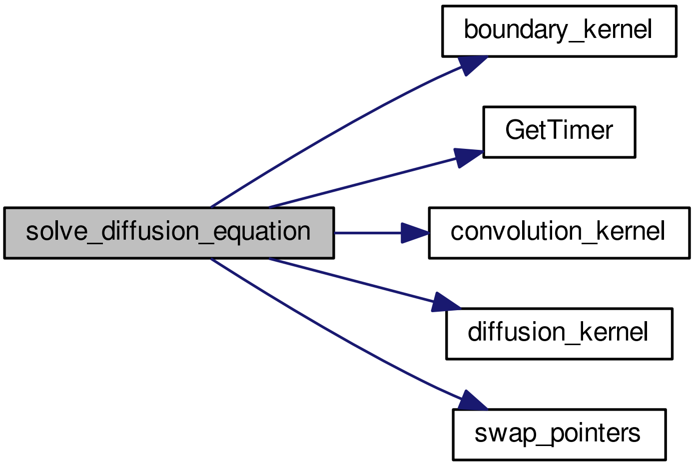

#### 4.18.2.5 solve_diffusion_equation()

```
void solve_diffusion_equation (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

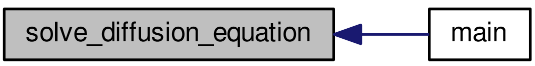
Update the scalar composition field using old and Laplacian values.

Definition at line 78 of file openacc_discretization.c.

Here is the call graph for this function:




Here is the caller graph for this function:

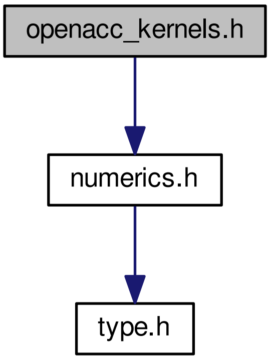



## 4.19 openacc_kernels.h File Reference

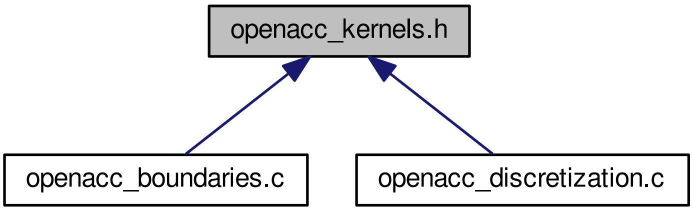
Declaration of functions to execute on the GPU (OpenACC kernels)

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

Include dependency graph for openacc_kernels.h:




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




### Functions

- void boundary_kernel (fp_t **conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Boundary condition kernel for execution on the GPU.*
- void convolution_kernel (fp_t **conc_old, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm)

  *Tiled convolution algorithm for execution on the GPU.*
- 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.*

#### 4.19.1 Detailed Description

Declaration of functions to execute on the GPU (OpenACC kernels)

### 4.19.2 Function Documentation

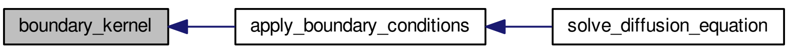
#### 4.19.2.1 boundary_kernel()

```
void boundary_kernel (
            fp_t ** conc,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2] )
```

Boundary condition kernel for execution on the GPU.

Definition at line 62 of file openacc_boundaries.c.

Here is the caller graph for this function:



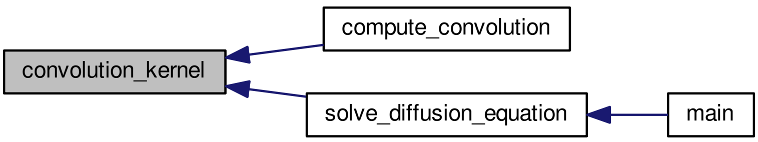

#### 4.19.2.2 convolution_kernel()

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

Tiled convolution algorithm for execution on the GPU.

Definition at line 32 of file openacc_discretization.c.

Here is the caller graph for this function:



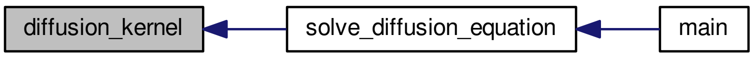
#### 4.19.2.3 diffusion_kernel()

```
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.

Definition at line 52 of file openacc_discretization.c.

Here is the caller graph for this function:

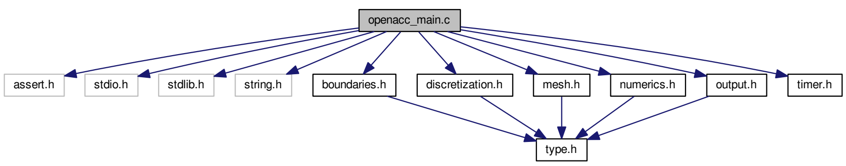



## 4.20 openacc_main.c File Reference

OpenACC implementation of semi-infinite diffusion equation.

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

Include dependency graph for openacc_main.c:




### Functions

- int main (int argc, char ∗argv[ ])

    *Run simulation using input parameters specified on the command line.*

### 4.20.1 Detailed Description

OpenACC implementation of semi-infinite diffusion equation.

### 4.20.2 Function Documentation

#### 4.20.2.1 main()

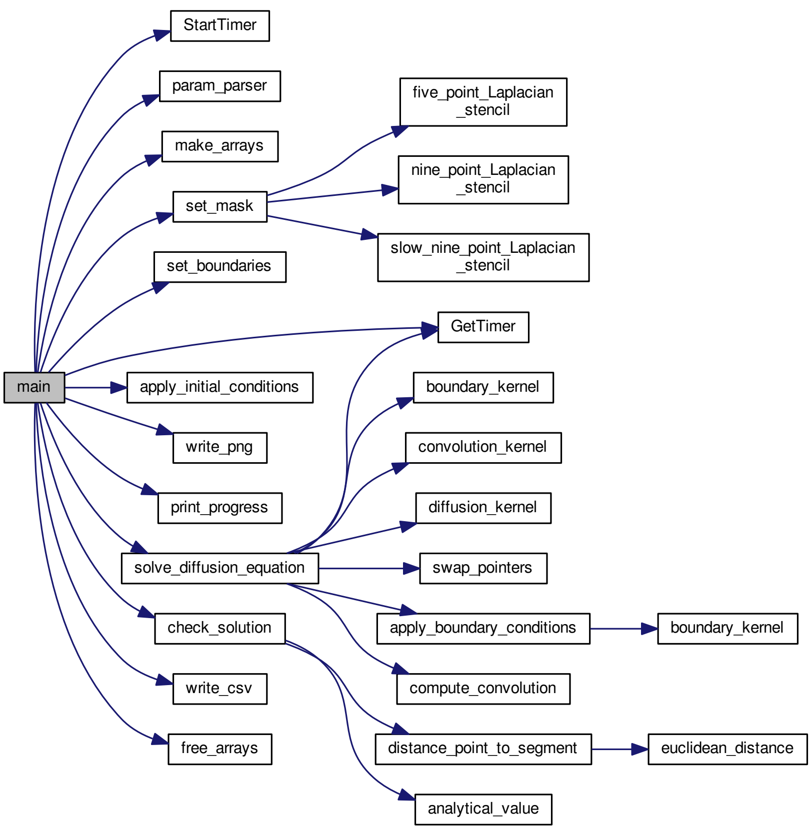
```
int main (
            int argc,
            char * argv[] )
```

Run simulation using input parameters specified on the command line.

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

Definition at line 48 of file openacc_main.c.

