Numerical Methods for Parallel Simulation of Diffusive Pollutant Transport from a Point Source

CyberTraining: Big Data + High-Performance Computing + Atmospheric Sciences

Noah Sienkiewicz[1], Arjun Pandya[2], Tim Brown[3],
Research assistant: Carlos Barajas[3], Faculty mentor: Matthias K. Gobbert[3]

[1]Department of Physics, UMBC
[2]Department of Information Systems, UMBC
[3]Department of Mathematics and Statistics, UMBC

Technical Report HPCF–2018–11, hpcf.umbc.edu > Publications

Abstract

In an interdisciplinary project combining Atmospheric Physics, High Performance Computing, and Big Data, we explore a numerical method for solving a physical system modeled by a partial differential equation. The application problem models the spread of pollution by a reaction-diffusion equation solved by the finite volume method. The numerical method is derived and tested on a known test problem in Matlab and then parallelized by MPI in C. We explore both closed and open systems of pollution, and show that the finite volume method is both mass conservative and has the ability to handle a point source modeled by the Dirac delta distribution. A parallel performance study confirms the scalability of the implementation to several compute nodes.

1 Introduction

In this report, we consider a problem that blends the areas of **Big Data**, **High Performance Computing**, and **Atmospheric Physics** of our CyberTraining program, as described in [3] and on the webpage www.cybertraining.com. The problem models diffusion of aerosols emitted from a pollution source into a stable environment (no wind), with the emission modeled as a point source in the center of a two-dimensional region on the scale of a city or county. This is sketched in Figure 1.1 as a square domain of size 100 km by 100 km, with a factory with a polluting smokestack indicated at the center. We model the scenario where the source pollutes with a given amount of material, κ kg per hour for the duration of a typical work day of 8 hours and then shuts off. We will simulate the model for 24 hours, beginning when the pollution starts. The simulation starts with no pollution present in the region. The environment is modeled initially as a closed system. That is, we enforce no-flow boundary conditions along the domain boundary; we will later show how to approximate the effect of an open system by increasing the size of the numerical domain such that the pollution does not reach the edge of the calculation domain and that sub domains can be considered as open systems.

This problem combines directly **Atmospheric Physics** in the modeled material flow and **High Performance Computing** to enable fast calculations of the problem with increasing domain size. The area of **Big Data** could be applied to ouput if, for instance, the value of the

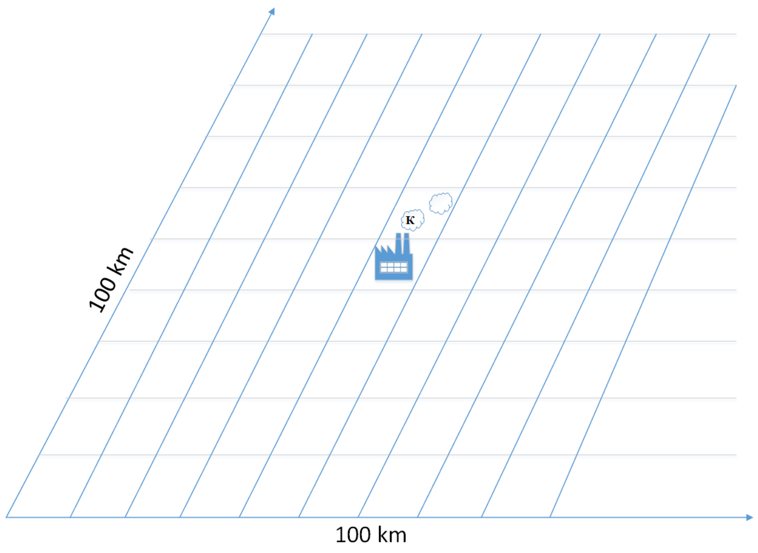

Figure 1.1: Pollution source at center of two-dimensional square region.

parameter κ is not assumed to be known. Section 2 provides the mathematical formulation of the model and specifies a test problem used to validate the numerical method.

Section 3 provides an introduction to the finite volume method (FVM) as a numerical method that is particularly appropriate to the type of partial differential equation in this report. Here, note that a point source is modeled mathematically by the Dirac delta distribution, which is zero everywhere except the location of the source and attains an infinite value at this location in such a way that the total integral over it has the value 1. Numerical methods such as the finite difference method cannot be set up for this model, since the value of the source term is necessarily infinite at the critical point. However, we show that the FVM guarantees mass conservation of the discrete approximations and also provides a smooth way to handle the point source.

Section 4 presents the results of simulations. Section 4.1 uses a test problem with smooth source term and a known true solution in closed form. This problem tests the implementation of the FVM and confirms that it converges in agreement with applicable theory. Then, Section 4.2 solves the pollution problem from Section 2.1 with its point source. We show how the mass conservation of the FVM confirms that also this problem is solved correctly, even though the solution is not known in closed form. Section 4.3 also solves the pollution problem, but on an enlarged numerical domain, which provides a demonstration how to simulate an open system on a domain of interest. This section then proceeds to use the capabilities developed to investigate the effect of an increasing parameter κ that controls the amount of pollution. Finally, Section 4.4 shows that parallel code using MPI on several compute nodes can reduce the simulation time dramatically, thus making the above simulations readily possible.

Section 5 summarizes our conclusions and outlines possible future applications of this model and method.

2 Mathematical Model

2.1 The Pollution Problem

This problem is described by the mathematical model given by a partial differential equation (PDE), a boundary condition (BC), and an initial condition (IC), for the concentration $u\,(x, y, t)$ of pollution, whose units are kg/km^2,

$$\begin{aligned} u_t - D\,\nabla \cdot (\nabla u) &= f\,(x, y, t) && \text{for } (x, y) \in \Omega \text{ and } t > 0, && (2.1)\\ \mathbf{n} \cdot \nabla u &= 0 && \text{for } (x, y) \in \partial\Omega \text{ and } t > 0, && (2.2)\\ u &= u_{ini}\,(x, y) && \text{for } (x, y) \in \overline{\Omega} \text{ at } t = 0, && (2.3) \end{aligned}$$

We use kilometers km for units on the spatial domain region $\Omega = (-50, 50) \times (-50, 50) \subset \mathbb{R}^2$ and hours h for the time domain where the time ranges such that $0 \le t \le 24$. In the PDE (2.1), the diffusivity coefficient D is a positive scalar constant in km^2/h and the point source is modeled by

$$f(x, y, t) = \kappa\, \delta_{(0,0)}\,(x, y)\ \chi_{[0,8]}\,(t) \qquad (2.4)$$

with output rate κ in kg km^{-2} h^{-1}. Here, we use the Dirac delta distribution in 2-D defined as $\delta_{(0,0)}\,(x, y) := \delta(x{-}0)\,\delta(y{-}0)$ to model the injection of pollution at center point $(0, 0) \in \Omega$, and an indicator function $\chi_{[0,8]}\,(t) = 1$ for times $0 \le t \le 8$ and 0 otherwise to control the switching. In the BC (2.2), the vector $\mathbf{n} \equiv \mathbf{n}\,(x, y)$ denotes the unit outward normal vector at $(x, y) \in \partial\Omega$. Since D is a constant scalar, this Neumann BC implements the no-flow condition of a closed system. In IC (2.3), setting $u_{ini} \equiv 0$ models the situation of no pollution present at the initial time $t = 0$.

2.2 Test Problem with Smooth Source Term

We will also, and in fact first, use a test problem that has the form of (2.1)–(2.3), but admits a chosen, known true solution $u(x, y, t)$ in closed form. Concretely, we choose

$$u(x, y, t) = (1 - e^{-(t/\tau)^2}) \cos^2(\pi x/100) \cos^2(\pi y/100), \qquad (2.5)$$

with $\tau = 8$, which is the solution of the problem with smooth source term

$$\begin{aligned} f(x, y, t) = {} & (2t/\tau^2)e^{-(t/\tau)^2} \cos^2(\pi x/100) \cos^2(\pi y/100) \\ & + D(1 - e^{-(t/\tau)^2})((-2\pi^2/100^2) \cos(2\pi x/100) \cos^2(\pi y/100) \\ & + (-2\pi^2/100^2) \cos^2(\pi x/100) \cos(2\pi y/100)) \qquad (2.6) \end{aligned}$$

Notice that D is chosen as in the pollution problem, and we also choose the same domain $\Omega = (-50, 50) \times (-50, 50)$ and initial condition $u_{ini} \equiv 0$.

3 Numerical Method

3.1 Spatial Discretization

We use the finite volume method (FVM) for the spatial discretization, because it conserves mass of the discrete concentration approximations $u_{ij}(t) \approx u(x_i, y_j, t)$, $i, j = 1, \ldots, N_0$, and because the FVM can directly and rigorously handle a point source modeled by a Dirac delta distribution.

