SIMULACIÓN NUMÉRICA DIRECTA DE TURBULENCIA HOMOGÉNEA EN UNA CAJA PERIODICA BIDIMENSIONAL USANDO TRANSFORMADAS DE FOURIER EN PARALELO

CRISTIAN FERNANDO LAGUADO HERNANDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA MECÁNICA
BUCARAMANGA
2013

SIMULACIÓN NUMÉRICA DIRECTA DE TURBULENCIA HOMOGÉNEA EN UNA CAJA PERIODICA BIDIMENSIONAL USANDO TRANSFORMADAS DE FOURIER EN PARALELO

CRISTIAN FERNANDO LAGUADO HERNANDEZ

Trabajo de grado para optar al título de
Ingeniero Mecánico

Director
Ph.D. David Alfredo Fuentes Díaz

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA MECÁNICA
BUCARAMANGA
2013

A mi familia, amigos,
compañeros y profesores.

AGRADECIMIENTOS

Agradecimientos a David Alfredo Fuentes Díaz, Ph.D., al Grupo de Investigación en Energía y Medio Ambiente, GIEMA., a la Unidad de Supercomputacion y Calculo Cientifico, SC3-UIS., a Ing. Monica Liliana Hernandez Ariza., a Jorge Luis Chacón Velasco, Ph.D., a Julian Ernesto Jaramillo Ibarra, Ph.D., a mi Familia y amigos., a David Alonso Barajas Solano. y a Khalay Cristina Chio Oi.

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE ALGORITMOS

pág.

RESUMEN

TÍTULO: SIMULACIÓN NUMÉRICA DIRECTA DE TURBULENCIA HOMOGENEA EN UNA CAJA PERIODICA BIDIMENSIONAL USANDO TRANSFORMADAS DE FOURIER EN PARALELO.*

AUTOR: CRISTIAN FERNANDO LAGUADO HERNANDEZ.†

PALABRAS CLAVE:
Simulación numérica directa, Turbulencia homogénea, Dos dimensiones, Trasformada de Fourier, Computación distribuida.

Se presenta una implementación de cómputo en paralelo con MPI (Message Passing Interface) para la solución numérica directa (DNS) del flujo turbulento homogéneo en una caja bidimensional utilizando condiciones de frontera periódicas mediante un método pseudo-espectral. Se utilizó la transformada de Fourier para la discretización espacial del dominio, controlando el error de solapamiento de frecuencias mediante el método de desplazamiento de fase y encontrando la solución temporal mediante un esquema Runge-Kutta híbrido de tercer orden. En la implementación se utilizaron las funciones para MPI de la librería de alto rendimiento FFTW (Fastest Fourier Transform in the West) para calcular la trasformada discreta de Fourier y su transformada inversa de forma eficiente.

El programa fue ejecutado en la supercomputadora GUANE-1 perteneciente a la plataforma GridUIS-2 de la Unidad de Supercomputación y Cálculo Científico de la Universidad Industrial de Santander, SC3-UIS.

Se realizó la simulación para una malla de 1024x1024 nodos, utilizando un número de Reynolds de 50000, cuyos resultados fueron validados utilizando mediciones globales de energía cinética y enstropía, mostrando un comportamiento físico satisfactorio.

Se evaluaron los efectos del error de solapamiento de frecuencias, mostrando que éste induce comportamientos no físicos en el sistema.

Adicionalmente se ejecutaron pruebas de rendimiento del programa en paralelo, obteniendo resultados de incremento de velocidad y eficiencias de escalamiento consistentes con lo esperado.

* Trabajo de Grado.

† Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería Mecánica. Director David Alfredo Fuentes Díaz, Ph.D.

ABSTRACT

TITLE: DIRECT NUMERICAL SIMULATION OF HOMOGENEOUS TURBULENCE IN A SQUARE PERIODIC BOX USING FOURIER TRANSFORMS IN PARALLEL.*

AUTHOR: CRISTIAN FERNANDO LAGUADO HERNANDEZ.†

KEYWORDS:
Direct numerical simulation, Homogeneous turbulence, two dimensional, Fourier transform, Parallel computing.

