

HiPerC
alpha

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	1
2.1 File List	1
3 Class Documentation	3
3.1 CudaData Struct Reference	3
3.1.1 Detailed Description	4
3.1.2 Member Data Documentation	4
3.2 OpenCLData Struct Reference	4
3.2.1 Detailed Description	5
3.2.2 Member Data Documentation	5
3.3 Stopwatch Struct Reference	7
3.3.1 Detailed Description	8
3.3.2 Member Data Documentation	8
4 File Documentation	9
4.1 boundaries.h File Reference	9
4.1.1 Detailed Description	10
4.1.2 Function Documentation	10
4.2 cuda_boundaries.cu File Reference	11
4.2.1 Detailed Description	12
4.2.2 Function Documentation	12
4.3 cuda_data.cu File Reference	14
4.3.1 Detailed Description	14
4.3.2 Function Documentation	14
4.4 cuda_data.h File Reference	16
4.4.1 Detailed Description	17
4.4.2 Function Documentation	17
4.5 cuda_discretization.cu File Reference	21
4.5.1 Detailed Description	22
4.5.2 Function Documentation	22
4.5.3 Variable Documentation	27
4.6 cuda_kernels.cuh File Reference	27
4.6.1 Detailed Description	29
4.6.2 Function Documentation	29
4.6.3 Variable Documentation	31
4.7 cuda_main.c File Reference	31
4.7.1 Detailed Description	32

4.7.2 Function Documentation	32
4.8 kernel_boundary.cl File Reference	33
4.8.1 Function Documentation	34
4.9 kernel_convolution.cl File Reference	34
4.9.1 Function Documentation	35
4.10 kernel_diffusion.cl File Reference	36
4.10.1 Function Documentation	36
4.11 mesh.c File Reference	37
4.11.1 Detailed Description	38
4.11.2 Function Documentation	38
4.12 mesh.h File Reference	40
4.12.1 Detailed Description	41
4.12.2 Function Documentation	41
4.13 numerics.c File Reference	43
4.13.1 Detailed Description	44
4.13.2 Function Documentation	44
4.14 numerics.h File Reference	50
4.14.1 Detailed Description	51
4.14.2 Macro Definition Documentation	51
4.14.3 Function Documentation	52
4.15 openacc_boundaries.c File Reference	59
4.15.1 Detailed Description	59
4.15.2 Function Documentation	59
4.16 openacc_discretization.c File Reference	61
4.16.1 Detailed Description	62
4.16.2 Function Documentation	62
4.17 openacc_kernels.h File Reference	63
4.17.1 Detailed Description	64
4.17.2 Function Documentation	64
4.18 openacc_main.c File Reference	65
4.18.1 Detailed Description	65
4.18.2 Function Documentation	65
4.19 opencl_boundaries.c File Reference	66
4.19.1 Detailed Description	67
4.19.2 Function Documentation	67
4.20 opencl_data.c File Reference	68
4.20.1 Detailed Description	69
4.20.2 Function Documentation	69
4.21 opencl_data.h File Reference	71

4.21.1 Detailed Description	73
4.21.2 Function Documentation	73
4.22 opencl_discretization.c File Reference	78
4.22.1 Detailed Description	78
4.22.2 Function Documentation	78
4.23 opencl_main.c File Reference	80
4.23.1 Detailed Description	81
4.23.2 Function Documentation	81
4.24 openmp_boundaries.c File Reference	82
4.24.1 Detailed Description	83
4.24.2 Function Documentation	83
4.25 openmp_discretization.c File Reference	84
4.25.1 Detailed Description	85
4.25.2 Function Documentation	85
4.26 openmp_main.c File Reference	86
4.26.1 Detailed Description	86
4.26.2 Function Documentation	86
4.27 output.c File Reference	87
4.27.1 Detailed Description	88
4.27.2 Function Documentation	88
4.28 output.h File Reference	91
4.28.1 Detailed Description	91
4.28.2 Function Documentation	92
4.29 serial_boundaries.c File Reference	94
4.29.1 Detailed Description	94
4.29.2 Function Documentation	95
4.30 serial_discretization.c File Reference	95
4.30.1 Detailed Description	96
4.30.2 Function Documentation	96
4.31 serial_main.c File Reference	97
4.31.1 Detailed Description	97
4.31.2 Function Documentation	98
4.32 tbb_boundaries.cpp File Reference	99
4.32.1 Detailed Description	99
4.32.2 Function Documentation	99
4.33 tbb_discretization.cpp File Reference	100
4.33.1 Detailed Description	101
4.33.2 Function Documentation	101
4.34 tbb_main.c File Reference	102

4.34.1 Detailed Description	103
4.34.2 Function Documentation	103
4.35 timer.c File Reference	105
4.35.1 Detailed Description	106
4.35.2 Macro Definition Documentation	106
4.35.3 Function Documentation	107
4.35.4 Variable Documentation	108
4.36 timer.h File Reference	108
4.36.1 Detailed Description	108
4.36.2 Function Documentation	108
4.37 type.h File Reference	109
4.37.1 Detailed Description	110
4.37.2 Typedef Documentation	110
Index	111

1 Class Index

1.1 Class List

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

CudaData	
Container for pointers to arrays on the GPU	3
OpenCLData	
Container for GPU array pointers and parameters	4
Stopwatch	7

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

boundaries.h	
Declaration of boundary condition function prototypes	9
cuda_boundaries.cu	
Implementation of boundary condition functions with OpenMP threading	11

cuda_data.cu	Implementation of functions to create and destroy CudaData struct	14
cuda_data.h	Declaration of CUDA data container	16
cuda_discretization.cu	Implementation of boundary condition functions with CUDA acceleration	21
cuda_kernels.cuh	Declaration of functions to execute on the GPU (CUDA kernels)	27
cuda_main.c	CUDA implementation of semi-infinite diffusion equation	31
kernel_boundary.cl		33
kernel_convolution.cl		34
kernel_diffusion.cl		36
mesh.c	Implementation of mesh handling functions for diffusion benchmarks	37
mesh.h	Declaration of mesh function prototypes for diffusion benchmarks	40
numerics.c	Implementation of Laplacian operator and analytical solution functions	43
numerics.h	Declaration of Laplacian operator and analytical solution functions	50
openacc_boundaries.c	Implementation of boundary condition functions with OpenMP threading	59
openacc_discretization.c	Implementation of boundary condition functions with OpenACC threading	61
openacc_kernels.h	Declaration of functions to execute on the GPU (OpenACC kernels)	63
openacc_main.c	OpenACC implementation of semi-infinite diffusion equation	65
opencl_boundaries.c	Implementation of boundary condition functions with OpenCL acceleration	66
opencl_data.c	Implementation of functions to create and destroy OpenCLData struct	68
opencl_data.h	Declaration of OpenCL data container	71
opencl_discretization.c	Implementation of boundary condition functions with OpenCL acceleration	78

opencl_main.c	OpenCL implementation of semi-infinite diffusion equation	80
openmp_boundaries.c	Implementation of boundary condition functions with OpenMP threading	82
openmp_discretization.c	Implementation of boundary condition functions with OpenMP threading	84
openmp_main.c	OpenMP implementation of semi-infinite diffusion equation	86
output.c	Implementation of file output functions for diffusion benchmarks	87
output.h	Declaration of output function prototypes for diffusion benchmarks	91
serial_boundaries.c	Implementation of boundary condition functions without threading	94
serial_discretization.c	Implementation of boundary condition functions without threading	95
serial_main.c	Serial implementation of semi-infinite diffusion equation	97
tbb_boundaries.cpp	Implementation of boundary condition functions with TBB threading	99
tbb_discretization.cpp	Implementation of boundary condition functions with TBB threading	100
tbb_main.c	Threading Building Blocks implementation of semi-infinite diffusion equation	102
timer.c	High-resolution cross-platform machine time reader	105
timer.h	Declaration of timer function prototypes for diffusion benchmarks	108
type.h	Definition of scalar data type and Doxygen diffusion group	109

3 Class Documentation

3.1 CudaData Struct Reference

Container for pointers to arrays on the GPU.

```
#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 pointers to arrays on the GPU.

Definition at line 21 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 24 of file [cuda_data.h](#).

3.1.2.2 [conc_new](#) [fp_t](#)* [CudaData::conc_new](#)

Definition at line 23 of file [cuda_data.h](#).

3.1.2.3 [conc_old](#) [fp_t](#)* [CudaData::conc_old](#)

Definition at line 22 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_context [context](#)
- cl_mem [conc_old](#)
- cl_mem [conc_new](#)
- cl_mem [conc_lap](#)
- cl_mem [mask](#)
- 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_command_queue [commandQueue](#)

3.2.1 Detailed Description

Container for GPU array pointers and parameters.

From the [OpenCL v1.2](#) spec:

- A *Context* is the environment within which the kernels execute and the domain in which synchronization and memory management is defined. The context includes a set of devices, the memory accessible to those devices, the corresponding memory properties and one or more command-queues used to schedule execution of a kernel(s) or operations on memory objects.
- A *Program Object* encapsulates the following information:
 - A reference to an associated context.
 - A program source or binary.
 - The latest successfully built program executable, the list of devices for which the program executable is built, the build options used and a build log.
 - The number of kernel objects currently attached.
- A *Kernel Object* encapsulates a specific `__kernel` function declared in a program and the argument values to be used when executing this `__kernel` function.

Definition at line 37 of file `opengl_data.h`.

3.2.2 Member Data Documentation

3.2.2.1 `boundary_kernel` `cl_kernel` `OpenCLData::boundary_kernel`

Boundary program executable for the GPU

Definition at line 59 of file `opengl_data.h`.

3.2.2.2 boundary_program `cl_program OpenCLData::boundary_program`

Boundary program source for JIT compilation on the GPU

Definition at line 52 of file `opencldata.h`.

3.2.2.3 commandQueue `cl_command_queue OpenCLData::commandQueue`

Queue for submitting OpenCL jobs to the GPU

Definition at line 66 of file `opencldata.h`.

3.2.2.4 conc_lap `cl_mem OpenCLData::conc_lap`

Copy of Laplacian field on the GPU

Definition at line 46 of file `opencldata.h`.

3.2.2.5 conc_new `cl_mem OpenCLData::conc_new`

Copy of new composition field on the GPU

Definition at line 44 of file `opencldata.h`.

3.2.2.6 conc_old `cl_mem OpenCLData::conc_old`

Copy of old composition field on the GPU

Definition at line 42 of file `opencldata.h`.

3.2.2.7 context `cl_context OpenCLData::context`

OpenCL interface to the GPU, hardware and software

Definition at line 39 of file `opencldata.h`.

3.2.2.8 convolution_kernel `cl_kernel OpenCLData::convolution_kernel`

Convolution program executable for the GPU

Definition at line 61 of file `openccl_data.h`.

3.2.2.9 convolution_program `cl_program OpenCLData::convolution_program`

Convolution program source for JIT compilation on the GPU

Definition at line 54 of file `openccl_data.h`.

3.2.2.10 diffusion_kernel `cl_kernel OpenCLData::diffusion_kernel`

Timestepping program executable for the GPU

Definition at line 63 of file `openccl_data.h`.

3.2.2.11 diffusion_program `cl_program OpenCLData::diffusion_program`

Timestepping program source for JIT compilation on the GPU

Definition at line 56 of file `openccl_data.h`.

3.2.2.12 mask `cl_mem OpenCLData::mask`

Copy of Laplacian mask on the GPU

Definition at line 49 of file `openccl_data.h`.

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

- [openccl_data.h](#)

3.3 Stopwatch Struct Reference

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

Public Attributes

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

3.3.1 Detailed Description

Container for timing data

Definition at line 27 of file type.h.

3.3.2 Member Data Documentation

3.3.2.1 **conv** [fp_t](#) Stopwatch::conv

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

Definition at line 31 of file type.h.

3.3.2.2 **file** [fp_t](#) Stopwatch::file

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

Definition at line 41 of file type.h.

3.3.2.3 **soln** [fp_t](#) Stopwatch::soln

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

Definition at line 46 of file type.h.

3.3.2.4 step `fp_t` `Stopwatch::step`

Cumulative time executing `solve_diffusion_equation()`

Definition at line 36 of file `type.h`.

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

- [type.h](#)

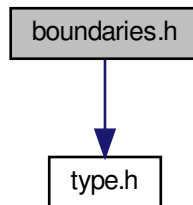
4 File Documentation

4.1 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 [apply_initial_conditions](#) (`fp_t` `**conc_old`, `const int nx`, `const int ny`, `const int nm`)
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) (`fp_t` `**conc_old`, `const int nx`, `const int ny`, `const int nm`)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.1.1 Detailed Description

Declaration of boundary condition function prototypes.

4.1.2 Function Documentation

4.1.2.1 `apply_boundary_conditions()` `void apply_boundary_conditions (`
 `fp_t ** conc_old,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

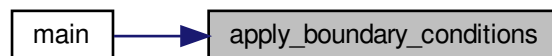
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

Definition at line 29 of file `serial_boundaries.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



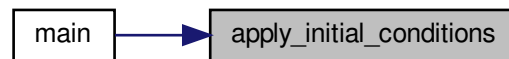
4.1.2.2 apply_initial_conditions() `void apply_initial_conditions (`
 `fp_t ** conc_old,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

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 14 of file serial_boundaries.c.

Here is the caller graph for this function:

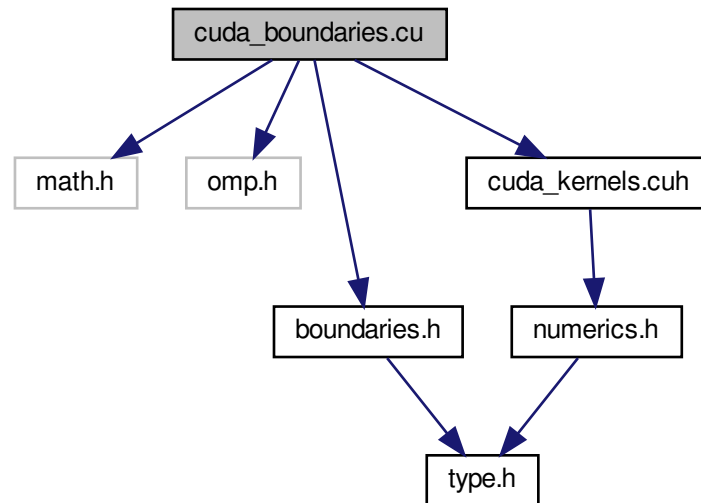


4.2 cuda_boundaries.cu File Reference

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 `apply_initial_conditions` (`fp_t **conc`, `const int nx`, `const int ny`, `const int nm`)
Initialize flat composition field with fixed boundary conditions.
- void `boundary_kernel` (`fp_t *d_conc`, `const int nx`, `const int ny`, `const int nm`)
Enable double-precision floats.

4.2.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.2.2 Function Documentation

4.2.2.1 apply_initial_conditions() `void apply_initial_conditions (`
 `fp_t ** conc_old,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

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 20 of file cuda_boundaries.cu.

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

Enable double-precision floats.

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

Definition at line 41 of file cuda_boundaries.cu.

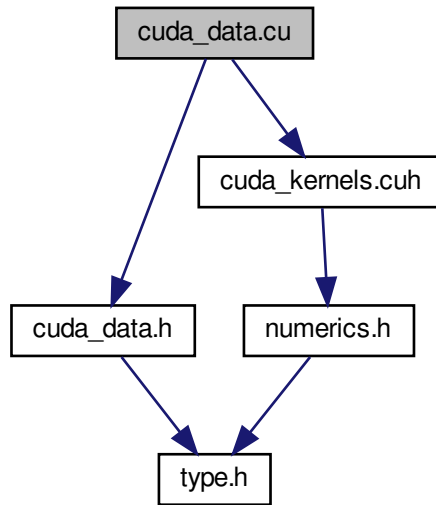
Here is the caller graph for this function:



4.3 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, const int nx, const int ny, const int nm, struct [CudaData](#) *dev)
Initialize CUDA device memory before marching.
- void `free_cuda` (struct [CudaData](#) *dev)
Free CUDA device memory after marching.

4.3.1 Detailed Description

Implementation of functions to create and destroy [CudaData](#) struct.

4.3.2 Function Documentation

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

Free CUDA device memory after marching.

Definition at line 33 of file cuda_data.cu.

Here is the caller graph for this function:



4.3.2.2 init_cuda() `void init_cuda (`
 `fp_t ** conc_old,`
 `fp_t ** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `struct CudaData * dev)`

Initialize CUDA device memory before marching.

Definition at line 17 of file cuda_data.cu.

Here is the caller graph for this function:

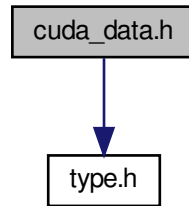


4.4 cuda_data.h File Reference

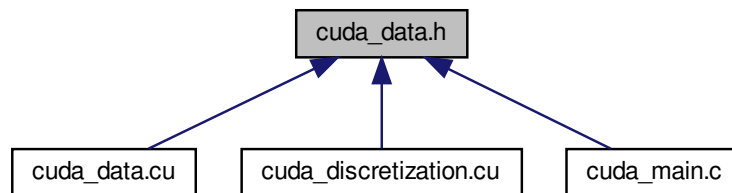
Declaration of CUDA data container.