Here is the call graph for this function:

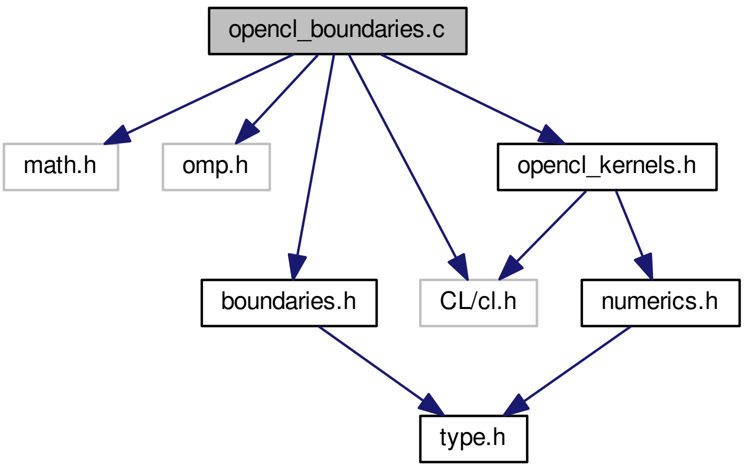


## 4.21 opencl_boundaries.c File Reference

Implementation of boundary condition functions with OpenCL acceleration.

```
#include <math.h>
#include <omp.h>
#include <CL/cl.h>
#include "boundaries.h"
#include "opencl_kernels.h"
```

Include dependency graph for opencl_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.*

### 4.21.1 Detailed Description

Implementation of boundary condition functions with OpenCL acceleration.

### 4.21.2 Function Documentation

#### 4.21.2.1 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 42 of file opencl_boundaries.c.

Here is the caller graph for this function:


#### 4.21.2.2 set_boundaries()

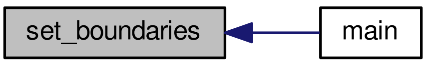
```
void set_boundaries (
            fp_t bc[2][2] )
```

Set values to be used along the simulation domain boundaries.

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

Definition at line 33 of file opencl_boundaries.c.

Here is the caller graph for this function:

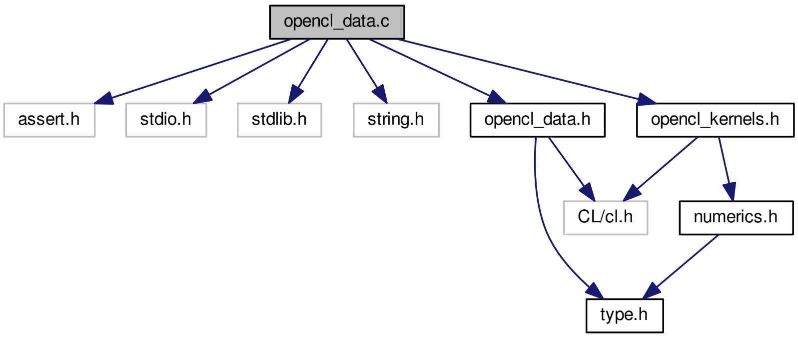


## 4.22 opencl_data.c File Reference

Implementation of functions to create and destroy OpenCLData struct.

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "opencl_data.h"
#include "opencl_kernels.h"
```

Include dependency graph for opencl_data.c:




### Functions

- void report_error (cl_int status, const char ∗message)

    *Report error code when status is not `CL_SUCCESS`.*
- void init_opencl (fp_t ∗∗conc_old, fp_t ∗∗mask_lap, fp_t bc[2][2], int nx, int ny, int nm, struct OpenCLData ∗dev)

    *Initialize OpenCL device memory before marching.*
- void build_program (const char ∗filename, cl_context context, cl_device_id gpu, cl_program program, cl_int ∗status)

    *Build kernel program from text input.*
- void free_opencl (struct OpenCLData ∗dev)

    *Free OpenCL device memory after marching.*

### 4.22.1 Detailed Description

Implementation of functions to create and destroy OpenCLData struct.

### 4.22.2 Function Documentation

#### 4.22.2.1 build_program()

```
void build_program (
            const char * filename,
            cl_context context,
            cl_device_id gpu,
            cl_program program,
            cl_int * status )
```

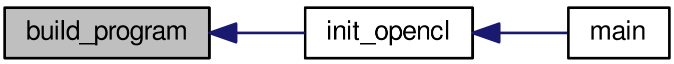
Build kernel program from text input.

Source follows the OpenCL Programming Book, https://www.fixstars.com/en/opencl/book/OpenCLProgrammingBook/calling-the-kernel/

Definition at line 123 of file opencl_data.c.

Here is the call graph for this function:


Here is the caller graph for this function:



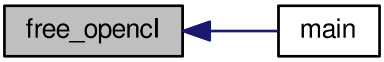

#### 4.22.2.2 free_opencl()

```
void free_opencl (
            struct OpenCLData * dev )
```

Free OpenCL device memory after marching.

Definition at line 160 of file opencl_data.c.

Here is the caller graph for this function:



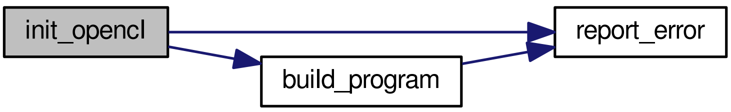

#### 4.22.2.3 init_opencl()

```
void init_opencl (
            fp_t ** conc_old,
            fp_t ** mask_lap,
            fp_t bc[2][2],
            int nx,
            int ny,
            int nm,
            struct OpenCLData * dev )
```

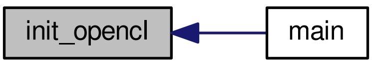
Initialize OpenCL device memory before marching.

Definition at line 52 of file opencl_data.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 4.22.2.4 report_error()

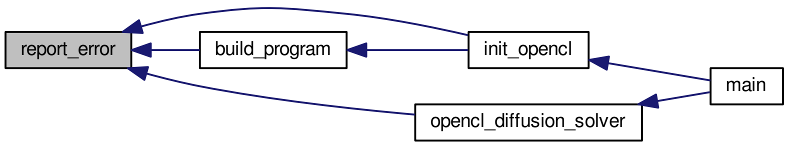
```
void report_error (
            cl_int error,
            const char * message )
```

Report error code when status is not `CL_SUCCESS`.

Refer to https://streamhpc.com/blog/2013-04-28/opencl-error-codes/ for help interpreting error codes.

Definition at line 33 of file opencl_data.c.

Here is the caller graph for this function:

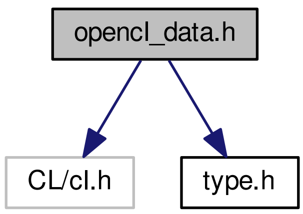



## 4.23 opencl_data.h File Reference

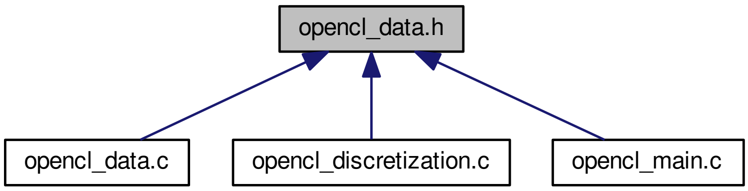
Declaration of OpenCL data container.

```
#include <CL/cl.h>
#include "type.h"
```

Include dependency graph for opencl_data.h:



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




### Classes

- struct OpenCLData

  *Container for GPU array pointers and parameters.*

### Macros

- #define MAX_PLATFORMS 4

  *Greatest number of expected platforms.*
- #define MAX_DEVICES 32

  *Greatest number of expected devices.*

### Functions

- void report_error (cl_int error, const char ∗message)

  *Report error code when status is not `CL_SUCCESS`.*
- void init_opencl (fp_t ∗∗conc_old, fp_t ∗∗mask_lap, fp_t bc[2][2], int nx, int ny, int nm, struct OpenCLData ∗dev)

  *Initialize OpenCL device memory before marching.*
- void opencl_diffusion_solver (struct OpenCLData ∗dev, fp_t ∗∗conc_new, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t ∗elapsed, struct Stopwatch ∗sw)

  *Specialization of solve_diffusion_equation() using OpenCL.*
- void free_opencl (struct OpenCLData ∗dev)

  *Free OpenCL device memory after marching.*

### 4.23.1 Detailed Description

Declaration of OpenCL data container.

### 4.23.2 Macro Definition Documentation

#### 4.23.2.1 MAX_DEVICES

```
#define MAX_DEVICES 32
```

Greatest number of expected devices.

Definition at line 42 of file opencl_data.h.

#### 4.23.2.2 MAX_PLATFORMS