An implementation of parallel distributed computing using the message passing interface (MPI) for the direct numerical simulation (DNS) of the turbulent homogeneous flow in a two-dimensional periodic box using a pseudo-spectral method has been it developed. The Fourier transform was used for the spatial discretization of the domain, controlling the aliasing error with the phase-shift method and integrating the solution in time with a hybrid third order Runge-Kutta scheme. The implementation uses the high performance FFTW (Fastest Fourier Transform in the West) MPI functions in order to efficiently compute the discrete Fourier transform and its inverse.

The routine was run in the GUANE-1 computer cluster, belonging to the GridUIS-2 platform in the Industrial University of Santander's Supercomputing and Scientific Calculation Unit, SC3-UIS.

The simulation was run for a 2D grid of 1024x1024 nodes and a Reynolds number of 50000, and its results were validated through global metrics of the kinetic energy and the enstrophy, showing satisfactory and coherent behavior.

The effects of the frequency aliasing error were evaluated, showing that this induces non-physical behaviors into the system.

In addition to this, parallel benchmark tests were executed for the code. It was observed that increase in speed and parallel scaling performance are consistent.

* Undergraduate Degree Work.

† Physical-Mechanical Engineering Faculty. Department of Mechanical Engineering. Director David Alfredo Fuentes Díaz, Ph.D.

INTRODUCCIÓN

El continuo incremento de la capacidad de los sistemas de cómputo hace posible la solución de problemas cada ves mas complejos, al tiempo que permite utilizar métodos de solución que en su momento, a pesar de su clara utilidad, resultaban imposibles de implementar debido a sus altas exigencias computacionales. El cómputo en paralelo, que antes era exclusivo de las más costosas estaciones de trabajo y supercomputadores, en la actualidad hace parte de la cotidianidad. Dispositivos móviles, computadores portátiles y de escritorio cuentan con procesadores de múltiples núcleos que permiten explotar los beneficios de la paralelización de algoritmos. Tener a disposición éstas tecnologías no es suficiente por sí solo, es necesaria la adquisición de los conocimientos necesarios para explotar este recurso al máximo, y los algoritmos numéricos deben ser capaces de utilizar esta capacidad. El uso de técnicas de programación en paralelo como MPI en conjunto con librerías optimizadas ofrecen la posibilidad de crear estrategias útiles para la solución de problemas numéricamente intensivos. Con la adquisición de recursos computacionales como la supercomputadora GUANE-1 la Universidad Industrial de Santander se ubica para atacar problemas numéricos intensivos de manera efectiva.

Una aplicación numéricamente intensiva posible de atacar con recursos modernos es el análisis directo de turbulencia. A pesar de que la turbulencia es un fenómeno tridimensional y no existe de forma bidimensional en la naturaleza, ni es posible crearla como tal en el laboratorio, este problema sirve como punto inicial para diseñar soluciones de turbulencia más complejas, y ha sido de gran utilidad para estudios atmosféricos y oceánicos en donde el grosor de la capa de fluido es muy bajo en comparación a su extensión superficial o el flujo se encuentra estratificado, haciendo que los efectos en el eje vertical sean suficientemente bajos para que los modelos de turbulencia en dos dimensiones sean apropiados para obtener resultados preliminares.

El presente trabajo de grado presenta la implementación y análisis de una solución numérica del problema de turbulencia homogénea en dos dimensiones, usando programación en paralelo con MPI. En el capítulo 1 se obtienen las ecuaciones que rigen el flujo turbulento homogéneo bidimensional; en el capítulo 2 se obtiene la formulación del problema en espacio espectral de Fourier y se explican los métodos utilizados para el cómputo del término no lineal y la integración temporal; en el capítulo 3 se explican los detalles de la implementación en cómputo en paralelo/MPI, del método de solución numérico; en el capítulo 4 se muestran y analizan los resultados obtenidos en la ejecución del programa implementado en GUANE-1.

Este trabajo de investigación es el primer trabajo numéricamente intensivo en mecánica de fluidos en la UIS, y se espera que sirva de plataforma para desarrollos productivos en el futuro.