```
#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 pointers to arrays on the GPU.

Functions

- void [init_cuda](#) ([fp_t](#) **conc_old, [fp_t](#) **mask_lap, const int nx, const int ny, const int nm, struct [CudaData](#) *dev)
Initialize CUDA device memory before marching.
- void [free_cuda](#) (struct [CudaData](#) *dev)
Free CUDA device memory after marching.

- void `device_boundaries` (`fp_t` *conc, const int nx, const int ny, const int nm, const int bx, const int by)
Apply boundary conditions on device.
- void `device_convolution` (`fp_t` *conc_old, `fp_t` *conc_lap, const int nx, const int ny, const int nm, const int bx, const int by)
Compute convolution on device.
- void `device_composition` (`fp_t` *conc_old, `fp_t` *conc_new, `fp_t` *conc_lap, const int nx, const int ny, const int nm, const int bx, const int by, const `fp_t` D, const `fp_t` dt)
Step diffusion equation on device.
- void `cuda_diffusion_solver` (struct `CudaData` *dev, `fp_t` **conc_new, const int bx, const int by, const int nm, const int nx, const int ny, const `fp_t` D, const `fp_t` dt, struct `Stopwatch` *sw)
Solve diffusion equation on the GPU.
- void `read_out_result` (`fp_t` **conc, `fp_t` *d_conc, const int nx, const int ny)
Read data from device.

4.4.1 Detailed Description

Declaration of CUDA data container.

4.4.2 Function Documentation

4.4.2.1 `cuda_diffusion_solver()` void `cuda_diffusion_solver` (
 struct `CudaData` * dev,
 `fp_t` ** conc_new,
 const int bx,
 const int by,
 const int nm,
 const int nx,
 const int ny,
 const `fp_t` D,
 const `fp_t` dt,
 struct `Stopwatch` * sw)

Solve diffusion equation on the GPU.

Solve diffusion equation on the GPU.

Compare `cuda_diffusion_solver()`: it accomplishes the same result, but without the memory allocation, data transfer, and array release. These are handled in `cuda_init()`, with arrays on the host and device managed through `CudaData`, which is a struct passed by reference into the function. In this way, device kernels can be called in isolation without incurring the cost of data transfers and with reduced risk of memory leaks.

Definition at line 219 of file `cuda_discretization.cu`.

Here is the call graph for this function:



4.4.2.2 device_boundaries() `void device_boundaries (`
 `fp_t * conc,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const int bx,`
 `const int by)`

Apply boundary conditions on device.

Definition at line 108 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



4.4.2.3 device_composition() `void device_composition (`
 `fp_t * conc_old,`
 `fp_t * conc_new,`
 `fp_t * conc_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const int bx,`
 `const int by,`

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

Step diffusion equation on device.

Definition at line 140 of file cuda_discretization.cu.

Here is the caller graph for this function:



4.4.2.4 device_convolution() void device_convolution (
 fp_t * conc_old,
 fp_t * conc_lap,
 const int nx,
 const int ny,
 const int nm,
 const int bx,
 const int by)

Compute convolution on device.

Definition at line 123 of file cuda_discretization.cu.

Here is the caller graph for this function:



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

Free CUDA device memory after marching.

Definition at line 33 of file `cuda_data.cu`.

Here is the caller graph for this function:



4.4.2.6 init_cuda() `void init_cuda (`
 `fp_t ** conc_old,`
 `fp_t ** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `struct CudaData * dev)`

Initialize CUDA device memory before marching.

Definition at line 17 of file `cuda_data.cu`.

Here is the caller graph for this function:




```

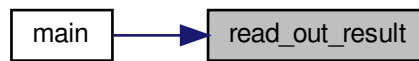
4.4.2.7 read_out_result() void read_out_result (
    fp_t ** conc,
    fp_t * d_conc,
    const int nx,
    const int ny )

```

Read data from device.

Definition at line 155 of file cuda_discretization.cu.

Here is the caller graph for this function:



4.5 cuda_discretization.cu File Reference

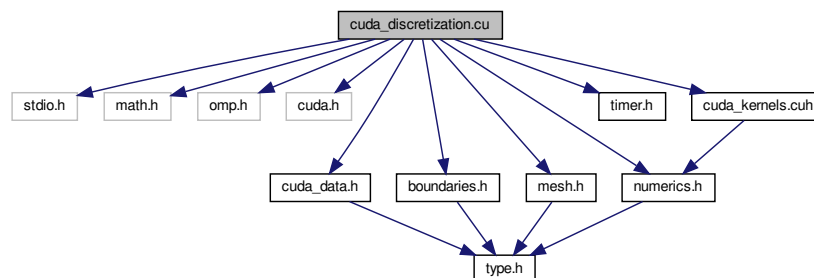
Implementation of boundary condition functions with CUDA acceleration.

```

#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <cuda.h>
#include "cuda_data.h"
#include "boundaries.h"
#include "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 `device_boundaries` (`fp_t` *conc, const int nx, const int ny, const int nm, const int bx, const int by)
Apply boundary conditions on device.
- void `device_convolution` (`fp_t` *conc_old, `fp_t` *conc_lap, const int nx, const int ny, const int nm, const int bx, const int by)
Compute convolution on device.
- void `device_composition` (`fp_t` *conc_old, `fp_t` *conc_new, `fp_t` *conc_lap, const int nx, const int ny, const int nm, const int bx, const int by, const `fp_t` D, const `fp_t` dt)
Step diffusion equation on device.
- void `read_out_result` (`fp_t` **conc, `fp_t` *d_conc, const int nx, const int ny)
Read data from device.
- void `compute_convolution` (`fp_t` **conc_old, `fp_t` **conc_lap, `fp_t` **mask_lap, const int bx, const int by, const int nm, const int nx, const int ny)
Reference showing how to invoke the convolution kernel.
- void `cuda_diffusion_solver` (struct `CudaData` *dev, `fp_t` **conc_new, const int bx, const int by, const int nm, const int nx, const int ny, const `fp_t` D, const `fp_t` dt, struct `Stopwatch` *sw)
Reference optimized code for solving the diffusion equation.

Variables

- `fp_t d_mask` [5 *5]
Convolution mask array on the GPU, allocated in protected memory.

4.5.1 Detailed Description

Implementation of boundary condition functions with CUDA acceleration.

4.5.2 Function Documentation

4.5.2.1 `compute_convolution()` `void compute_convolution (`
`fp_t ** conc_old,`
`fp_t ** conc_lap,`
`fp_t ** mask_lap,`
`const int bx,`
`const int by,`
`const int nm,`
`const int nx,`
`const int ny)`

Reference showing how to invoke the convolution kernel.

A stand-alone function like this incurs the cost of host-to-device data transfer each time it is called: it is a teaching tool, not reusable code. It is the basis for `cuda_diffusion_solver()`, which achieves much better performance by bundling CUDA kernels together and intelligently managing data transfers between the host (CPU) and device (GPU).

Definition at line 170 of file `cuda_discretization.cu`.

4.5.2.2 `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.

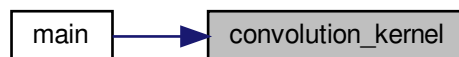
This function accesses 1D data rather than the 2D array representation of the scalar composition field, mapping into 2D tiles on the GPU with halo cells before computing the convolution.

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 28 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



```
4.5.2.3 cuda_diffusion_solver() void cuda_diffusion_solver (
    struct CudaData * dev,
    fp_t ** conc_new,
    const int bx,
    const int by,
    const int nm,
    const int nx,
    const int ny,
    const fp_t D,
    const fp_t dt,
    struct Stopwatch * sw )
```

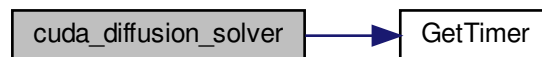
Reference optimized code for solving the diffusion equation.

Solve diffusion equation on the GPU.

Compare `cuda_diffusion_solver()`: it accomplishes the same result, but without the memory allocation, data transfer, and array release. These are handled in `cuda_init()`, with arrays on the host and device managed through `CudaData`, which is a struct passed by reference into the function. In this way, device kernels can be called in isolation without incurring the cost of data transfers and with reduced risk of memory leaks.

Definition at line 219 of file `cuda_discretization.cu`.

Here is the call graph for this function:



```
4.5.2.4 device_boundaries() void device_boundaries (
    fp_t * conc,
    const int nx,
    const int ny,
    const int nm,
    const int bx,
    const int by )
```

Apply boundary conditions on device.

Definition at line 108 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



4.5.2.5 device_composition() void device_composition (

```
    fp_t * conc_old,  
    fp_t * conc_new,  
    fp_t * conc_lap,  
    const int nx,  
    const int ny,  
    const int nm,  
    const int bx,  
    const int by,  
    const fp_t D,  
    const fp_t dt )
```

Step diffusion equation on device.

Definition at line 140 of file cuda_discretization.cu.

Here is the caller graph for this function:



4.5.2.6 device_convolution() `void device_convolution (`
 `fp_t * conc_old,`
 `fp_t * conc_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const int bx,`
 `const int by)`

Compute convolution on device.

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

Here is the caller graph for this function:



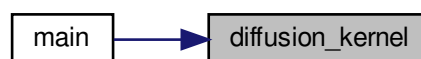
4.5.2.7 diffusion_kernel() `void diffusion_kernel (`
 `fp_t * conc_old,`
 `fp_t * conc_new,`
 `fp_t * conc_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const fp_t D,`
 `const fp_t dt)`

Vector addition algorithm for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field. Memory allocation, data transfer, and array release are handled in `cuda_init()`, with arrays on the host and device managed through [CudaData](#), which is a struct passed by reference into the function. In this way, device kernels can be called in isolation without incurring the cost of data transfers and with reduced risk of memory leaks.

Definition at line 85 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



4.5.2.8 read_out_result() `void read_out_result (`
 `fp_t ** conc,`
 `fp_t * d_conc,`
 `const int nx,`
 `const int ny)`

Read data from device.

Definition at line 155 of file cuda_discretization.cu.

Here is the caller graph for this function:



4.5.3 Variable Documentation

4.5.3.1 d_mask `fp_t d_mask[5 * 5]`

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

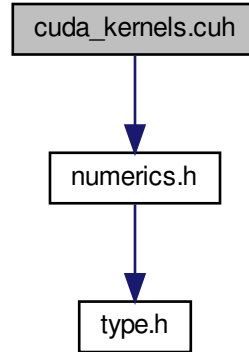
Definition at line 26 of file cuda_discretization.cu.

4.6 cuda_kernels.cuh File Reference

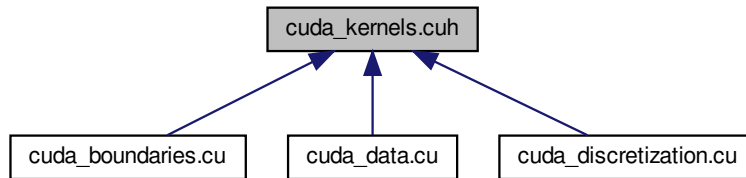
Declaration of functions to execute on the GPU (CUDA kernels)

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

Include dependency graph for cuda_kernels.cuh:



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



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` [5 *5]
Convolution mask array on the GPU, allocated in protected memory.

4.6.1 Detailed Description

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

4.6.2 Function Documentation

4.6.2.1 boundary_kernel() `void boundary_kernel (`
 `fp_t * d_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

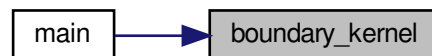
Boundary condition kernel for execution on the GPU.

Boundary condition kernel for execution on the GPU

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

Definition at line 41 of file cuda_boundaries.cu.

Here is the caller graph for this function:



4.6.2.2 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 28 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



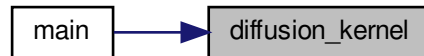
4.6.2.3 diffusion_kernel() `void diffusion_kernel (`
 `fp_t * conc_old,`
 `fp_t * conc_new,`
 `fp_t * conc_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const fp_t D,`
 `const fp_t dt)`

Vector addition algorithm for execution on the GPU.

This function accesses 1D data rather than the 2D array representation of the scalar composition field. Memory allocation, data transfer, and array release are handled in `cuda_init()`, with arrays on the host and device managed through `CudaData`, which is a struct passed by reference into the function. In this way, device kernels can be called in isolation without incurring the cost of data transfers and with reduced risk of memory leaks.

Definition at line 85 of file `cuda_discretization.cu`.

Here is the caller graph for this function:



4.6.3 Variable Documentation

4.6.3.1 d_mask `fp_t d_mask[5 * 5] [extern]`

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

Definition at line 26 of file `cuda_discretization.cu`.

4.7 cuda_main.c File Reference

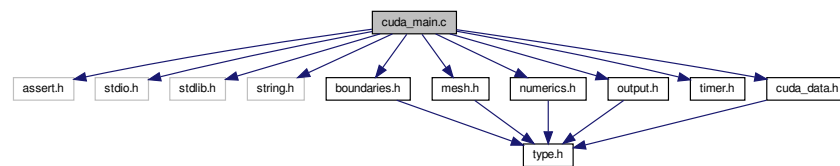
CUDA implementation of semi-infinite diffusion equation.

