Source code can be found in the file ‘advection2D-final.c’ and is attached to the bottom of this file.

A plot of the initial conditions u(x, y) from section 2.2Chart, histogram

Description automatically generated

A plot of the final values of u(x, y) from section 2.2Chart

Description automatically generated

A plot of the final values of u(x, y) from section 2.3A picture containing graphical user interface

Description automatically generated

A plot showing the vertically averaged profile of u from section 2.4 Chart, line chart, histogram

Description automatically generated

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2D advection example program which advects a Gaussian u(x,y) at a fixed velocity

Outputs: initial.dat - inital values of u(x,y)

final.dat - final values of u(x,y)

The output files have three columns: x, y, u

Compile with: gcc -o advection2D -std=c99 advection2D.c -lm

Notes: The time step is calculated using the CFL condition

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Include header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <omp.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(){

/\* Grid properties \*/

const int NX=1000; // Number of x points

const int NY=1000; // Number of y points

const float xmin=0.0; // Minimum x value

const float xmax=30.0; // Maximum x value

const float ymin=0.0; // Minimum y value

const float ymax=30.0; // Maximum y value

/\* Parameters for the Gaussian initial conditions \*/

const float x0=3.0; // Centre(x)

const float y0=15.0; // Centre(y)

const float sigmax=1.0; // Width(x)

const float sigmay=5.0; // Width(y)

const float sigmax2 = sigmax \* sigmax; // Width(x) squared

const float sigmay2 = sigmay \* sigmay; // Width(y) squared

/\* Boundary conditions \*/

const float bval\_left=0.0; // Left boudnary value

const float bval\_right=0.0; // Right boundary value

const float bval\_lower=0.0; // Lower boundary

const float bval\_upper=0.0; // Upper bounary

/\* Time stepping parameters \*/

const float CFL=0.9; // CFL number

const int nsteps=800; // Number of time steps

/\* Velocity \*/

const float velx=1.0; // Velocity in x direction

const float vely=0.0; // Velocity in y direction

/\* Arrays to store variables. These have NX+2 elements

to allow boundary values to be stored at both ends \*/

float x[NX+2]; // x-axis values

float y[NX+2]; // y-axis values

float u[NX+2][NY+2]; // Array of u values

float dudt[NX+2][NY+2]; // Rate of change of u

float x2; // x squared (used to calculate iniital conditions)

float y2; // y squared (used to calculate iniital conditions)

/\* Calculate distance between points \*/

float dx = (xmax-xmin) / ( (float) NX);

float dy = (ymax-ymin) / ( (float) NY);

/\* Calculate time step using the CFL condition \*/

/\* The fabs function gives the absolute value in case the velocity is -ve \*/

float dt = CFL / ( (fabs(velx) / dx) + (fabs(vely) / dy) );

float modified\_velx = 0.0; // our modified horizontal velocity

const float ustar = 0.2; // friction velocity

const float k = 0.41; // Von Karman's constant

const float z0 = 1.0; // roughness length

/\*\*\* Report information about the calculation \*\*\*/

printf("Grid spacing dx = %g\n", dx);

printf("Grid spacing dy = %g\n", dy);

printf("CFL number = %g\n", CFL);

printf("Time step = %g\n", dt);

printf("No. of time steps = %d\n", nsteps);

printf("End time = %g\n", dt\*(float) nsteps);

printf("Distance advected x = %g\n", velx\*dt\*(float) nsteps);

printf("Distance advected y = %g\n", vely\*dt\*(float) nsteps);

/\*\*\* Place x points in the middle of the cell \*\*\*/

/\* LOOP 1 \*/

// Variables scoped as shared by default

// Each loop reads/writes to a different element of array 'x' so there is no loop carried dependency

#pragma omp parallel for default(shared)

for (int i=0; i<NX+2; i++){

x[i] = ( (float) i - 0.5) \* dx;

}

/\*\*\* Place y points in the middle of the cell \*\*\*/

/\* LOOP 2 \*/

// Variables scoped as shared by default

// Each loop reads/writes to a different element of array 'y' so there is no loop carried dependency

#pragma omp parallel for default(shared)

for (int j=0; j<NY+2; j++){

y[j] = ( (float) j - 0.5) \* dy;

