Comparison of Lagrangian Coherent Structures and Relative Dispersion in a Mixing Layer

Tadbhagya Kumar

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Engineering

University of Washington

2016

Reading Committee:

James J. Riley, Chair

Steven L. Brunton, Chair

Duane Storti

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Comparison of Lagrangian Coherent Structures
and Relative Dispersion in a Mixing Layer

Tadbhagya Kumar

Co-Chairs of the Supervisory Committee:
Professor James J. Riley
Mechanical Engineering

Assistant Professor Steven L. Brunton
Mechanical Engineering

Lagrangian coherent structures (LCS) and relative dispersion are two widely used tools to study mixing and have been successfully applied to a wide variety of flows. Their computation requires particle advection by numerical integration of the velocity field. We have developed a CUDA-based 2D flow solver using a Fourier-spectral method with a second-order Adams-Bashforth time-stepping method to solve the Navier-Stokes and scalar diffusion equations for a perturbed, temporally-growing, two-dimensional incompressible mixing layer. The resulting simulation data are used to compute LCS and relative dispersion R^2. LCS is computed using finite time Lyapunov exponents (σ). Contours plots are used to visualize these two scalar fields and it is found that maximum dispersion (R^2) values correspond to the ridges in the FTLE field. We further compute normalized relative dispersion (λ_d) and compare it with σ. It is shown that λ_d and σ provide qualitatively the same information. Another quantity Γ, defined as the ratio of λ_d and σ, is computed to relate the two quantities. The implications and future directions of our research are suggested

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am thankful to my research advisers James J. Riley and Steven L. Brunton for their continuous support, guidance and encouragement over the research period. They have been a huge source of inspiration and have helped me to do good work on this project. I would also like to thank Duane Storti, for his class introduced me to the world of parallel computing and provided me access to machines running cutting edge GPUs. Thanks are due to the Mechanical Engineering Department for providing me with Teaching Assistant opportunities.

This work would not have been possible without the constant support of my family and friends. My sister Nalini Kumar has been inspirational and has helped me through various stagnant points in research. I thank my parents for their moral, emotional and financial support throughout my stay in Seattle. I thank Anirudh Gupta, Atinder Saini, Srivatsa CV and Maanas Maheshwari for sharing graduate school anxiety and being there in good and bad times over the course of two years. I would also like to acknowledge the support of those who are far especially Amit Pawar, Mohit Yadav and Umang Rawat for being close friends to talk with. Lastly, I would like to thank all others who made this masters journey cherishable.

Chapter 1
INTRODUCTION

1.1 Overview

Understanding how turbulent mixing occurs for a wide variety of flows has been one of the fundamental research problems in fluid mechanics. *Shear flows* are perhaps one of the most widely occurring class of flows and are frequently encountered both in nature and engineering applications. Like all other turbulent flows, they exhibit complex behavior and are characterized by a large range of length and time scales. Shear flows can be classified as wall-bounded and free shear, both of which are commonly occurring and have garnered attention over the years. Free shear flows are have mean velocity gradients that develop in the absence of boundaries and are usually turbulent. The jet of air issuing out of one's nostrils, the smoke plume from a chimney, the wake behind moving objects, and the exhaust from engines are some examples of free shear flows which occur frequently around us. One of their most important features is the presence of large scale structures or eddies, which play an important role in mixing processes.

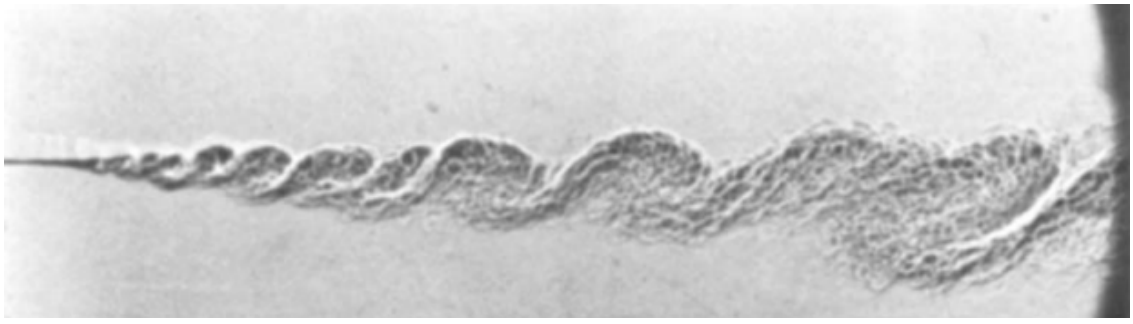


Figure 1.1: Shadow graph image of mixing layer between Nitrogen and Helium (adapted from Brown and Roshko, 1974).

The mixing layer serves as a model problem for studying instabilities in free shear flows. Alternatively referred to as a shear layer, the mixing layer consists of two parallel streams of fluid moving at different velocities with the presence of disturbance at their interface. Due to presence of the disturbance, the layer rolls up forming vortex-like structures. Michalke and Wille [1966], Freymuth [1966], and Brown and Roshko [1974] experimentally observed large coherent eddies in plane turbulent mixing layers. Based on their observations, they suggested that turbulent mixing is a process of large-scale entrainment resulting from entanglement of coherent structures. These and several other seminal works laid the foundation for subsequent research in turbulent mixing and the process of entrainment.

The mixing caused by these eddies raises the question: How much mixing has actually occurred? An important measure of mixing is the amount of dispersion by the flow field. Dispersion measures the amount of spreading of particles by the flow and can provide insight into its spatial structure and key transport features. Thus, dispersion has been a well studied measure of mixing for almost a century. The idea was first introduced by G.I. Taylor for a single particle, and was extended into two particle relative dispersion by Richardson. Batchelor [1952], Binnie and Phillips [1955], Elder [1959], Crowe et al. [1988] are some who have worked on dispersion and mixing in turbulent shear flows. There is a depth of literature available on dispersion and is discussed in more detail in Chapter 2.

Alternative methods involving particle trajectories have also been developed to study mixing. Finite size Lyapunov exponent (FSLE) (e.g., Lacorata et al. 2001; LaCasce 2008) and finite time Lyapunov exponent (FTLE) (e.g., Haller 2001, 2002; Shadden et al. 2005) are two metrics which make use of particle trajectories. These concepts have been adopted from dynamical systems theory and are based on how far these trajectories move apart in a certain interval of time (FTLE) or the time taken by them to move apart by certain distance (FSLE).

FSLE and dispersion are diagnostics that are often considered together for studying par-

ticle pair statistics. FSLE has the advantage that it contains information only from a single regime. FTLE is also related to particle separation. More recently, FTLE has been used to compute Lagrangian coherent structures (LCS), which are locally maximizing regions of stretching in the flow and act as transport barriers. They have powerful implications for transport and also provide information about underlying structures in the flow field. LCS has been successfully applied to a range of flows such as transport in the ocean (Lekien et al. 2007; Olascoaga and Haller 2012; Mezić et al. 2010), atmospheric mixing (Beron-Vera et al. 2010, 2013), cardiovascular flow (Shadden and Taylor, 2008), animal swimming (Peng and Dabiri, 2008), turbulent channel flow (Green et al., 2007), and vortex shedding (Lipinski et al. 2008). These structures often govern the stretching and folding mechanisms that control mixing. While there has been significant interest in utilizing LCS for different flows, much less attention has been put in analyzing its similarities with relative dispersion. Early research on FTLE focused on determining material lines that divided the flow into regions of different dynamics and formalizing their definitions mathematically. Bowman [1999], Von Hardenberg et al. [2000], Haller and Yuan [2000] showed strong correlations between maximum relative dispersion and FTLE. Orre et al. [2006] and Waugh et al. [2012] have compared relative dispersion and FTLE for stirring in ocean flows. They found that peaks in relative dispersion plots correspond to the material lines identified by FTLE and the growth rate inferred from both calculations is comparable.

1.2 Scope of the Study

While there has been much interest in applying these mixing diagnostics to various flows, not much attention has been put to quantify the relationship between FTLE and dispersion and comparing the information that they provide. Early work used a different formulation of FTLE and the method for their calculation has evolved since. Waugh et al. [2012] have tried to bound the relative dispersion by using relations derived from uniform strain flow. However, their method of FTLE computation utilizes velocity gradient which is different to the the method used in this work.

The scope of the study is to establish bounds on the value of relative dispersion using Lagrangian coherent structures derived from FTLE for a mixing layer. We are interested in analyzing the relationship between these two mixing diagnostics. Since these are computationally intensive tasks, an alternative focus is on utilizing GPUs to explore high performance parallel computing.

1.3 Organization of Thesis

This work is divided into six chapters. Chapter 1 talks briefly about shear flows and diagnostic measures commonly used in applied flows. Chapter 2 provides theoretical background of instability mechanism, review of dispersion and LCS, and GPU terminology. Chapter 3 discusses the numerical method used for solving Navier-Stokes equation, flow solver setup (serial and parallel), and the results obtained from simulations. In chapter 4, we discuss the computation of mixing diagnostics from velocity data and present results for both. In chapter 5, we present our findings on the relationship between the two metrics. The work culminates by summarizing our findings and a note on the future work in chapter 6.

Chapter 2

BACKGROUND

2.1 Inviscid instability of a shear layer

The process of fluid instability has intrigued and motivated researchers to seek answers about laminar to turbulent transition and to develop models to predict this onset. Linear hydrodynamics theory initiated by Rayleigh, Kelvin and Helmholtz, laid the foundation for theoretical work on transition. They considered the mixing layer as a prototype for their analysis: two parallel fluid streams of different velocities separated by an interface such that mean velocity gradients exists. At high enough Reynolds number, this flow becomes sensitive to the growth of two-dimensional sinusoidal disturbance and becomes unstable. However, this theory only describes the onset since development to fully turbulent flow is a three-dimensional process.

Figure 2.1: Plane mixing layer with mean velocity gradient (adapted from Brown and Roshko 1974)

Rayleigh [1880] derived the equation for two dimensional disturbances in an inviscid

flow with an initial velocity profile $u(y)$ and superimposed disturbances $u_1(y)(\ll u)$ and $v_1(y)(\ll u)$ on the basic state. here u is a character value of the basic state. A streamfunction ψ_1 of the disturbance can be defined as

$$u_1 = \partial\psi_1/\partial y, \quad v_1 = -\partial\psi_1/\partial x \tag{2.1}$$

Now, the stream function can be written as $\psi_1 = \phi(y)\ e^{i\alpha(x-ct)}$ where $\phi_1(y)$ is the amplitude of the disturbance, α is the wavenumber of the disturbance, $c = c_r + ic_i$ is generally complex, c_r is the phase velocity and c_i is the amplification rate of the disturbance. Substituting equation 2.1 in Euler equation and neglecting second order terms, we obtain Rayleigh stability equation.

$$[u - c][\phi'' - \alpha^2\phi] - u''\phi = 0, \tag{2.2}$$

where primes denote differentiation with respect to y. For unbounded flow, ϕ must vanish at infinity. Thus, Equation 2.2 is an eigenvalue problem with ϕ as the eigenfunction and c as the eigenvalue. Rayleigh showed that a necessary condition for the initial state $u(y)$ to be unstable to disturbances is the presence of an inflexion point. Michalke[1964] provided comprehensive analysis of instability mechanism of free shear layer. He numerically integrated Rayleigh stability equation using the initial velocity profile

$$u(y) = 0.5[1 + \tanh(y)] \tag{2.3}$$

Setting $c_r = u(0) = 0.5$, the eigenvalues, growth rate, and eigenfunctions($\phi_r(y)$ and $\phi_y(y)$) were computed for different wavenumbers. He found that maximum amplification is $\alpha c_i = 0.0949$ and occurs at $\alpha = 0.4446$ (Figure. It may be noted that this formulation is non-dimensional. The dimensional problem is defined as:

$$u = u_0[1 + \tanh(y/\delta)] \tag{2.4}$$

where $u_0 = 0.5$, $\delta = \alpha/n$, and n is an integer (the wavenumber of the most unstable mode). Further, the disturbance function becomes

$$\psi_1 = e^{\alpha c_i t}(\phi_r(y/\delta)\cos n(x - c_r t) - \phi_i(y/\delta)\sin n(x - c_r t)) \tag{2.5}$$

We have used curve fitting to determine eigenfunctions as : $\phi_r(y) = \text{sech}(\alpha y)$ and $\phi_y(y) = \tanh(y)e^{-\alpha y}$. This form of streamfunction was used to initialize disturbance in our flow solver. Rayleigh instability theory successfully described the initial roll-up of inflectional velocity profile. While his theory works well for free shear flows in which instability characteristics are determined by inviscid theory, it fails in the case of wall bounded shear flows which require viscosity for initial instability. Orr and Sommerfeld generalized Rayleighs theory for viscous incompressible fluid. The Orr Sommerfeld equation is more difficult to solve even for the simplest of the cases. In the limit of high Reynolds number, the equation reduces to Rayleigh's equation. Readers are suggested to read Bayly and Orszag [1988], who provide an excellent review of the instability mechanisms in shear flows.