```

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "boundaries.h"
#include "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.

4.7.1 Detailed Description

CUDA implementation of semi-infinite diffusion equation.

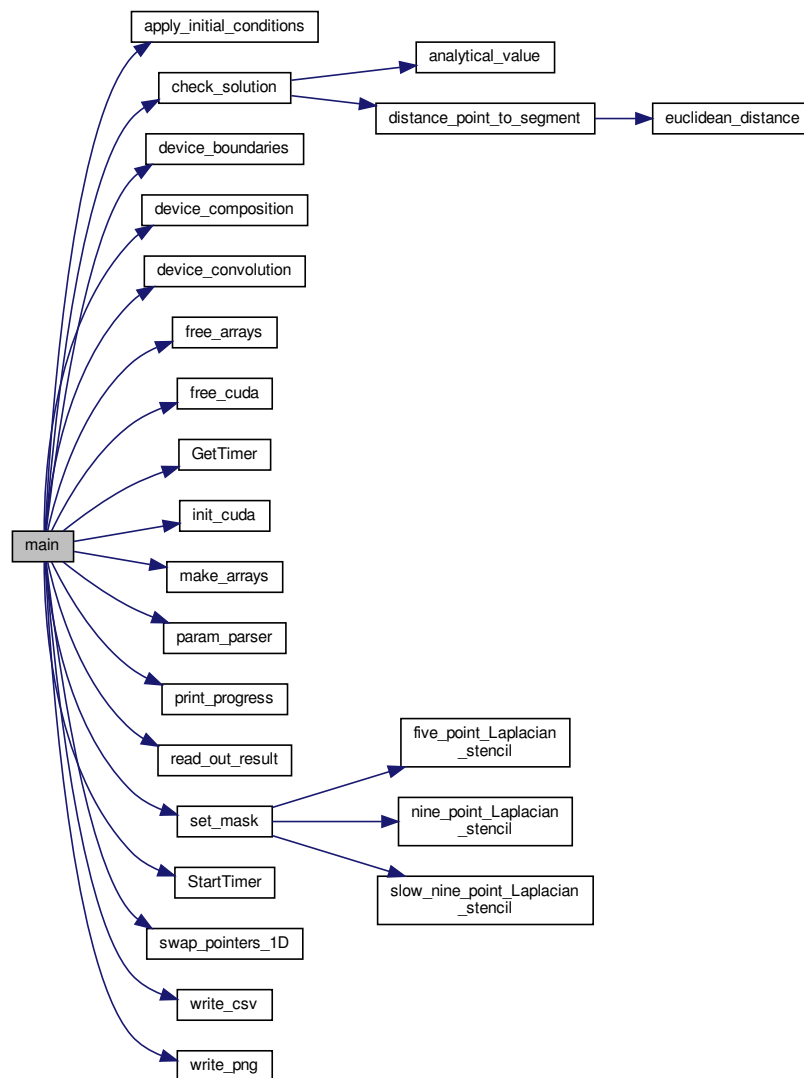
4.7.2 Function Documentation

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

Run simulation using input parameters specified on the command line.

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

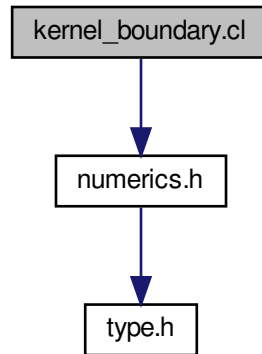
Here is the call graph for this function:



4.8 kernel_boundary.cl File Reference

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

Include dependency graph for kernel_boundary.cl:



Functions

- `__kernel void boundary_kernel (__global fp_t *d_conc, const int nx, const int ny, const int nm)`

4.8.1 Function Documentation

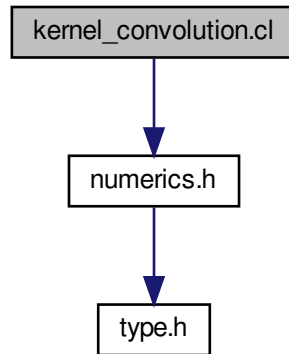
4.8.1.1 boundary_kernel() `__kernel void boundary_kernel (`
 `__global fp_t * d_conc,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

Definition at line 24 of file kernel_boundary.cl.

4.9 kernel_convolution.cl File Reference

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

Include dependency graph for kernel_convolution.cl:



Functions

- `__kernel void convolution_kernel (__global fp_t *d_conc_old, __global fp_t *d_conc_lap, __constant fp_t *d_mask, __local fp_t *d_conc_tile, const int nx, const int ny, const int nm)`
Enable double-precision floats.

4.9.1 Function Documentation

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

Enable double-precision floats.

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:

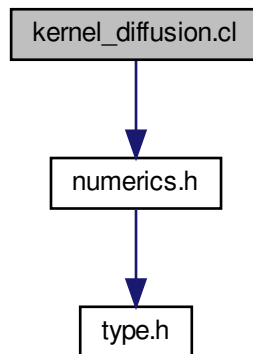
- The source matrix (*d_conc_old*) and destination matrix (*d_conc_lap*) must be identical in size
- One OpenCL worker operates on one array index: there is no nested loop over matrix elements
- The halo ($nm/2$ perimeter cells) in *d_conc_lap* are unallocated garbage
- The same cells in *d_conc_old* 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 37 of file `kernel_convolution.cl`.

4.10 kernel_diffusion.cl File Reference

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

Include dependency graph for `kernel_diffusion.cl`:



Functions

- `__kernel void diffusion_kernel (__global fp_t *d_conc_old, __global fp_t *d_conc_new, __global fp_t *d_conc_lap, const int nx, const int ny, const int nm, const fp_t D, const fp_t dt)`
Enable double-precision floats.

4.10.1 Function Documentation

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

Enable double-precision floats.

Diffusion equation kernel for execution on the GPU

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

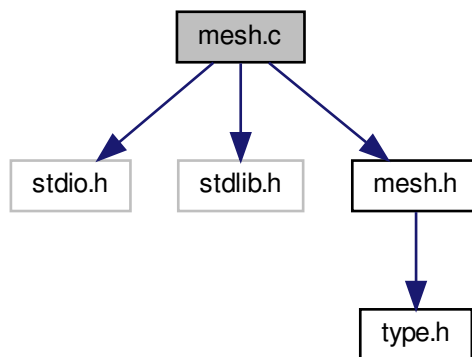
Definition at line 23 of file kernel_diffusion.cl.

4.11 mesh.c File Reference

Implementation of mesh handling functions for diffusion benchmarks.

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

Include dependency graph for mesh.c:



Functions

- void `make_arrays` (`fp_t ***conc_old`, `fp_t ***conc_new`, `fp_t ***conc_lap`, `fp_t ***mask_lap`, `const int nx`, `const int ny`, `const 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.

4.11.1 Detailed Description

Implementation of mesh handling functions for diffusion benchmarks.

4.11.2 Function Documentation

4.11.2.1 `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 44 of file mesh.c.

Here is the caller graph for this function:



4.11.2.2 make_arrays() `void make_arrays (`
 `fp_t *** conc_old,`
 `fp_t *** conc_new,`
 `fp_t *** conc_lap,`
 `fp_t *** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const 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 15 of file mesh.c.

Here is the caller graph for this function:



4.11.2.3 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 59 of file mesh.c.

Here is the caller graph for this function:



4.11.2.4 swap_pointers_1D() void swap_pointers_1D (

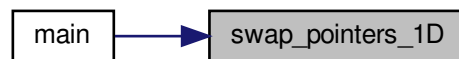
```
fp_t ** conc_old,
fp_t ** conc_new )
```

Swap pointers to data underlying 1D arrays.

Rather than copy data from `f_p_t* conc_old[0]` into `f_p_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 68 of file mesh.c.

Here is the caller graph for this function:

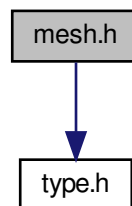


4.12 mesh.h File Reference

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`, `const int nx`, `const int ny`, `const 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.

4.12.1 Detailed Description

Declaration of mesh function prototypes for diffusion benchmarks.

4.12.2 Function Documentation

4.12.2.1 `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 44 of file mesh.c.

Here is the caller graph for this function:



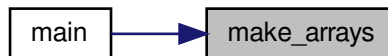
4.12.2.2 make_arrays() `void make_arrays (`
 `fp_t *** conc_old,`
 `fp_t *** conc_new,`
 `fp_t *** conc_lap,`
 `fp_t *** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

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 15 of file mesh.c.

Here is the caller graph for this function:



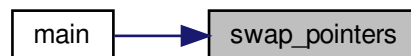
4.12.2.3 swap_pointers() `void swap_pointers (`
 `fp_t *** conc_old,`
 `fp_t *** conc_new)`

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 59 of file mesh.c.

Here is the caller graph for this function:



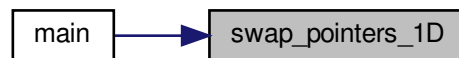
4.12.2.4 swap_pointers_1D() `void swap_pointers_1D (`
 `fp_t ** conc_old,`
 `fp_t ** conc_new)`

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 68 of file `mesh.c`.

Here is the caller graph for this function:

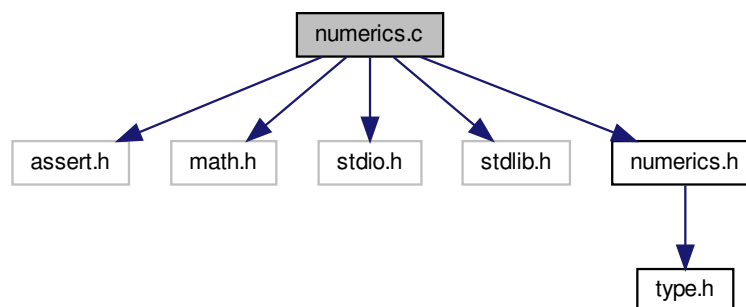


4.13 numerics.c File Reference

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

4.13.1 Detailed Description

Implementation of Laplacian operator and analytical solution functions.

4.13.2 Function Documentation

4.13.2.1 analytical_value() void analytical_value (

```

    const fp_t x,
    const fp_t t,
    const fp_t D,
    fp_t * c )
```

Analytical solution of the diffusion equation for a carburizing process.

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

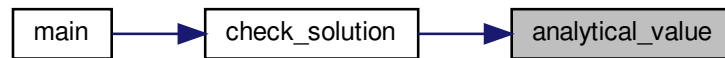
$$c(x, t) = c_0 - (c_0 - c_\infty) \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right)$$

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

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

Definition at line 109 of file numerics.c.

Here is the caller graph for this function:



4.13.2.2 `check_solution()`

```
void check_solution (
    fp_t ** conc_new,
    fp_t ** conc_lap,
    const int nx,
    const int ny,
    const fp_t dx,
    const fp_t dy,
    const int nm,
    const fp_t elapsed,
    const fp_t D,
    const fp_t * rss )
```

Compare numerical and analytical solutions of the diffusion equation.

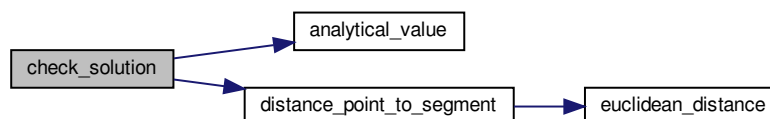
Returns

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

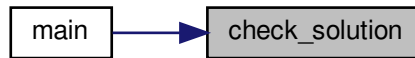
Overwrites `conc_lap`, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 114 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.2.3 distance_point_to_segment() `fp_t distance_point_to_segment (`
 `const fp_t ax,`
 `const fp_t ay,`
 `const fp_t bx,`
 `const fp_t by,`
 `const fp_t px,`
 `const 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 96 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:

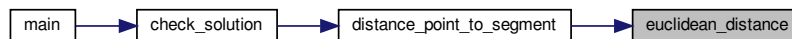


4.13.2.4 euclidean_distance() `fp_t euclidean_distance (`
 `const fp_t ax,`
 `const fp_t ay,`
 `const fp_t bx,`
 `const fp_t by)`

Compute Euclidean distance between two points, *a* and *b*.

Definition at line 84 of file numerics.c.

Here is the caller graph for this function:



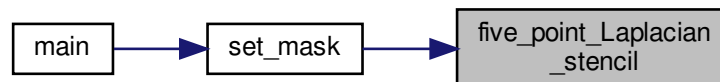
4.13.2.5 five_point_Laplacian_stencil() `void five_point_Laplacian_stencil (`
 `const fp_t dx,`
 `const fp_t dy,`
 `fp_t ** mask_lap,`
 `const int nm)`

Write 5-point Laplacian stencil into convolution mask.

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

Definition at line 37 of file numerics.c.

Here is the caller graph for this function:



4.13.2.6 `manhattan_distance()` `fp_t` `manhattan_distance` (

```

    const fp_t ax,
    const fp_t ay,
    const fp_t bx,
    const fp_t by )

```

Compute Manhattan distance between two points, *a* and *b*.

Definition at line 90 of file `numerics.c`.

4.13.2.7 `nine_point_Laplacian_stencil()` `void` `nine_point_Laplacian_stencil` (

```

    const fp_t dx,
    const fp_t dy,
    fp_t ** mask_lap,
    const int nm )

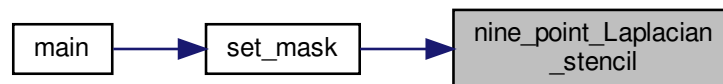
```

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 48 of file `numerics.c`.

Here is the caller graph for this function:



4.13.2.8 `set_mask()` `void` `set_mask` (

```

    const fp_t dx,
    const fp_t dy,
    const int code,
    fp_t ** mask_lap,
    const int nm )

```

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

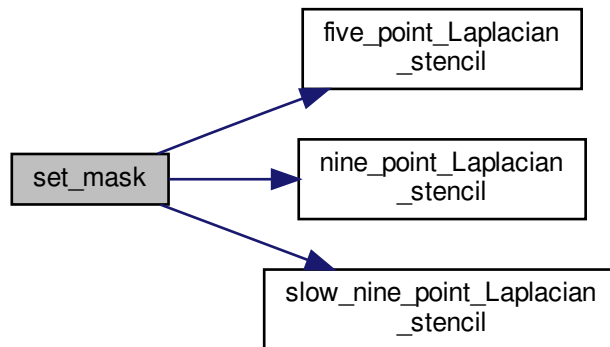
The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, e.g. 53 specifies [five_point_Laplacian_stencil\(\)](#), while 93 specifies [nine_point_Laplacian_stencil\(\)](#).