The problem (2.1)–(2.3) is stated on a two-dimensional domain

$$\Omega = (x_{min}, x_{\max}) \times (y_{min}, y_{\max}) \subset \mathbb{R}^2 \tag{3.1}$$

that is assumed to be square and symmetric about $(0, 0)$ in both x- and y-direction. The symmetric property of Ω requires that $x_{min} = -x_{\max}$ and $y_{min} = -y_{\max}$. For the domain to be square requires furthermore that $x_{min} = y_{min}$ and $x_{\max} = y_{\max}$.

To derive the finite volume method, we define the primal mesh points as $(x_i, y_j) \in \overline{\Omega}$ with $x_i = x_{min} + (i-1)\,h$, $i = 1, \ldots, N_0$, and $y_j = y_{min} + (j-1)\,h$, $j = 1, \ldots, N_0$, with uniform mesh spacing $h = L/(N_0 - 1)$, where $L = x_{\max} - x_{min}$. We restrict N_0 to an odd integer, so that center point $(0, 0)$ is guaranteed to be a mesh point. Then we define the dual mesh on triangulation $\mathcal{T}_h = \cup_{(i,j)} \Omega_{ij}$ of cells around each mesh point (x_i, y_j), $i, j = 1, \ldots, N_0$.

The finite volume method starts by integrating the PDE (2.1) over each cell $\Omega_{ij} \in \mathcal{T}_h$ to obtain the conservative integral form

$$\iint_{\Omega_{ij}} \frac{\partial u}{\partial t}\, dx\, dy - \iint_{\Omega_{ij}} D\, \nabla \cdot (\nabla u)\, dx\, dy = \iint_{\Omega_{ij}} f\, dx\, dy. \tag{3.2}$$

We apply the divergence theorem to the diffusion term in the conservative integral form (3.2) to explicitly track the flow through the boundary $\partial\Omega_{ij}$ of each cell Ω_{ij}. The divergence theorem states $\iint_W \nabla \cdot J\, dx\, dy = \int_{\partial W} \mathbf{n} \cdot J\, dS$ with vector field $J : W \to \mathbb{R}^2$ and $W \subset \mathbb{R}^2$. Taking $J = \nabla u$ and $W = \Omega$, we obtain

$$\iint_{\Omega_{ij}} \frac{\partial u}{\partial t}\, dx\, dy - D \int_{\partial\Omega_{ij}} \mathbf{n} \cdot (\nabla u)\, dS = \iint_{\Omega_{ij}} f\, dx\, dy \tag{3.3}$$

for each cell $\Omega_{ij} \in \mathcal{T}_h$, $i, j = 1, \ldots, N_0$, of the dual mesh.

3.1.1 Semi-Discretization of a Interior Cell

All interior cells have the form $\Omega_{ij} = (x_i - \frac{h}{2}, x_i + \frac{h}{2}) \times (y_j - \frac{h}{2}, y_j + \frac{h}{2})$, $i, j = 2, \ldots, N_0 - 1$, that is, a square of size h^2 centered around each primal mesh point (x_i, y_j). The first integral in (3.3) yields for the integral of the time discretization the approximation

$$\iint_{\Omega_{ij}} \frac{\partial u(x,y,t)}{\partial t}\, dx\, dy \approx \int_{y_j - h/2}^{y_j + h/2} \int_{x_i - h/2}^{x_i + h/2} 1\, dx\, dy\, \frac{du(x_i, y_j, t)}{dt} \approx h^2\, \frac{du_{ij}(t)}{dt} \tag{3.4}$$

and similarly for the integral of the source term

$$\iint_{\Omega_{ij}} f\,dx\,dy \approx \int_{y_j-h/2}^{y_j+h/2} \int_{x_i-h/2}^{x_i+h/2} 1\,dx\,dy\, f(x_i,y_j,t) = h^2\, f(x_i,y_j,t). \tag{3.5}$$

The surface integral over the diffusion term in (3.3) is calculated by considering each of the four linear segments of $\partial\Omega_{ij}$. Consider the left-hand segment, which is at $x = x_i - \frac{h}{2}$ for $y_j - \frac{h}{2} \le y \le y_j + \frac{h}{2}$, with outward unit normal vector $\mathbf{n} = (-1,0)^T$. Then we approximate the integral

$$\begin{aligned}\int_{y_j-h/2}^{y_j+h/2} \mathbf{n}\cdot(\nabla u)\;dy &= \int_{y_j-h/2}^{y_j+h/2} (-1,0)\begin{pmatrix}u_x\\u_y\end{pmatrix}\;dy = -\int_{y_j-h/2}^{y_j+h/2} u_x(x_i-\frac{h}{2},y)\,dy\\ &\approx -\int_{y_j-h/2}^{y_j+h/2} 1\,dy\,\frac{u_{ij}-u_{i-1j}}{h} = -h\,\frac{u_{ij}-u_{i-1j}}{h}\\ &= -(u_{ij}-u_{i-1j}) = (u_{i-1j}-u_{ij})\end{aligned} \tag{3.6}$$

Together with analogous derivations of the other segments of $\partial\Omega_{ij}$, this yield in total for the diffusion term

$$\begin{aligned}\int_{\partial\Omega_{ij}} \mathbf{n}\cdot(\nabla u)\;dS &\approx \big(u_{ij-1}-u_{ij}\big) + \big(u_{i-1j}-u_{ij}\big) + \big(u_{i+1j}-u_{ij}\big) + \big(u_{ij+1}-u_{ij}\big)\\ &= u_{ij-1}+u_{i-1j}-4u_{ij}+u_{i+1j}+u_{ij+1}.\end{aligned} \tag{3.7}$$

Insert all approximations (3.4), (3.6), and (3.7) into the integral equation (3.3) and divide by h^2 to get the semi-discretization

$$\frac{du_{ij}(t)}{dt} + \frac{D}{h^2}\Big(-u_{ij-1}(t) - u_{i-1j}(t) + 4u_{ij}(t) - u_{i+1j}(t) - u_{ij+1}(t)\Big) = f(x_i,y_j,t) \tag{3.8}$$

for all interior cells Ω_{ij}, $i,j = 2,\dots,N_0-1$.

3.1.2 Semi-Discretization of a Boundary Cell

Consider the conservative integral form (3.3) in order to approximate a bottom boundary cell. The bottom segment of the boundary cell would be a subset of $\partial\Omega$, $i = 2,\dots,N_0-1$, $j = 1$ and the boundary cell's form would be $\Omega_{ij} = (x_i - \frac{h}{2}, x_i + \frac{h}{2}) \times (0, \frac{h}{2})$ which has size $h^2/2$, so the first and last integrals of (3.3) approximate to:

$$\iint_{\Omega_{ij}} \frac{\partial u(x,y,t)}{\partial t}\,dx\,dy \approx \int_0^{h/2}\int_{x_i-h/2}^{x_i+h/2} 1\,dx\,dy\,\frac{du(x_i,y_j,t)}{dt} \approx \frac{h^2}{2}\,\frac{du_{ij}(t)}{dt}, \tag{3.9}$$

$$\iint_{\Omega_{ij}} f\,dx\,dy \approx \int_0^{h/2}\int_{x_i-h/2}^{x_i+h/2} 1\,dx\,dy\,f(x_i,y_j,t) = \frac{h^2}{2}\,f(x_i,y_j,t). \tag{3.10}$$

The surface integral over the diffusion term in (3.3) is calculated by considering each of the four linear segments of $\partial\Omega_{ij}$, one of which is a subset of the domain boundary $\partial\Omega$. The segment $x_i - \frac{h}{2} \le x \le x_i + \frac{h}{2}$, $y_j = 0$ with the outward unit vector $\mathbf{n} = (0, -1)^T$ lies on the domain boundary $\partial\Omega$ thus with the Neumann boundary condition (2.2) is

$$\int_{x_i-h/2}^{x_i+h/2} \mathbf{n} \cdot \nabla u \, dx = 0. \tag{3.11}$$

Now again consider the left-segment which is at $x = x_i - \frac{h}{2}$ for $0 \le y \le \frac{h}{2}$ with the outward unit normal vector $\mathbf{n} = (-1, 0)^T$, then

$$\begin{aligned}\int_0^{h/2} \mathbf{n} \cdot \nabla u \, dy &= \int_0^{h/2} (-1, 0) \begin{pmatrix} u_x \\ u_y \end{pmatrix} dy = -\int_0^{h/2} u_x(x_i - \frac{h}{2}, y) \, dy \\ &\approx -\int_0^{h/2} 1 \, dy \, \frac{u_{ij} - u_{i-1j}}{h} = -\frac{h}{2} \, \frac{u_{ij} - u_{i-1j}}{h} \\ &= -\tfrac{1}{2}(u_{ij} - u_{i-1j}) = \tfrac{1}{2}(u_{i-1j} - u_{ij})\end{aligned} \tag{3.12}$$