2.2 Relative Dispersion

One way of understanding how mixing occurs is to study how particles move in the flow field. It is far more easy to comprehend particle trajectories than the entire velocity field. The study of particle motion has drawn the attention of the greatest minds in fluid mechanics. Taylor's [1992] paper on single particle dispersion gave rise to a number of statistical tools that are still used in studying turbulence. This was crucial as dispersion of particles gives us information about the structure of turbulence. However, it was Richardson's [1926] paper that laid the foundation of the study of two particle dispersion. He examined the motion of two particles in isotropic homogeneous turbulence. He considered the mean square separation of a pair of marked particles and used the probability density function for separation of a pair of particles to develop his famous fourth thirds law of diffusion.

The equations governing the separation distance vector $\mathbf{r}$(t) are as follows :

$$\frac{d\mathbf{r}}{dt} = \mathbf{u}(t),\ \mathbf{r}(t) = \mathbf{r}_0 + \int_0^t \mathbf{u}(t')\,dt' \tag{2.6}$$

where $\mathbf{u}$(t) is the relative velocity between the two particles. Since our main interest is in obtaining the mean squared separation, the ensemble average of squared distances can be

performed. This becomes

$$\frac{1}{2}\langle|\mathbf{r}(t)-\mathbf{r}_0|^2\rangle = \int_0^t \int_0^t \langle \mathbf{u}(t')\mathbf{u}(t'')\rangle \, dt' \, dt''. \tag{2.7}$$

In the above equation, the principal challenge is to model the relationship between separation vector and relative velocity, which is implicitly dependent on the separation vector. The process of dispersion can be divided into three regimes: a) the dissipation subrange (r(t) $\ll$ η), b) the inertial subrange ($\eta \ll r(t) \ll L$), c)the diffusion subrange ($r(t) \gg L$), where L is the integral length scale and η is the Kolmogorov length scale. Richardson's 4/3 law can also be obtained from Kolmogorov's [1941] theory of local isotropy. According to his theory, there is an equillibrium range of scales which are dominated only by the viscosity ν and kinetic energy dissipation rate $\bar{\epsilon}$. Kolmogorov's range consists of two subranges : the dissipation subrange and inertial subrange for scales larger than dissipation range and smaller than energy containing scales. Obukhov [1941] and Batchelor [1950] established power laws for mean-square separation for small and intermediate times. Later, Batchelor [1952] extended Richardson's relative dispersion concepts to three dimensions. Much of the recent work focuses on direct numerical simulation of turbulent flows. Boffetta and Sokolov [2002] investigated relative dispersion in DNS of two-dimensional inverse cascade turbulence and demonstrated good agreement of results with Richardsons description of diffusion. Salazar and Collins [2009] have provided a good review of two particle dispersion in turbulent flows.

2.3 Lagrangian Coherent Structures

Another technique which has emerged recently and has been applied successfully to study fluid advection is Lagrangian coherent structures, which addresses material surfaces characterized by their attracting or repelling nature. This method was first developed by Haller [2000]. This approach is Lagrangian in the sense that it uses particle trajectories to determine underlying flow features, which are more efficient than using only Eulerian criteria. Also these are termed *coherent* because they have distinguished stability compared to other

surfaces in the flow. Since these surfaces are often the most attracting/repelling locally in the domain, they have strong influence on how transport occurs. The discussion here will be regarding two dimensional flows, although it can be extended to n dimensions (Lekien 2007).

The fluid advection can by represented by the equation:

$$\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}, t) \tag{2.8}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the Eulerian velocity field and $\mathbf{x}(t)$ is the trajectory of fluid particle, starting at position $\mathbf{x}_0$. In applications, the trajectory is usually obtained by numerically integrating equation (2.7) using, for example Adams-Bashforth or Runge-Kutta method. In dynamical systems theory, hyperbolic fixed points and their corresponding stable and unstable manifolds are known to be organizing structures. Thus, the set of hyperbolic fixed points and their stable and unstable manifolds behave as separatrices which partition the flow into regions with similar dynamics. A separatrix is a trajectory in the phase plane that acts as a boundary between regions of distinct characteristics. Thus, computing these separatrices is key to understanding transport in dynamical systems.

2.3.1 Finite-Time Lyapunov Exponents

Lyapunov exponents have long been used to determine sensitivity of a dynamical system to initial conditions. If two points are initially separated by a small distance $|\zeta_0|$ at time t_0, then the separation at some later time is given by

$$|\zeta_t| \approx e^{\sigma(t-t_0)}|\zeta_0|, \tag{2.9}$$

where σ denotes the finite-time Lyapunov exponent (FTLE). The precise separation between particles at any time t, which are initially separated by ζ_0 is given by

$$\zeta_t = F_{t_0}^t(\mathbf{x}_0 + \zeta_0) - F_{t_0}^t(\mathbf{x}_0). \tag{2.10}$$

where $F_{t_0}^t$ denotes the flow map. Integrating equation(2.7) yields

$$F_{t_0}^t : \mathbf{x}_0 \rightarrow \mathbf{x}\ (\mathbf{x}_0, t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{u}(\mathbf{x}(\tau), \tau)\ d\tau, \tag{2.11}$$

The flow map ($F_{t_0}^t$) maps the particles from initial position ($\mathbf{x}_0(t_0)$) to final position $\mathbf{x}$ at time t. By Taylor expanding the function about ζ_0, the distance becomes

$$|\zeta_t| = |\ \nabla F_{t_0}^t(\mathbf{x}_0) \cdot\ \zeta_0|\ +\ O(|\zeta_0|^2), \tag{2.12}$$

which can be written as

$$|\zeta_t| = \sqrt{\mathbf{e}^T \cdot\ \nabla F_{t_0}^t(\mathbf{x}_0)^T .\ \nabla F_{t_0}^t(\mathbf{x}_0) \cdot \mathbf{e}}\ |\zeta_0| +\ +\ O(|\zeta_0|^2). \tag{2.13}$$

where $\mathbf{e} = \zeta_0/|\zeta_0|$ and T denotes transpose. The coefficient of expansion for small $|\zeta_0|$ then becomes

$$\lim_{|\zeta_0|\rightarrow 0} \frac{|\zeta_t|}{|\zeta_0|} = \sqrt{\mathbf{e}^T .\ \nabla F_{t_0}^t(\mathbf{x}_0)^T \cdot\ \nabla F_{t_0}^t(\mathbf{x}_0) \cdot \mathbf{e}}\ |\zeta_0|. \tag{2.14}$$

The Cauchy-Green strain tensor is defined as

$$C(\mathbf{x}_0, t_0, t) =\ \nabla F_{t_0}^t(\mathbf{x}_0)^T .\ \nabla F_{t_0}^t(\mathbf{x}_0), \tag{2.15}$$

where $\lambda^i(\mathbf{x}_0, t_0, t)$ denotes the i^{th} eigenvalue and $\mathbf{e}^i(\mathbf{x}_0, t_0, t)$ the associated eigenvectors. If the largest eigenvalue of C is greater than 0, the associated eigenvector points in the direction of greatest stretching. Thus, eigenvalues of C are a measure of rate of stretching. FTLE are defined as the natural logarithm of the coefficients of expansion and, dividing by the length of integration time gives average expansion rate

$$\sigma^i(\mathbf{x}_0, t_0, t) = \frac{1}{|t - t_0|} \log \sqrt{\lambda^i(\zeta_0, t_0, t)}. \tag{2.16}$$

The FTLE defined in this manner is a measure of how much the trajectories separate in the flow over the time interval (t_0, t). Shadden et al. [2005] formalized the definition of Lagrangian coherent structure (LCS) as ridges of the FTLE field. Interestingly, this definition of FTLE is valid for both forward and backward integration in time. For positive values of T,

the FTLE measures expansion in forward time and yields repelling surfaces. For backward integration, FTLE measures expansion backward in time and provides surfaces which are attracting in forward time.

Since these computations are intensive and often time consuming, we explored the applicability of high performance parallel computing using GPUs in this work. A high level introduction of GPU terminology is provided followed by its implementation in flow solver and mixing diagnostics.

2.4 GPU Computing

While GPUs have been extensively used traditionally for graphics rendering, their parallel architecture can be used for general purpose computing and achieve massive speedup. The main idea behind general purpose GPU computing is to break a large problem into smaller tasks and execute them simultaneously. **CUDA** (Compute Unified Device Architecture) is a parallel computing platform and Application Programming Interface (API) provided by NVIDIA allows developers to use NVIDIA GPUs for general purpose computing. The hardware aspect of CUDA involves graphics cards and the software aspect consists of the CUDA Toolkit provided by NVIDIA. The advantage with general purpose GPU computing over serial computing is that we have thousands of processors at our disposal to do the computing task. Since shifting from serial to parallel computing requires changes in hardware, we will discuss commonly used GPU terminology.

2.4.1 Terminology and Architecture

As mentioned above, the biggest advantage with a GPU is the availability of thousands of computing units. Since the speedup occurs by utilizing these processing units for subtasks, knowledge of GPU architecture is required. The main architecture difference between a CPU and a GPU is that the CPU has fewer computing cores (processors) and a large cache and control unit to help those cores run fast. Meanwhile, a GPU is designed such that more cores are dedicated to data processing rather than cache and control. Thus, GPUs become

well suited for computations which are computationally intensive.

A GPU consists of numerous computing units called cores and each core consists of Arithmetic Logical Units (shown as green boxes in figure 2.2) and smaller caches and control units than a CPU. Cores are grouped together to form streaming multiprocessors (SMs). The parallelism is achieved by spawning threads. A thread is a task or one independent path of execution in the code. Threads are organized into blocks and blocks are divided into warps whose size is equal to the number of cores in SMs. Threads, blocks and memory are the main architectural components of the GPU. CUDA uses Single Instruction Multiple

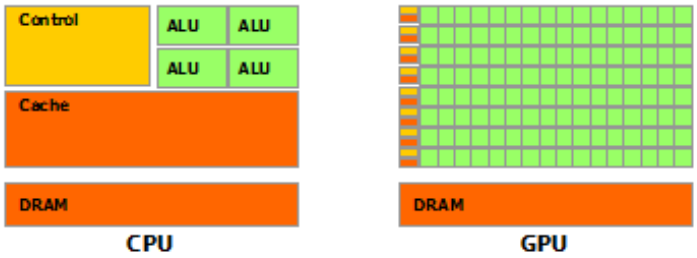


Figure 2.2: Architecture difference between CPU and GPU (adapted from CUDA Programming Guide, NVIDIA Corp).

Thread (SIMT) method of parallelization, which means that same instruction is processed by threads simultaneously. However, the task performed by a single thread is not redundant since each thread performs the computation for a different index value. This approach is scalable because massive parallelism can be achieved by adding more SMs to perform computations. The time spent by a core waiting for data is termed as latency and CPUs are designed to reduce latency by increasing their clock cycle and having data storage units (cache) located closer to the cores. GPUs are designed to hide latency by increasing the computing throughput rather than speed of individual core.