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

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

Definition at line 17 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:



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

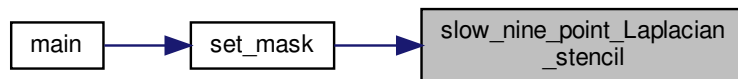
Write 9-point Laplacian stencil into convolution mask.

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

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

Definition at line 65 of file numerics.c.

Here is the caller graph for this function:

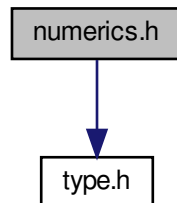


4.14 numerics.h File Reference

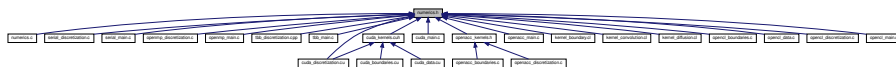
Declaration of Laplacian operator and analytical solution functions.

```
#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` (const `fp_t` dx, const `fp_t` dy, const int code, `fp_t` **mask_lap, const int nm)
Specify which stencil (mask) to use for the Laplacian (convolution)
- void `five_point_Laplacian_stencil` (const `fp_t` dx, const `fp_t` dy, `fp_t` **mask_lap, const int nm)
Write 5-point Laplacian stencil into convolution mask.
- void `nine_point_Laplacian_stencil` (const `fp_t` dx, const `fp_t` dy, `fp_t` **mask_lap, const int nm)
Write 9-point Laplacian stencil into convolution mask.
- void `slow_nine_point_Laplacian_stencil` (const `fp_t` dx, const `fp_t` dy, `fp_t` **mask_lap, const int nm)
Write 9-point Laplacian stencil into convolution mask.
- void `compute_convolution` (`fp_t` **const conc_old, `fp_t` **const conc_lap, `fp_t` **const mask_lap, const int nx, const int ny, const int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void `update_composition` (`fp_t` **const conc_old, `fp_t` **const conc_lap, `fp_t` **const conc_new, const int nx, const int ny, const int nm, const `fp_t` D, const `fp_t` dt)
Update composition field using explicit Euler discretization (forward-time centered space)
- `fp_t` `euclidean_distance` (const `fp_t` ax, const `fp_t` ay, const `fp_t` bx, const `fp_t` by)
Compute Euclidean distance between two points, a and b.
- `fp_t` `manhattan_distance` (const `fp_t` ax, const `fp_t` ay, const `fp_t` bx, const `fp_t` by)
Compute Manhattan distance between two points, a and b.
- `fp_t` `distance_point_to_segment` (const `fp_t` ax, const `fp_t` ay, const `fp_t` bx, const `fp_t` by, const `fp_t` px, const `fp_t` py)
Compute minimum distance from point p to a line segment bounded by points a and b.
- void `analytical_value` (const `fp_t` x, const `fp_t` t, const `fp_t` D, `fp_t` *c)
Analytical solution of the diffusion equation for a carburizing process.
- void `check_solution` (`fp_t` **const conc_new, `fp_t` **const conc_lap, const int nx, const int ny, const `fp_t` dx, const `fp_t` dy, const int nm, const `fp_t` elapsed, const `fp_t` D, `fp_t` *rss)
Compare numerical and analytical solutions of the diffusion equation.

4.14.1 Detailed Description

Declaration of Laplacian operator and analytical solution functions.

4.14.2 Macro Definition Documentation

4.14.2.1 MAX_MASK_H `#define MAX_MASK_H 5`

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

Definition at line 26 of file numerics.h.

4.14.2.2 MAX_MASK_W `#define MAX_MASK_W 5`

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

Definition at line 21 of file numerics.h.

4.14.3 Function Documentation

4.14.3.1 analytical_value() `void analytical_value (` `const fp_t x,` `const fp_t t,` `const fp_t D,` `fp_t * c)`

Analytical solution of the diffusion equation for a carburizing process.

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

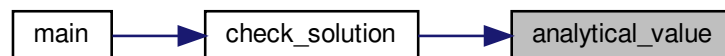
$$c(x, t) = c_0 - (c_0 - c_\infty) \operatorname{erf} \left(\frac{x}{\sqrt{4Dt}} \right)$$

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

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

Definition at line 109 of file numerics.c.

Here is the caller graph for this function:



4.14.3.2 check_solution() `void check_solution (`
 `fp_t ** conc_new,`
 `fp_t ** conc_lap,`
 `const int nx,`
 `const int ny,`
 `const fp_t dx,`
 `const fp_t dy,`
 `const int nm,`
 `const fp_t elapsed,`
 `const fp_t D,`
 `fp_t * rss)`

Compare numerical and analytical solutions of the diffusion equation.

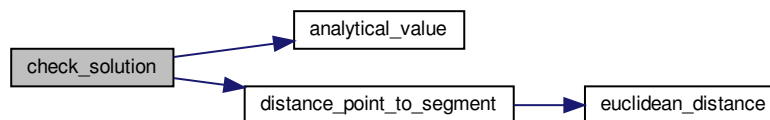
Returns

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

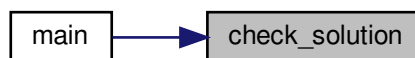
Overwrites *conc_lap*, into which the point-wise RSS is written. Normalized RSS is then computed as the sum of the point-wise values.

Definition at line 114 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:



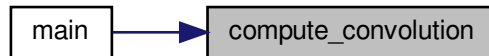
4.14.3.3 compute_convolution() `void compute_convolution (`
`fp_t **const conc_old,`
`fp_t ** conc_lap,`
`fp_t **const mask_lap,`
`const int nx,`
`const int ny,`
`const int nm)`

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 17 of file `serial_discretization.c`.

Here is the caller graph for this function:



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

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

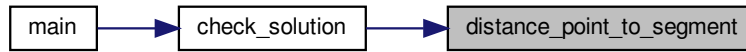
This function computes the projection of *p* onto *ab*, limiting the projected range to [0, 1] to handle projections that fall outside of *ab*. Implemented after Grumdrig on Stackoverflow, <https://stackoverflow.com/a/1501725>.

Definition at line 96 of file `numerics.c`.

Here is the call graph for this function:



Here is the caller graph for this function:

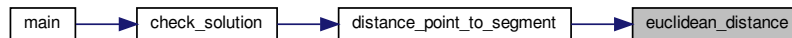


4.14.3.5 euclidean_distance() `fp_t euclidean_distance (`
 `const fp_t ax,`
 `const fp_t ay,`
 `const fp_t bx,`
 `const fp_t by)`

Compute Euclidean distance between two points, *a* and *b*.

Definition at line 84 of file `numerics.c`.

Here is the caller graph for this function:



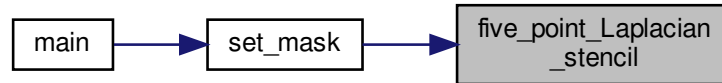
4.14.3.6 five_point_Laplacian_stencil() `void five_point_Laplacian_stencil (`
 `const fp_t dx,`
 `const fp_t dy,`
 `fp_t ** mask_lap,`
 `const int nm)`

Write 5-point Laplacian stencil into convolution mask.

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

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

Here is the caller graph for this function:



4.14.3.7 `manhattan_distance()` `fp_t` `manhattan_distance` (
 const `fp_t` `ax`,
 const `fp_t` `ay`,
 const `fp_t` `bx`,
 const `fp_t` `by`)

Compute Manhattan distance between two points, *a* and *b*.

Definition at line 90 of file `numerics.c`.

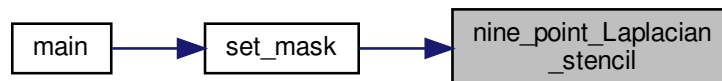
4.14.3.8 `nine_point_Laplacian_stencil()` `void` `nine_point_Laplacian_stencil` (
 const `fp_t` `dx`,
 const `fp_t` `dy`,
 `fp_t` ** `mask_lap`,
 const `int` `nm`)

Write 9-point Laplacian stencil into convolution mask.

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

Definition at line 48 of file `numerics.c`.

Here is the caller graph for this function:



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

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

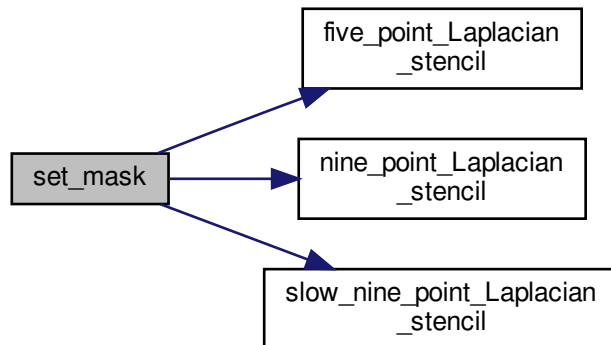
The mask corresponding to the numerical code will be applied. The suggested encoding is mask width as the ones digit and value count as the tens digit, e.g. 53 specifies [five_point_Laplacian_stencil\(\)](#), while 93 specifies [nine_point_Laplacian_stencil\(\)](#).

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

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

Definition at line 17 of file numerics.c.

Here is the call graph for this function:



Here is the caller graph for this function:



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

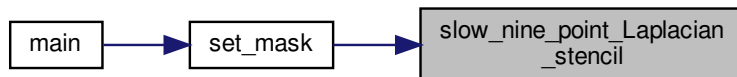
Write 9-point Laplacian stencil into convolution mask.

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

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

Definition at line 65 of file numerics.c.

Here is the caller graph for this function:



4.14.3.11 update_composition() `void update_composition (`
`fp_t ** conc_old,`
`fp_t ** conc_lap,`
`fp_t ** conc_new,`
`const int nx,`
`const int ny,`
`const int nm,`
`const fp_t D,`
`const fp_t dt)`

Update composition field using explicit Euler discretization (forward-time centered space)

Definition at line 33 of file serial_discretization.c.

Here is the caller graph for this function:

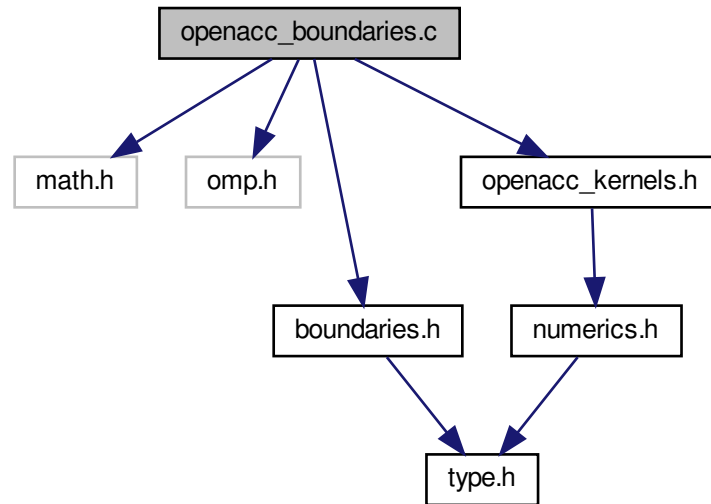


4.15 openacc_boundaries.c File Reference

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 `apply_initial_conditions` (`fp_t` **conc, const int nx, const int ny, const int nm)
Initialize flat composition field with fixed boundary conditions.
- void `boundary_kernel` (`fp_t` **__restrict__ conc, const int nx, const int ny, const int nm)
- void `apply_boundary_conditions` (`fp_t` **conc, const int nx, const int ny, const int nm)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.15.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

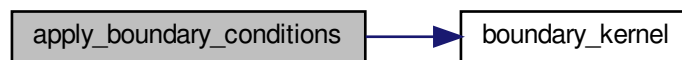
4.15.2 Function Documentation

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

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

Definition at line 90 of file `openacc_boundaries.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.2.2 apply_initial_conditions() `void apply_initial_conditions (`
 `fp_t ** conc_old,`
 `const int nx,`
 `const int ny,`
 `const int nm)`

Initialize flat composition field with fixed boundary conditions.

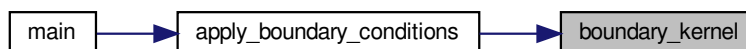
The boundary conditions are fixed values of c_{hi} along the lower-left half and upper-right half walls, no flux everywhere else, with an initial values of c_{lo} everywhere. These conditions represent a carburizing process, with partial exposure (rather than the entire left and right walls) to produce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

Definition at line 16 of file `openacc_boundaries.c`.

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

Definition at line 37 of file openacc_boundaries.c.

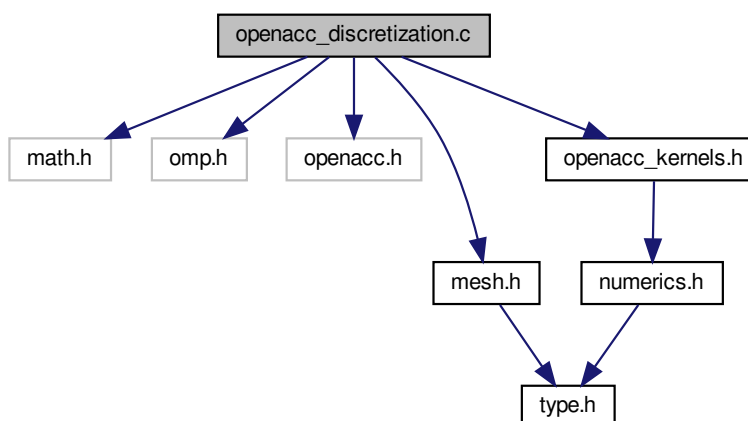
Here is the caller graph for this function:



4.16 openacc_discretization.c File Reference

Implementation of boundary condition functions with OpenACC threading.

```
#include <math.h>
#include <omp.h>
#include <openacc.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, 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.

4.16.1 Detailed Description

Implementation of boundary condition functions with OpenACC threading.

4.16.2 Function Documentation

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

Tiled convolution algorithm for execution on the GPU.

Definition at line 17 of file openacc_discretization.c.

4.16.2.2 `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.

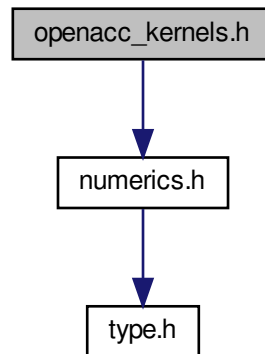
Definition at line 41 of file openacc_discretization.c.