1. FORMULACIÓN DEL PROBLEMA

El flujo turbulento homogéneo es aquel cuyas propiedades medias no cambian con la posición, es decir, si se tiene un conjunto de puntos en un lugar del dominio de flujo, y éstos puntos son desplazados una cantidad constante igual para todos ellos, las propiedades medias se mantienen invariables en la nueva posición (Mathieu y Scott, 2000). Para lograr ésta condición, se requiere que el dominio de flujo carezca de fronteras físicas que introduzcan alteraciones que violen la homogeneidad.

Turbulencia homogénea en flujo es una condición idealizada, por tanto no se puede observar directamente en la naturaleza y carece de aplicaciones prácticas de ingeniería. Aún así, su estudio teórico es útil ya que sirve como punto de partida para desarrollar la teoría y entender las características de casos más complejos (Batchelor, 1953).

1.1. TURBULENCIA BIDIMENSIONAL

Las ecuaciones de Navier-Stokes para un flujo incompresible de densidad uniforme ρ en el dominio Ω se pueden escribir como

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nabla \cdot \tau + \mathbf{f}, \tag{1.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1.2}$$

donde $\mathbf{u}$ es el vector de velocidad, p es la presión (dividida por ρ), τ es el tensor de esfuerzos, y $\mathbf{f}$ son las fuerzas volumétricas (divididas por ρ). Para el caso de viscosidad uniforme, el término viscoso se reduce a

$$\nabla \cdot \tau = \nu \nabla^2 \mathbf{u}$$

donde $\nu = \mu/\rho$ es la viscosidad cinemática.

La vorticidad de un flujo se denota con $\boldsymbol{\omega}$ y está dada por

$$\boldsymbol{\omega} = \nabla \times \mathbf{u}, \tag{1.3}$$

y representa la velocidad local de rotación del fluido. Para el caso de viscosidad uniforme y fuerzas volumétricas conservativas, de (1.1) se obtiene

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \mathbf{u} \cdot \nabla \boldsymbol{\omega} = \boldsymbol{\omega} \cdot \nabla \mathbf{u} + \nu \nabla^2 \boldsymbol{\omega}, \tag{1.4}$$

$$\nabla \cdot \boldsymbol{\omega} = 0. \tag{1.5}$$

Para un flujo bidimensional, se puede escribir $\mathbf{u} = (u, v, 0)^T$ y $\boldsymbol{\omega} = (0, 0, \omega)^T$, donde

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}. \tag{1.6}$$

En éste caso, el primer término del lado derecho de (1.4) desaparece, obteniendo la expresión simplificada

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \nu \nabla^2 \omega. \tag{1.7}$$

Para simplificar la descripción del flujo bidimensional, se puede introducir la función de corriente ψ definida por las relaciones

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{1.8}$$

Reemplazando éstas relaciones en (1.6) y (1.4), se obtiene el sistema de ecuaciones

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \nu \nabla^2 \omega, \tag{1.9}$$

$$-\nabla^2 \psi = \omega. \tag{1.10}$$

Las ecuaciones (1.9) y (1.10), junto con la definición de la función de corriente ψ, definen de manera completa la vorticidad en un flujo de las características anteriormente indicadas.

1.2. GEOMETRÍA DEL DOMINIO Y CONDICIONES DE FRONTERA

El concepto de turbulencia homogénea implica que el flujo se desarrolla en un dominio de tamaño infinito que como tal no puede ser modelado. La adopción de condiciones de frontera periódicas sobre un dominio finito representa una aproximación del dominio infinito suponiendo que éste último se trata de una repetición infinita de cajas periódicas. Adicionalmente, hacer el espacio periódico permite el uso de la transformada de Fourier, que simplifica la discretización de las ecuaciones de Navier-Stokes y posibilita el uso de algoritmos eficientes como la transformada rápida de Fourier.