2.4.2 Software Aspect

To launch threads for performing computations, a special kind of function called a **kernel** is used. A kernel creates a computational grid composed of blocks of threads. CUDA provides each thread with a built-in index variable which is used by the kernel to distribute subtasks. These index variables are analogous to loop indices in a serial code. Kernels are called on the host (CPU) and executed on the device (GPU). A kernel is called by specifying the name of the function and the dimensions of the computational grid. A typical kernel call looks like

```
kernelName <<< DimGrid, DimBlock >>> (arguments)
```

where `DimGrid` specifies the number of blocks in the grid and `DimBlock`, the number of threads in each block. These together specify the dimensions of the kernel launch. Since kernels are called from the device, type qualifiers have to be used to differentiate them from function calls. CUDA provides the following function type qualifiers:

- `__global__` to specify kernels (called from the host, executed on the device
- `__host__` for functions called from the host and executed on the host (default qualifier)
- `__device__` for functions called from the device and executed on the device

Some other properties of kernels are:

- They cannot return a value. Hence, the return type is always void. The kernel declaration is as follows:
  `__global__ void kernelName(arguments)`
- Dimension and Index variables provided by CUDA :
  - **Grid Dimension variables**
    `gridDim` (number of blocks in the grid) and `blockDim` (number of threads in each block)

  - **Index variables**:
    `blockIdx` (index of block in the grid) and `threadIdx` (index of thread in the block)

- They execute on the device and do not have access to data stored on host. As a result, data has to copied from the host to the device.

- Since the computations are parallel, the order of execution is random. However, CUDA provides with functions to synchronize (`__syncthreads()`, `cudaDeviceSynchronize()`) and coordinate execution

Since the kernel cannot access data residing on host, mirror arrays have to be created on the device. `cudaMemcpy()` is used to perform this operation. This process of copying data from the host to device is time consuming. Thus to use CUDA productively, one aim should be to minimize the number of copy operations between the device and the host. Also, the kernel should be launched in such a manner that it does significant amount of work to utilize the perks of massive parallelism against the cost of data transfer. Although there is an alternative of unified memory which allows for the array to be accessible from both host and device, it has particular system requirements and is not always feasible.

2.4.3 Memory Hierarchy and Stencil Computations

Now that we are familiar with the basic hardware and software aspect of CUDA, we would like to focus on the using it for our problem. But first, we look at the types of memory available on the device. An understanding of the memory hierarchy allows us to efficiently use CUDA in applications such as stencil computations. By stencil, we mean computations in which operation at each point on computational grid requires information about neighboring grid points. A kernel involving stencil computations would have threads that are dependent on neighboring threads for data. Since threads are reading /writing from the same device array, this competition for data access can introduce bottlenecks. Another roadblock is that

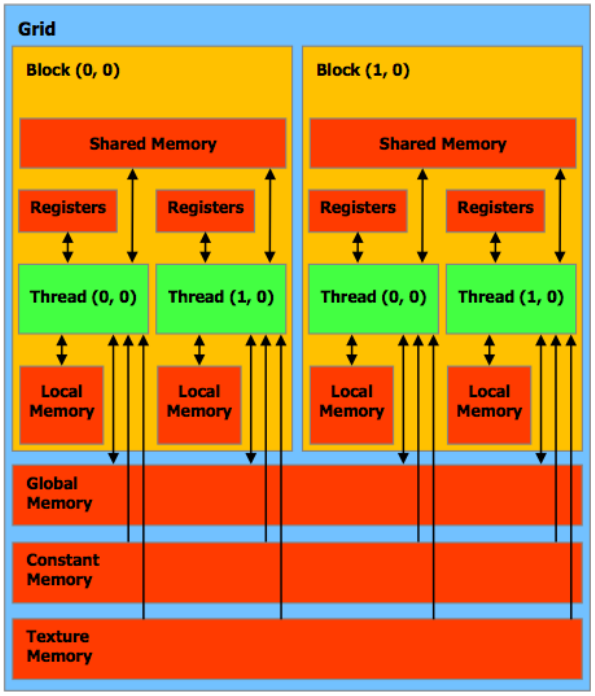


Figure 2.3: Typical GPU architecture and Memory Hierarchy(adapted from CUDA Programming Guide, NVIDIA).

in SIMT parallelism, each thread has access only to its own version of kernel variable. This is overcome by defining device arrays in global memory. Global memory provides most of the storage capacity but is located far from the SMs. Register is located closest to SMs but it can only be accessed by the a single thread. Shared memory resolves this divide between memory access and speed. Each block has shared memory located adjacent to the SM and is accessible to all threads in that block. This is of particular interest in stencil computations. For more details, refer to Storti and Yurtoglu [2015]. In the following chapters, we use this knowledge of applied GPU computing using CUDA to solve the Navier-Stokes on a two dimensional grid for a shear layer and use the simulation data for LCS and dispersion computations. We have used both serial and parallel approaches to set up the flow solver. A comparison of execution time using these approaches is made in chapter 3.

Chapter 3

MIXING LAYER

In this chapter, we describe the numerical method used to integrate the Navier-Stokes equation for a mixing layer. Further, we discuss the serial and parallel implementation of the flow solver and results from the simulations.

3.1 Numerical Method for Flow Solver

The flow solver uses the Fourier-spectral method for solving Navier-Stokes equation for an incompressible, constant density flow in two dimensions. A traditional approach for two-dimensional flows is to use the streamfunction-vorticity form of the Navier-Stokes equations. However, we refrain from using this formulation because it cannot be easily extended to three dimensions. An incompressible flow of a Newtonian fluid satisfying Navier-Stokes equations obeys:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u}, \tag{3.1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{3.2}$$

Here, $Re = Ul/\nu$ is the Reynolds number, $U$is the characteristic velocity, l is a characteristic length scale, and ν is the kinematic viscosity. In the equations, U is used to non-dimensionalize all velocities, l for length scales, and l/U for time scale. It is assumed that the flow is periodic in both spatial directions (x, y). The convective terms in equation (3.1) can be written as:

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\omega \times \mathbf{u} + \nabla \frac{|\mathbf{u}|^2}{2}, \tag{3.3}$$

where $\omega = \nabla \times \mathbf{u}$ is the vorticity vector. With this identity, equation (3.1) can be written as

$$\frac{\partial \mathbf{u}}{\partial t} - \omega \times \mathbf{u} = -\nabla P + \frac{1}{Re}\nabla^2 \mathbf{u}. \tag{3.4}$$

with $P = p + |\mathbf{u}^2|/2$ is the dynamic pressure. Equation (3.4) has the advantage that it conserves kinetic energy exactly in the numerical scheme used for time stepping. Defining

$$\mathbf{B} = \omega \times \mathbf{u} + \frac{1}{Re}\nabla^2 \mathbf{u}, \tag{3.5}$$

and integrating Equation (3.4) from time t to time $t + \Delta$t gives:

$$\int_t^{t+\Delta t} \frac{\partial \mathbf{u}}{\partial t} dt' = \mathbf{u}^{n+1} - \mathbf{u}^n = -\Delta t \ \nabla \tilde{P} + \int_t^{t+\Delta t} \mathbf{B} \ dt', \tag{3.6}$$

where

$$-\nabla \int_t^{t+\Delta t} P dt' = -\Delta t \ \nabla \left[\frac{1}{\Delta t} \int_t^{t+\Delta t} P dt' \right] = -\Delta t \ \nabla \tilde{P}. \tag{3.7}$$

where $\tilde{P}$ is the dynamic pressure averaged over the time period $(t, t + \Delta t)$. Now we proceed to numerically integrate equation (3.6) using a second order Adams-Bashforth scheme.

3.1.1 Adams-Bashforth Method for Time Stepping

To evaluate the time integral in equation (3.6), the last term $\mathbf{B}$ is expanded using Taylor series about time t, assuming B is known at the previous time steps n and *n-1*, i.e.,

$$\mathbf{B}(t') = \mathbf{A}_1 + \mathbf{A}_2(t' - t). \tag{3.8}$$

where $\mathbf{A}_1$ and $\mathbf{A}_2$ are constants that need to be determined. Evaluating this series at time t_n where $\mathbf{B}(t_n) = \mathbf{B}^n$ and $\mathbf{B}(t - \Delta t(n-1)) = \mathbf{B}^{n-1}$, then

$$\mathbf{B} = \mathbf{B}^n - \frac{1}{\Delta t}(\mathbf{B}^{n-1} - \mathbf{B}^n)(t' - t). \tag{3.9}$$

Substituting this back into the integral gives

$$\int_t^{t+\Delta t} \mathbf{B} dt' = \Delta t \left(\frac{3}{2}\mathbf{B}^n - \frac{1}{2}\mathbf{B}^{n-1} \right). \tag{3.10}$$

Using this result in equation (3.6) gives:

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \left[-\nabla \tilde{P} + \frac{3}{2}\mathbf{B}^n - \frac{1}{2}\mathbf{B}^{n-1} \right]. \tag{3.11}$$

This form of equation can be used for time stepping assuming $\mathbf{B}^n$ and $\mathbf{B}^{n-1}$ are known. However, $\nabla \tilde{P}$ is unknown and is obtained by taking divergence of equation (3.11). This gives:

$$\nabla^2 \tilde{P} = \frac{3}{2}\nabla \cdot (\omega \times \mathbf{u})^n - \frac{1}{2}\nabla \cdot (\omega \times \mathbf{u})^{n-1}. \tag{3.12}$$

The numerical scheme proceeds as follows:

1. It is assumed that $\mathbf{u}^n$, $(\omega \times \mathbf{u})^n$ and $(\omega \times \mathbf{u})^{n-1}$ are known at time step n. Note that $\omega = \nabla \times \mathbf{u}$ and can be calculated using derivative properties of Fourier transform. For a two dimensional flow with velocity $\mathbf{u} = u(x,y)\hat{i} + v(x,y)\hat{j}$, the vorticity (ω) is given by:

   $$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{3.13}$$

   The derivatives of velocity components can be calculated by transforming to spectral domain by using Fourier transforms. Recall that in spectral domain,

   $$\frac{\partial \hat{u}}{\partial x} = ik_x\hat{u}, \quad \frac{\partial \hat{u}}{\partial y} = ik_y\hat{u}. \tag{3.14}$$

   where $\hat{u}$ is the Fourier transform of u and k_x, k_y are the wavenumbers in spectral domain. These results are used to calculate ω in equation (3.13). Once ω is known, $(\omega \times \mathbf{u})^n$ and $(\omega \times \mathbf{u})^{n-1}$ can be calculated similarly by using derivative properties of Fourier transforms.

2. Poisson's equation for pressure (equation 3.12) is solved next by using results from the previous step and Fourier transforming both sides of the equation into wavenumber domain. the resulting equation can then be solved for Fourier transform of $\tilde{P}$ and later transformed back into physical space.

3. The above results can then be used in equation (3.6) to time step and solve for $\mathbf{u}^{n+1}$.

We now proceed to outline the implementation of the above numerical method using a traditional serial approach and a parallel implementation using CUDA.

3.2 Implementation using serial approach

Here we describe the flow solver developed for the purpose of numerically integrating the Navier-Stokes equations and generating velocity field data. This solver is written in both Matlab and C. The C version extensively uses the FFTW library for computing Fast Fourier Transforms (FFTs). C was preferred language of choice because of its portability to CUDA and flexibility in terms of memory management for higher grid sizes when compared to an interpreted language such as Matlab. The solver consists of the following functions and subroutines:

`Initialize()` : This function is used to initialize the grid points and initial velocity profile using data from Michalke [1964], as described in chapter 2. An option is present to include the harmonic and sub-harmonic disturbances to the hyperbolic tangent shear layer (equation (2.5).

`navstk_serial()`: This subroutine time steps the velocity field in physical space using equation 3.11. It uses two numerical methods : 1) **Forward Euler** for the first time step, 2) **Adams Bashforth** for all successive timesteps. It may be noted that it is necessary to use Euler method initially because Adams Bashforth requires information from two previous time steps. Both the time stepping routines utilize the following major functions:

1. `grad_laplacian()` : This function takes velocity arrays (u, v) as input and calculates the gradient $(\nabla u, \nabla v)$ and the laplacian $(\nabla^2 u, \nabla^2 v)$. It uses the FFTW library to compute discrete FFTs and makes use of their derivative properties as described earlier. For this task, three main subfunctions are used:

- `real_to_complex()`: Convert real data (velocity, pressure) to complex fftw specific arrays in Fourier space. The imaginary components of these arrays are zero.
- `grad&lap()`: Perform the Fourier transform. Use this transformed data to calculate the gradient and the Laplacian of input array in the spectral domain.
- `complex_to_real()`: Inverse FFT the complex gradient and Laplacian arrays to physical domain

2. `omega_and_A()` : Uses velocity gradients from the previous step to calculate the vorticity (ω) and $\mathbf{A}$ ($\omega \times \mathbf{u}$). These variables are used for solving the Poisson's equation for pressure, which is the next step in numerical scheme.