4.17 openacc_kernels.h File Reference

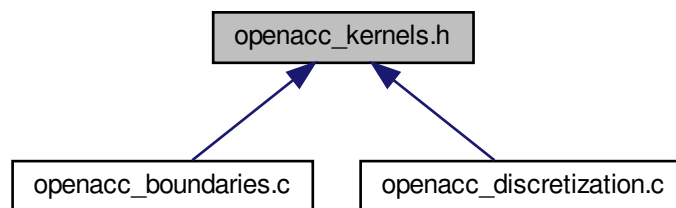
Declaration of functions to execute on the GPU (OpenACC kernels)

```
#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, 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, [fp_t](#) **mask_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.

4.17.1 Detailed Description

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

4.17.2 Function Documentation

4.17.2.1 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.

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

Tiled convolution algorithm for execution on the GPU.

Definition at line 17 of file openacc_discretization.c.

4.17.2.3 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.

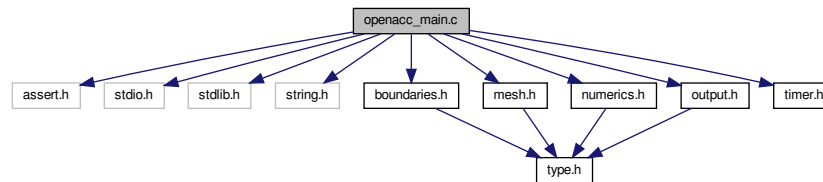
Definition at line 41 of file openacc_discretization.c.

4.18 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 "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.

4.18.1 Detailed Description

OpenACC implementation of semi-infinite diffusion equation.

4.18.2 Function Documentation

```

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

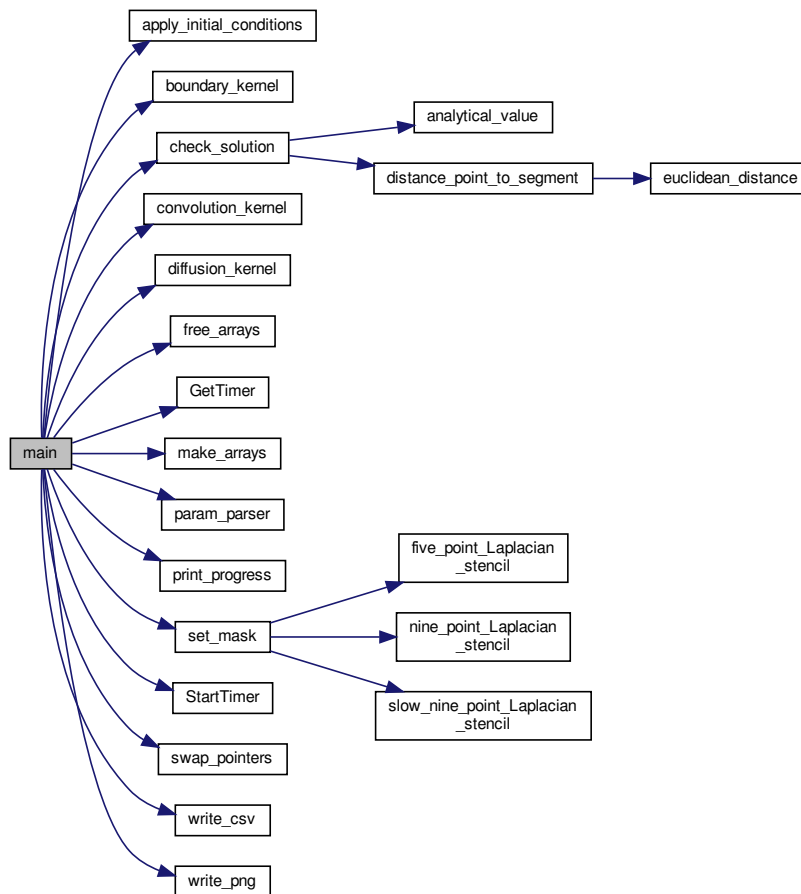
```

Run simulation using input parameters specified on the command line.

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

Definition at line 33 of file `openacc_main.c`.

Here is the call graph for this function:

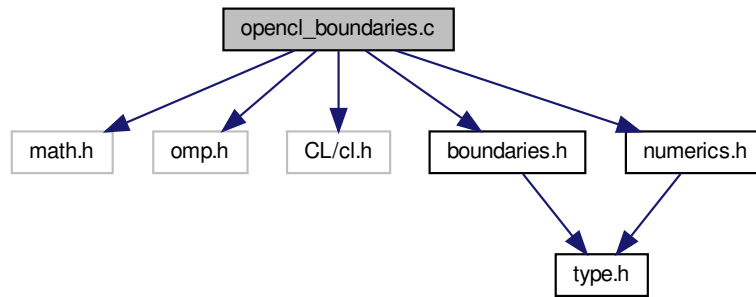


4.19 opencil_boundaries.c File Reference

Implementation of boundary condition functions with OpenCL acceleration.

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

Include dependency graph for `openc1_boundaries.c`:



Functions

- void `apply_initial_conditions` (`fp_t` **conc, const int nx, const int ny, const int nm)
Initialize flat composition field with fixed boundary conditions.

4.19.1 Detailed Description

Implementation of boundary condition functions with OpenCL acceleration.

4.19.2 Function Documentation

4.19.2.1 `apply_initial_conditions()` void `apply_initial_conditions` (
`fp_t` ** conc_old,
const int nx,
const int ny,
const int nm)

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 17 of file `openc1_boundaries.c`.

Here is the caller graph for this function:



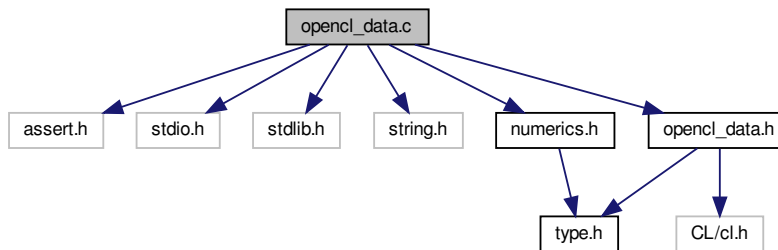
4.20 openc1_data.c File Reference

Implementation of functions to create and destroy `OpenCLData` struct.

```

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "numerics.h"
#include "openc1_data.h"
  
```

Include dependency graph for `openc1_data.c`:



Functions

- void `report_error` (`cl_int` status, `const char *`message)
Report error code when status is not `CL_SUCCESS`.
- void `init_openc1` (`fp_t **`conc_old, `fp_t **`mask_lap, `const int` nx, `const int` ny, `const 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_opengl` (struct `OpenCLData` *dev)
Free OpenCL device memory after marching.

4.20.1 Detailed Description

Implementation of functions to create and destroy `OpenCLData` struct.

4.20.2 Function Documentation

4.20.2.1 `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.

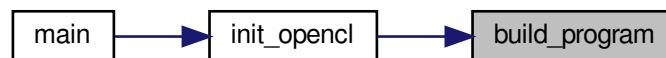
Source follows the OpenCL Programming Book, <https://www.fixstars.com/en/opengl/book/OpenCLProgrammingBook/calling-the-kernel/>

Definition at line 137 of file `opengl_data.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.20.2.2 free_opengl() `void free_opengl (`
 `struct OpenGLData * dev)`

Free OpenGL device memory after marching.

Definition at line 211 of file `opengl_data.c`.

Here is the caller graph for this function:

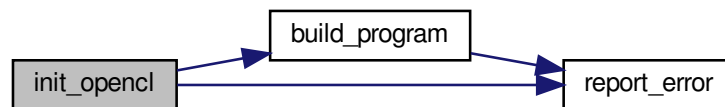


4.20.2.3 init_opengl() `void init_opengl (`
 `fp_t ** conc_old,`
 `fp_t ** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `struct OpenGLData * dev)`

Initialize OpenGL device memory before marching.

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

Here is the call graph for this function:



Here is the caller graph for this function:



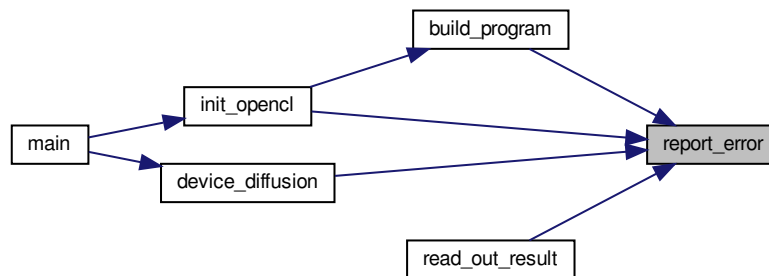
4.20.2.4 report_error() void report_error (
 cl_int error,
 const char * message)

Report error code when status is not CL_SUCCESS.

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

Definition at line 18 of file opengl_data.c.

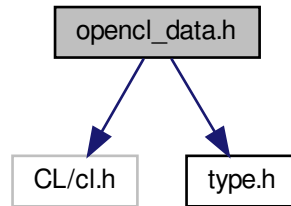
Here is the caller graph for this function:



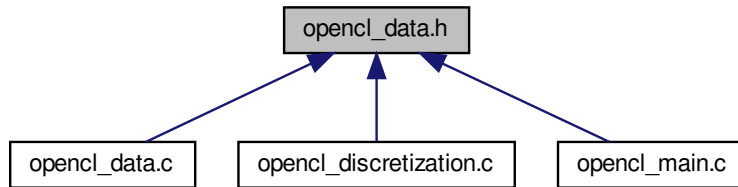
4.21 opengl_data.h File Reference

Declaration of OpenGL data container.

```
#include <CL/cl.h>
#include "type.h"
Include dependency graph for openc1_data.h:
```



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



Classes

- struct [OpenCLData](#)
Container for GPU array pointers and parameters.

Functions

- void [report_error](#) (cl_int error, const char *message)
Report error code when status is not CL_SUCCESS.
- 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 [init_openc1](#) (fp_t **conc_old, fp_t **mask_lap, const int nx, const int ny, const int nm, struct [OpenCLData](#) *dev)
Initialize OpenCL device memory before marching.

- void `device_boundaries` (struct `OpenCLData` *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by)
Apply boundary conditions on OpenCL device.
- void `device_convolution` (struct `OpenCLData` *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by)
Compute convolution on OpenCL device.
- void `device_diffusion` (struct `OpenCLData` *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by, const `fp_t` D, const `fp_t` dt)
Solve diffusion equation on OpenCL device.
- void `read_out_result` (struct `OpenCLData` *dev, const int flip, `fp_t` **conc_new, const int nx, const int ny)
Copy data out of OpenCL device.
- void `free_openc1` (struct `OpenCLData` *dev)
Free OpenCL device memory after marching.

4.21.1 Detailed Description

Declaration of OpenCL data container.

4.21.2 Function Documentation

4.21.2.1 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.

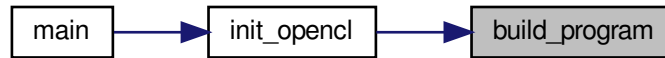
Source follows the OpenCL Programming Book, <https://www.fixstars.com/en/openc1/book/OpenCLProgrammingBook/calling-the-kernel/>

Definition at line 137 of file openc1_data.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.21.2.2 device_boundaries() `void device_boundaries (`
 `struct OpenCLData * dev,`
 `const int flip,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const int bx,`
 `const int by)`

Apply boundary conditions on OpenCL device.

Definition at line 27 of file `opengl_discretization.c`.

4.21.2.3 device_convolution() `void device_convolution (`
 `struct OpenCLData * dev,`
 `const int flip,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const int bx,`
 `const int by)`

Compute convolution on OpenCL device.

Definition at line 54 of file `opengl_discretization.c`.

4.21.2.4 device_diffusion() void device_diffusion (
 struct `OpenGLData` * dev,
 const int flip,
 const int nx,
 const int ny,
 const int nm,
 const int bx,
 const int by,
 const `fp_t` D,
 const `fp_t` dt)

Solve diffusion equation on OpenGL device.

Definition at line 83 of file `opengl_discretization.c`.

Here is the call graph for this function:



Here is the caller graph for this function:

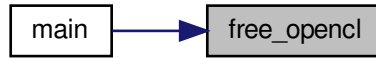


4.21.2.5 free_opengl() void free_opengl (
 struct `OpenGLData` * dev)

Free OpenGL device memory after marching.

Definition at line 211 of file `opengl_data.c`.

Here is the caller graph for this function:

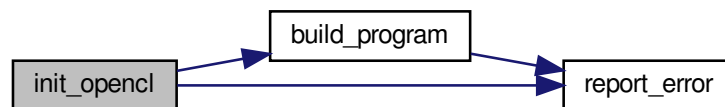


4.21.2.6 init_opengl() `void init_opengl (`
 `fp_t ** conc_old,`
 `fp_t ** mask_lap,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `struct OpenCLData * dev)`

Initialize OpenCL device memory before marching.

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

Here is the call graph for this function:



Here is the caller graph for this function:



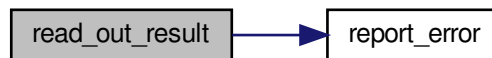
4.21.2.7 read_out_result() void read_out_result (

```
    struct OpenGLData * dev,  
    const int flip,  
    fp_t ** conc_new,  
    const int nx,  
    const int ny )
```

Copy data out of OpenGL device.

Definition at line 114 of file opengl_discretization.c.

Here is the call graph for this function:



4.21.2.8 report_error() void report_error (

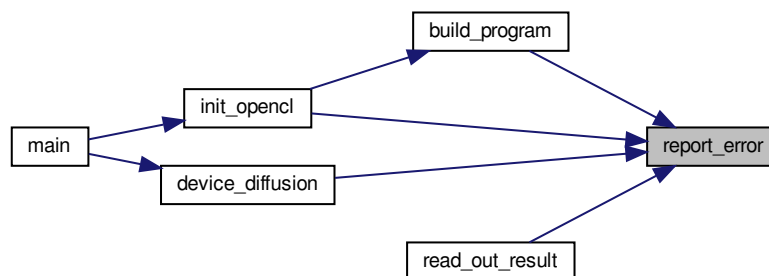
```
    cl_int error,  
    const char * message )
```

Report error code when status is not CL_SUCCESS.

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

Definition at line 18 of file opengl_data.c.

Here is the caller graph for this function:

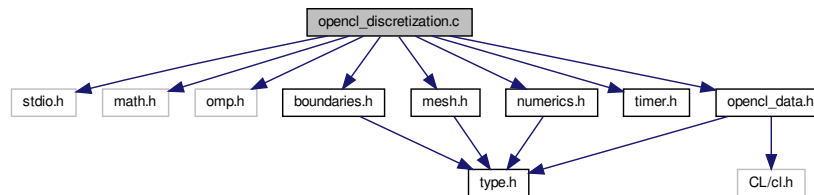


4.22 opencil_discretization.c File Reference

Implementation of boundary condition functions with OpenCL acceleration.

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

Include dependency graph for opencil_discretization.c:



Functions

- void [device_boundaries](#) (struct [OpenCLData](#) *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by)
Apply boundary conditions on OpenCL device.
- void [device_convolution](#) (struct [OpenCLData](#) *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by)
Compute convolution on OpenCL device.
- void [device_diffusion](#) (struct [OpenCLData](#) *dev, const int flip, const int nx, const int ny, const int nm, const int bx, const int by, const [fp_t](#) D, const [fp_t](#) dt)
Solve diffusion equation on OpenCL device.
- void [read_out_result](#) (struct [OpenCLData](#) *dev, const int flip, [fp_t](#) **conc, const int nx, const int ny)
Copy data out of OpenCL device.

4.22.1 Detailed Description

Implementation of boundary condition functions with OpenCL acceleration.

4.22.2 Function Documentation

4.22.2.1 device_boundaries() void device_boundaries (