}

/\*\*\* Set up Gaussian initial conditions \*\*\*/

/\* LOOP 3 \*/

// Can write to a shared array using the unique index

// Collapses the nested for loop into a single loop without using nested parallelism

// Compiler forms a single loop and then parallelises this

#pragma omp parallel for collapse(2) default(shared)

for (int i=0; i<NX+2; i++){

for (int j=0; j<NY+2; j++){

x2 = (x[i]-x0) \* (x[i]-x0);

y2 = (y[j]-y0) \* (y[j]-y0);

u[i][j] = exp( -1.0 \* ( (x2/(2.0\*sigmax2)) + (y2/(2.0\*sigmay2)) ) );

}

}

/\*\*\* Write array of initial u values out to file \*\*\*/

FILE \*initialfile;

initialfile = fopen("initial.dat", "w");

/\* LOOP 4 \*/

// Cannot be parallelised - Parallel loop iterations are not carried out in the order specified by the loop iterator

// The outer loop will mismatch the 'i' and 'j' indicies because they are processsed by different threads.

// These print statements need to happen in order so the loop iterations must take place in order.

// We can’t parallelise this loop.

for (int i=0; i<NX+2; i++){

for (int j=0; j<NY+2; j++){

fprintf(initialfile, "%g %g %g\n", x[i], y[j], u[i][j]);

}

}

fclose(initialfile);

/\*\*\* Update solution by looping over time steps \*\*\*/

/\* LOOP 5 \*/

// Cannot perfectly collapse loop due to differing inner loop depths

// We can’t parallelise this loop.

for (int m=0; m<nsteps; m++){

/\*\*\* Apply boundary conditions at u[0][:] and u[NX+1][:] \*\*\*/

/\* LOOP 6 \*/

// Variables scoped as shared by default

// Can write to a shared array using the unique index

#pragma omp parallel for default(shared)

for (int j=0; j<NY+2; j++){

u[0][j] = bval\_left;

u[NX+1][j] = bval\_right;

}

/\*\*\* Apply boundary conditions at u[:][0] and u[:][NY+1] \*\*\*/

/\* LOOP 7 \*/

// Variables scoped as shared by default

// Can write to a shared array using the unique index

#pragma omp parallel for default(shared)

for (int i=0; i<NX+2; i++){

u[i][0] = bval\_lower;

u[i][NY+1] = bval\_upper;

}

/\*\*\* Calculate rate of change of u using leftward difference \*\*\*/

/\* Loop over points in the domain but not boundary values \*/

/\* LOOP 8 \*/

// Variables scoped as shared by default

// Each inner loop reads/writes to a different element of array 'duct' so there is no loop carried dependency

// Collapses the nested for loop into a single loop without using nested parallelism

// Compiler forms a single loop and then parallelises this

#pragma omp parallel for collapse(2) default(shared)

for (int i=1; i<NX+1; i++){

for (int j=1; j<NY+1; j++){

// TASK 3 - Adding Vertical Shear

if (y[j] > 1) {

// modified\_velx = (ustar / k) \* (log(y[j]) / 1.66); // close to sqr root (e)

// modified\_velx = (ustar / k) \* (log(y[j]) / sqrt(M\_E)); // was not working on remote linux

modified\_velx = (ustar / k) \* (log(y[j]) / sqrt(exp(z0)));

} else { // this else loop isn't strictly necessary but helps with readability

modified\_velx = 0.0;

}

dudt[i][j] = -modified\_velx \* (u[i][j] - u[i-1][j]) / dx

- vely \* (u[i][j] - u[i][j-1]) / dy;

}

}

/\*\*\* Update u from t to t+dt \*\*\*/

/\* Loop over points in the domain but not boundary values \*/

/\* LOOP 9 \*/

// Variables scoped as shared by default

// Each inner loop reads/writes to a different element of array 'u' so there is no loop carried dependency

// Can write to a shared array using the unique index

// Collapses the nested for loop into a single loop without using nested parallelism

// Compiler forms a single loop and then parallelises this

#pragma omp parallel for collapse(2) default(shared)

for (int i=1; i<NX+1; i++){

for (int j=1; j<NY+1; j++){

u[i][j] = u[i][j] + dudt[i][j] \* dt;

}

}

} // time loop

/\*\*\* Write array of final u values out to file \*\*\*/

FILE \*finalfile;

finalfile = fopen("final.dat", "w");

/\* LOOP 10 \*/

// Cannot be parallelised - Parallel loop iterations are not carried out in the order specified by the loop iterator

// The outer loop will mismatch the 'i' and 'j' indicies because they are processsed by different threads.