3. `poisson_AB()/poisson_euler()`: Solves the Poisson's equation by first converting the input real arrays into complex arrays (imaginary component being zero), transforming into spectral domain using FFT, and solving for the dynamic pressure in spectral domain. This is further used to calculate the pressure gradient by using the derivative property. Lastly, the inverse Fourier transform is performed to obtain the pressure gradient in physical space. This function uses the same subfunctions as `grad_laplacian()`.

4. `adamsbash()/euler()`: Time steps the velocity field using data from previous steps.

At the end of each time step, the data can be saved to the disk which is further processed for computing dispersion and LCS. The conversion of this serial approach to a parallel CUDA code requires some modifications. These are elaborated in the next section.

3.3 Parallel Implementation using CUDA

The skeleton of the CUDA code is very similar to the serial approach. However, since the hardware entities (CPU and GPU) are different, the implementation of subroutines requires modifications. Some of the modifications in parallel code are:

- The computational grid is defined using index and dimension variables. This is done as follows:
  `#define TPB 32` defines 32 threads per block
  `dim3 dimgrid (N/TPB, N/TPB)` declares a two dimensional computational grid and `dim3 dimblock (TPB,TPB)` declares size of each block

- Mirror arrays are created on the device and memory locations are copied to the device. For example:
  `cudaMemcpy(u_device, u_host, sizeof(double)*N*N, cudaMemcpyHosttoDevice);`
  is used make a copy of the velocity array (`u_host`) of type `double` and dimensions `N X N` on the device (`u_device`)

- Functions are replaced by kernels. The function call is changed to kernel call and type classifier is added to the function declaration. Hence,
  `void real_to_complex()` becomes `__global__void real_to_complex()`
  and the function call
  `real_to_complex(arguments)` is replaced by kernel call `real_to_complex <<< dimgrid, dimblock >>> (arguments)`

- Loop indices are replaced with index and dimension variables available in CUDA API. Specifically the nested loops in serial code:
  `for(int i=0; i < N; i++)`
  `for(int j=0; j < N; j++)`
  are replaced by
  `int i = threadIdx.x + blockIdx.x*blockDim.x;`
  `int j = threadIdx.y + blockIdx.y*blockDim.y;`
  where `threadIdx`, `blockIdx` are the index variables and `blockDim` is used to define the dimensions of the computational grid.

- At the end of each computation, the device array is copied to the host and saved to the disk. This is achieved using:
  `cudaMemcpy(u_host, u_device, sizeof(double)*N*N, cudaMemcpyDevicetoHost);`

- In place of FFTW, CUDA's own GPU accelerated library cuFFT is used to perform discrete FFT computations on the device.

The workflow in the parallel code remains the same as the serial approach. Velocity field is initialized using `vel_initialize()` after which `navstkCUDA()` is called for time stepping. It utilizes the modified versions of previous functions and subroutines for calculating the velocity gradients, solving for Poisson's equation and lastly time stepping using Euler/Adams Bashforth method. Like the serial approach, data files are saved to disk for post processing. We now proceed to discuss the results from our simulations.

3.4 Results

Mixing layer with perturbations

An unperturbed shear layer would diffuse out under the action of viscosity. However, the profile rolls up in the presence of external perturbations. These perturbations can either be in the form of forced disturbances or noise. Here, we look at the case of forced perturbation on an initial hyperbolic tangent velocity profile. The sinusoidal disturbance that we use has dimensional wavenumber ($k = 4$) and its subharmonic ($k = 2$). The initial velocity can then be defined using equation (2.1) and (2.5) as:

$$u_{initial} = u + u_1,\ v_{intial} = v_1. \tag{3.15}$$

Since the most unstable mode corresponds to $k = 4$, we expect four initial vortices which roll up into two vortices (due to the presence of subharmonic, $k = 2$). These vortical structures are largely responsible for mixing of the two fluids and entrainment. As the shear layer

rolls up, large amount of neighboring fluid is stretched and folded, and drawn into vortex cores. The simulations are performed on a [0, 2π] x [-2π, 2π] domain with $\Delta t = 0.002$ and periodic boundary conditions. The solver makes use of free slip boundary condition in y direction using the symmetry of the initial flow field. Thus, it is set up in a manner such that $u(-y) = u(y)$ and $v(-y) = -v(y)$. The velocity data is stored at every 10^{th} time step. As can be seen figure 3.1, the shear layer rolls up under the action of imposed perturbations

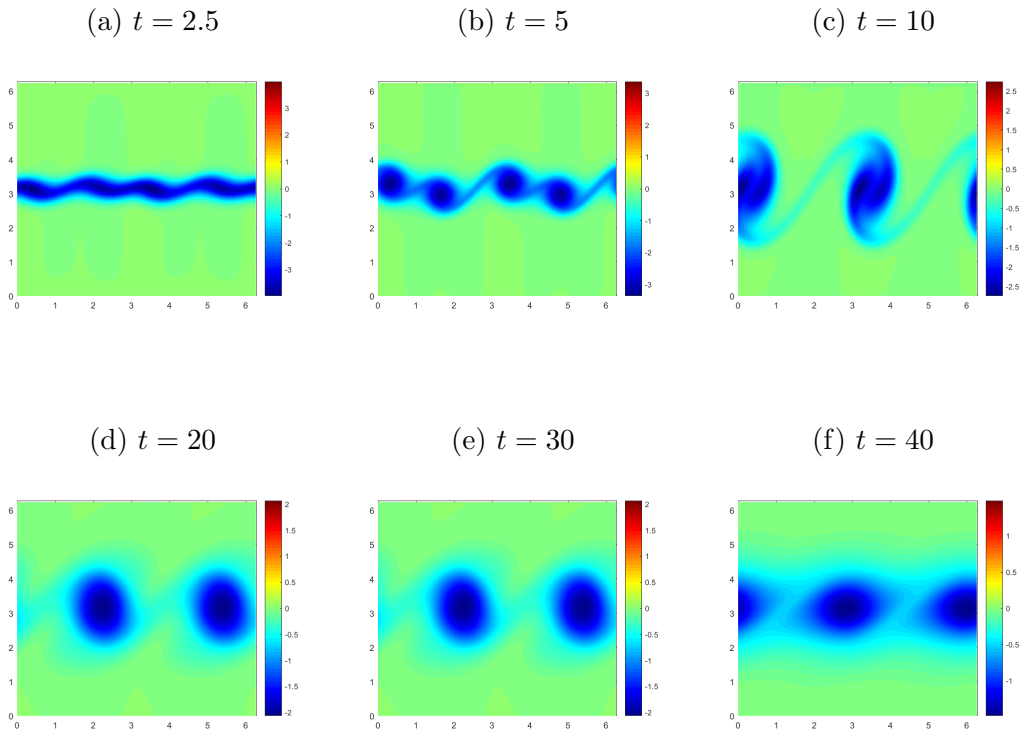


Figure 3.1: Vorticity at different times during the roll up of a mixing layer with sinusoidal perturbations.

Mixing layer with noise

In this case, we add noise to an unperturbed shear layer. Due to the presence of background noise, the shear layer rolls up although the manner of roll up is different from the perturbed noise-free case. The simulations are performed using the same numerical parameters (grid size, Δt) as the previous case but the perturbations are replaced by background noise whose energy spectrum is defined as

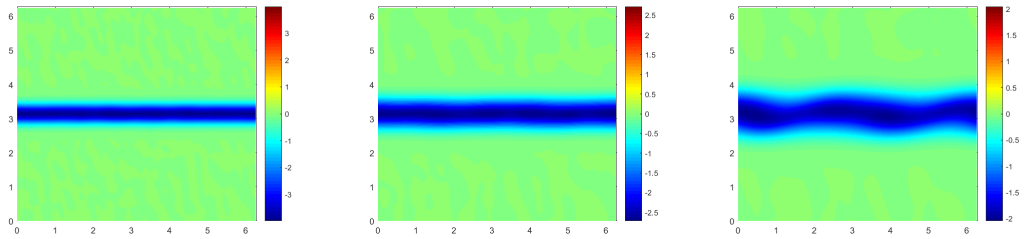
$$E(k) = \tan(k)\ e^{-\eta k}. \tag{3.16}$$

where $k = 1, 2, 3, 4...N$ is the wavenumber and $\eta = 0.05$. The energy function is defined such that the peak of the curve is at the sub-harmonic ($k = 2$). Due to the presence of a large spectrum of wavenumbers, the shear layer rolls up unevenly till there are two large vortices (corresponding to $k = 2$) in the domain. Subsequently, they pair up to form one large vortex.

3.4.1 Benchmarking

Here we compare the CPU and GPU implementation for different 2D grid sizes (N^2). The results are reported using an NVIDIA GeForce GTX670 unit and a Tesla K unit. For GPU implementation, only registers and global memory were utilized. We perform time stepping on the GPU and copy back data to the device which is then stored on disk. The serial code was run on an Intel Core i5-3450 3.1 GHz processor. It can be seen from figure 3.3 that the runtime on GPU is significantly lower for higher grids. This can be attributed to the number of computations that needs to performed serially vs their simultaneous computation on GPU. Also, the cuFFT library is optimized such that maximum speedup occurs when N = 1024 (NVIDIA Corporation). It can be seen that a speed up of roughly 6 x can be obtained using GPUs. However, the speedup is computed using execution time (in seconds) whereas a better measure is to compare the throughput (operations/sec).

(a) $t = 2.5$ (b) $t = 5$ (c) $t = 10$



(d) $t = 20$ (e) $t = 30$ (f) $t = 40$

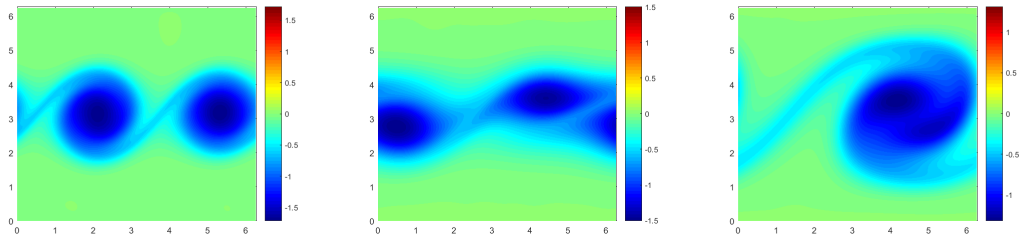


Figure 3.2: Vorticity at different times during the roll up of a mixing layer with noise.

3.4.2 Mixture fraction

The flow solver can be also be used to determine the evolution of a scalar field with the flow. In our case, we have determined mixing of two streams (fuel and air). Assume a chemical reaction initially with the fuel in the upper stream and air in the lower stream. The mixture fraction (Z) is same as the initial condition used for the velocity field:

$$Z(x, y) = \frac{1}{2}[1 + \tanh(\frac{y}{\delta})]. \tag{3.17}$$

Here $Z = 0$ and Z=1 correspond to pure oxidant and pure fuel, respectively. At any particular time, Z gives the amount of intermixing of the fuel and air. The time evolution of

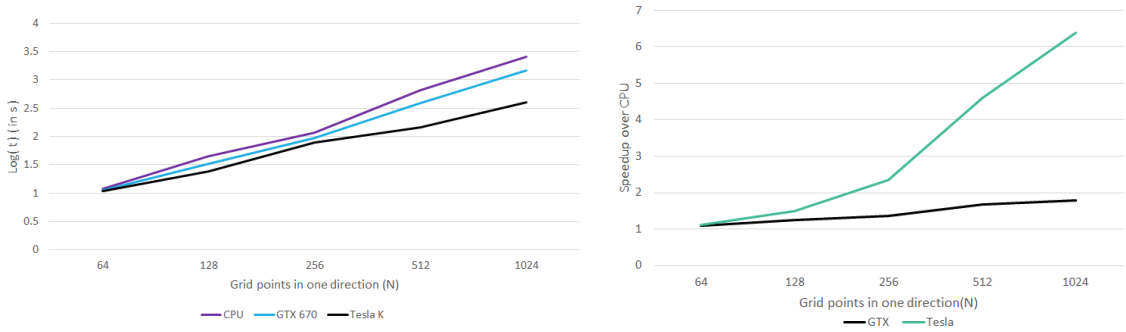


Figure 3.3: Runtime comparison ($log(t)$) between serial and parallel implementation of flow solver for different grid sizes (N^2) for 1000 time steps. The figure on the right shows the corresponding speedup

scalar field with diffusivity (D) can be obtained by numerically integrating:

$$\frac{\partial Z}{\partial t} + (\mathbf{u} \cdot \nabla)Z = D\nabla^2 Z, \tag{3.18}$$

where D is the diffusivity of the species. We assume that both molecular species (fuel and oxidant) have the same diffusivity. Note that this can equation can solved simultaneously while solving for velocity. As the flow develops, more product is generated and the mass fraction of product in the field can be determined, assuming infinte reaction rate as:

$$Y_p = \begin{cases} 2Z, & 0 \leq Z \leq 0.5. \\ 2(1-Z), & 0.5 < Z \leq 1. \end{cases} \tag{3.19}$$