Analogous derivations for the other segments of $\partial\Omega_{ij}$ yield the diffusion term

$$\begin{aligned}\int_{\partial\Omega_{ij}} \mathbf{n} \cdot (\nabla u) \, dS &\approx \tfrac{1}{2}\big(u_{i-1j} - u_{ij}\big) + \tfrac{1}{2}\big(u_{i+1j} - u_{ij}\big) + \big(u_{ij+1} - u_{ij}\big) \\ &= \tfrac{1}{2}u_{i-1j} - 2u_{ij} + \tfrac{1}{2}u_{i+1j} + u_{ij+1}.\end{aligned} \tag{3.13}$$

Insert all approximations, (3.9), (3.13), (3.10), into the integral equation (3.3) and divide by h^2 to get the semi-discretization

$$\frac{1}{2}\frac{du_{ij}(t)}{dt} + \frac{D}{h^2}\Big(-\tfrac{1}{2}u_{i-1j}(t) + 2u_{ij}(t) - \tfrac{1}{2}u_{i+1j}(t) - u_{ij+1}(t)\Big) = \frac{1}{2} f(x_i, y_j, t) \tag{3.14}$$

or after multiplying by 2 to make the leading coefficient a 1

$$\frac{du_{ij}(t)}{dt} + \frac{D}{h^2}\Big(-u_{i-1j}(t) + 4u_{ij}(t) - u_{i+1j}(t) - 2u_{ij+1}(t)\Big) = f(x_i, y_j, t) \tag{3.15}$$

for all boundary mesh points (x_i, y_j), $i = 2, \ldots, N_0 - 1$, $j = 1$. This derivation applies to a cell at the bottom boundary of Ω. Analogous derivations are applied to the top, left, and right boundaries.

3.1.3 Semi-Discretization of a Corner Cell

For the corner cell at the left bottom of boundary $\partial\Omega$, $i = 1$, $j = 1$, $\Omega_{ij} = (0, \frac{h}{2}) \times (0, \frac{h}{2})$ has size $h^2/4$, and the first and last integrals of (3.3) yield the approximations

$$\iint_{\Omega_{ij}} \frac{\partial u(x, y, t)}{\partial t} \, dx \, dy \approx \int_0^{h/2}\int_0^{h/2} 1 \, dx \, dy \, \frac{du(x_i, y_j, t)}{dt} \approx \frac{h^2}{4} \, \frac{du_{ij}(t)}{dt}, \tag{3.16}$$

$$\iint_{\Omega_{ij}} f \, dx \, dy \approx \int_0^{h/2}\int_0^{h/2} 1 \, dx \, dy \, f(x_i, y_j, t) = \frac{h^2}{4} \, f(x_i, y_j, t). \tag{3.17}$$

The surface integral over the diffusion term in (3.3) is calculated by considering each of the four linear segments of $\partial\Omega_{ij}$, two of which are subsets of the domain boundary $\partial\Omega$.

Consider one of the two boundary segments, namely the bottom segment $0 \le x \le \frac{h}{2}$, $y_j = 0$ with $\mathbf{n} = (0,-1)^T$ lies on the domain boundary $\partial\Omega$, thus by the boundary condition (2.2):

$$\int_0^{h/2} \mathbf{n} \cdot \nabla u \, dx = 0. \tag{3.18}$$

One of the non-boundary segments is at $x = x_i + \frac{h}{2}$ for $0 \le y \le \frac{h}{2}$ with outward unit normal vector $\mathbf{n} = (1,0)^T$, then

$$\begin{aligned} \int_0^{h/2} \mathbf{n} \cdot \nabla u \, dy &= \int_0^{h/2} (1,0) \begin{pmatrix} u_x \\ u_y \end{pmatrix} dy = \int_0^{h/2} u_x(x_i + \frac{h}{2}, y) \, dy \\ &\approx \int_0^{h/2} 1 \, dy \, \frac{u_{i+1j} - u_{ij}}{h} = \frac{h}{2} \, \frac{u_{i+1j} - u_{ij}}{h} = \tfrac{1}{2}(u_{i+1j} - u_{ij}). \end{aligned} \tag{3.19}$$

After the analogous derivations of the other segments of $\partial\Omega_{ij}$ the result yields

$$\begin{aligned} \int_{\partial\Omega_{ij}} \mathbf{n} \cdot (\nabla u) \; dS &\approx \tfrac{1}{2}\big(u_{i+1j} - u_{ij}\big) + \tfrac{1}{2}\big(u_{ij+1} - u_{ij}\big) \\ &= -u_{ij} + \tfrac{1}{2}u_{i+1j} + \tfrac{1}{2}u_{ij+1}. \end{aligned} \tag{3.20}$$

Insert all approximations, (3.16), (3.20), (3.17), into the integral equation (3.3) and divide by h^2 to get the semi-discretization

$$\frac{1}{4}\frac{du_{ij}(t)}{dt} + \frac{D}{h^2}\Big(u_{ij}(t) - \tfrac{1}{2}u_{i+1j}(t) - \tfrac{1}{2}u_{ij+1}(t)\Big) = \frac{1}{4}\, f(x_i, y_j, t) \tag{3.21}$$

or after multiplying by 4 to make the leading coefficient a 1

$$\frac{du_{ij}(t)}{dt} + \frac{D}{h^2}\Big(4u_{ij}(t) - 2u_{i+1j}(t) - 2u_{ij+1}(t)\Big) = f(x_i, y_j, t) \tag{3.22}$$

for the corner mesh point (x_i, y_j), $i = 1$, $j = 1$. This derivation applies to a cell at the left bottom corner of $\partial\Omega$. Analogous derivations are applied to the right bottom, left top, and right top corners of the boundary.

3.1.4 System of All Semi-Discretizations

Define the column-vector $\mathbf{u}(t) = (\mathbf{u}_k(t)) \in \mathbb{R}^N$, $k = 1, \dots N$ with $N := N_0^2$, with components $\mathbf{u}_k(t) = u_{ij}(t)$, ordered by $k = i + N_0\,(j-1)$ for $i, j = 1, \dots, N_0$. Also analogously define the shorthand notation $\mathbf{f}(t) = (\mathbf{f}_k(t)) \in \mathbb{R}^N$ with $\mathbf{f}_k(t) = f_{ij}(t) = f(x_i, y_j, t)$ using the same ordering of the components. Assembling all types of semi-discretizations (3.8), (3.15), (3.22) yields then in vector form

$$\frac{d\mathbf{u}(t)}{dt} + \frac{D}{h^2}\, A\,\mathbf{u}(t) = \mathbf{f}(t), \tag{3.23}$$

with $A = I \otimes T + T \otimes I \in \mathbb{R}^{N \times N}$ computed as sum of Kronecker products between the identity matrix $I \in \mathbb{R}^{N_0 \times N_0}$ and the tri-diagonal matrix

$$T = \begin{bmatrix} 2 & -2 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -2 & 2 \end{bmatrix} \in \mathbb{R}^{N_0 \times N_0} \tag{3.24}$$

To see in what sense the FVM conserves mass at the discrete level, recall what it means at the continuous level: If $u(x, y, t)$ denotes the concentration of a quantity throughout domain Ω, then the total mass in the domain $\iint_\Omega u\,dx\,dy$ is conserved if the time derivative of the total is zero for all $t > 0$. We see that the PDE (2.1) without a source by $f \equiv 0$ satisfies this condition by integrating and applying the divergence theorem in

$$\frac{d}{dt} \iint_\Omega u\,dx\,dy = \iint_\Omega \frac{\partial u}{\partial t}\,dx\,dy = \iint_\Omega D\,\nabla \cdot (\nabla u)\,dx\,dy = D \iint_\Omega \mathbf{n} \cdot \nabla u\,dS = 0, \tag{3.25}$$

which is 0 by BC (2.2). The analogue condition on the discrete level starts by noting that Ω is the union of all cells Ω_{ij}, that is, $\Omega = \biguplus_{ij} \Omega_{ij}$, and thus by linearity of the integral

$$\frac{d}{dt} \iint_\Omega u\,dx\,dy = \frac{d}{dt} \sum_{(i,j)} \iint_{\Omega_{ij}} u\,dx\,dy = \sum_{(i,j)} \frac{d}{dt} \iint_{\Omega_{ij}} u\,dx\,dy \approx \sum_{(i,j)} |\Omega_{ij}|\,\frac{du_{ij}}{dt}. \tag{3.26}$$

Since the area of each cell $|\Omega_{ij}|$ is h^2 for interior cell, $h^2/2$ for boundary cell, and $h^2/4$ for corner cell, the terms $|\Omega_{ij}|\,\frac{du_{ij}}{dt}$ under the sums are, aside from common factor h^2, exactly the time derivative with pre-factor in (3.8), (3.14), (3.21), respectively. Replacing each $|\Omega_{ij}|\,\frac{du_{ij}}{dt}$ by the sum of the approximations from the diffusion term (noting that $f \equiv 0$) yields a large sum of approximations with common factor $-D$ (since the h^2 cancel). To see that this sum is 0, notice for instance that each interior approximation u_{ij} has factor 4 in (3.8) and cancels against the $-u_{ij-1}$, $-u_{i-1j}$, $-u_{i+1j}$, $-u_{ij+1}$ from the four neighboring cells; this remains true if the neighboring cell is a boundary cell with (3.14). In turn, boundary approximations u_{ij} have factor 2 in (3.14) and cancel against one term from the neighboring interior cell with factor 1 and one term each from the two neighboring boundary cells with factor 1/2; this remains true if one of the neighboring cells is a corner cell with (3.21). In this way, the final sum in (3.26) is 0, which shows the mass conservation on the discrete level.