```
#define MAX_PLATFORMS 4
```

Greatest number of expected platforms.

Definition at line 37 of file opencl_data.h.

### 4.23.3 Function Documentation

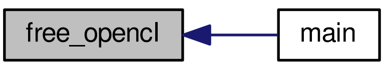
#### 4.23.3.1 free_opencl()

```
void free_opencl (
            struct OpenCLData * dev )
```

Free OpenCL device memory after marching.

Definition at line 160 of file opencl_data.c.

Here is the caller graph for this function:



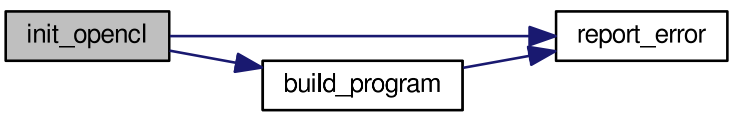
#### 4.23.3.2 init_opencl()

```
void init_opencl (
            fp_t ** conc_old,
            fp_t ** mask_lap,
            fp_t bc[2][2],
            int nx,
            int ny,
            int nm,
            struct OpenCLData * dev )
```

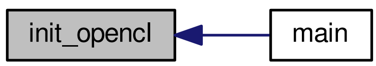
Initialize OpenCL device memory before marching.

Definition at line 52 of file opencl_data.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 4.23.3.3 opencl_diffusion_solver()

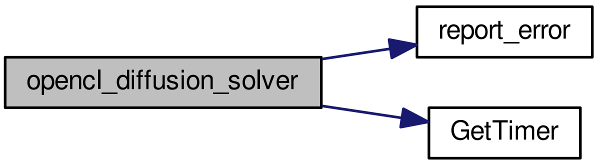
```
void opencl_diffusion_solver (
            struct OpenCLData * dev,
            fp_t ** conc_new,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
```

```
            fp_t * elapsed,
            struct Stopwatch * sw )
```

Specialization of solve_diffusion_equation() using OpenCL.

Definition at line 35 of file opencl_discretization.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 4.23.3.4 report_error()

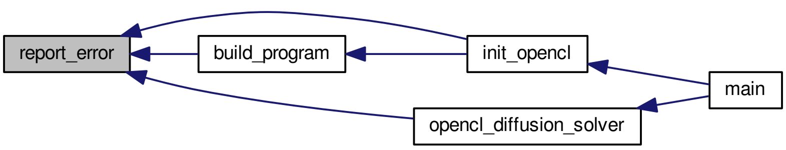
```
void report_error (
            cl_int error,
            const char * message )
```

Report error code when status is not `CL_SUCCESS`.

Refer to `https://streamhpc.com/blog/2013-04-28/opencl-error-codes/` for help interpreting error codes.

Definition at line 33 of file opencl_data.c.

Here is the caller graph for this function:

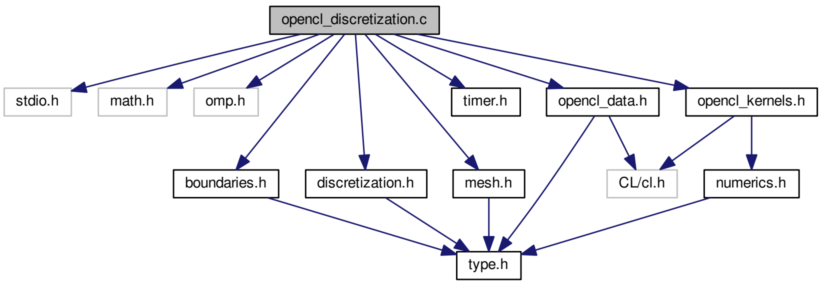


## 4.24 opencl_discretization.c File Reference

Implementation of boundary condition functions with OpenCL acceleration.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include "boundaries.h"
#include "discretization.h"
#include "mesh.h"
#include "timer.h"
#include "opencl_data.h"
#include "opencl_kernels.h"
```

Include dependency graph for opencl_discretization.c:




### Functions

- void opencl_diffusion_solver (struct OpenCLData ∗dev, fp_t ∗∗conc_new, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t ∗elapsed, struct Stopwatch ∗sw)

  *Specialization of solve_diffusion_equation() using OpenCL.*
- void check_solution (fp_t ∗∗conc_new, fp_t ∗∗conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t ∗rss)

  *Compare numerical and analytical solutions of the diffusion equation.*

### 4.24.1 Detailed Description

Implementation of boundary condition functions with OpenCL acceleration.

### 4.24.2 Function Documentation

#### 4.24.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

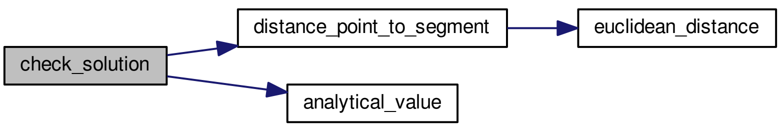
**Returns**

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

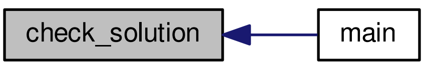
Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 122 of file opencl_discretization.c.

Here is the call graph for this function:




Here is the caller graph for this function:



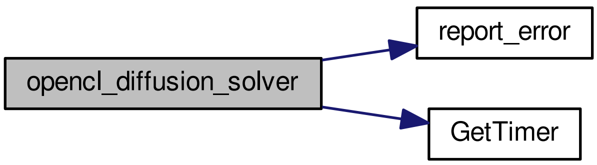
#### 4.24.2.2 opencl_diffusion_solver()

```
void opencl_diffusion_solver (
            struct OpenCLData * dev,
            fp_t ** conc_new,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

Specialization of solve_diffusion_equation() using OpenCL.

Definition at line 35 of file opencl_discretization.c.

Here is the call graph for this function:



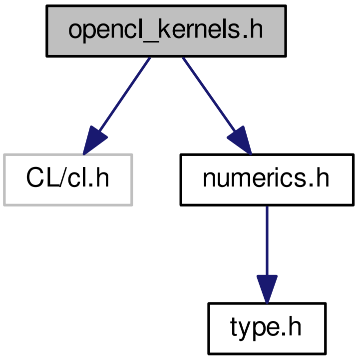

Here is the caller graph for this function:


## 4.25 opencl_kernels.h File Reference

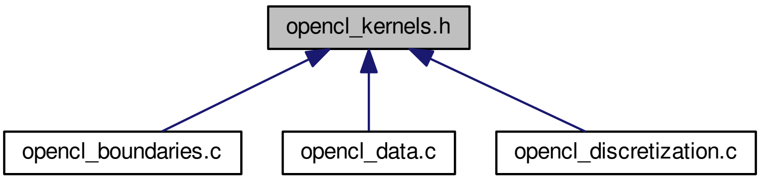
Declaration of functions to execute on the GPU (OpenCL kernels)

```
#include <CL/cl.h>
#include "numerics.h"
```

Include dependency graph for opencl_kernels.h:




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




### Macros

- #define TILE_W 32

  *Width of an input tile, including halo cells, for GPU memory allocation.*
- #define TILE_H 32

  *Height of an input tile, including halo cells, for GPU memory allocation.*

### Functions

- void build_program (const char ∗filename, cl_context context, cl_device_id gpu, cl_program program, cl_int ∗status)

  *Build kernel program from text input.*
- void boundary_kernel (fp_t ∗d_conc, fp_t d_bc[2][2], int nx, int ny, int nm)

  *Boundary condition kernel for execution on the GPU.*
- void convolution_kernel (fp_t ∗d_conc_old, fp_t ∗d_conc_lap, fp_t ∗∗d_mask, int nx, int ny, int nm)

  *Tiled convolution algorithm for execution on the GPU.*
- void diffusion_kernel (fp_t ∗d_conc_old, fp_t ∗d_conc_new, fp_t ∗d_conc_lap, int nx, int ny, int nm, fp_t D, fp_t dt)

  *Diffusion equation kernel for execution on the GPU.*

### 4.25.1 Detailed Description

Declaration of functions to execute on the GPU (OpenCL kernels)

### 4.25.2 Macro Definition Documentation

#### 4.25.2.1 TILE_H

```
#define TILE_H 32
```

Height of an input tile, including halo cells, for GPU memory allocation.

Definition at line 42 of file opencl_kernels.h.

#### 4.25.2.2 TILE_W