Here Y_p is the mass fraction of product in the field at any time. Time evolution of product field can be calculated using equation (3.18) and (3.19). The amount of product, and hence mixing at the molecular level in the computational domain can be obtained by summation of mass fraction (Y_p) over the entire grid. As we can see in figure 3.4 and 3.5, that contours in product mass fraction plots resemble the structures in vorticity plots very closely. Initial

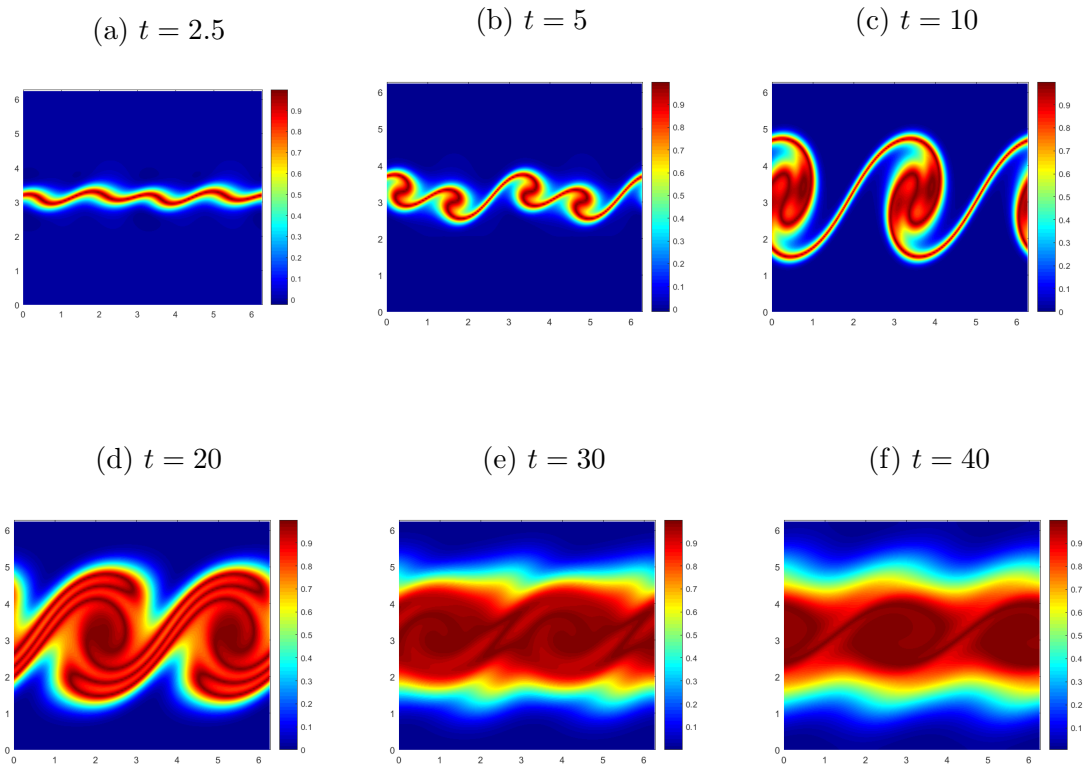
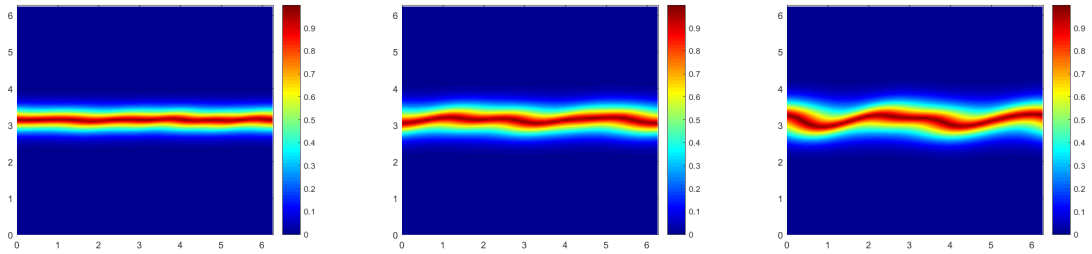


Figure 3.4: Mass fraction of product (Y_p) at different times (t) in the flow for a shear layer with sinusoidal perturbation

product formation is dominated by shear layer roll up while diffusion is predominantly responsible for later times. Now that an analytic measure of mixing is available, we shift our focus to other two mixing measures.

(a) $t = 2.5$ (b) $t = 5$ (c) $t = 10$



(d) $t = 20$ (e) $t = 30$ (f) $t = 40$

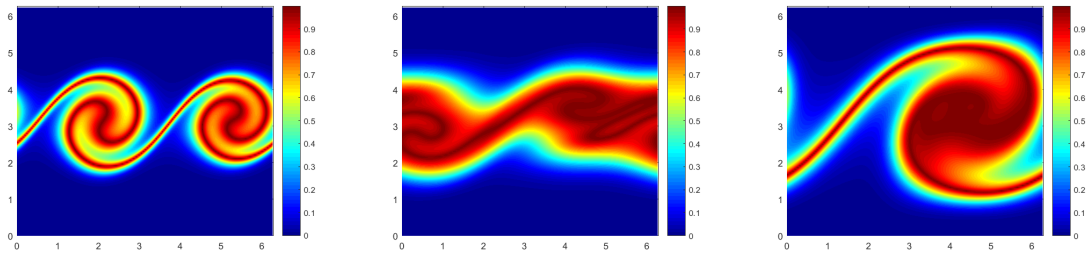


Figure 3.5: Mass fraction of product (Y_p) at different times (t) in the flow for a shear layer with noise.

Chapter 4

FINITE TIME LYAPUNOV EXPONENTS AND DISPERSION

Using velocity data from the solver, particles are advected forward and backward in time and the resulting displacements are used to compute FTLE (σ) and relative dispersion (R^2). We obtain the time series of their evolution by seeding particles at various times and advecting them for a fixed integration time (T = 15) using the Adams-Bashforth method.

4.1 FTLE: Computation and Implementation

The algorithm for FTLE computation is discussed in section 2.3.1. A grid of particles is initialized on the domain and advected with the flow. The velocity field describing the flow is usually available, either in the form of analytic model, or from numerically solving Navier-Stokes equations. The flow map is computed by integrating the particles forward or backward in time. We have used the Adams-Bashforth scheme for this purpose. Once the flow map is obtained, its differentiation with respect to initial separation yields the flow map Jacobian ($\nabla F_{t0}^{t}(\mathbf{x}_0)$) which is then used to compute the Cauchy-Green strain tensor ($C = \nabla F_{t0}^{t}(\mathbf{x}_0)^T \cdot \nabla F_{t0}^{t}(\mathbf{x}_0)$. Eigenvalues of C yields FTLE at each point in the flow domain. LCS then appears as ridges in this FTLE field. This procedure is repeated by seeding particles at different times and advected to obtain time evolution of FTLE. It may be noted that high value of T tends to sharpen the ridges and clutter the field with too many intricate structures. Thus, choosing the optimum value of T is important to balance the information provided by their plots. The major subroutines used in FTLE calculations are:

- `Flowmap()` is used to calculate the flow map. The subroutine initializes the particle grid in the domain, advects them in forward/backward direction and returns the particle position arrays at the end of time stepping. For velocity interpolation in C,

interpolation package in `alglib` library is used for bilinear/bicubic interpolation. For Matlab implementation, the built in function `interp2` is used to perform bicubic interpolations.

- `FTLE()` is used to calculate the flow map jacobian using finite difference. We use central difference for interior particles and forward/backward difference for particles on the boundary. For each point, this constitutes a 2×2 matrix. The Cauchy-Green tensor (C) is calculated next and eigenvalues are computed. For implementation in C, `eigen` library is used for this task while Matlab has an inbuilt function called `eig()`. Time averaged logarithm of the square root of eigenvalues gives the FTLE.

4.2 Results: Finite Time Lyapunov Exponents and LCS

Using the implementation described above, we obtain the FTLE in forward and backward time for the perturbed mixing layer. For small integration time (T = 5), the structures are poorly defined whereas for longer time integration (T = 30), the domain gets cluttered with too many structures. Hence, we use an integration time of T = 15 to obtain time evolution of FTLE field in both forward and backward time.

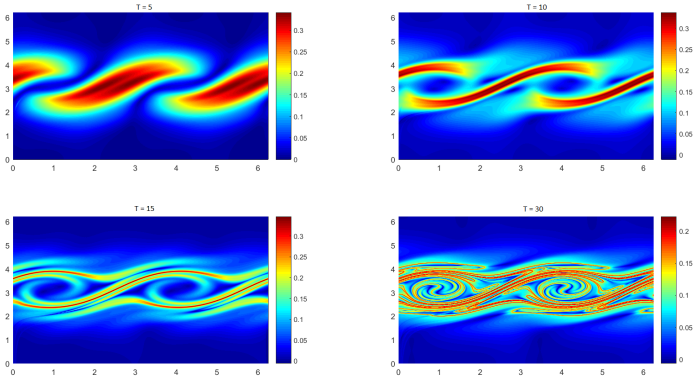
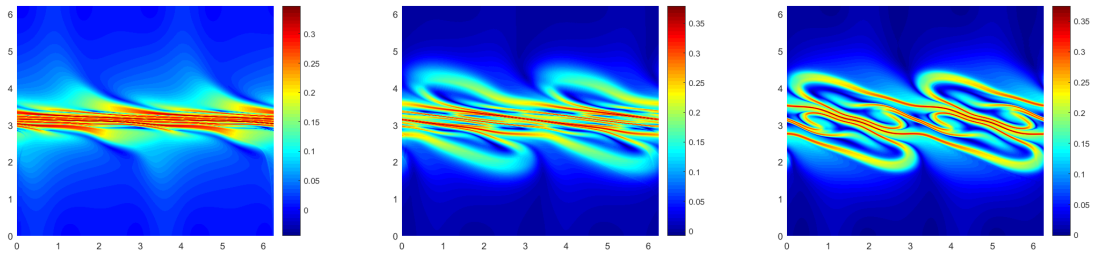


Figure 4.1: Backward FTLE calculations from particle advection in the mixing layer using different integration times (T = $5, 10, 15, 30$) from $t_0 = 0$ to $t = 5, 10, 15, 30$ in the flow.

Forward LCS

To obtain Lagrangian Coherent Structures in forward time, we seed an initial grid particles (512^2) at $t = 0$ and integrate particles forward with time step $\Delta t = 0.02$. The FTLE computation at the end of each time step is performed as described in section 4.1. The time series is obtained by re-initializing the particle grid at various times ($t_0 = 0.5, 1, 1.5...45$) and integrating for a time period, T = 15. FTLEs calculated in forward time in this manner (figure 4.2) provide the time evolution of most repelling structures in the flow.

(a) $t_0 = 1$ (b) $t_0 = 2.5$ (c) $t_0 = 5$



(d) $t_0 = 8$ (e) $t_0 = 10$ (f) $t_0 = 17.5$

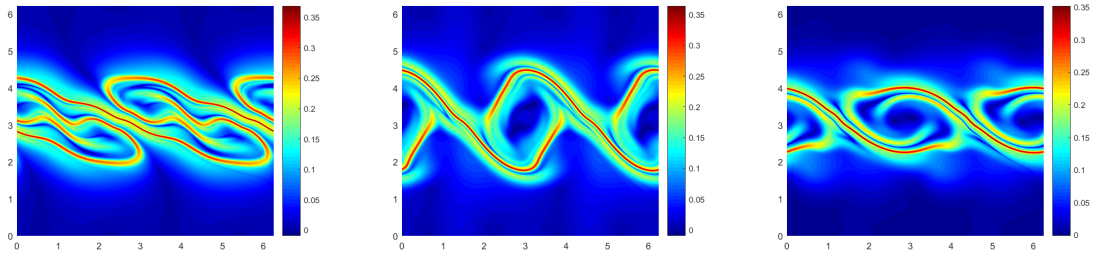


Figure 4.2: Time evolution of Forward FTLE in the mixing layer obtained by advecting particles from $t_0 = 1, 2.5, 5, 8, 10, 17.5$ to $t = 16, 18.5, 20, 23, 25, 32.5$ using T = 15. These structures are the most repelling regions in the flow.