To see in what sense the FVM is able to handle a point source in the right-hand side function $f(x, y, t)$ in (2.1), consider first how the finite difference method would have a problem: Recall that with assumed N_0 to be an odd integer, such that the center point $(0, 0)$ of the domain Ω in (3.1) is a mesh point $(x_{i_c}, y_{j_c}) = (0, 0)$ for some (i_c, j_c). If one attempts to discretize $f(x, y, t) = \kappa\,\delta_{(0,0)}(x, y)\,\chi_{[0,8]}(t)$ from Section 2.1 by the finite difference method, $f(x_i, y_j, t)$ involves $\delta_{(0,0)}(x_i, y_j)$, which is infinite at for $(i, j) = (i_c, j_c)$. By contrast, for a FVM discretization, we integrate also over the right-hand side of the PDE (2.1) as

$$\iint_{\Omega_{ij}} f\,dx\,dy = \kappa \iint_{\Omega_{ij}} \delta_{(0,0)}(x, y)\;dx\,dy\;\chi_{[0,8]}(t) = \kappa\,\delta_{ii_c}\,\delta_{jj_c}\,\chi_{[0,8]}(t), \tag{3.27}$$

which is 0 if $(i,j) \neq (i_c,j_c)$ or $t > 8$ and is κ if $(i,j) = (i_c,j_c)$ and $0 \leq t \leq 8$. This result now replaces (3.5) in the derivation of (3.8), thus noting the division by h^2 in the last step of this derivation, the right-hand side of (3.8) reads κ/h^2 for $(i,j) = (i_c,j_c)$ and $0 \leq t \leq 8$ and 0 otherwise. Therefore, the right-hand side of (3.23) has components $\mathbf{f}_k(t) = \kappa/h^2$ for $k = i_c + N_0(j_c - 1)$ and $0 \leq t \leq 8$ and 0 otherwise, which clearly involves no infinite or undefined values and makes the right-hand side vector $\mathbf{f}(t) \in \mathbb{R}^N$ in (3.23) well-defined.

3.2 Time Discretization

Since the semi-discretization of a parabolic PDE such as (2.1) is necessarily a stiff system of ODEs, we need an appropriate ODE solver, namely an ODE solver that is approriate for stiff ODEs. There are sophisticated choices available, such as the family of NDFk methods implemented in Matlab's `ode15s` function. We choose here the simplest stiff ODE solver, namely the implicit Euler method, because this time discretization will result in a full discretization that can be particulary easily coded, if code for a stationary Poisson equation is available.

The implicit Euler method is based on using the backward finite difference

$$\frac{d\mathbf{u}(t+\Delta t)}{dt} \approx \frac{\mathbf{u}(t+\Delta t) - \mathbf{u}(t)}{\Delta t} \tag{3.28}$$

of the time derivative with a positive constant Δt in the ODE system. For a method with constant time step Δt, we introduce the time discretization by the discrete times $t_n := n\,\Delta t$ for $n = 0, 1, \ldots, N_t$, where Δt and N_t are such that $N_t\,\Delta t = t_{fin}$ for the final time t_{fin}. The implicit Euler method is then derived by first evaluating the semi-discretization (3.23) at time $t = t_{n+1} = t_n + \Delta t$ to yield

$$\frac{\mathbf{u}(t_{n+1}) - \mathbf{u}(t_n)}{\Delta t} + \frac{D}{h^2}\,A\,\mathbf{u}(t_{n+1}) \approx \mathbf{f}(t_{n+1}) \tag{3.29}$$

and second using this as defining equation for the approximations $\mathbf{u}^{(n)} \approx \mathbf{u}(t_n)$ as

$$\frac{\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}}{\Delta t} + \frac{D}{h^2}\,A\,\mathbf{u}^{(n+1)} = \mathbf{f}^{(n+1)} \tag{3.30}$$

where we also use the shorthand notation $\mathbf{f}^{(n)} := \mathbf{f}(t_n)$. Multiplying (3.30) by the time step Δt to produce

$$\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)} + \frac{D\Delta t}{h^2}\,A\,\mathbf{u}^{(n+1)} = \Delta t\,\mathbf{f}^{(n+1)}.$$

At this point this equation has unknowns and knowns on both sides. Organizing such that the unknown vector $\mathbf{u}^{(n+1)} \in \mathbb{R}^N$ appears only on the left-hand side and all terms with $\mathbf{u}^{(n)} \in \mathbb{R}^N$ and $\mathbf{f}^{(n+1)} \in \mathbb{R}^N$ that are known at time t_n on the right-hand side yields

$$\mathbf{u}^{(n+1)} + D\,\frac{\Delta t}{h^2}\,A\,\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \Delta t\,\mathbf{f}^{(n+1)}.$$

This can finally be arranged in the form of a system of linear equations that needs to be solved at every time step t_n, $n = 0, 1, \ldots, N_t - 1$, to give the outline of the algorithm: Initialize $\mathbf{u}^{(0)} = (\mathbf{u}_k(0)) = (u_{ij}(0)) = (u_{ini}(x_i, y_j)) \in \mathbb{R}^N$, then

$$\text{Solve } \left(I + D\tfrac{\Delta t}{h^2}\, A\right) \mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \Delta t\, \mathbf{f}^{(n+1)} \qquad \text{for } n = 0, 1, \ldots, N_t - 1. \tag{3.31}$$

Here, the matrix $A = I \otimes T + T \otimes I \in \mathbb{R}^{N \times N}$ is computed as sum of Kronecker products between the identity matrix $I \in \mathbb{R}^{N_0 \times N_0}$ and the tri-diagonal matrix $T \in \mathbb{R}^{N_0 \times N_0}$ in (3.24).

3.3 Linear Solver

We observe that the system (3.31) that has to be solved at every time step t_n is linear in the unknown vector $\mathbf{u}^{(n+1)} \in \mathbb{R}^N$. Moreover, the system matrix $I + D\frac{\Delta t}{h^2}\, A \in \mathbb{R}^{N \times N}$ is large and sparse. Therefore, the conjugate gradient (CG) method is a suitable linear solver, which iteratively computes a solution to a linear system, starting from a chosen initial guess, such that the Euclidean norm of the relative residual of the solution is less than a selected tolerance.

3.4 Implementation in Matlab

The Matlab implementation of the method is based on the code supplied for the Poisson problem $-\Delta u = f$ in [2]. The `setupA` function in that code can be readily modified to implement the calculation of A, as it appears here, then A is multiplied by scalar $D\Delta t/h^2$ and 1 added to all diagonal elements. We use here that the system matrix $I + D\frac{\Delta t}{h^2}\, A \in \mathbb{R}^{N \times N}$ is the same at all time steps. The main code is then extended to enclose the call to the CG method in a for loop on the time step index $n = 0, 1, \ldots, N_t - 1$, where the right-hand side of the system is computed each in time step as $\mathbf{b} = \mathbf{u}^{(n)} + \Delta t\, \mathbf{f}^{(n+1)}$. Note that the solution $\mathbf{u}^{(n)}$ at the current time step t_n is used as initial guess for the CG method that computes $\mathbf{u}^{(n+1)}$. For reasonably small time steps and for a smooth solution in time, this initial guess is typically very good, and only few iterations per time step are required.

3.5 Implementation in parallel using C and MPI

Using the working and debugged Matlab code as a guide, we proceeded to write a serial C code that was tested to give the same results as the Matlab code for a test problem with known solution. This C code was the parallelized with MPI to allow for speedup by using multiple compute nodes. We used for both C codes an available implementation of a parallel CG method provied by Dr. Gobbert. This code uses a matrix-free implementation of the linear solver, in which the system matrix of the linear system is not formed as a matrix, but only the operation of a matrix-vector product with a vector is coded. This approach saves memory, since no matrix is ever stored.