```
#define TILE_W 32
```

Width of an input tile, including halo cells, for GPU memory allocation.

Definition at line 37 of file opencl_kernels.h.

### 4.25.3 Function Documentation

#### 4.25.3.1 boundary_kernel()

```
void boundary_kernel (
             fp_t * d_conc,
             fp_t d_bc[2][2],
             int nx,
             int ny,
             int nm )
```

Boundary condition kernel for execution on the GPU.

Boundary condition kernel for execution on the GPU

This function accesses 1D data rather than the 2D array representation of the scalar composition field.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

#### 4.25.3.2 build_program()

```
void build_program (
            const char * filename,
            cl_context context,
            cl_device_id gpu,
            cl_program program,
            cl_int * status )
```

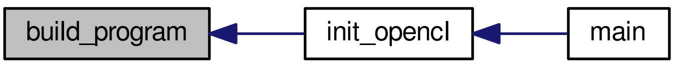
Build kernel program from text input.

Source follows the OpenCL Programming Book, https://www.fixstars.com/en/opencl/book/↩
OpenCLProgrammingBook/calling-the-kernel/

Definition at line 123 of file opencl_data.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 4.25.3.3 convolution_kernel()

```
void convolution_kernel (
            fp_t * d_conc_old,
            fp_t * d_conc_lap,
            fp_t ** d_mask,
            int nx,
            int ny,
            int nm )
```

Tiled convolution algorithm for execution on the GPU.

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

Note:

- The source matrix (*d_conc_old*) and destination matrix (*d_conc_lap*) must be identical in size
- One OpenCL worker operates on one array index: there is no nested loop over matrix elements
- The halo (*nm/2* perimeter cells) in *d_conc_lap* are unallocated garbage
- The same cells in *d_conc_old* are boundary values, and contribute to the convolution
- *d_conc_tile* is the shared tile of input data, accessible by all threads in this block
- The *__local* specifier allocates the small *d_conc_tile* array in cache
- The *__constant* specifier allocates the small *d_mask* array in cache

#### 4.25.3.4 diffusion_kernel()

```
void diffusion_kernel (
            fp_t * d_conc_old,
            fp_t * d_conc_new,
            fp_t * d_conc_lap,
            int nx,
            int ny,
            int nm,
            fp_t D,
            fp_t dt )
```

Diffusion equation kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

Diffusion equation kernel for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field

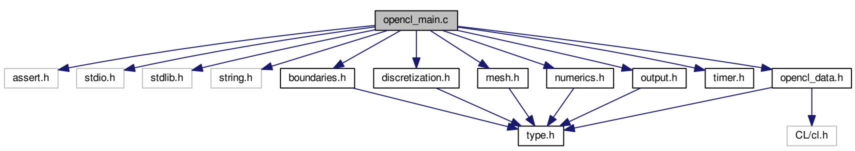
Definition at line 104 of file cuda_discretization.cu.

## 4.26 opencl_main.c File Reference

OpenCL implementation of semi-infinite diffusion equation.

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

Include dependency graph for opencl_main.c:



### Functions

- int main (int argc, char ∗argv[ ])
  *Run simulation using input parameters specified on the command line.*

### 4.26.1 Detailed Description

OpenCL implementation of semi-infinite diffusion equation.

### 4.26.2 Function Documentation

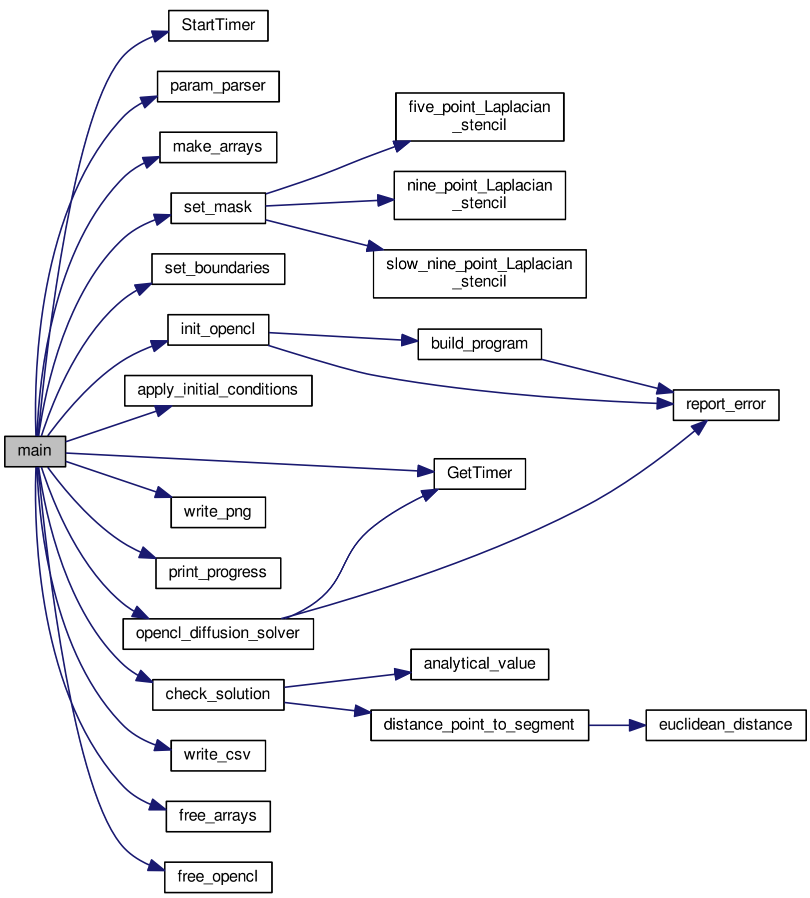
#### 4.26.2.1 main()

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

Run simulation using input parameters specified on the command line.

Definition at line 45 of file opencl_main.c.

Here is the call graph for this function:

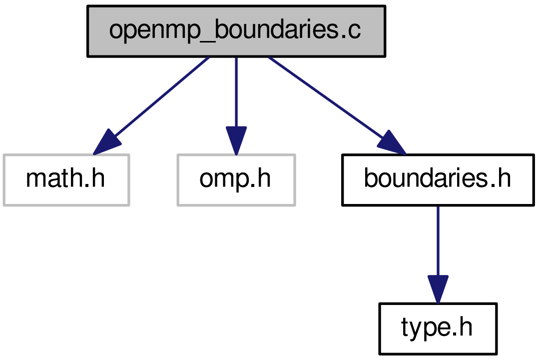


## 4.27 openmp_boundaries.c File Reference

Implementation of boundary condition functions with OpenMP threading.

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

Include dependency graph for openmp_boundaries.c:




### Functions

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

  *Set values to be used along the simulation domain boundaries.*
- void apply_initial_conditions (fp_t **conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Initialize flat composition field with fixed boundary conditions.*
- void apply_boundary_conditions (fp_t **conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.*

### 4.27.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

### 4.27.2 Function Documentation

#### 4.27.2.1 apply_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.

Definition at line 62 of file openmp_boundaries.c.

#### 4.27.2.2 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 39 of file openmp_boundaries.c.

#### 4.27.2.3 set_boundaries()

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

Set values to be used along the simulation domain boundaries.

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

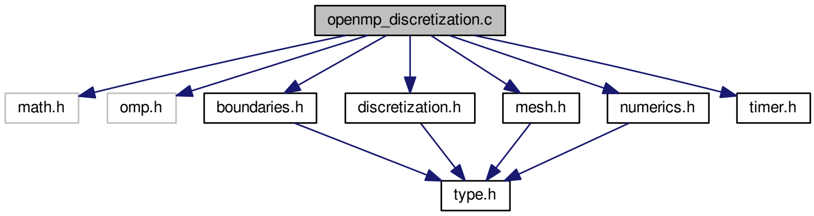
Definition at line 30 of file openmp_boundaries.c.

## 4.28 openmp_discretization.c File Reference

Implementation of boundary condition functions with OpenMP threading.

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

Include dependency graph for openmp_discretization.c:




### Functions

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

  *Perform the convolution of the mask matrix with the composition matrix.*
- void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t *elapsed, struct Stopwatch *sw)

  *Update the scalar composition field using old and Laplacian values.*
- void check_solution (fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t *rss)

  *Compare numerical and analytical solutions of the diffusion equation.*

### 4.28.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

### 4.28.2 Function Documentation

#### 4.28.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

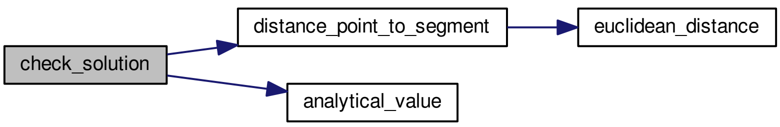
**Returns**

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

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

Definition at line 84 of file openmp_discretization.c.

Here is the call graph for this function:




#### 4.28.2.2 compute_convolution()

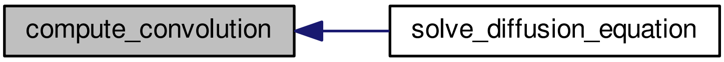
```
void compute_convolution (
            fp_t ** conc_old,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm )
```

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

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

Definition at line 33 of file openmp_discretization.c.

Here is the caller graph for this function:



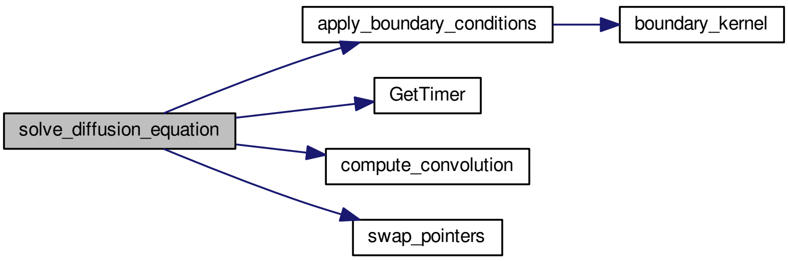

#### 4.28.2.3 solve_diffusion_equation()