```
    struct OpenCLData * dev,  
    const int flip,  
    const int nx,  
    const int ny,  
    const int nm,  
    const int bx,  
    const int by )
```

Apply boundary conditions on OpenCL device.

Definition at line 27 of file opencil_discretization.c.

4.22.2.2 device_convolution() void device_convolution (

```
    struct OpenCLData * dev,  
    const int flip,  
    const int nx,  
    const int ny,  
    const int nm,  
    const int bx,  
    const int by )
```

Compute convolution on OpenCL device.

Definition at line 54 of file opencil_discretization.c.

4.22.2.3 device_diffusion() void device_diffusion (

```
    struct OpenCLData * dev,  
    const int flip,  
    const int nx,  
    const int ny,  
    const int nm,  
    const int bx,  
    const int by,  
    const fp_t D,  
    const fp_t dt )
```

Solve diffusion equation on OpenCL device.

Definition at line 83 of file opencil_discretization.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.22.2.4 read_out_result() `void read_out_result (`
 `struct OpenCLData * dev,`
 `const int flip,`
 `fp_t ** conc,`
 `const int nx,`
 `const int ny)`

Copy data out of OpenCL device.

Definition at line 114 of file `openc1_discretization.c`.

Here is the call graph for this function:

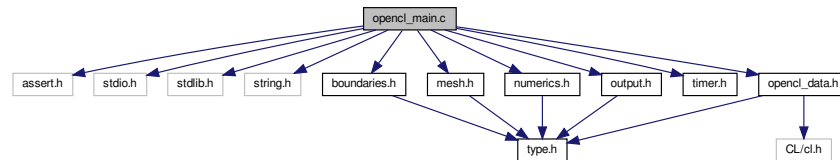


4.23 openc1_main.c File Reference

OpenCL implementation of semi-infinite diffusion equation.

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

```
#include "timer.h"
#include "opencil_data.h"
Include dependency graph for opencil_main.c:
```



Functions

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

Run simulation using input parameters specified on the command line.

4.23.1 Detailed Description

OpenCL implementation of semi-infinite diffusion equation.

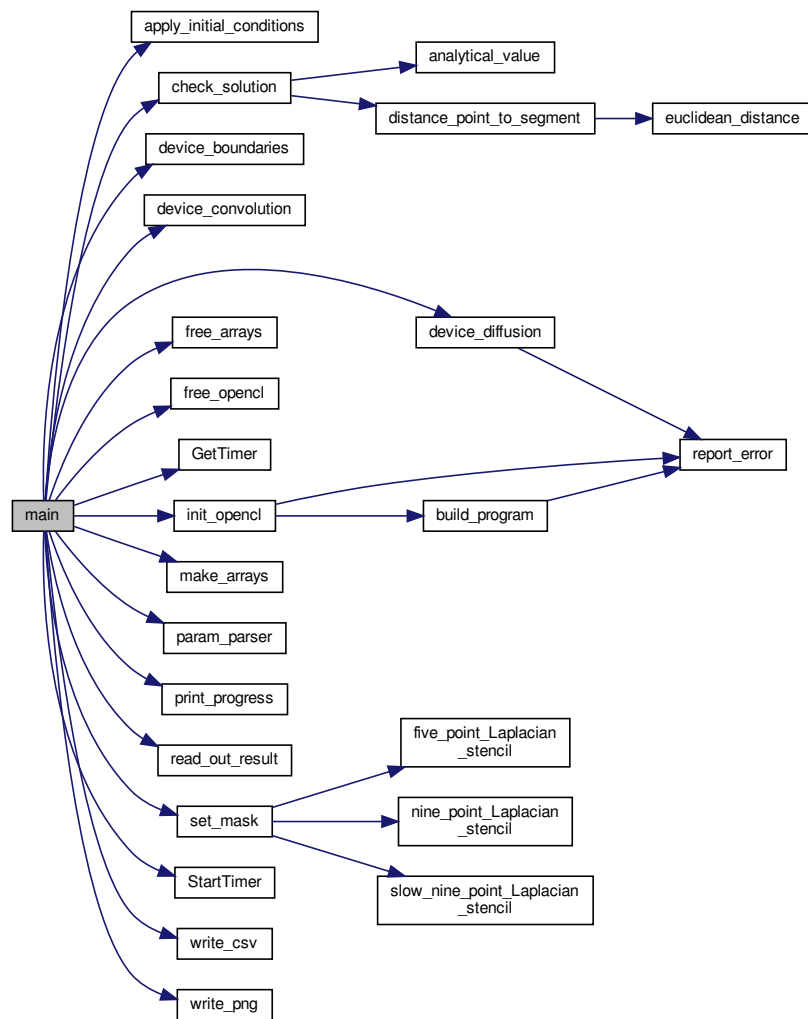
4.23.2 Function Documentation

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

Run simulation using input parameters specified on the command line.

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

Here is the call graph for this function:



4.24 openmp_boundaries.c File Reference

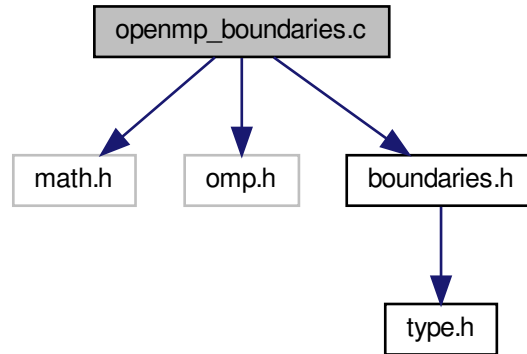
Implementation of boundary condition functions with OpenMP threading.

```

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

```

Include dependency graph for openmp_boundaries.c:



Functions

- void [apply_initial_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.24.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.24.2 Function Documentation

4.24.2.1 [apply_boundary_conditions\(\)](#) void [apply_boundary_conditions](#) (
[fp_t](#) ** conc,
 const int nx,
 const int ny,
 const int nm)

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

Definition at line 36 of file openmp_boundaries.c.

4.24.2.2 apply_initial_conditions() void apply_initial_conditions (

```

    fp_t ** conc_old,
    const int nx,
    const int ny,
    const int nm )

```

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 15 of file openmp_boundaries.c.

4.25 openmp_discretization.c File Reference

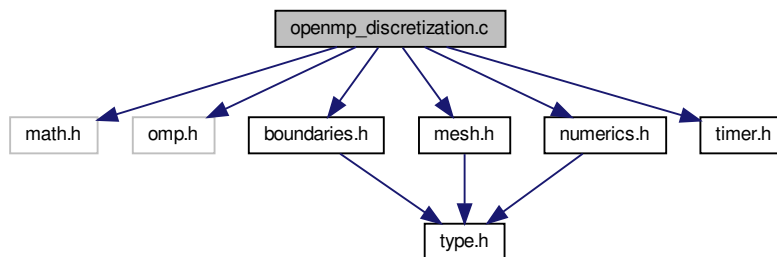
Implementation of boundary condition functions with OpenMP threading.

```

#include <math.h>
#include <omp.h>
#include "boundaries.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, const int nx, const int ny, const int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void **update_composition** (fp_t **conc_old, fp_t **conc_lap, fp_t **conc_new, const int nx, const int ny, const int nm, const fp_t D, const fp_t dt)
Update composition field using explicit Euler discretization (forward-time centered space)

4.25.1 Detailed Description

Implementation of boundary condition functions with OpenMP threading.

4.25.2 Function Documentation

4.25.2.1 compute_convolution() `void compute_convolution (`
 `fp_t **const conc_old,`
 `fp_t ** conc_lap,`
 `fp_t **const mask_lap,`
 `const int nx,`
 `const int ny,`
 `const 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 18 of file openmp_discretization.c.

4.25.2.2 update_composition() `void update_composition (`
 `fp_t ** conc_old,`
 `fp_t ** conc_lap,`
 `fp_t ** conc_new,`
 `const int nx,`
 `const int ny,`
 `const int nm,`
 `const fp_t D,`
 `const fp_t dt)`

Update composition field using explicit Euler discretization (forward-time centered space)

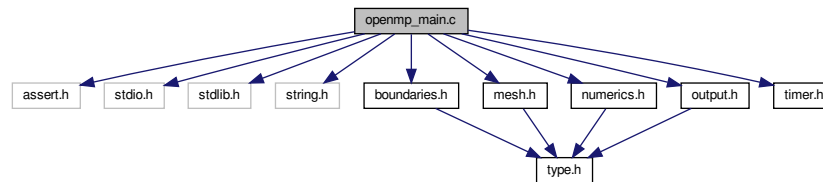
Definition at line 38 of file openmp_discretization.c.

4.26 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 "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.

4.26.1 Detailed Description

OpenMP implementation of semi-infinite diffusion equation.

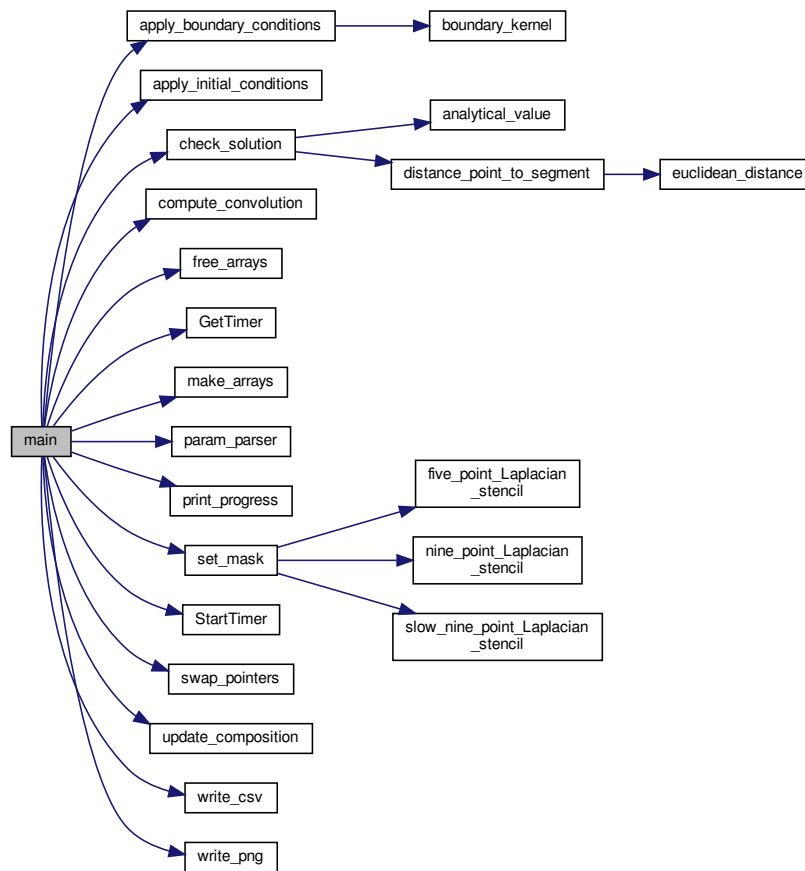
4.26.2 Function Documentation

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

Run simulation using input parameters specified on the command line.

Definition at line 25 of file `openmp_main.c`.

Here is the call graph for this function:



4.27 output.c File Reference

Implementation of file output functions for diffusion benchmarks.

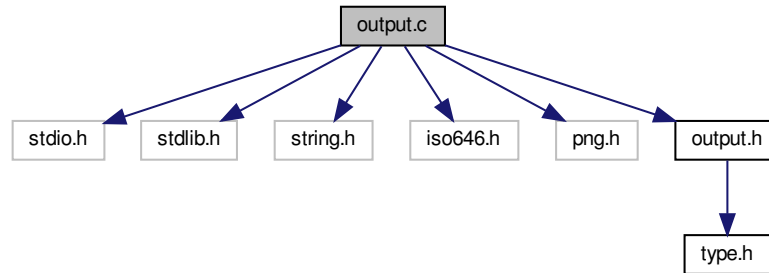
```

#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 *bx, int *by, int *checks, int *code, `fp_t` *D, `fp_t` *dx, `fp_t` *dy, `fp_t` *linStab, int *nm, int *nx, int *ny, int *steps)
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, const int nx, const int ny, const `fp_t` dx, const `fp_t` dy, const int step)
Writes scalar composition field to diffusion.????????.csv.
- void `write_png` (`fp_t` **conc, const int nx, const int ny, const int step)
Writes scalar composition field to diffusion.????????.png.

4.27.1 Detailed Description

Implementation of file output functions for diffusion benchmarks.

4.27.2 Function Documentation

4.27.2.1 param_parser() void param_parser (

```

    int argc,
    char * argv[],
    int * bx,
    int * by,
    int * checks,
    int * code,
    fp_t * D,

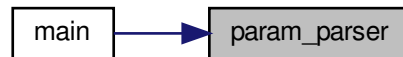
```

```
fp_t * dx,  
fp_t * dy,  
fp_t * linStab,  
int * nm,  
int * nx,  
int * ny,  
int * steps )
```

Read parameters from file specified on the command line.

Definition at line 18 of file output.c.

Here is the caller graph for this function:



4.27.2.2 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 124 of file output.c.

Here is the caller graph for this function:



4.27.2.3 write_csv() `void write_csv (`
 `fp_t ** conc,`
 `const int nx,`
 `const int ny,`
 `const fp_t dx,`
 `const fp_t dy,`
 `const 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:



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

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

Definition at line 181 of file output.c.

Here is the caller graph for this function:

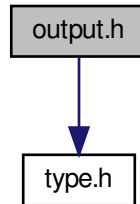


4.28 output.h File Reference

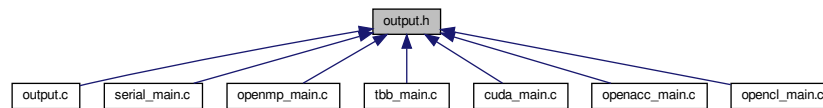
Declaration of output function prototypes for diffusion benchmarks.

```
#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 *bx, int *by, int *checks, int *code, [fp_t](#) *D, [fp_t](#) *dx, [fp_t](#) *dy, [fp_t](#) *linStab, int *nm, int *nx, int *ny, int *steps)
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, const int nx, const int ny, const [fp_t](#) dx, const [fp_t](#) dy, const int step)
Writes scalar composition field to diffusion.???????.csv.
- void [write_png](#) ([fp_t](#) **conc, const int nx, const int ny, const int step)
Writes scalar composition field to diffusion.???????.png.

4.28.1 Detailed Description

Declaration of output function prototypes for diffusion benchmarks.

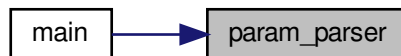
4.28.2 Function Documentation

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

Read parameters from file specified on the command line.

Definition at line 18 of file output.c.

Here is the caller graph for this function:



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

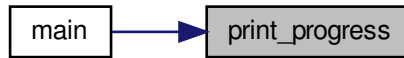
Prints timestamps and a 20-point progress bar to stdout.

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 124 of file output.c.

Here is the caller graph for this function:



4.28.2.3 write_csv() void write_csv (
 fp_t ** conc,
 const int nx,
 const int ny,
 const fp_t dx,
 const fp_t dy,
 const 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:



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

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

Definition at line 181 of file output.c.