4 Results

4.1 Test Problem with Smooth Source Term

With the finite volume method derived in Section 3, the next task is to implement it. To do so we utilize Matlab as it has straight forward syntax and a number of useful built-in functions which we can utilize. Namely, it has the conjugate gradient (CG) linear solver and the Kroneker product, the first of which is important for solving the matrix equation derived as (3.31) and the second which we use to create the matrix A in that same expression.

To show that our method is generally applicable and accurately solves a partial differential equation of the form of (2.1)–(2.3), we utilize the test problem specified in Section 2.2. This test problem satisfies both Neumann and Dirichlet boundary conditions, making it ideal for a general examination of our method's flexibility. Note that the test problem has a number of constants in it whose value we can set. The most important of these is $\tau = 8$. This parameter controls how quickly the test problem approaches a steady state, thereby defining the time evolution we expect to see. We have already defined our primary problem statement as a mass injection such that the function u is growing from $t = 0$ to $t = 8$ hours and diffusing afterward, hence why we set our test problem now to grow in that same interval with $\tau = 8$. We choose $D = 10$, domain $\Omega = (-50, 50) \times (-50, 50)$, and initial condition $u_{ini} \equiv 0$ as in the pollution problem.

There must also be defined a few numerical parameters, once we have written our method. Namely, we must choose the size of our timesteps ($\Delta t = 10^{-1}$) such that there is minimal build up of error as we iterate over time, the tolerance of our CG method ($\text{tol} = 10^{-9}$) such that the method accurately converges to the right solution, and the computational size of our mesh (here 129×129 cells, to start with). The first two of these parameters are set mainly by observation and go unchanged in our tests, while the third (coded as N_0) is later shown to be usefully varied for the adjustment of spatial resolution.

The output of our test problem, solved by finite volume method (FVM), is shown in Figure 4.1. Note that our choice of τ was consistent with significant growth until $t = 8$ followed by an approach to a steady state. Intuitively, this plot looks reasonable, but the value of using a test problem is that it has a known solution, (2.5). Using this easily verified function, it is straightforward in Matlab to subtract the known solution from the calculated solution everywhere and show the difference.

In Figure 4.2 the results of such a comparison are shown. Here it is important to note the scale of the result (above the vertical axis). Namely, the entirety of the structure shown is on the order of about 10^{-3}. That is to say: the error is generally small. This is good, but it is worth noting the structure as well. Namely, we see the largest error at the center, where the problem is changing rapidly, and around the edges where our boundary approximations are made deriving the FVM.

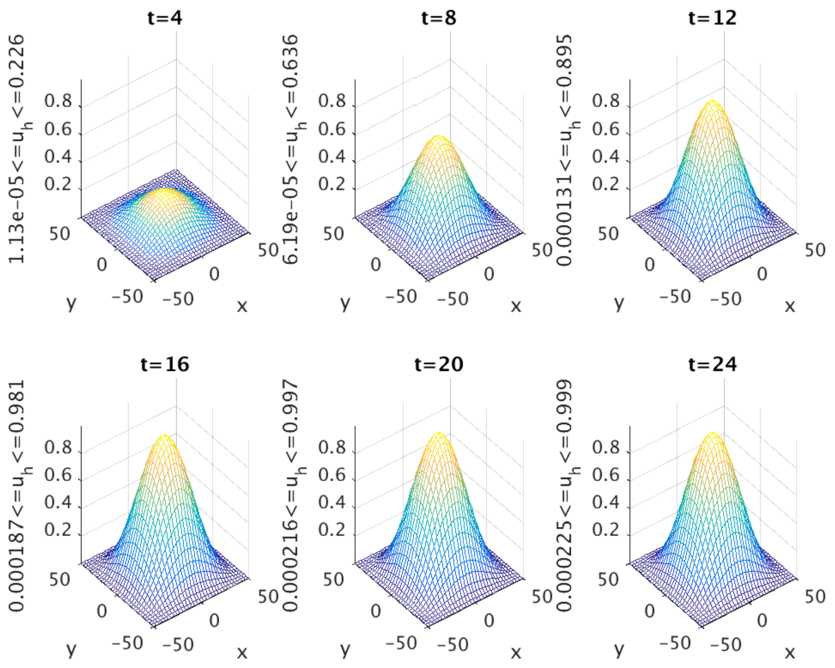

Figure 4.1: Numerical solution of the test problem at $t = 4, 8, \ldots, 24$.

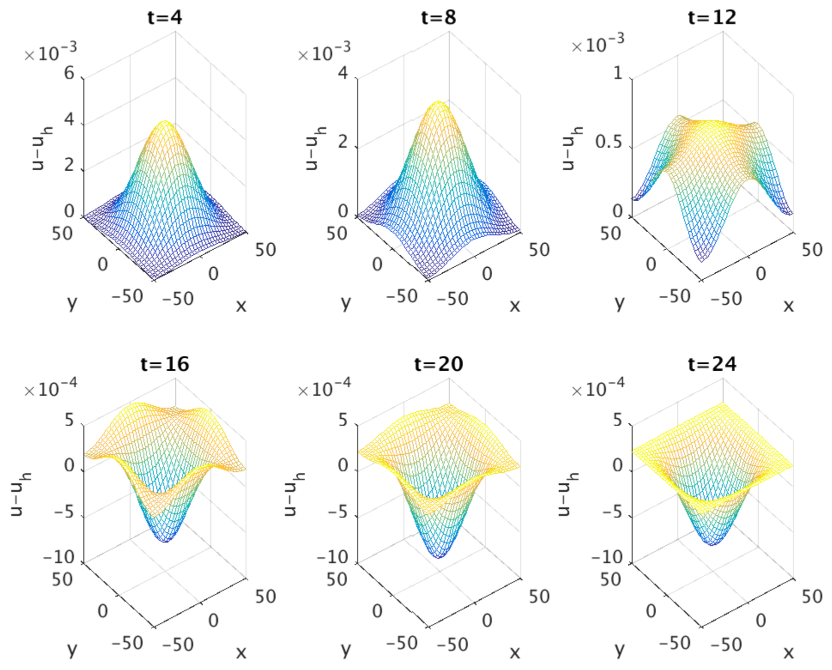

Figure 4.2: Numerical error of the test problem at $t = 4, 8, \ldots, 24$.

It is also useful to see how our backend methods are working and what the exact error peaks as. To do this we have Matlab output a number of useful data points shown in Table 4.1. Here, the column "enorminf" references the maximum of the absolute (sign independent) error shown in Figure 4.2. The values n are the number of steps through time (with t_n then being the time at that step), "it" is the number of iterations of the CG method at time step t_n, and "cumit" is the cumulative iteration count. As one might expect the values "min" and "max" are simply the minimum and maximum of the numerical solution of u.

Table 4.1: Numerical data of the test problem at $t = 4, 8, \ldots, 24$.

n	t_n	it	cumit	min(u)	max(u)	enorminf
39	4.0	7	273	1.131256e-05	2.256441e-01	4.444927e-03
79	8.0	7	561	6.185924e-05	6.356576e-01	3.537069e-03
119	12.0	8	850	1.314239e-04	8.953037e-01	7.029653e-04
159	16.0	7	1140	1.866904e-04	9.810095e-01	6.748332e-04
199	20.0	5	1420	2.164897e-04	9.972224e-01	8.470974e-04
239	24.0	5	1619	2.250605e-04	9.991729e-01	7.037356e-04

4.2 Pollution Problem on Original Numerical Domain

Confident now in the fact that our method can solve a PDE of the form of (2.1)–(2.3), it is time to return to the pollution problem defined in Section 2.1. There, our source function f is the point source (2.4) with amount κ material injected per hour into the domain at the center point $(0, 0)$. Recall that the FVM was pointedly chosen because of its ability to handle such a source and here we will begin exploring the results of that choice. To begin, we choose the value $\kappa = 10$ kg km^{-2} h^{-1} and run the simulation on $\Omega = (-50, 50) \times (-50, 50)$ in units of km. The snapshots of the concentration across the domain at six times are shown in Figure 4.3.

When examining the result, we first look to see that it makes intuitive sense. Namely, the point source is present from $t = 0$ to $t = 8$ h, and afterward the mass it has injected begins to spread out. Figure 4.3 shows this behavior, but to be truly sure of the result we need to examine quantitatively the numerical output.

Again we use direct Matlab output and put the results in Table 4.2. This table is similar to the output from the test problem shown in Table 4.1. Note though that we do not have a true solution to compare to and therefore no error. So instead we check the mass conservation of the system by integrating over the entire calculation space. The result is that the mass increases by κ for each hour to reach 80 kg by $t = 8$ h, and then that value is constant afterward, corresponding to when the source is turned off and the mass is just spreading out. This verifies not only that the FVM can handle point sources, but that it is mass conservative as well, making it ideal for physical modelling.