```
void solve_diffusion_equation (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

Update the scalar composition field using old and Laplacian values.

Definition at line 56 of file openmp_discretization.c.

Here is the call graph for this function:

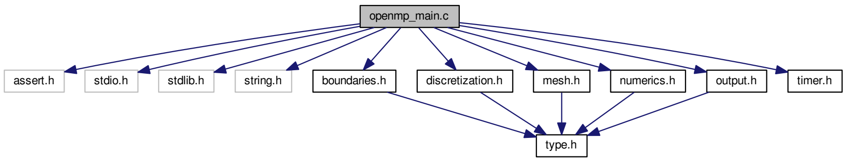


## 4.29 openmp_main.c File Reference

OpenMP implementation of semi-infinite diffusion equation.

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

Include dependency graph for openmp_main.c:




### Functions

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

  *Run simulation using input parameters specified on the command line.*

### 4.29.1 Detailed Description

OpenMP implementation of semi-infinite diffusion equation.

### 4.29.2 Function Documentation

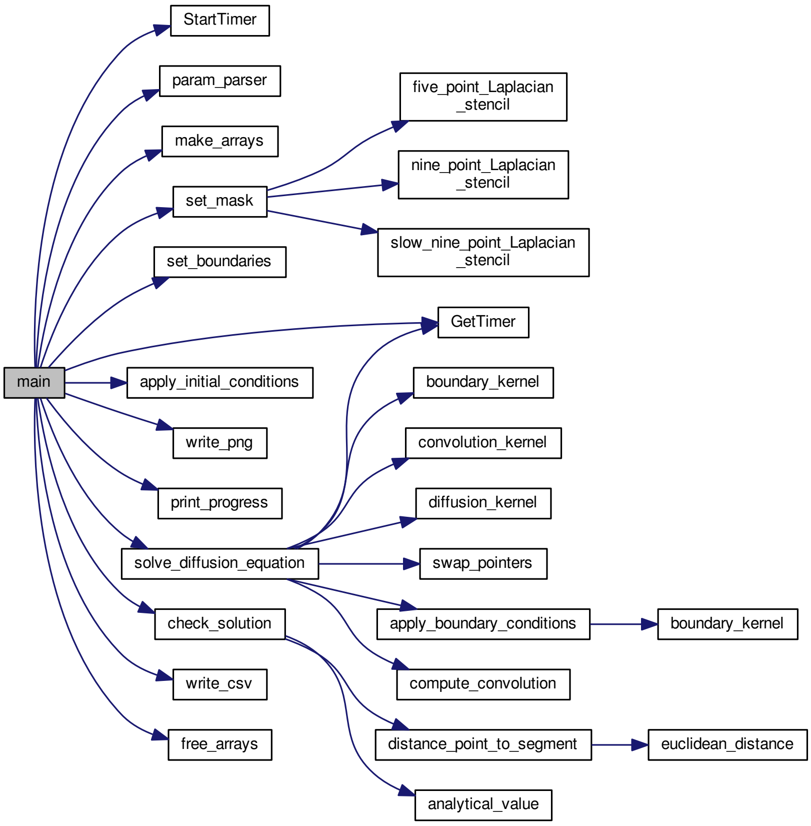
#### 4.29.2.1 main()

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

Run simulation using input parameters specified on the command line.

Definition at line 40 of file openmp_main.c.

Here is the call graph for this function:

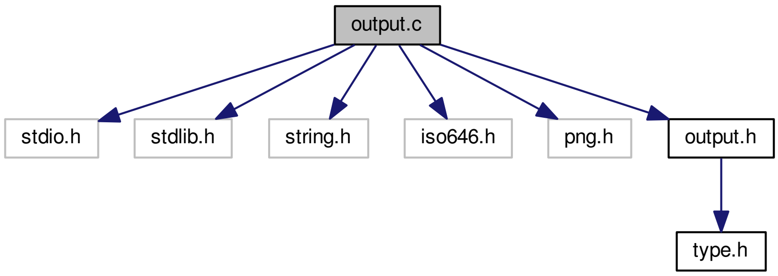



## 4.30 output.c File Reference

Implementation of file output functions for diffusion benchmarks.

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

Include dependency graph for output.c:




## Functions

- void param_parser (int argc, char ∗argv[ ], int ∗nx, int ∗ny, int ∗nm, int ∗code, fp_t ∗dx, fp_t ∗dy, fp_t ∗D, fp_t ∗linStab, int ∗steps, int ∗checks)

    *Read parameters from file specified on the command line.*
- void print_progress (const int step, const int steps)

    *Prints timestamps and a 20-point progress bar to stdout.*
- void write_csv (fp_t ∗∗conc, int nx, int ny, fp_t dx, fp_t dy, int step)

    *Writes scalar composition field to diffusion.???????.csv.*
- void write_png (fp_t ∗∗conc, int nx, int ny, int step)

    *Writes scalar composition field to diffusion.???????.png.*

### 4.30.1 Detailed Description

Implementation of file output functions for diffusion benchmarks.

### 4.30.2 Function Documentation

#### 4.30.2.1 param_parser()

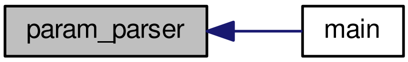
```
void param_parser (
             int argc,
             char ∗ argv[ ],
             int ∗ nx,
             int ∗ ny,
             int ∗ nm,
             int ∗ code,
             fp_t ∗ dx,
             fp_t ∗ dy,
             fp_t ∗ D,
             fp_t ∗ linStab,
```

```
            int * steps,
            int * checks )
```

Read parameters from file specified on the command line.

Definition at line 32 of file output.c.

Here is the caller graph for this function:




#### 4.30.2.2 print_progress()

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

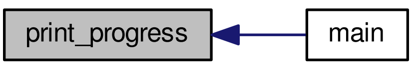
Prints timestamps and a 20-point progress bar to stdout.

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

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

Definition at line 123 of file output.c.