Backward LCS

Backward LCS is obtained in a manner similar to the computation of forward time LCS. In this case, we integrate particles back in time starting from $t = 0$. The time series is obtained by integrating from times $t_0 = 0, 1, 2, 3, 4..45$ back in time for time period (T= 15). For $t < 0$, initial velocity is used to advect particles. FTLE calculated in backward time (figure 4.3) gives the most attracting structures in the flow in forward time. It is interesting

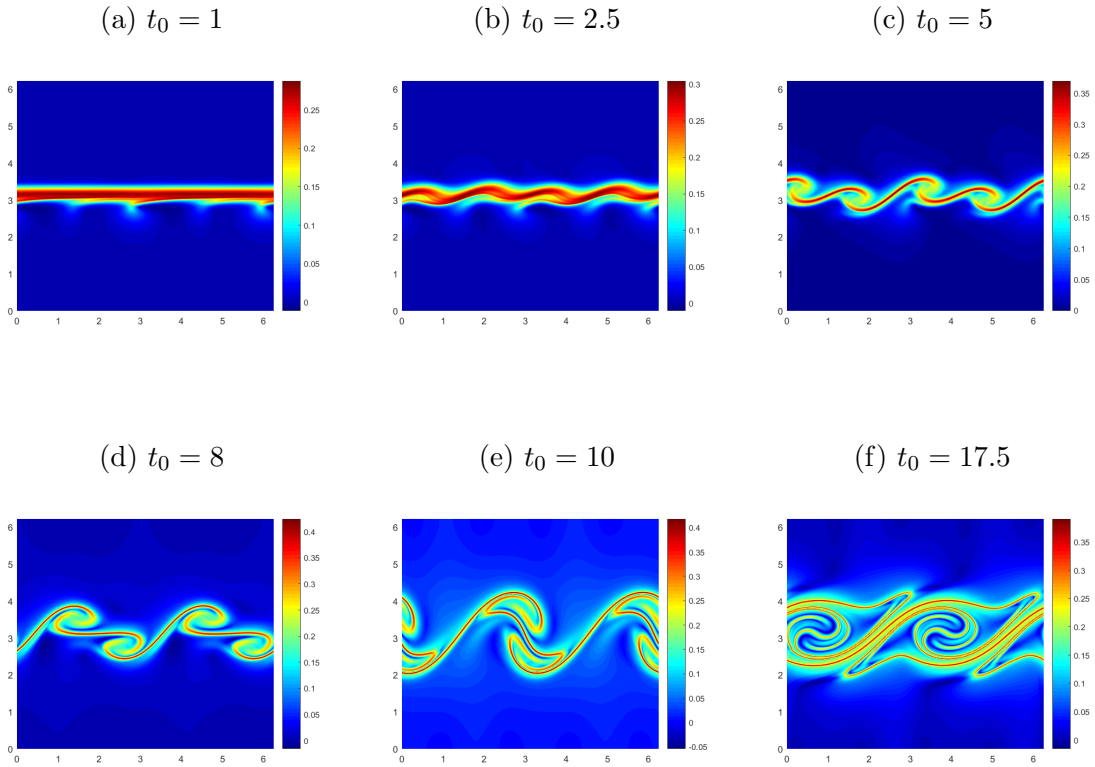


Figure 4.3: Time evolution of Backward FTLE in the mixing layer obtained by advecting particles backward from $t_0 = 1, 2.5, 5, 8, 10, 17.5$ to $t = -14, -12.5, -10, -7, -5, 2.5$ using T = 15. These structures are the most attracting regions in the flow.

to note that contour plots for backward LCS closely resemble the vorticity and mixing

fraction plots. It can be seen that the most attracting features in the flow also correspond to the regions where most mixing occurs (figure 3.4). Thus, the mixing regions in the flow are correctly identified by backward LCS which is consistent with existing literature on Lagrangian coherent structures. As can be seen in figure 4.4, forward LCS (black) envelops the backward LCS (red) and separates the mixed regions from the ambient fluid streams. The points of intersection of these two provide the saddle points in the flow. The fluid is drawn along the attracting structures and is fed to the vortex cores where more product is formed. Once the roll up finishes, the structures begin to vanish from the domain.

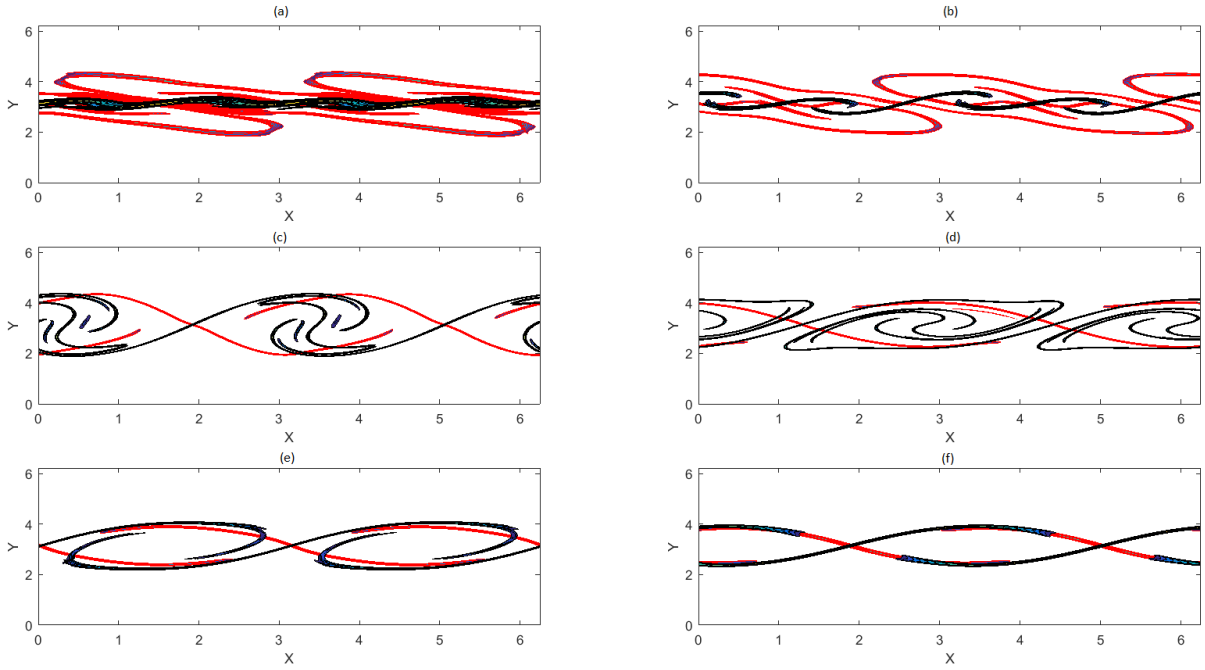


Figure 4.4: Forward FTLE (red) and Backward FTLE (black) in the mixing layer revealing the most repelling and attracting structures corresponding to times: a) $t_0 = 2.5$, (b) $t_0 = 5$, (c) $t_0 = 11$, (d) $t_0 = 15$, (e) $t_0 = 20$, (f) $t_0 = 30$. These were obtained using T = 15.

4.3 Relative Dispersion: Computation and Implementation

To compute relative dispersion, we measure the separation between neighboring particles in the grid. The relative dispersion (R^2) calculations are performed simultaneously with forward/backward FTLE computations. A grid of particles is defined over the velocity domain and advected forward/backward in time by using a numerical integration scheme (Euler/Adams-Bashforth). The final position of particles are used to calculate the average squared distance of each particle with its four neighbors. Particle position, $\mathbf{r}(t)$ at $(n+1)^{th}$ time step is given by:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{u}_n dt \tag{4.1}$$

Here, $\mathbf{r}_{n+1}(t) = (x_{n+1}, y_{n+1})$ is the position of the particle at $(n+1)^{th}$ time step. Using final position of particles, relative dispersion is calculated as

$$D_2 = \frac{1}{4} \sum_{over 4 neighboring points} ||\mathbf{r}(t) - \mathbf{r}_i(t)||^2 \tag{4.2}$$

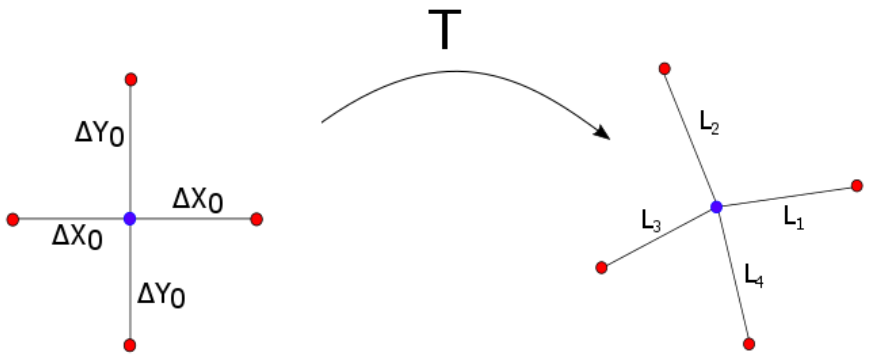


Figure 4.5: Relative dispersion of a particle with respect to its neighbors after time T

Here $\mathbf{r}(t) = (x, y)$ represents the position of particle at time t, $\mathbf{r}_i(t)$ is the position of the

i^{th} neighbor. Using figure 4.5, the relative dispersion of a particle (blue dot) can be defined as:

$$R^2 = \frac{(L_1^2 + L_2^2 + L_3^2 + L_4^2)}{4} \tag{4.3}$$

In our implementation, dispersion computations are performed simultaneously while computing the flowmap for FTLE calculations and the subroutine `flowmap()` is reused with slight modifications to compute distance of each particle with its four neighbors.

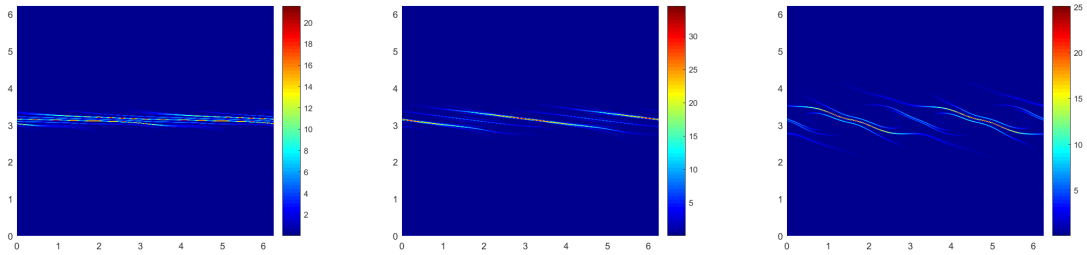
Forward Relative Dispersion

In a manner similar to forward LCS a grid of particles (512^2) is advected with the flow using Adams-Bashforth numerical scheme with time step, $\Delta t = 0.02$ for T = 15. At the end of integration, the particle distances from their neighbors are calculated and stored in an array. Contour plots (figure 4.5) are then used to visualize the points which have separated the most in the flow. It is interesting to note that R^2 plots produce the same ridges as LCS plots (figure 4.2), although the structures in the domain are not as detailed and intricate. Maximum separation occurs for particles which are near the ridge and are repelled exponentially in the flow.

Backward Relative Dispersion

Backward dispersion gives the separation of particles in backward time. Like LCS, backward dispersion (figure 4.6) is maximum along the FTLE ridges. Hence, we may hypothesize that both quantities provide qualitatively the same information about the attracting/repelling regions in the flow which dominate the mixing process.