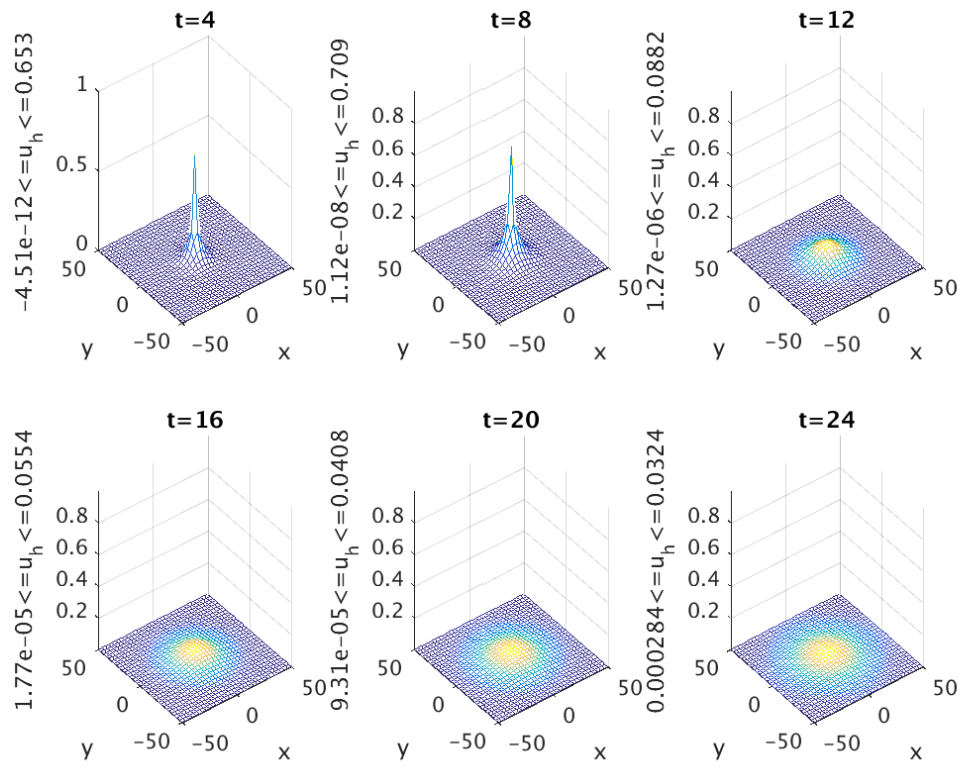

Figure 4.3: Numerical solution of pollution problem with $\kappa = 10$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$ using 129×129 mesh, and on the numerical domain $(-50, 50) \times (-50, 50)$ km^2.

Table 4.2: Numerical data of pollution problem with $\kappa = 10$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$ using 129×129 mesh, and on the numerical domain $(-50, 50) \times (-50, 50)$ km^2.

n	t_n	it	cumit	min(u)	max(u)	mass
39	4.0	12	755	-4.510617e-12	6.534136e-01	4.000000e+01
79	8.0	11	1182	1.117690e-08	7.091521e-01	8.000000e+01
119	12.0	11	1926	1.272000e-06	8.819662e-02	8.000000e+01
159	16.0	10	2357	1.767457e-05	5.544710e-02	8.000000e+01
199	20.0	10	2753	9.312632e-05	4.080380e-02	8.000000e+01
239	24.0	8	3135	2.842079e-04	3.236277e-02	8.000000e+01

4.3 Pollution Problem on Enlarged Numerical Domain

In the previous subsection, the pollution problem is solved on the domain $(-50, 50) \times (-50, 50)$, which is both the numerical domain and the domain of interest. Our Neumann boundary condition at the boundary of the numerical domain by definition prevents material flow across the numerical boundary, thereby making our system closed. But in reality, pollution should not be contained by the system, but be allowed to spread beyond the boundary of the domain of interest. We simulate this effect of an open system by increasing the size of the numerical domain to $(-800, 800) \times (-800, 800)$, while only considering the domain of interest $(-50, 50) \times (-50, 50)$ in the results. In this way, the effect of the Neumann boundary condition at the numerical domain is far removed from the domain of interest, and material can flow across its boundary, which simulates an open system. This is what is done in Figures 4.4 through 4.7 and the associated Tables 4.3 through 4.6 in this section. Note that the numerical domain is discretized by a $N_0 \times N_0 = 129 \times 129$ mesh with uniform mesh spacing $h = 100/128 = 0.78125$ km in the previous section. In order to use the same mesh spacing, we use $N_0 \times N_0 = 2049 \times 2049$ in this section, since this gives the same value $h = 1600/2048 = 0.78125$ km.

The result in Figure 4.4 looks similar to the previous one, Figure 4.3, but has slightly smaller minima along the boundary of the plots. This is confirmed when we look at the output table, Table 4.3. Namely, the very small negative numbers for the minimum of the solution in the table show that the concentration at the boudnary of the larger numerical domain is still essentially 0 there, i.e., pollution injected at center has not reached the now larger numerical boundary. Note that the numerical method does not enforce non-negativity of solution, despite concentration of course physically being a non-negative quantity. So, the observed behavior constitutes a good check of the credibility of the numerical method on the numerical domain. Now, examine also the ranges of values displayed in the plots in Figure 4.4, which are the minimum and maximum values on the domain of interest $(-50, 50) \times (-50, 50)$ only. Those exhibit smaller minima than in Figure 4.3, thus confirming that the close system there had held back more material in the domain. The maxima in both Figures 4.3 and 4.4 and Tables 4.2 and 4.3 are very close though, which confirms that the effect of the injection at the center itself is represented equivalently in both cases.

In the remaining results in this section, we simulate progressively increasing amounts of pollutants being injected into the domain by larger values of $\kappa = 20, 40, 80$ kg km^{-2} h^{-1}. Results are shown in Figures 4.5 through 4.7 and Tables 4.4 through 4.6, respectively. Note that for one thing, we see that the maximum values of the solution increase, namely about double for doubling κ, in Figures 4.4 through 4.7. More precisely, Tables 4.3 through 4.6 show that the total mass exactly doubles for each doubling of κ. This confirms the physical accuracy of the simulations. Moreover, this increasing κ tests how long our method of simulating an open system remains valid. We can see that even for the largest κ considered, the minimum values of the concentration on the numerical boundary are reported as small negative numbers, which confirms that the pollution has not reached there, thus the plots on the smaller domain of interest are still reliable, since no mass has diffused back there from the numerical boundary.

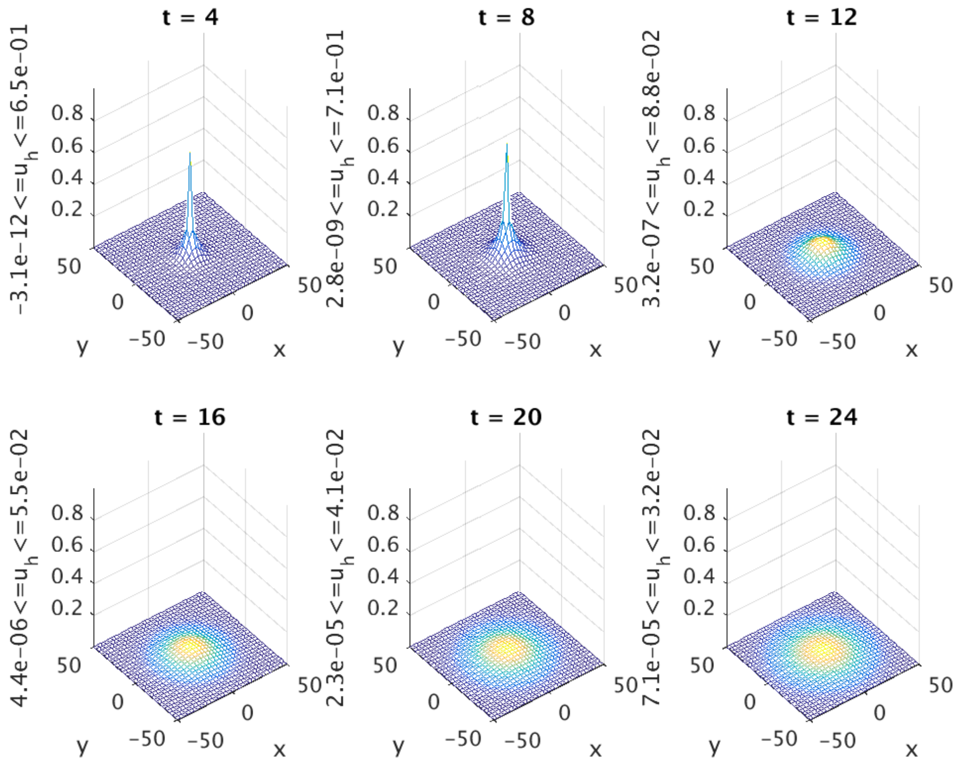

Figure 4.4: Numerical solution of pollution problem with $\kappa = 10$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