// These print statements need to happen in order so the loop iterations must take place in order.

// We can’t parallelise this loop.

for (int i=0; i<NX+2; i++){

for (int j=0; j<NY+2; j++){

fprintf(finalfile, "%g %g %g\n", x[i], y[j], u[i][j]);

}

}

fclose(finalfile);

// TASK 4 - Calculating the Vertically Averaged Distribution

float avg[NY]; // variable store avg intensity at each value of x

float intensity\_sum; // allows us to average the value of u(x,y) for the entire range of y values at each value of x

/\* Loop over points in the domain but not boundary values \*/

for(int i=1; i<NX+1; i++){

intensity\_sum = 0.0;

for (int j=1; j<NY+1; j++){

intensity\_sum += u[i][j]; // sum the y attribute of u(x, y) at each value of x

}

avg[i] = intensity\_sum / (float) NY; // average the values of intensity in the y direction at each value of x

// printf("%d\n", NY);

// printf("%f\n", intensity\_sum);

// printf("%f\n\n", avg[i]);

}

/\*\*\* Write array of vertically averaged u values out to file \*\*\*/

FILE \*vertavgfile;

vertavgfile = fopen("vertavg.dat", "w");

for (int i=0; i<NX+1; i++){

for (int j=0; j<NY+1; j++){

fprintf(vertavgfile, "%g %g\n", x[i], avg[i]);

}

}

fclose(vertavgfile);

return 0;

}

/\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# This gnuplot script plots the results from the first coursework assignment.

# It is assumed that the data to be plotted are in a file called

# final.dat which contains 3 columns: x,y,u

# The plot is sent to a PNG file called final.png

# To use this file copy it to the directory/folder containing

# final.dat and run the command:

# gnuplot plot\_final

# Send output to a PNG file

set terminal png enhanced

# Set ranges and labels for axes

set xrange [0:30.0]

set yrange [0:30.0]

set xlabel "x(m)"

set ylabel "y(m)"

# Enforce an aspect ratio of 1

set size square

# Set the range of the colour scale

set cbrange [0:1]

#------ TASK 2 ------- Uncomment as appropriate

# # Set the title of the figure for task 2 initial conditions

# set title "Modifying the Calculation - Initial Conditions u(x, y)" offset 0,0.1

# # Set the name of the output file

# set output "initial-two.png"

# # Plot the data

# plot "initial.dat" with image

#------ TASK 2 ------- Uncomment as appropriate

# # Set the title of the figure for task 2 final values

# set title "Modifying the Calculation - Final Values u(x, y)" offset 0,0.1

# # Set the name of the output file

# set output "final-two.png"

# # Plot the data

# plot "final.dat" with image

#------ TASK 3 ------- Uncomment as appropriate

# Set the title of the figure for task 3 final values

set title "Adding Vertical Shear - Final Values of u(x, y)" offset 0,0.1

# Set the name of the output file

set output "final-three.png"

# Plot the data

plot "final.dat" with image

# End of file

# This gnuplot script plots the results from the first coursework assignment.

# It is assumed that the data to be plotted are in a file called

# final.dat which contains 3 columns: x,y,u

# The plot is sent to a PNG file called final.png

# To use this file copy it to the directory/folder containing

# final.dat and run the command:

# gnuplot plot\_final

# Send output to a PNG file

set terminal png enhanced

# Set the name of the output file

set output "vertavg.png"

# Set ranges and labels for axes

set xrange [0:30.0]

set yrange [0:0.2]

set ytics 0,0.025,0.2

set xlabel "x(m)"

set ylabel "vertically averaged values of u"

# Enforce an aspect ratio of 1

set size square

# Set linestyle 1 to blue (#0060ad)

set style line 1 \

linecolor rgb '#0060ad' \

linetype 1 \

set title "Vertically Averaged Distribution of u(x,y)" offset 0,0.1

# Plot the data

plot 'vertavg.dat' with linespoints pt 0 lw 2 title ''

# End of file