(a) $t_0 = 1$ (b) $t_0 = 2.5$ (c) $t_0 = 5$



(d) $t_0 = 8$ (e) $t_0 = 10$ (f) $t_0 = 17.5$

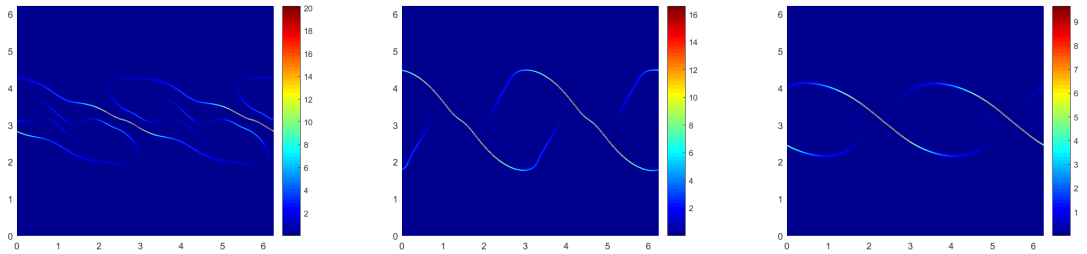
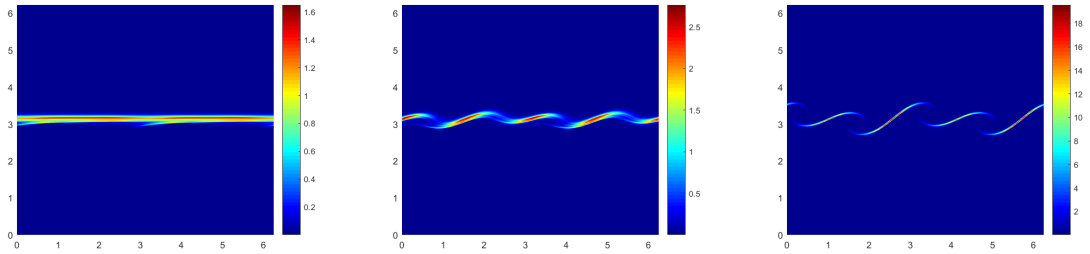


Figure 4.6: Time evolution of forward relative dispersion (R^2) in the mixing layer obtained by integrating from $t_0 = 1, 2.5, 5, 8, 10, 17.5$ to $t = 16, 18.5, 20, 23, 25, 32.5$ using T = 15.

(a) $t_0 = 1$ (b) $t_0 = 2.5$ (c) $t_0 = 5$



(d) $t_0 = 8$ (e) $t_0 = 10$ (f) $t_0 = 17.5$

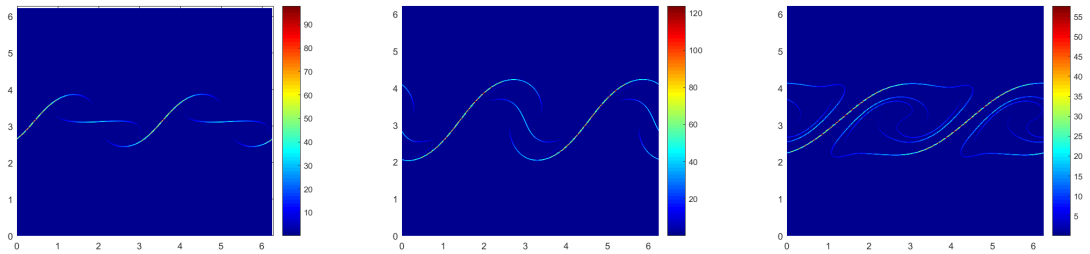


Figure 4.7: Time evolution of backward relative dispersion(R^2) in the flow obtained by integrating backward from $t_0 = 1, 2.5, 5, 8, 10, 17.5$ to $t = -14, -12.5, -10, -7, -5, 2.5$ using T = 15.

Chapter 5

COMPARISON OF FTLE AND RELATIVE DISPERSION

In the previous chapter, we saw that both quantities successfully identify the *coherent* structures in the flow which are responsible for mixing and hence we now proceed to study the relationship between the two. Any particle near an FTLE ridge undergoes maximum stretching as a result of which the R^2 contour plots have peaks at FTLE ridges. While R^2 depends on spreading of particle pairs and LCS measures maximum separation rate, the information provided by both of them is qualitatively similar. To further understand their relationship, we employ backward time LCS and R^2 for a mixing layer with sinusoidal perturbations and vary the length of particle advection time ($\mathrm{T} = 5, 15, 30$). In short times, the particle separation is approximately exponential whereas for longer time, a more statistical outlook needs to be taken while comparing the two quantities.

This idea of the apparent connection between the two quantities is not new. Von Hardenberg et al. [2000] noticed similarities between the two while studying chaotic mixing during a baroclinic life cycle. They noticed that regions of strong dispersion were associated with regions of high shear and strong advection. Haller and Yuan [2000] made similar observations while comparing another measure of LCS, called hyperbolicity time, with relative dispersion in two dimensional isotropic turbulence. In their results, they noticed that relative dispersion produces fuzzy curves while LCS boundaries are sharp. The key difference is that the methodology used by them to compute LCS differs from the one used in this work. LaCasce et al. [2005] used absolute dispersion, relative dispersion, and FTLE to characterize chaotic mixing in a coastal tidal model. Their results also indicate strong correlation between relative dispersion and FTLE as both techniques revealed similar structures. Ad-

ditionally, the growth rate revealed by relative dispersion for first tidal cycle is comparable to maximum FTLE values from which it can be inferred that maximum relative dispersion reflects regions of maximum strain. These observations are consistent with our observations and hypothesis that these two quantities are related. Recently, Waugh et al. [2012] used normalized dispersion (λ_d) and established its bound using FTLE (σ). It is useful to express dispersion in terms of λ_d as it brings both quantities on the same non-dimensional scale.

$$\lambda_d = \frac{1}{2T}\log\left(\frac{R^2}{x_0^2}\right) \tag{5.1}$$

Defined in such a way, λ_d would measure the rate of exponential separation of particle pairs

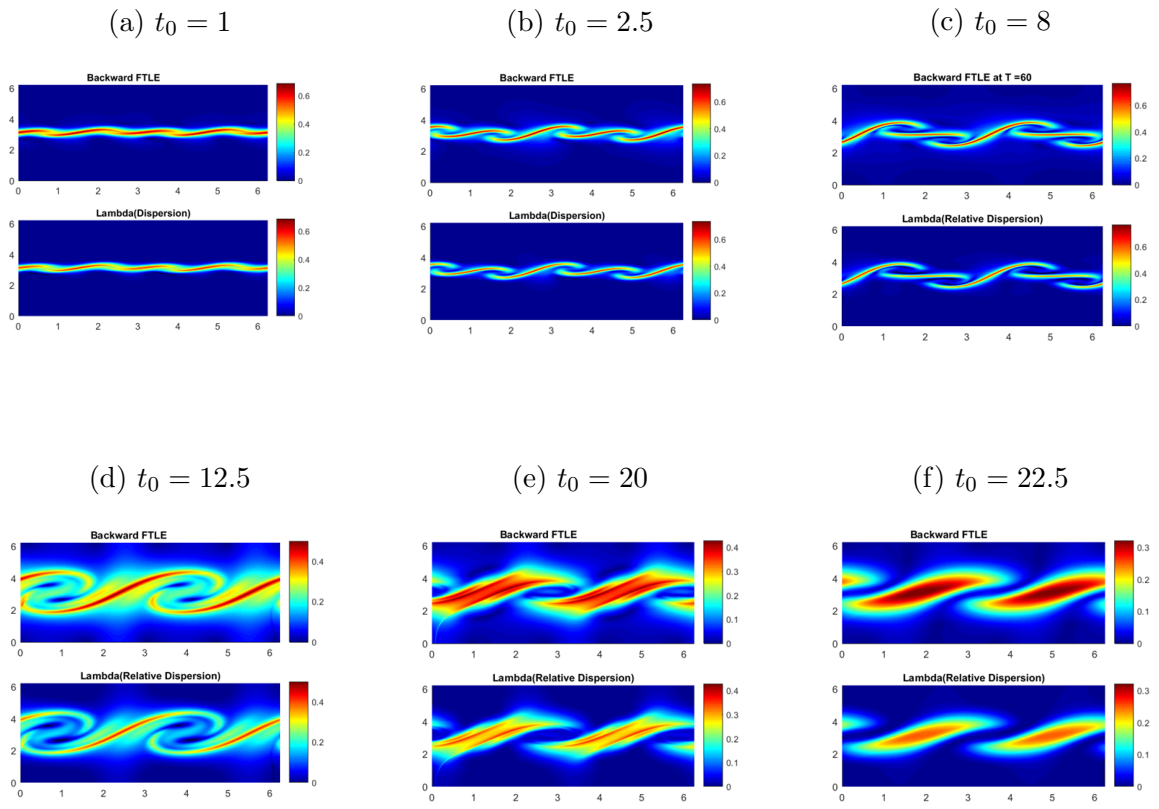


Figure 5.1: Backward FTLE (σ) and dispersion (λ_d) at different times using T = 15.

normalized by initial separation (x_0) for integration time (T).

5.1 Comparison of σ and λ_d for short times

Using equation 5.1 we proceed to calculate λ_d at different times in the flow. For short integration times (T = 5), the particles are still close enough to make an analytic approximation about λ_d and σ. Using the formulation of Waugh et al. [2012] for a uniform strain flow,

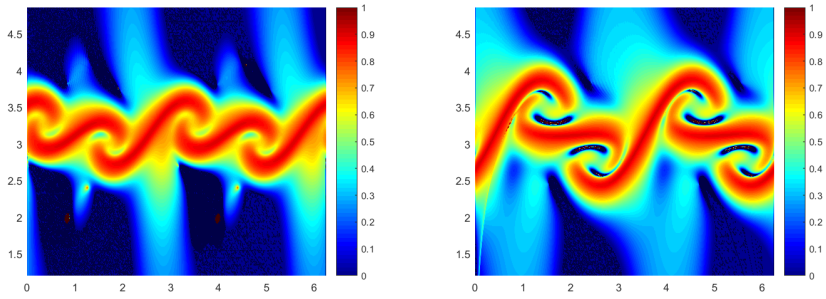
$$\sigma_{min} \leq \lambda_d \leq \sigma_{max} \tag{5.2}$$

where σ_{min} and σ_{max} are the minimum and maximum FTLE values obtained using equation (2.14). It can be seen from the contour plots of σ and λ_d (figure 5.1) that both quantities are indeed very close to each other. We proceed to calculate their ratio, defined as $\Gamma = \lambda_d/\sigma$. As can be seen in figure 5.2, the value of Γ is very close to 1 on the ridge and gradually falls off as we move away. There are some dark regions in the plots which correspond to those regions where particles are attracted and $\frac{R^2}{x0^2} < 1$.

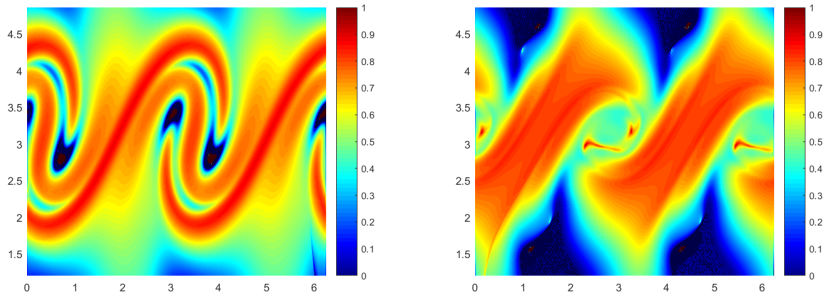
5.2 Effect of integration time

In this case, we proceed to advect particles using different integration times $T = 5, 15, 30$ and determine Γ. This is carried out at three different starting times in the flow ($t_0 = 5, 10, 20$). For each time, we use three integration times (T) and plot Γ. It can be noticed that even as integration time increases, the value of Γ at the ridge is still very close to one. It is in line with the observation of Haller [2001]. He postulated that long term calculation of FTLE is more statistical owing to growing error in particle advection and is strongly correlated with relative dispersion.