Table 4.3: Numerical data of pollution problem with $\kappa = 10$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

n	t_n	it	cumit	min(u)	max(u)	mass
39	4.0	12	757	-3.0767e-12	6.5341e-01	4.0000e+01
79	8.0	10	1181	-2.8013e-12	7.0915e-01	8.0000e+01
119	12.0	11	1925	-5.4151e-13	8.8197e-02	8.0000e+01
159	16.0	10	2356	-6.6028e-13	5.5447e-02	8.0000e+01
199	20.0	10	2750	-7.0221e-13	4.0804e-02	8.0000e+01
239	24.0	8	3133	-3.0485e-13	3.2362e-02	8.0000e+01

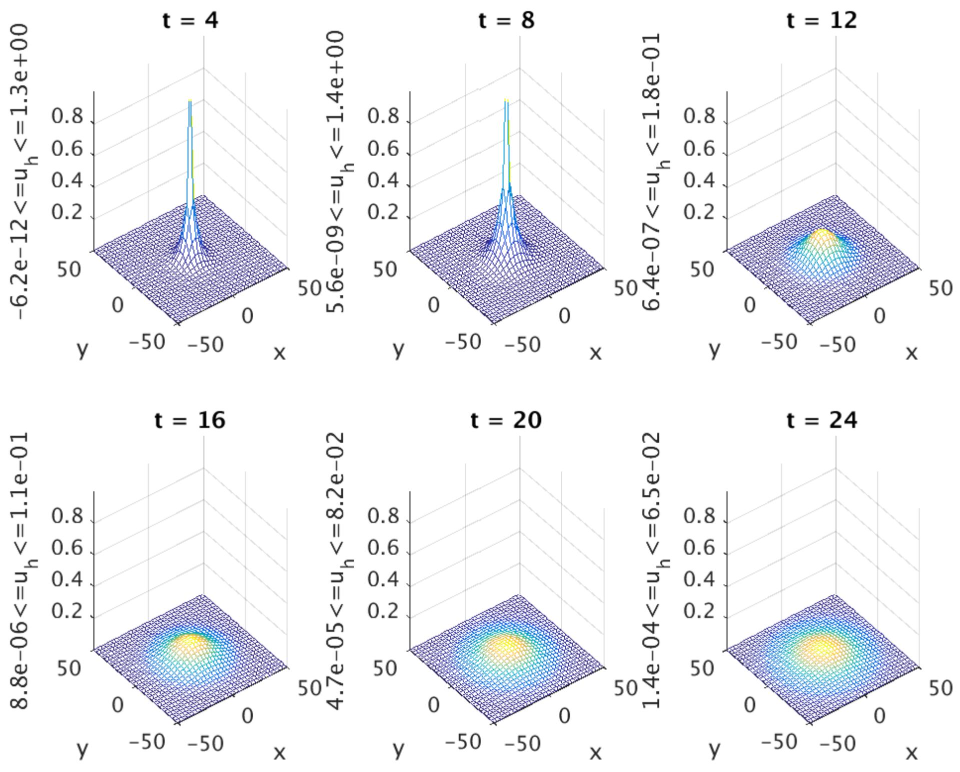

Figure 4.5: Numerical solution of pollution problem with $\kappa = 20$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

Table 4.4: Numerical data of pollution problem with $\kappa = 20$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

n	t_n	it	cumit	min(u)	max(u)	mass
39	4.0	12	757	-6.1534e-12	1.3068e+00	8.0000e+01
79	8.0	10	1181	-5.6026e-12	1.4183e+00	1.6000e+02
119	12.0	11	1925	-1.0830e-12	1.7639e-01	1.6000e+02
159	16.0	10	2356	-1.3206e-12	1.1089e-01	1.6000e+02
199	20.0	10	2750	-1.4044e-12	8.1607e-02	1.6000e+02
239	24.0	8	3133	-6.0971e-13	6.4723e-02	1.6000e+02

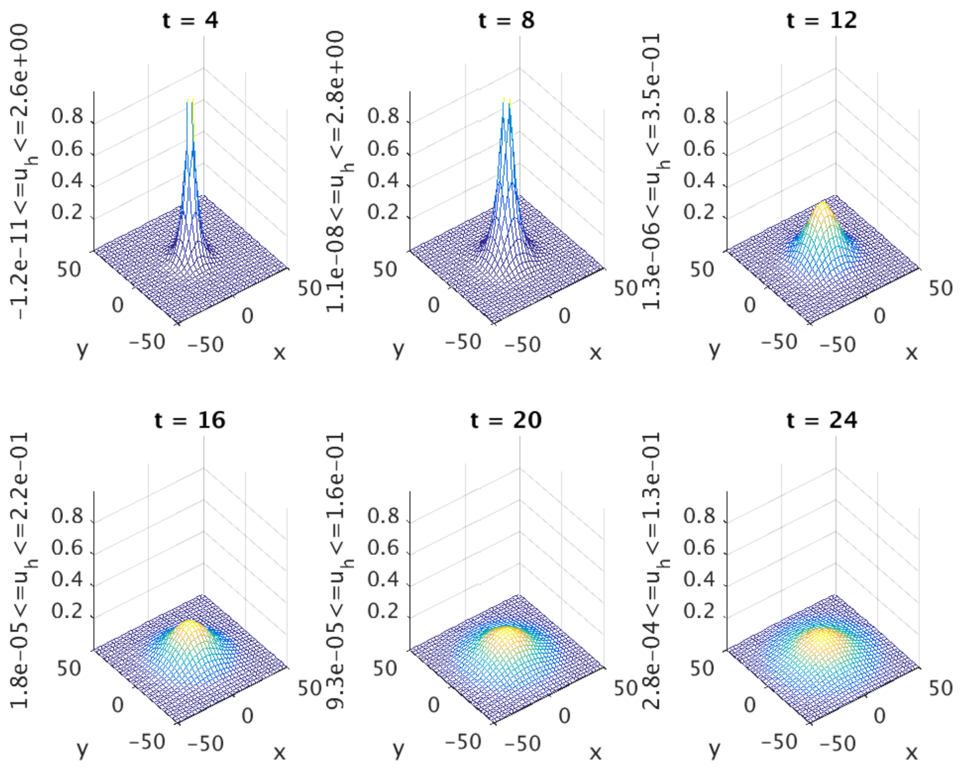

Figure 4.6: Numerical solution of pollution problem with $\kappa = 40$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

Table 4.5: Numerical data of pollution problem with $\kappa = 40$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

n	t_n	it	cumit	min(u)	max(u)	mass
39	4.0	12	757	-1.2307e-11	2.6137e+00	1.6000e+02
79	8.0	10	1181	-1.1205e-11	2.8366e+00	3.2000e+02
119	12.0	11	1925	-2.1660e-12	3.5279e-01	3.2000e+02
159	16.0	10	2356	-2.6411e-12	2.2179e-01	3.2000e+02
199	20.0	10	2750	-2.8088e-12	1.6321e-01	3.2000e+02
239	24.0	8	3133	-1.2194e-12	1.2945e-01	3.2000e+02

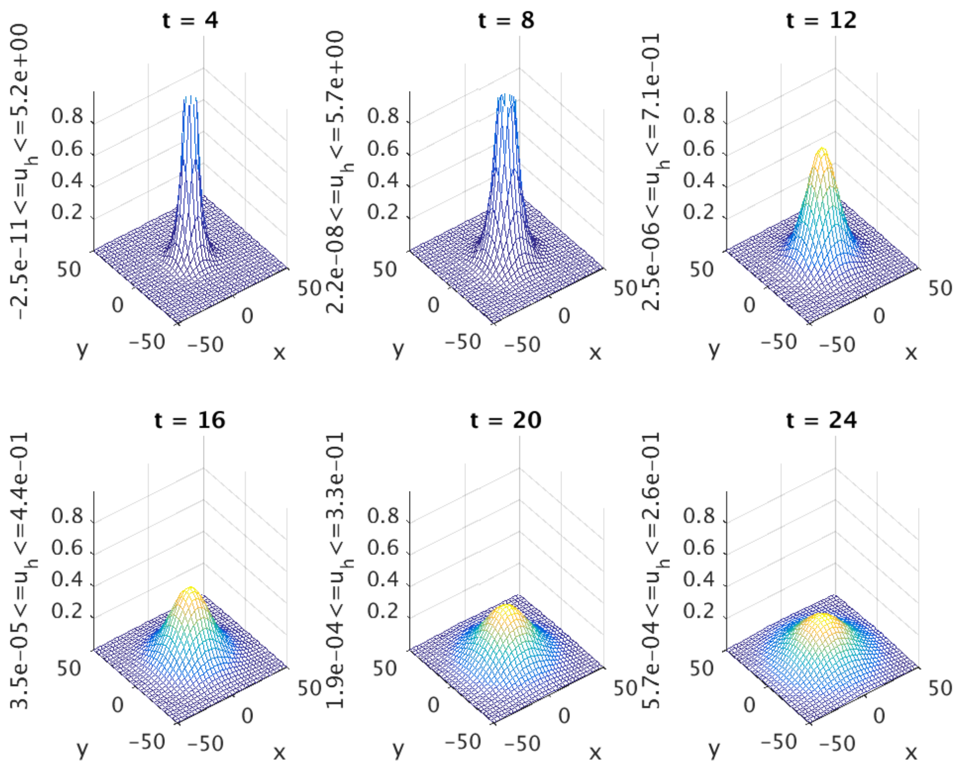

Figure 4.7: Numerical solution of pollution problem with $\kappa = 80$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