Here is the caller graph for this function:



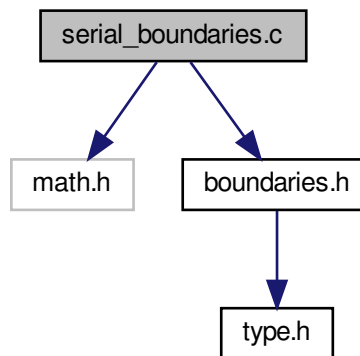
4.29 serial_boundaries.c File Reference

Implementation of boundary condition functions without threading.

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

```
#include "boundaries.h"
```

Include dependency graph for `serial_boundaries.c`:



Functions

- void [apply_initial_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.29.1 Detailed Description

Implementation of boundary condition functions without threading.

4.29.2 Function Documentation

4.29.2.1 apply_boundary_conditions() void apply_boundary_conditions (
 fp_t ** conc,
 const int nx,
 const int ny,
 const int nm)

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

Definition at line 29 of file serial_boundaries.c.

4.29.2.2 apply_initial_conditions() void apply_initial_conditions (
 fp_t ** conc_old,
 const int nx,
 const int ny,
 const int nm)

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 14 of file serial_boundaries.c.

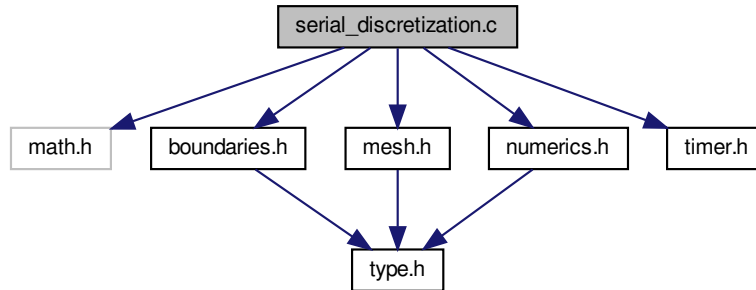
4.30 serial_discretization.c File Reference

Implementation of boundary condition functions without threading.

```
#include <math.h>
#include "boundaries.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`, `const int nx`, `const int ny`, `const int nm`)
Perform the convolution of the mask matrix with the composition matrix.
- void `update_composition` (`fp_t **conc_old`, `fp_t **conc_lap`, `fp_t **conc_new`, `const int nx`, `const int ny`, `const int nm`, `const fp_t D`, `const fp_t dt`)
Update composition field using explicit Euler discretization (forward-time centered space)

4.30.1 Detailed Description

Implementation of boundary condition functions without threading.

4.30.2 Function Documentation

4.30.2.1 compute_convolution() `void compute_convolution (`
`fp_t **const conc_old,`
`fp_t ** conc_lap,`
`fp_t **const mask_lap,`
`const int nx,`
`const int ny,`
`const 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 17 of file `serial_discretization.c`.

4.30.2.2 update_composition() void update_composition (

```

    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** conc_new,
    const int nx,
    const int ny,
    const int nm,
    const fp_t D,
    const fp_t dt )

```

Update composition field using explicit Euler discretization (forward-time centered space)

Definition at line 33 of file serial_discretization.c.

4.31 serial_main.c File Reference

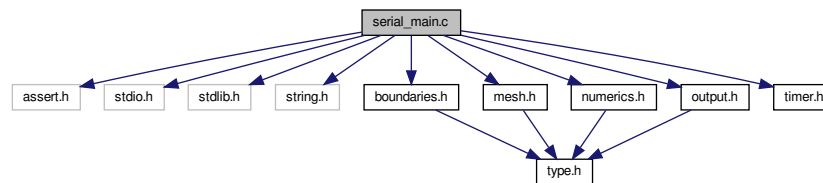
Serial implementation of semi-infinite diffusion equation.

```

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "boundaries.h"
#include "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.

4.31.1 Detailed Description

Serial implementation of semi-infinite diffusion equation.

4.31.2 Function Documentation

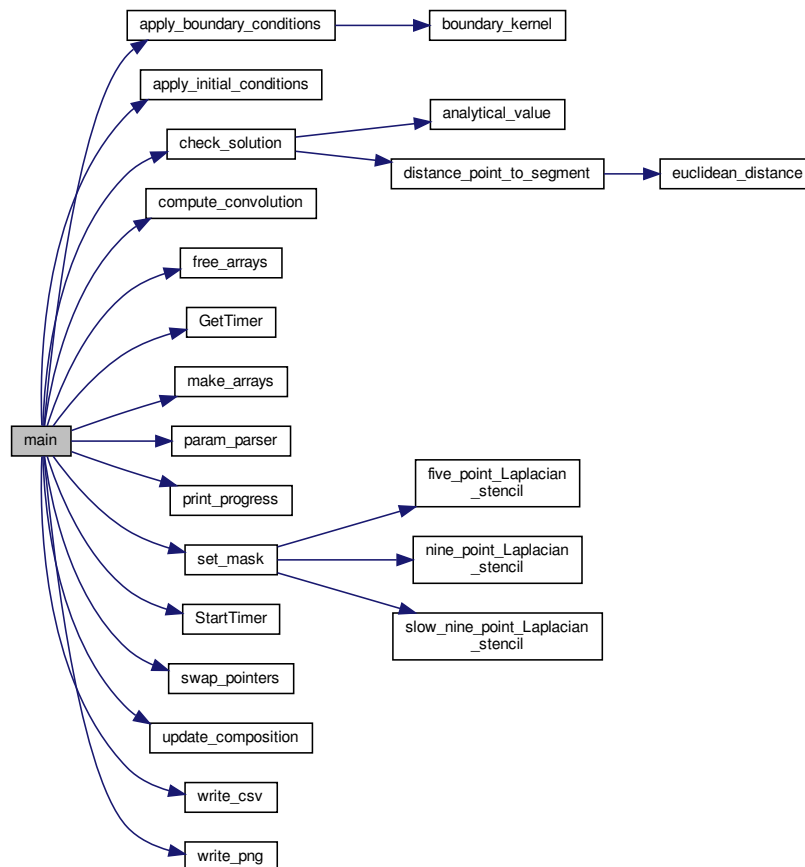
4.31.2.1 main() `int main (`
`int argc,`
`char * argv[])`

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 33 of file serial_main.c.

Here is the call graph for this function:

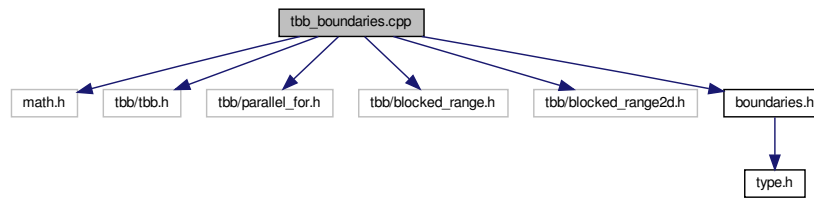


4.32 tbb_boundaries.cpp File Reference

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 [apply_initial_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Initialize flat composition field with fixed boundary conditions.
- void [apply_boundary_conditions](#) ([fp_t](#) **conc, const int nx, const int ny, const int nm)
Set fixed value (c_{hi}) along left and bottom, zero-flux elsewhere.

4.32.1 Detailed Description

Implementation of boundary condition functions with TBB threading.

4.32.2 Function Documentation

4.32.2.1 [apply_boundary_conditions\(\)](#) void [apply_boundary_conditions](#) (
[fp_t](#) ** conc,
 const int nx,
 const int ny,
 const int nm)

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

Definition at line 54 of file tbb_boundaries.cpp.

4.32.2.2 apply_initial_conditions() void apply_initial_conditions (

```

    fp_t ** conc_old,
    const int nx,
    const int ny,
    const int nm )

```

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 18 of file tbb_boundaries.cpp.

4.33 tbb_discretization.cpp File Reference

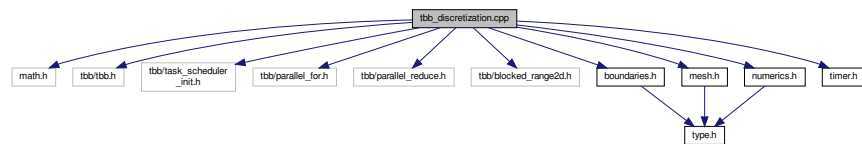
Implementation of boundary condition functions with TBB threading.

```

#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 "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, const int nx, const int ny, const int nm)
Perform the convolution of the mask matrix with the composition matrix.
- void **update_composition** (fp_t **conc_old, fp_t **conc_lap, fp_t **conc_new, const int nx, const int ny, const int nm, const fp_t D, const fp_t dt)
Update composition field using explicit Euler discretization (forward-time centered space)
- void **check_solution_lambda** (fp_t **conc_new, fp_t **conc_lap, const int nx, const int ny, const fp_t dx, const fp_t dy, const int nm, const fp_t elapsed, const fp_t D, fp_t *rss)

4.33.1 Detailed Description

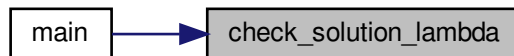
Implementation of boundary condition functions with TBB threading.

4.33.2 Function Documentation

4.33.2.1 check_solution_lambda() `void check_solution_lambda (`
 `fp_t ** conc_new,`
 `fp_t ** conc_lap,`
 `const int nx,`
 `const int ny,`
 `const fp_t dx,`
 `const fp_t dy,`
 `const int nm,`
 `const fp_t elapsed,`
 `const fp_t D,`
 `fp_t * rss)`

Definition at line 59 of file tbb_discretization.cpp.

Here is the caller graph for this function:



4.33.2.2 compute_convolution() `void compute_convolution (`
 `fp_t **const conc_old,`
 `fp_t ** conc_lap,`
 `fp_t **const mask_lap,`
 `const int nx,`
 `const int ny,`
 `const 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 22 of file `tbb_discretization.cpp`.

Here is the caller graph for this function:



```
4.33.2.3 update_composition() void update_composition (
    fp_t ** conc_old,
    fp_t ** conc_lap,
    fp_t ** conc_new,
    const int nx,
    const int ny,
    const int nm,
    const fp_t D,
    const fp_t dt )
```

Update composition field using explicit Euler discretization (forward-time centered space)

Definition at line 43 of file `tbb_discretization.cpp`.

Here is the caller graph for this function:



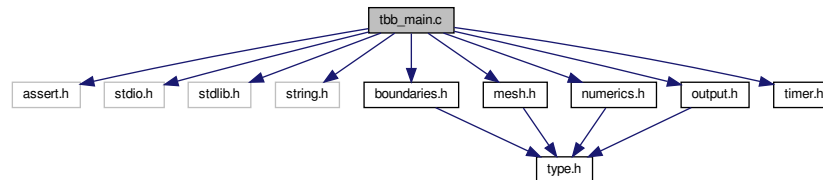
4.34 tbb_main.c File Reference

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 "mesh.h"
#include "numerics.h"
#include "output.h"
#include "timer.h"
```

Include dependency graph for tbb_main.c:



Functions

- void `check_solution_lambda` (`fp_t **conc_new`, `fp_t **conc_lap`, `const int nx`, `const int ny`, `const fp_t dx`, `const fp_t dy`, `const int nm`, `const fp_t elapsed`, `const fp_t D`, `fp_t *rss`)
- int `main` (`int argc`, `char *argv[]`)

Run simulation using input parameters specified on the command line.

4.34.1 Detailed Description

Threading Building Blocks implementation of semi-infinite diffusion equation.

4.34.2 Function Documentation

4.34.2.1 `check_solution_lambda()` void `check_solution_lambda` (
`fp_t ** conc_new`,
`fp_t ** conc_lap`,
`const int nx`,
`const int ny`,
`const fp_t dx`,
`const fp_t dy`,
`const int nm`,
`const fp_t elapsed`,
`const fp_t D`,
`fp_t * rss`)

Definition at line 59 of file `tbb_discretization.cpp`.

Here is the caller graph for this function:

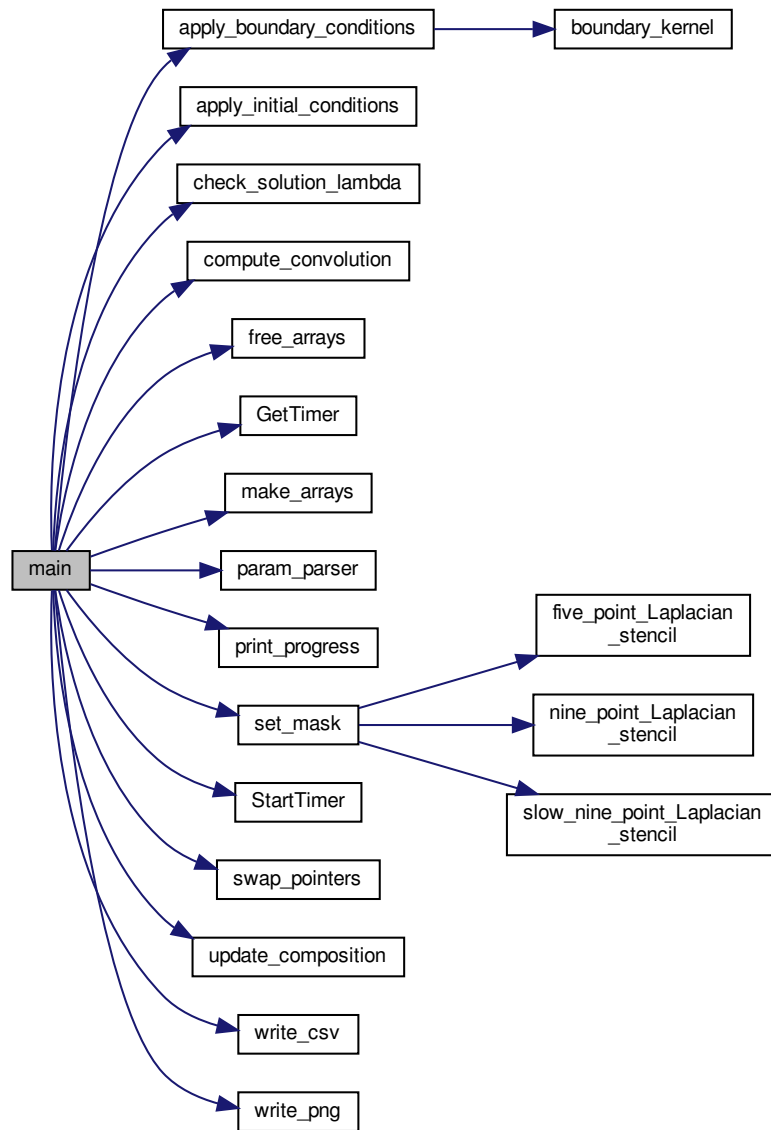


4.34.2.2 main() `int main (`
 `int argc,`
 `char * argv[])`

Run simulation using input parameters specified on the command line.

Definition at line 29 of file `tbb_main.c`.