Here is the caller graph for this function:



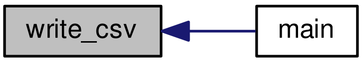
#### 4.30.2.3 write_csv()

```
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.

Definition at line 148 of file output.c.

Here is the caller graph for this function:



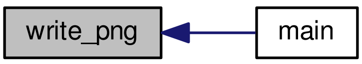

#### 4.30.2.4 write_png()

```
void write_png (
            fp_t ** conc,
            int nx,
            int ny,
            int step )
```

Writes scalar composition field to diffusion.???????.png.

Definition at line 182 of file output.c.

Here is the caller graph for this function:

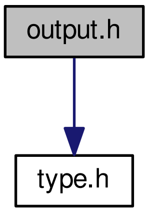


## 4.31 output.h File Reference

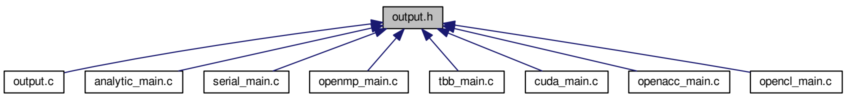
Declaration of output function prototypes for diffusion benchmarks.

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

Include dependency graph for output.h:




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




### Functions

- void param_parser (int argc, char ∗argv[ ], int ∗nx, int ∗ny, int ∗nm, int ∗code, fp_t ∗dx, fp_t ∗dy, fp_t ∗D, fp_t ∗linStab, int ∗steps, int ∗checks)

    *Read parameters from file specified on the command line.*
- void print_progress (const int step, const int steps)

    *Prints timestamps and a 20-point progress bar to stdout.*
- void write_csv (fp_t ∗∗conc, int nx, int ny, fp_t dx, fp_t dy, int step)

    *Writes scalar composition field to diffusion.???????.csv.*
- void write_png (fp_t ∗∗conc, int nx, int ny, int step)

    *Writes scalar composition field to diffusion.???????.png.*

### 4.31.1 Detailed Description

Declaration of output function prototypes for diffusion benchmarks.

### 4.31.2 Function Documentation

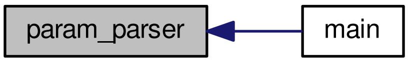
#### 4.31.2.1 param_parser()

```
void param_parser (
            int argc,
            char * argv[ ],
            int * nx,
            int * ny,
            int * nm,
            int * code,
            fp_t * dx,
            fp_t * dy,
            fp_t * D,
            fp_t * linStab,
            int * steps,
            int * checks )
```

Read parameters from file specified on the command line.

Definition at line 32 of file output.c.

Here is the caller graph for this function:




#### 4.31.2.2 print_progress()

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

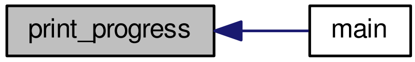
Prints timestamps and a 20-point progress bar to stdout.

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

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

Definition at line 123 of file output.c.

Here is the caller graph for this function:



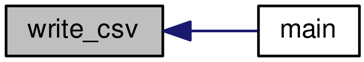

#### 4.31.2.3 write_csv()

```
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.

Definition at line 148 of file output.c.

Here is the caller graph for this function:



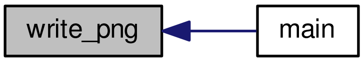

#### 4.31.2.4 write_png()

```
void write_png (
            fp_t ** conc,
            int nx,
            int ny,
            int step )
```

Writes scalar composition field to diffusion.???????.png.

Definition at line 182 of file output.c.

Here is the caller graph for this function:

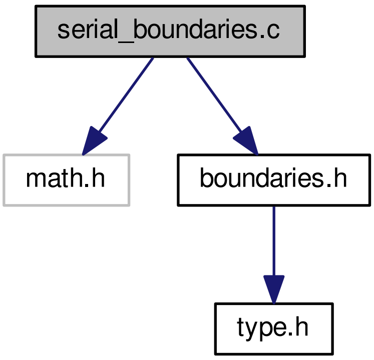



## 4.32 serial_boundaries.c File Reference

Implementation of boundary condition functions without threading.

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

Include dependency graph for serial_boundaries.c:




### Functions

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

  *Set values to be used along the simulation domain boundaries.*
- void apply_initial_conditions (fp_t $**$conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Initialize flat composition field with fixed boundary conditions.*
- void apply_boundary_conditions (fp_t $**$conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.*

#### 4.32.1 Detailed Description

Implementation of boundary condition functions without threading.

### 4.32.2 Function Documentation

#### 4.32.2.1 apply_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.

Definition at line 54 of file serial_boundaries.c.

#### 4.32.2.2 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 37 of file serial_boundaries.c.

#### 4.32.2.3 set_boundaries()

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

Set values to be used along the simulation domain boundaries.

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

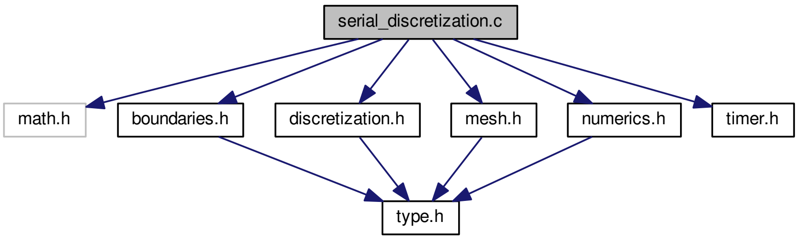
Definition at line 28 of file serial_boundaries.c.

## 4.33 serial_discretization.c File Reference

Implementation of boundary condition functions without threading.

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

Include dependency graph for serial_discretization.c:




### Functions

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

  *Perform the convolution of the mask matrix with the composition matrix.*
- void solve_diffusion_equation (fp_t **conc_old, fp_t **conc_new, fp_t **conc_lap, fp_t **mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t *elapsed, struct Stopwatch *sw)

  *Update the scalar composition field using old and Laplacian values.*
- void check_solution (fp_t **conc_new, fp_t **conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t *rss)

  *Compare numerical and analytical solutions of the diffusion equation.*

### 4.33.1 Detailed Description

Implementation of boundary condition functions without threading.

### 4.33.2 Function Documentation

#### 4.33.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

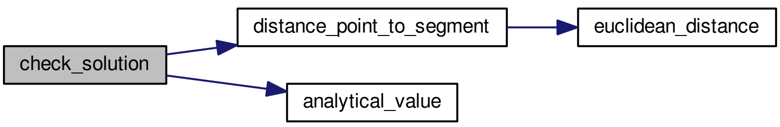
**Returns**

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

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

Definition at line 78 of file serial_discretization.c.

Here is the call graph for this function:




#### 4.33.2.2 compute_convolution()

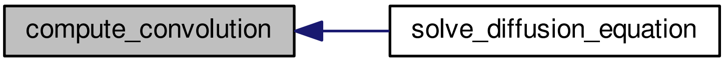
```
void compute_convolution (
            fp_t ** conc_old,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm )
```

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

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

Definition at line 32 of file serial_discretization.c.

Here is the caller graph for this function:



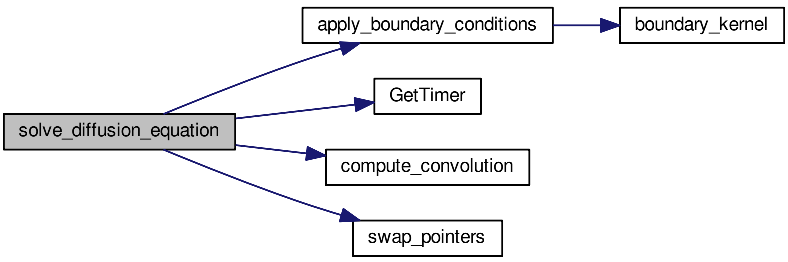

#### 4.33.2.3 solve_diffusion_equation()

```
void solve_diffusion_equation (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

Update the scalar composition field using old and Laplacian values.

Definition at line 51 of file serial_discretization.c.

Here is the call graph for this function:

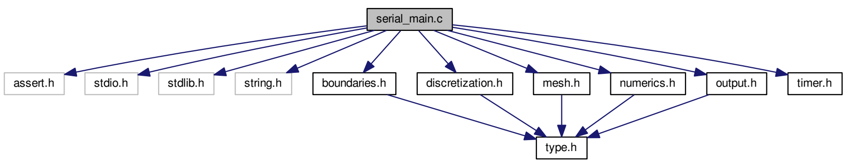


## 4.34 serial_main.c File Reference

Serial implementation of semi-infinite diffusion equation.

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

Include dependency graph for serial_main.c:




### Functions

- int main (int argc, char ∗argv[ ])

    *Run simulation using input parameters specified on the command line.*

### 4.34.1 Detailed Description

Serial implementation of semi-infinite diffusion equation.

### 4.34.2 Function Documentation

#### 4.34.2.1 main()

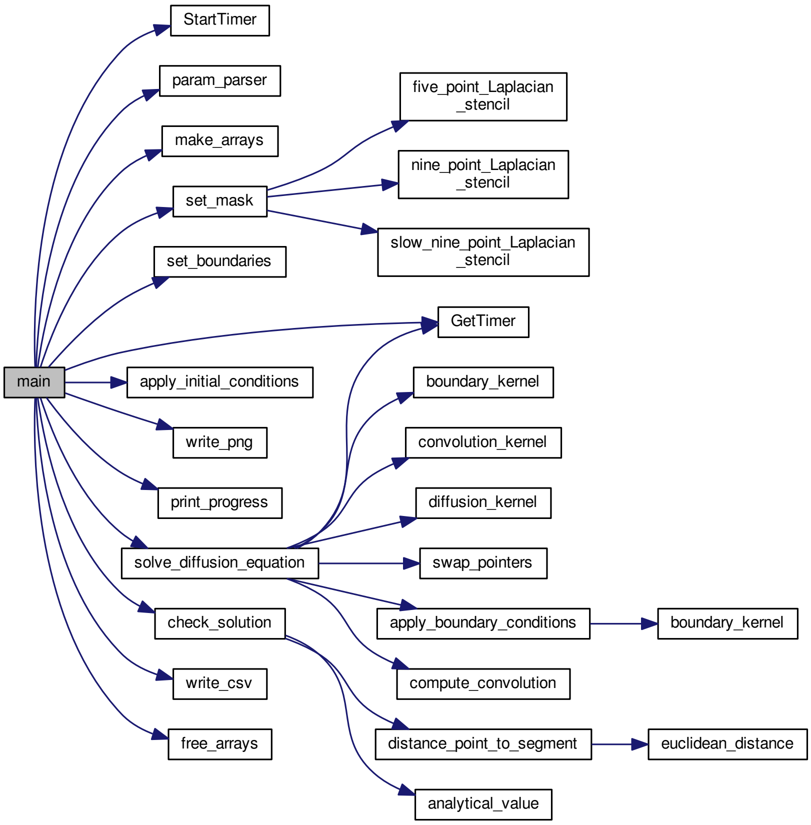
```
int main (
            int argc,
            char * argv[ ] )
```

Run simulation using input parameters specified on the command line.

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

Definition at line 48 of file serial_main.c.

Here is the call graph for this function:

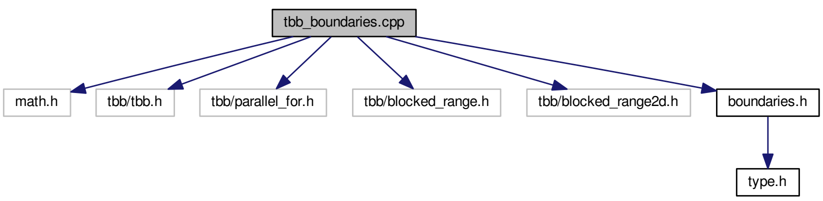



## 4.35 tbb_boundaries.cpp File Reference

Implementation of boundary condition functions with TBB threading.

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

Include dependency graph for tbb_boundaries.cpp:



#### Functions

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

  *Set values to be used along the simulation domain boundaries.*
- void apply_initial_conditions (fp_t **conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Initialize flat composition field with fixed boundary conditions.*
- void apply_boundary_conditions (fp_t **conc, int nx, int ny, int nm, fp_t bc[2][2])

  *Set fixed value $(c_{hi})$ along left and bottom, zero-flux elsewhere.*

### 4.35.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

### 4.35.2 Function Documentation

#### 4.35.2.1 apply_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.

Definition at line 77 of file tbb_boundaries.cpp.

#### 4.35.2.2 apply_initial_conditions()

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

Initialize flat composition field with fixed boundary conditions.

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

Definition at line 41 of file tbb_boundaries.cpp.

#### 4.35.2.3 set_boundaries()

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

Set values to be used along the simulation domain boundaries.

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

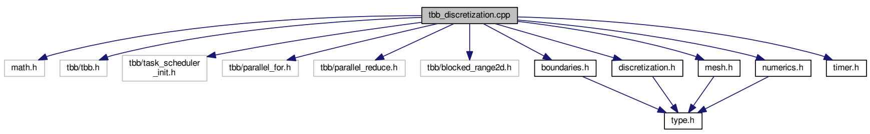
Definition at line 32 of file tbb_boundaries.cpp.

## 4.36 tbb_discretization.cpp File Reference

Implementation of boundary condition functions with TBB threading.

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

Include dependency graph for tbb_discretization.cpp:




### Functions

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

    *Perform the convolution of the mask matrix with the composition matrix.*
- void solve_diffusion_equation (fp_t $**$conc_old, fp_t $**$conc_new, fp_t $**$conc_lap, fp_t $**$mask_lap, int nx, int ny, int nm, fp_t bc[2][2], fp_t D, fp_t dt, int checks, fp_t $*$elapsed, struct Stopwatch $*$sw)

    *Update the scalar composition field using old and Laplacian values.*
- void check_solution (fp_t $**$conc_new, fp_t $**$conc_lap, int nx, int ny, fp_t dx, fp_t dy, int nm, fp_t elapsed, fp_t D, fp_t bc[2][2], fp_t $*$rss)

    *Compare numerical and analytical solutions of the diffusion equation.*

### 4.36.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

### 4.36.2 Function Documentation

#### 4.36.2.1 check_solution()

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

Compare numerical and analytical solutions of the diffusion equation.

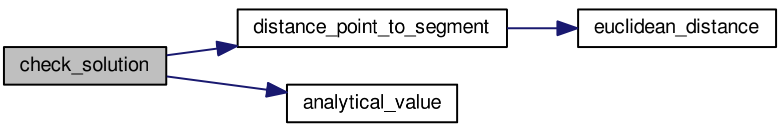
**Returns**

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

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

Definition at line 91 of file tbb_discretization.cpp.

Here is the call graph for this function:




#### 4.36.2.2 compute_convolution()

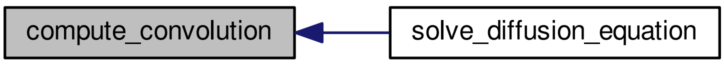
```
void compute_convolution (
            fp_t ** conc_old,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm )
```

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

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

Definition at line 37 of file tbb_discretization.cpp.

Here is the caller graph for this function:



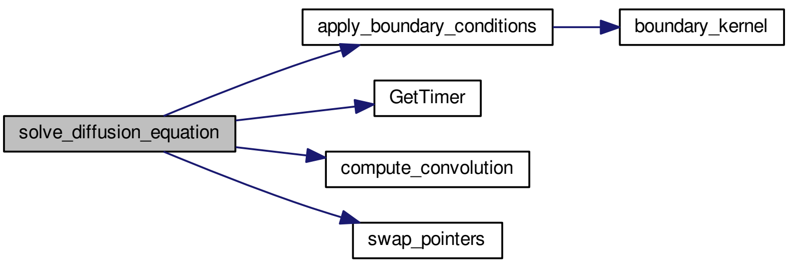

#### 4.36.2.3 solve_diffusion_equation()