Table 4.6: Numerical data of pollution problem with $\kappa = 80$ kg km^{-2} h^{-1} at $t = 4, 8, \ldots, 24$. 2049×2049 mesh, on the numerical domain $(-800, 800) \times (-800, 800)$ km^2.

n	t_n	it	cumit	min(u)	max(u)	mass
39	4.0	12	757	-2.4614e-11	5.2273e+00	3.2000e+02
79	8.0	10	1181	-2.2410e-11	5.6732e+00	6.4000e+02
119	12.0	11	1925	-4.3321e-12	7.0557e-01	6.4000e+02
159	16.0	10	2356	-5.2822e-12	4.4358e-01	6.4000e+02
199	20.0	10	2750	-5.6177e-12	3.2643e-01	6.4000e+02
239	24.0	8	3133	-2.4388e-12	2.5889e-01	6.4000e+02

4.4 Parallel Performance Study

The previous section demonstrated the need for an enlarged numerical domain. This necessitates a larger numerical mesh in order to maintain the same numerical mesh spacing and thus same spatial error. Unsurprisingly, the calculation time of those simulations is longer the larger the mesh is. This motivates the parallelization of the code, since by spreading the work caused by the increasing mesh sizes across multiple nodes, the simulation times will not increase with the mesh, but can actually be decreased by using more nodes.

Thankfully, the CG method can be readily parallelized. Working with matrices and arrays also invites us to split these arrays up across multiple processes for rapid calculation on them. The next step of the project is therefore, now that we have proven the concept, to take our code in Matlab and translate it to C where we can use MPI to do just such a parallelization and improve our runtimes for larger and more precise mesh sizes. This speedup can be best seen by a comparison between numerous data runs of various mesh sizes on various numbers of processes.

This section describes a parallel performance study for the solution of the test problem from Section 2.2 on the 2013 portion of maya in the UMBC High Performance Computing Facility (`hpcf.umbc.edu`). The compute nodes are made up of two eight-core 2.6 GHz Intel E5–2650v2 Ivy Bridge CPUs. The 64 GB of the node's memory is formed by eight 8 GB DIMMs, four of which are connected to each CPU. The two CPUs of a node are connected to each other by two QPI (quick path interconnect) links. The nodes in maya 2013 are connected by a quad-data rate InfiniBand interconnect [1]

The results in this section use the default Intel compiler and Intel MPI. The SLURM submission script uses the srun command to start the job. The number of nodes are controlled by the `--nodes` option in the SLURM submission script, and the number of processes per node by the `--ntasks-per-node` option. Each node that is used is dedicated to the job with remaining cores idling, using `--exclusive`. The assignment of the MPI processes to the cores of the two CPUs on the node uses the default assignment, in which consecutive processes are distributed in alternating fashion between the two CPUs. We conduct numerical experiments of the test problem for three progressively finer meshes of $N_0 = 2048$, 4096, and 8192. For each mesh resolution, the parallel implementation of the test problem is run on all possible combinations of 1, 2, 4, 8, and 16 nodes with 1, 2, 4, 8, 16 processes per node.

Table 4.7 collects the results of the performance study. The table summarizes the observed wall clock time (total time to execute the code) in HH:MM:SS (hours:minutes:seconds) format. Note that the table for 4096×4096 shows very nearly linear speedup for the upper two rows. Also the columns show speedup, sometimes even better than linear agreement is seen (e.g., from 1 to 2 processes per node in many cases), though eventually the number of cores on a CPU in use exceeds the number of memory channels, leading to no more gain (from 8 to 16 processes per node in most cases). If the problem is too small, such as the 2048×2048 mesh, there are diminishing returns to parallelization, since the cost of passing messages and accessing memory begins to take more time than is gained by distributing the work. Interestingly, also for the largest mesh considered, the benefit of using more processes per node is eventually limited, though speedup along the rows continues to be excellent.

Table 4.7: Run times for MPI code on maya using test problem in HH:MM:SS format. ET indicates that the run requires excessive time (over 4 hours).

2048×2048 mesh	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	00:45:37	00:21:08	00:09:29	00:03:37	00:01:54
2 processes per node	00:26:55	00:11:00	00:03:43	00:01:52	00:01:03
4 processes per node	00:15:11	00:05:02	00:01:59	00:01:03	00:00:40
8 processes per node	00:15:00	00:03:27	00:01:01	00:00:42	00:00:34
16 processes per node	00:07:26	00:02:56	00:00:52	00:00:44	00:00:48
4096×4096 mesh	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	05:32:08	04:03:17	01:59:11	01:00:06	00:25:19
2 processes per node	03:36:05	01:54:10	00:55:51	00:20:40	00:09:37
4 processes per node	02:08:38	00:59:16	00:29:27	00:11:22	00:04:33
8 processes per node	02:01:09	00:34:32	00:17:34	00:07:44	00:03:50
16 processes per node	01:02:55	00:32:07	00:17:09	00:07:08	00:03:01
8192×8192 mesh	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	ET	ET	ET	ET	03:03:53
2 processes per node	ET	ET	ET	02:14:52	01:27:15
4 processes per node	ET	ET	03:18:48	01:18:37	00:50:25
8 processes per node	ET	03:20:35	01:44:31	01:00:33	00:41:29
16 processes per node	ET	03:09:49	01:43:47	01:03:01	00:54:55

5 Conclusions

In conclusion, we show that the finite volume method (FVM) is an ideal numerical scheme for computing the solutions of mass conservative partial differential equations. The method also performs well in handling the usage of delta distributions, and both these properties make it a good method for the handling of physical situations like those of the pollution problem defined in Section 2.

The CG method used to solve the linear systems from the derived form of the FVM in Section 3 strongly invites the use of paralellization using MPI, where the conjugate gradient method and linearity of the FVM are easy to distribute across threads. Further, these explorations show that the solution to our pollution problem by FVM can have flexible boundaries that allow for the modeling of material in a closed system, or material flow out of an open system. We can simulate open boundaries with flow allowed across them by increasing the size of our numerical domain and doing analysis only on the smaller subdomain of interst. The efficiency granted by MPI allows this nicely, as such problems require large meshes compared to the sub domain of interest to prevent return flow of material, and to maintain fine spatial resolution on such a mesh we must increase the number of mesh cells proportionally.

The resultant solver of the pollution problem created here thereby represents a sturdy stepping stone for future projects. Confident now in the stability and accuracy of our solution, it is now possible to begin exploring the physical ramifications of the system. Ideally, we might combine our results with **Big Data** analysis to show that it should be possible to predict κ (assuming first that it is not known) from the value of u at a set of points in the mesh.

Ultimately, this project satisfyingly demonstrates the applicability of the FVM and provides a means for future projects to examine a simple atmospheric system with applications in urban planning and perhaps geological pursuits (if the point source is perhaps considered a volcano instead of a factory). The flexibility to add multiple source terms, adjust the domains of interest, and readily change the source function f gives this solution a wide range of possible uses.

Acknowledgments

This work is supported by the grant CyberTraining: DSE: Cross-Training of Researchers in Computing, Applied Mathematics and Atmospheric Sciences using Advanced Cyberinfrastructure Resources from the National Science Foundation (grant no. OAC–1730250). Co-author Noah Sienkiewicz additionally acknowledges a GAANN Fellowship from the Department of Education (P200A150003). The hardware in the UMBC High Performance Computing Facility (HPCF) is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258, CNS–1228778, and OAC–1726023) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `hpcf.umbc.edu` for more information on HPCF and the projects using its resources. Co-author Carlos Barajas was supported as HPCF RA.

References

[1] Samuel Khuvis and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on the cluster maya. Technical Report HPCF–2015–6, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015.

[2] Sai K. Popuri and Matthias K. Gobbert. A comparative evaluation of Matlab, Octave, R, and Julia on Maya. Technical Report HPCF–2017–3, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2017.

[3] Jianwu Wang, Matthias K. Gobbert, Zhibo Zhang, Aryya Gangopadhyay, and Glenn G. Page. Multidisciplinary education on Big Data + HPC + Atmospheric Sciences. *EduHPC-17: Workshop on Education for High-Performance Computing*, 8 pages, in press (2017).