Here is the call graph for this function:



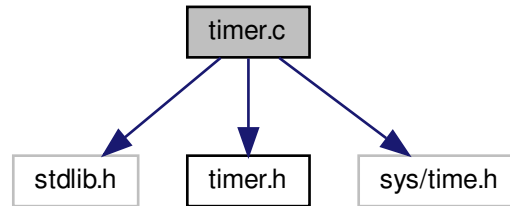
4.35 timer.c File Reference

High-resolution cross-platform machine time reader.

```
#include <stdlib.h>
#include "timer.h"
```

```
#include <sys/time.h>
```

Include dependency graph for timer.c:



Macros

- `#define` [__USE_BSD](#)
- `#define` [__USE_MISC](#)

Functions

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

Variables

- struct timeval [timerStart](#)

4.35.1 Detailed Description

High-resolution cross-platform machine time reader.

Author

NVIDIA

4.35.2 Macro Definition Documentation

4.35.2.1 `__USE_BSD` `#define __USE_BSD`

Definition at line 37 of file timer.c.

4.35.2.2 `__USE_MISC` `#define __USE_MISC`

Definition at line 40 of file timer.c.

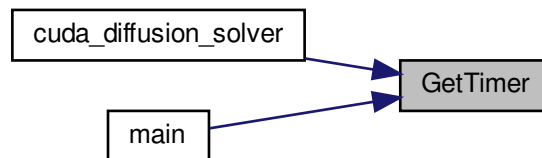
4.35.3 Function Documentation

4.35.3.1 `GetTimer()` `double GetTimer ()`

Return elapsed time in seconds.

Definition at line 71 of file timer.c.

Here is the caller graph for this function:



4.35.3.2 `StartTimer()` `void StartTimer ()`

Set CPU frequency and begin timing.

Definition at line 55 of file timer.c.

Here is the caller graph for this function:



4.35.4 Variable Documentation

4.35.4.1 **timerStart** `struct timeval timerStart`

Platform-dependent data type of hardware time value

Definition at line 1 of file timer.c.

4.36 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.

4.36.1 Detailed Description

Declaration of timer function prototypes for diffusion benchmarks.

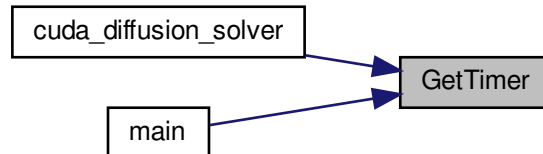
4.36.2 Function Documentation

4.36.2.1 GetTimer() `double GetTimer ()`

Return elapsed time in seconds.

Definition at line 71 of file timer.c.

Here is the caller graph for this function:



4.36.2.2 StartTimer() `void StartTimer ()`

Set CPU frequency and begin timing.

Definition at line 55 of file timer.c.

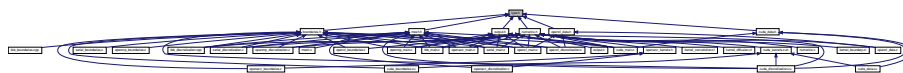
Here is the caller graph for this function:



4.37 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](#)

4.37.1 Detailed Description

Definition of scalar data type and Doxygen diffusion group.

4.37.2 Typedef Documentation

4.37.2.1 `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 22 of file type.h.

Index

- `__USE_BSD`
 - `timer.c`, [106](#)
 - `__USE_MISC`
 - `timer.c`, [107](#)
- `analytical_value`
 - `numerics.c`, [44](#)
 - `numerics.h`, [52](#)
- `apply_boundary_conditions`
 - `boundaries.h`, [10](#)
 - `openacc_boundaries.c`, [59](#)
 - `openmp_boundaries.c`, [83](#)
 - `serial_boundaries.c`, [95](#)
 - `tbb_boundaries.cpp`, [99](#)
- `apply_initial_conditions`
 - `boundaries.h`, [10](#)
 - `cuda_boundaries.cu`, [12](#)
 - `openacc_boundaries.c`, [60](#)
 - `opencl_boundaries.c`, [67](#)
 - `openmp_boundaries.c`, [83](#)
 - `serial_boundaries.c`, [95](#)
 - `tbb_boundaries.cpp`, [99](#)
- `boundaries.h`, [9](#)
 - `apply_boundary_conditions`, [10](#)
 - `apply_initial_conditions`, [10](#)
- `boundary_kernel`
 - `cuda_boundaries.cu`, [13](#)
 - `cuda_kernels.cuh`, [29](#)
 - `kernel_boundary.cl`, [34](#)
 - `openacc_boundaries.c`, [60](#)
 - `openacc_kernels.h`, [64](#)
 - `OpenCLData`, [5](#)
- `boundary_program`
 - `OpenCLData`, [5](#)
- `build_program`
 - `opencl_data.c`, [69](#)
 - `opencl_data.h`, [73](#)
- `check_solution`
 - `numerics.c`, [45](#)
 - `numerics.h`, [52](#)
- `check_solution_lambda`
 - `tbb_discretization.cpp`, [101](#)
 - `tbb_main.c`, [103](#)
- `commandQueue`
 - `OpenCLData`, [6](#)
- `compute_convolution`
 - `cuda_discretization.cu`, [22](#)
 - `numerics.h`, [53](#)
 - `openmp_discretization.c`, [85](#)
 - `serial_discretization.c`, [96](#)
 - `tbb_discretization.cpp`, [101](#)
- `conc_lap`
 - `CudaData`, [4](#)
 - `OpenCLData`, [6](#)
- `conc_new`
 - `CudaData`, [4](#)
 - `OpenCLData`, [6](#)
- `conc_old`
 - `CudaData`, [4](#)
 - `OpenCLData`, [6](#)
- `context`
 - `OpenCLData`, [6](#)
- `conv`
 - `Stopwatch`, [8](#)
- `convolution_kernel`
 - `cuda_discretization.cu`, [23](#)
 - `cuda_kernels.cuh`, [29](#)
 - `kernel_convolution.cl`, [35](#)
 - `openacc_discretization.c`, [62](#)
 - `openacc_kernels.h`, [64](#)
 - `OpenCLData`, [6](#)
- `convolution_program`
 - `OpenCLData`, [7](#)
- `cuda_boundaries.cu`, [11](#)
 - `apply_initial_conditions`, [12](#)
 - `boundary_kernel`, [13](#)
- `cuda_data.cu`, [14](#)
 - `free_cuda`, [14](#)
 - `init_cuda`, [15](#)
- `cuda_data.h`, [16](#)
 - `cuda_diffusion_solver`, [17](#)
 - `device_boundaries`, [18](#)
 - `device_composition`, [18](#)
 - `device_convolution`, [19](#)
 - `free_cuda`, [19](#)
 - `init_cuda`, [20](#)
 - `read_out_result`, [20](#)
- `cuda_diffusion_solver`
 - `cuda_data.h`, [17](#)
 - `cuda_discretization.cu`, [23](#)
- `cuda_discretization.cu`, [21](#)
 - `compute_convolution`, [22](#)
 - `convolution_kernel`, [23](#)
 - `cuda_diffusion_solver`, [23](#)
 - `d_mask`, [27](#)
 - `device_boundaries`, [24](#)
 - `device_composition`, [25](#)
 - `device_convolution`, [25](#)
 - `diffusion_kernel`, [26](#)
 - `read_out_result`, [27](#)
- `cuda_kernels.cuh`, [27](#)

- boundary_kernel, 29
 - convolution_kernel, 29
 - d_mask, 31
 - diffusion_kernel, 30
- cuda_main.c, 31
 - main, 32
- CudaData, 3
 - conc_lap, 4
 - conc_new, 4
 - conc_old, 4
- d_mask
 - cuda_discretization.cu, 27
 - cuda_kernels.cuh, 31
- device_boundaries
 - cuda_data.h, 18
 - cuda_discretization.cu, 24
 - opengl_data.h, 74
 - opengl_discretization.c, 78
- device_composition
 - cuda_data.h, 18
 - cuda_discretization.cu, 25
- device_convolution
 - cuda_data.h, 19
 - cuda_discretization.cu, 25
 - opengl_data.h, 74
 - opengl_discretization.c, 79
- device_diffusion
 - opengl_data.h, 74
 - opengl_discretization.c, 79
- diffusion_kernel
 - cuda_discretization.cu, 26
 - cuda_kernels.cuh, 30
 - kernel_diffusion.cl, 36
 - openacc_discretization.c, 62
 - openacc_kernels.h, 64
 - OpenCLData, 7
- diffusion_program
 - OpenCLData, 7
- distance_point_to_segment
 - numerics.c, 46
 - numerics.h, 54
- euclidean_distance
 - numerics.c, 46
 - numerics.h, 55
- file
 - Stopwatch, 8
- five_point_Laplacian_stencil
 - numerics.c, 47
 - numerics.h, 55
- fp_t
 - type.h, 110
- free_arrays
 - mesh.c, 38
 - mesh.h, 41
- free_cuda
 - cuda_data.cu, 14
 - cuda_data.h, 19
- free_opengl
 - opengl_data.c, 69
 - opengl_data.h, 75
- GetTimer
 - timer.c, 107
 - timer.h, 108
- init_cuda
 - cuda_data.cu, 15
 - cuda_data.h, 20
- init_opengl
 - opengl_data.c, 70
 - opengl_data.h, 76
- kernel_boundary.cl, 33
 - boundary_kernel, 34
- kernel_convolution.cl, 34
 - convolution_kernel, 35
- kernel_diffusion.cl, 36
 - diffusion_kernel, 36
- main
 - cuda_main.c, 32
 - openacc_main.c, 65
 - opengl_main.c, 81
 - openmp_main.c, 86
 - serial_main.c, 98
 - tbb_main.c, 104
- make_arrays
 - mesh.c, 38
 - mesh.h, 41
- manhattan_distance
 - numerics.c, 47
 - numerics.h, 56
- mask
 - OpenCLData, 7
- MAX_MASK_H
 - numerics.h, 51
- MAX_MASK_W
 - numerics.h, 52
- mesh.c, 37
 - free_arrays, 38
 - make_arrays, 38
 - swap_pointers, 39
 - swap_pointers_1D, 39
- mesh.h, 40
 - free_arrays, 41
 - make_arrays, 41
 - swap_pointers, 42

- swap_pointers_1D, 42
- nine_point_Laplacian_stencil
 - numerics.c, 48
 - numerics.h, 56
- numerics.c, 43
 - analytical_value, 44
 - check_solution, 45
 - distance_point_to_segment, 46
 - euclidean_distance, 46
 - five_point_Laplacian_stencil, 47
 - manhattan_distance, 47
 - nine_point_Laplacian_stencil, 48
 - set_mask, 48
 - slow_nine_point_Laplacian_stencil, 49
- numerics.h, 50
 - analytical_value, 52
 - check_solution, 52
 - compute_convolution, 53
 - distance_point_to_segment, 54
 - euclidean_distance, 55
 - five_point_Laplacian_stencil, 55
 - manhattan_distance, 56
 - MAX_MASK_H, 51
 - MAX_MASK_W, 52
 - nine_point_Laplacian_stencil, 56
 - set_mask, 56
 - slow_nine_point_Laplacian_stencil, 57
 - update_composition, 58
- openacc_boundaries.c, 59
 - apply_boundary_conditions, 59
 - apply_initial_conditions, 60
 - boundary_kernel, 60
- openacc_discretization.c, 61
 - convolution_kernel, 62
 - diffusion_kernel, 62
- openacc_kernels.h, 63
 - boundary_kernel, 64
 - convolution_kernel, 64
 - diffusion_kernel, 64
- openacc_main.c, 65
 - main, 65
- opencl_boundaries.c, 66
 - apply_initial_conditions, 67
- opencl_data.c, 68
 - build_program, 69
 - free_opengl, 69
 - init_opengl, 70
 - report_error, 71
- opencl_data.h, 71
 - build_program, 73
 - device_boundaries, 74
 - device_convolution, 74
 - device_diffusion, 74
 - free_opengl, 75
 - init_opengl, 76
 - read_out_result, 76
 - report_error, 77
- opencl_discretization.c, 78
 - device_boundaries, 78
 - device_convolution, 79
 - device_diffusion, 79
 - read_out_result, 80
- opencl_main.c, 80
 - main, 81
- OpenCLData, 4
 - boundary_kernel, 5
 - boundary_program, 5
 - commandQueue, 6
 - conc_lap, 6
 - conc_new, 6
 - conc_old, 6
 - context, 6
 - convolution_kernel, 6
 - convolution_program, 7
 - diffusion_kernel, 7
 - diffusion_program, 7
 - mask, 7
- openmp_boundaries.c, 82
 - apply_boundary_conditions, 83
 - apply_initial_conditions, 83
- openmp_discretization.c, 84
 - compute_convolution, 85
 - update_composition, 85
- openmp_main.c, 86
 - main, 86
- output.c, 87
 - param_parser, 88
 - print_progress, 89
 - write_csv, 89
 - write_png, 90
- output.h, 91
 - param_parser, 92
 - print_progress, 92
 - write_csv, 93
 - write_png, 93
- param_parser
 - output.c, 88
 - output.h, 92
- print_progress
 - output.c, 89
 - output.h, 92
- read_out_result
 - cuda_data.h, 20
 - cuda_discretization.cu, 27
 - opencl_data.h, 76
 - opencl_discretization.c, 80

- report_error
 - openc1_data.c, 71
 - openc1_data.h, 77
- serial_boundaries.c, 94
 - apply_boundary_conditions, 95
 - apply_initial_conditions, 95
- serial_discretization.c, 95
 - compute_convolution, 96
 - update_composition, 96
- serial_main.c, 97
 - main, 98
- set_mask
 - numerics.c, 48
 - numerics.h, 56
- slow_nine_point_Laplacian_stencil
 - numerics.c, 49
 - numerics.h, 57
- soln
 - Stopwatch, 8
- StartTimer
 - timer.c, 107
 - timer.h, 109
- step
 - Stopwatch, 8
- Stopwatch, 7
 - conv, 8
 - file, 8
 - soln, 8
 - step, 8
- swap_pointers
 - mesh.c, 39
 - mesh.h, 42
- swap_pointers_1D
 - mesh.c, 39
 - mesh.h, 42
- tbb_boundaries.cpp, 99
 - apply_boundary_conditions, 99
 - apply_initial_conditions, 99
- tbb_discretization.cpp, 100
 - check_solution_lambda, 101
 - compute_convolution, 101
 - update_composition, 102
- tbb_main.c, 102
 - check_solution_lambda, 103
 - main, 104
- timer.c, 105
 - __USE_BSD, 106
 - __USE_MISC, 107
 - GetTimer, 107
 - StartTimer, 107
 - timerStart, 108
- timer.h, 108
 - GetTimer, 108
 - StartTimer, 109
- timerStart
 - timer.c, 108
- type.h, 109
 - fp_t, 110
- update_composition
 - numerics.h, 58
 - openmp_discretization.c, 85
 - serial_discretization.c, 96
 - tbb_discretization.cpp, 102
- write_csv
 - output.c, 89
 - output.h, 93
- write_png
 - output.c, 90
 - output.h, 93