(a) $t_0 = 5$ (b) $t_0 = 8$



(c) $t_0 = 10$ (d) $t_0 = 20$



(e) $t_0 = 25$ (f) $t_0 = 30$

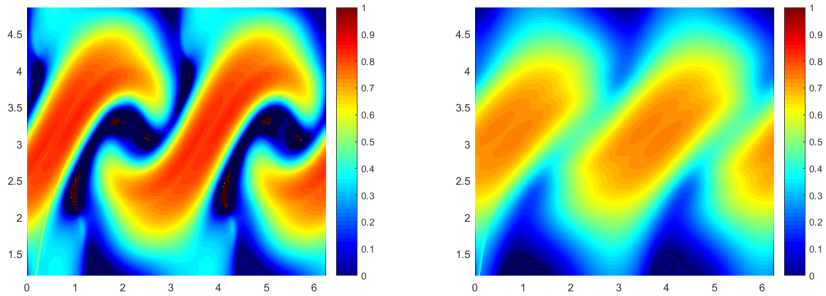


Figure 5.2: $\Gamma = \lambda_d/\sigma$ obtained by integrating backwards in time from $t_0 = 5, 8, 10, 20, 25, 30$ to $t = 0, 3, 5, 15, 20, 25$ using $\mathrm{T} = 5$.

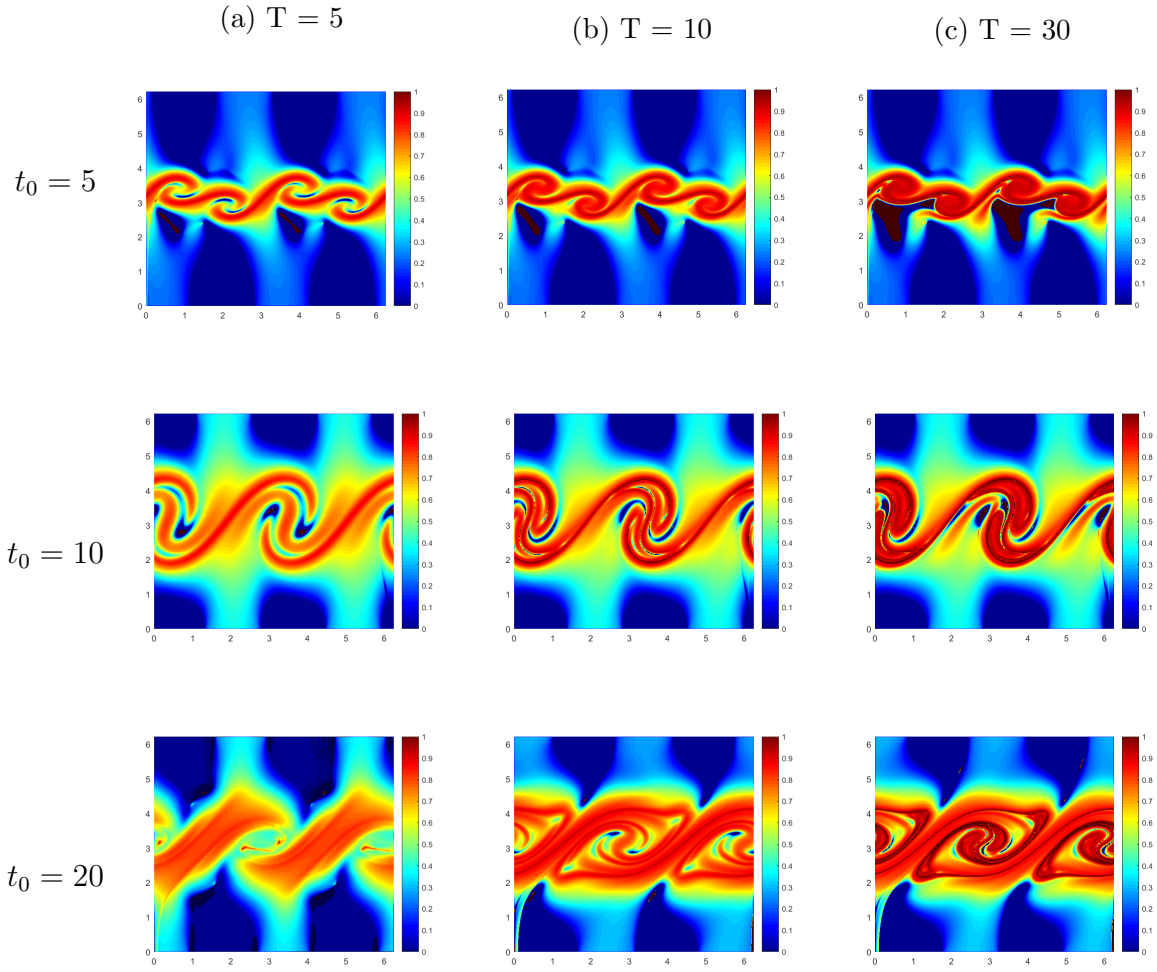


Figure 5.3: $\Gamma = \sigma/\lambda_d$ obtained using different integration times (T $= 5, 10, 30$) for different times in the backward integration ($t_0 = 5$, $t_0 = 10$ and $t_0 = 20$).

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

In this work, we have tried to understand and quantify the relationship between relative dispersion and finite time Lyapunov exponents (FTLE) for a perturbed, temporally-growing, two-dimensional mixing layer. We have developed a CUDA-based 2D flow solver using Fourier-spectral methods to solve the Navier-Stokes equation. The runtimes were calculated for a serial implementation for the same solver and it was found that GPU implementation is approximately 6 times faster.

Using velocity data from the flow solver, we were able to compute finite time Lyapunov exponents (σ) and relative dispersion in forward and backward time. It was found that maximum R^2 values correspond to the ridges in the FTLE field. We further compute normalized relative dispersion (λ_d) and compare it with σ. It was shown that λ_d and σ provide qualitatively the same information, and the structures in their contour plots are very close. Another quantity, Γ, defined as the ratio of λ_d and σ, is computed and it is found that $\Gamma = 1$ at the ridges and close to one in the vicinity of FTLE ridges. The exercise was repeated using different integration times ($\mathrm{T} = 5, 15, 30$) and at different starting points ($t_0 = 5, 10, 20$). Using information from Γ plots, we can conclude that the value of λ_d is bounded by σ although an exhaustive analysis is required at this stage.

6.2 Future Work

The future work as planned but not limited to includes verification of Γ for different diffusion regimes. FTLE calculations are dependent on flow map Jacobian and require a dense

grid of particles for approximation whereas diffusion depends on time and length scales in the flow. Deriving diffusion information using an LCS plot at a given time which, although ambitious, is something we would like to explore. Another task is to mathematically verify the relationship between λ_d and σ for intermediate and longer times using a model other than uniform strain flow. In this work, a serial code was used to perform LCS and dispersion computation. We would like to extend the CUDA based flow solver to include these computations and obtain speedup. Lastly, we would attempt to validate dispersion laws using LCS.

BIBLIOGRAPHY

GK Batchelor. Diffusion in a field of homogeneous turbulence. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 48, pages 345–362. Cambridge Univ Press, 1952.

GK Batchelor, AM Binnie, and OM Phillips. The mean velocity of discrete particles in turbulent flow in a pipe. *Proceedings of the Physical Society. Section B*, 68(12):1095, 1955.

Bruce J Bayly, Steven A Orszag, and Thorwald Herbert. Instability mechanisms in shear-flow transition. *Annual review of fluid mechanics*, 20(1):359–391, 1988.

Francisco J Beron-Vera, María J Olascoaga, Michael G Brown, Huseyin Koçak, and Irina I Rypina. Invariant-tori-like lagrangian coherent structures in geophysical flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):017514, 2010.

Francisco J Beron-Vera, Yan Wang, María J Olascoaga, Gustavo J Goni, and George Haller. Objective detection of oceanic eddies and the agulhas leakage. *Journal of Physical Oceanography*, 43(7):1426–1438, 2013.

G Boffetta and IM Sokolov. Relative dispersion in fully developed turbulence: the richardsons law and intermittency corrections. *Physical review letters*, 88(9):094501, 2002.

Garry L Brown and Anatol Roshko. On density effects and large structure in turbulent mixing layers. *Journal of Fluid Mechanics*, 64(04):775–816, 1974.

Blake M Cardwell and Kamran Mohseni. Vortex shedding over a two-dimensional airfoil: Where the particles come from. *AIAA journal*, 46(3):545–547, 2008.

CT Crowe, JN Chung, and TR Troutt. Particle mixing in free shear flows. *Progress in energy and combustion science*, 14(3):171–194, 1988.

JW Elder. The dispersion of marked fluid in turbulent shear flow. *Journal of fluid mechanics*, 5(04):544–560, 1959.

Peter Freymuth. On transition in a separated laminar boundary layer. *Journal of Fluid Mechanics*, 25(04):683–704, 1966.

Melissa A Green, Clarence W Rowley, and George Haller. Detection of lagrangian coherent structures in three-dimensional turbulence. *Journal of Fluid Mechanics*, 572(1):111–120, 2007.

George Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.

George Haller. Lagrangian coherent structures from approximate velocity data. *Physics of Fluids (1994-present)*, 14(6):1851–1861, 2002.

George Haller and FJ Beron-Vera. Coherent lagrangian vortices: The black holes of turbulence. *Journal of Fluid Mechanics*, 731:R4, 2013.

George Haller and Francisco J Beron-Vera. Geodesic theory of transport barriers in two-dimensional flows. *Physica D: Nonlinear Phenomena*, 241(20):1680–1702, 2012.

George Haller and Guocheng Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3):352–370, 2000.

Andrei N Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. In *Dokl. Akad. Nauk SSSR*, volume 30, pages 301–305. JSTOR, 1941.

JH LaCasce. Statistics from lagrangian observations. *Progress in Oceanography*, 77(1):1–29, 2008.

Francois Lekien, Shawn C Shadden, and Jerrold E Marsden. Lagrangian coherent structures in n-dimensional systems. *Journal of Mathematical Physics*, 48(6):065404, 2007.

Igor Mezić, S Loire, Vladimir A Fonoberov, and P Hogan. A new mixing diagnostic and gulf oil spill movement. *Science*, 330(6003):486–489, 2010.

A MICHALKE and R WILLE. Flow processes in the laminar-turbulent transition region of freestream boundary layers(laminar turbulent transition in freestream boundary layer behind axisymmetric and plane nozzle). 1966.

Alfons Michalke. On the inviscid instability of the hyperbolictangent velocity profile. *Journal of Fluid Mechanics*, 19(04):543–556, 1964.

NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.

AM Obukhov. On the distribution of energy in the spectrum of turbulent flow. In *Dokl. Akad. Nauk SSSR*, volume 32, pages 22–24, 1941.

María J Olascoaga and George Haller. Forecasting sudden changes in environmental pollution patterns. *Proceedings of the National Academy of Sciences*, 109(13):4738–4743, 2012.

Steinar Orre, Bjørn Gjevik, and Joseph H LaCasce. Characterizing chaotic dispersion in a coastal tidal model. *Continental Shelf Research*, 26(12):1360–1374, 2006.

Jifeng Peng and John O Dabiri. An overview of a lagrangian method for analysis of animal wake dynamics. *Journal of Experimental Biology*, 211(2):280–287, 2008.

Lewis F Richardson. Atmospheric diffusion shown on a distance-neighbour graph. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 110(756):709–737, 1926.

Juan PLC Salazar and Lance R Collins. Two-particle dispersion in isotropic turbulent flows. *Annual Review of Fluid Mechanics*, 41:405–432, 2009.

SC Shadden, K Katija, M Rosenfeld, JE Marsden, and JO Dabiri. Transport and stirring induced by vortex formation. *Journal of Fluid Mechanics*, 593:315–331, 2007.

Shawn C Shadden and Charles A Taylor. Characterization of coherent structures in the cardiovascular system. *Annals of biomedical engineering*, 36(7):1152–1162, 2008.

Shawn C Shadden, Francois Lekien, and Jerrold E Marsden. Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3):271–304, 2005.

Duane Storti and Mete Yurtoglu. *CUDA for Engineers: An Introduction to High-performance Parallel Computing*. Addison-Wesley Professional, 2015.

Geoffrey I Taylor. Diffusion by continuous movements. *Proc. London Math. Soc*, 20(1): 196–212, 1922.

J Von Hardenberg, K Fraedrich, F Lunkeit, and A Provenzale. Transient chaotic mixing during a baroclinic life cycle. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 10(1):122–134, 2000.

Darryn W Waugh, Shane R Keating, and Mei-Lin Chen. Diagnosing ocean stirring: comparison of relative dispersion and finite-time lyapunov exponents. *Journal of Physical Oceanography*, 42(7):1173–1185, 2012.