```
void solve_diffusion_equation (
            fp_t ** conc_old,
            fp_t ** conc_new,
            fp_t ** conc_lap,
            fp_t ** mask_lap,
            int nx,
            int ny,
            int nm,
            fp_t bc[2][2],
            fp_t D,
            fp_t dt,
            int checks,
            fp_t * elapsed,
            struct Stopwatch * sw )
```

Update the scalar composition field using old and Laplacian values.

Definition at line 58 of file tbb_discretization.cpp.

Here is the call graph for this function:

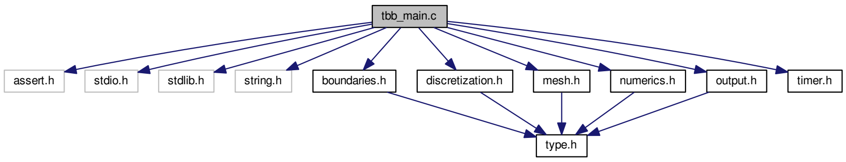


## 4.37 tbb_main.c File Reference

Threading Building Blocks implementation of semi-infinite diffusion equation.

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

Include dependency graph for tbb_main.c:




### Functions

- int main (int argc, char ∗argv[ ])

  *Run simulation using input parameters specified on the command line.*

### 4.37.1 Detailed Description

Threading Building Blocks implementation of semi-infinite diffusion equation.

### 4.37.2 Function Documentation

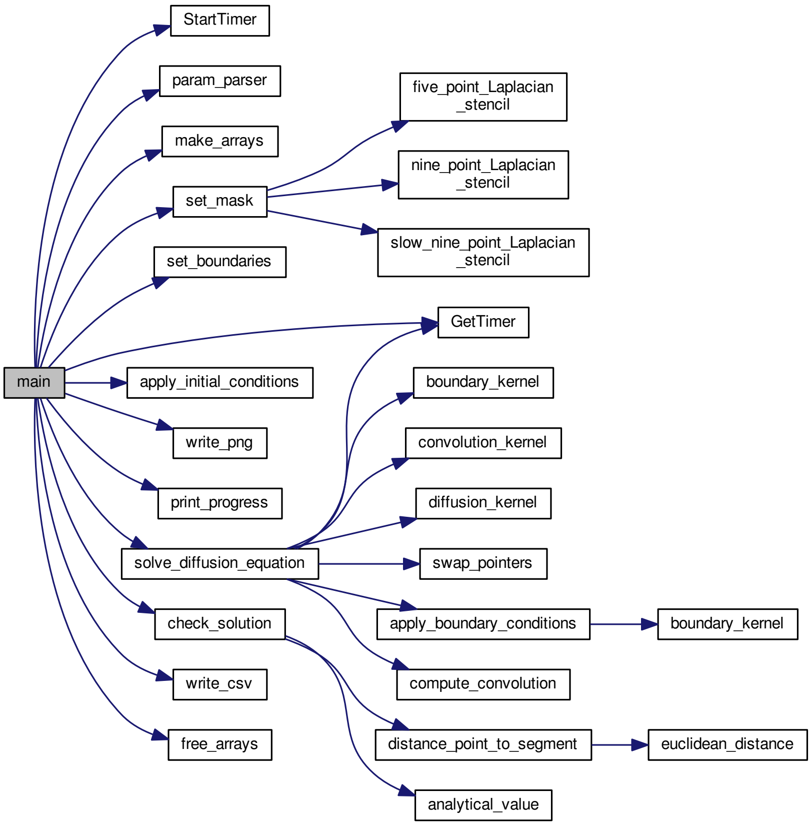
#### 4.37.2.1 main()

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

Run simulation using input parameters specified on the command line.

Definition at line 40 of file tbb_main.c.

Here is the call graph for this function:

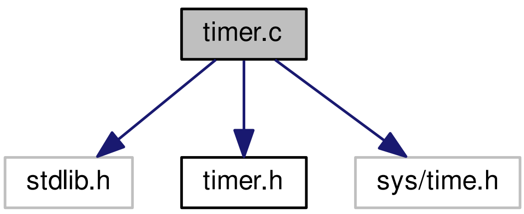



## 4.38 timer.c File Reference

High-resolution cross-platform machine time reader.

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

Include dependency graph for timer.c:



### Functions

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

### Variables

- struct timeval timerStart

### 4.38.1 Detailed Description

High-resolution cross-platform machine time reader.

**Author**

NVIDIA

### 4.38.2 Function Documentation

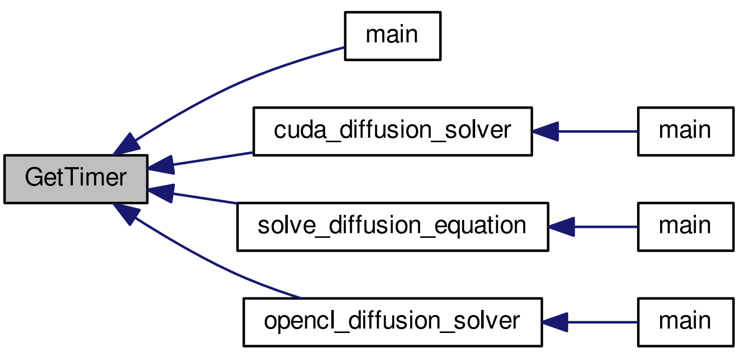
#### 4.38.2.1 GetTimer()

```
double GetTimer ( )
```

Return elapsed time in seconds.

Definition at line 64 of file timer.c.

Here is the caller graph for this function:



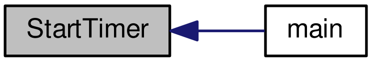
#### 4.38.2.2 StartTimer()

`void StartTimer ( )`

Set CPU frequency and begin timing.

Definition at line 48 of file timer.c.

Here is the caller graph for this function:




### 4.38.3 Variable Documentation

#### 4.38.3.1 timerStart

`struct timeval timerStart`

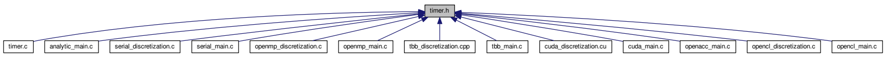
Platform-dependent data type of hardware time value

Definition at line 45 of file timer.c.

## 4.39 timer.h File Reference

Declaration of timer function prototypes for diffusion benchmarks.

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




### Functions

- void StartTimer ()

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

  *Return elapsed time in seconds.*

#### 4.39.1 Detailed Description

Declaration of timer function prototypes for diffusion benchmarks.

#### 4.39.2 Function Documentation

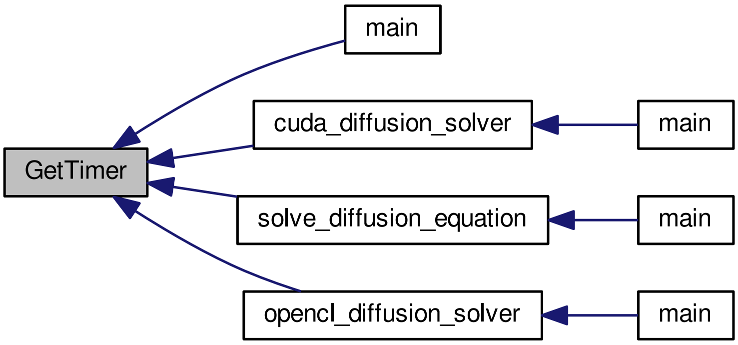
##### 4.39.2.1 GetTimer()

```
double GetTimer ( )
```

Return elapsed time in seconds.

Definition at line 64 of file timer.c.

Here is the caller graph for this function:



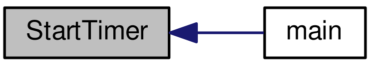

##### 4.39.2.2 StartTimer()

```
void StartTimer ( )
```

Set CPU frequency and begin timing.

Definition at line 48 of file timer.c.

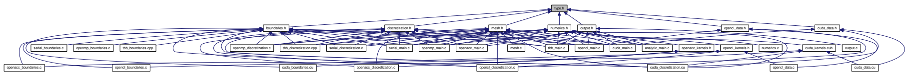
Here is the caller graph for this function:



## 4.40 type.h File Reference

Definition of scalar data type and Doxygen diffusion group.

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



### Classes

- struct Stopwatch

### Typedefs

- typedef double fp_t

### 4.40.1 Detailed Description

Definition of scalar data type and Doxygen diffusion group.

### 4.40.2 Typedef Documentation

#### 4.40.2.1 fp_t

```
typedef double fp_t
```

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

Definition at line 36 of file type.h.

# Index