El estudio presente se limitará al caso de turbulencia bidimensional en la caja rectangular $\Omega = [0, L] \times [0, L]$ con condiciones de frontera periódicas, esto es, $\omega(x, 0) = \omega(x, L)$ y $\omega(0, y) = \omega(L, y)$ para todo x y y. Con el fin de generalizar el análisis a realizar, se introducen las cantidades adimensionales

$$x^* = x/\lambda, \quad u^* = u/u', \quad \omega^* = \lambda/u'\omega,$$
$$y^* = y/\lambda, \quad v^* = v/u', \quad t^* = u't/\lambda,$$

donde L y u' sirven como escalas de longitud y velocidad, respectivamente. Además, se empleará el factor de escala de longitudes $\lambda = L/2\pi$ para transformar el dominio de flujo a la caja $\Omega = [0, 2\pi) \times [0, 2\pi)$. En términos de estas cantidades adimensionales, y después de eliminar los asteriscos, el sistema de ecuaciones (1.9)–(1.10) se convierte en

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \frac{2\pi}{\mathrm{Re}}\nabla^2\omega, \tag{1.11}$$

$$-\nabla^2\psi = \omega. \tag{1.12}$$

donde $\mathrm{Re} = u'L/\nu$ es el número de Reynolds del flujo.

A lo largo de todo el texto, se hará referencia a cantidades adimensionales sin utilizar asteriscos.

1.3. SOLUCIÓN NUMÉRICA

El problema de valor en la frontera (1.11)–(1.12) no admite en general soluciones analíticas para condiciones iniciales arbitrarias, excepto en casos específicos como los vórtices de Taylor-Green (Taylor y Green, 1937). Es necesario por tanto contar con una estrategia de solución numérica para estudiar el fenómeno de turbulencia de manera rigurosa.

Existen varios métodos para simular el fenómeno de turbulencia numéricamente, siendo las más comunes LES (Large Eddy Simulation), RANS (Reynolds Averaged Navier-Stokes) y DNS (Direct Numerical Simulation) (Pope, 2000; Moin y Mahesh, 1998). LES consiste en obtener una solución al problema de flujo incluyendo directamente sólo vórtices de un cierto tamaño y mayores, e indirectamente escalas más pequeñas mediante modelos matemáticos de sus efectos energéticos en las escalas más grandes. En RANS (Reynolds Averaged Navier-Stokes), las propiedades del flujo se dividen en dos componentes, una media en el tiempo y una fluctuación, y requiere modelos empíricos de las interacciones entre el flujo medio y las fluctuaciones. La exactitud y aplicabilidad de LES y RANS depende fuertemente en la disponibilidad de modelos empíricos de alta calidad para los fenómenos de turbulencia que éstas estrategias no incluyen directamente.

DNS ofrece una alternativa más robusta, en la que se asume que las ecuaciones de Navier-Stokes describen completamente el flujo en todas las escalas temporales y espaciales, por lo que no es necesario el uso de modelos o suposiciones fuera de Navier-Stokes. Siendo un método directo, la exactitud de DNS es teóricamente ideal, pero resolver todas las escalas de espacio y tiempo requiere el uso de una malla espacial muy fina y un avance temporal muy pequeño, lo que acarrea un costo computacional en general prohibitivo. Por lo tanto, DNS no es de uso práctico en

aplicaciones de ingeniería, pero es una poderosa herramienta para investigación de los mecanismos de turbulencia y el desarrollo de modelos para LES y RANS.

En el caso de la solución de turbulencia homogénea mediante DNS, los métodos pseudo-espectrales, que se tratarán en el siguiente capítulo, ofrecen una gran precisión a un menor costo en comparación con otros métodos.

2. SOLUCIÓN NUMÉRICA PSEUDO-ESPECTRAL

Para el caso de turbulencia en un dominio periódico, los métodos pseudo-espectrales son preferidos por su alta precisión y costo computacional relativamente bajo. En éste capítulo se introducirán los componentes del método pseudo-espectral utilizado en éste trabajo.

Los método pseudo-espectrales de Fourier se basan en una representación del campo de vorticidad ω como serie discreta de Fourier de N frecuencias en cada dirección, para un total de N^2 modos $\hat{\omega}$. Dicha discretización se presenta en la sección 2.1.1. La transformada discreta de Fourier (DFT) (sección 2.1.2) permite calcular numéricamente de manera eficiente los N^2 modos en espacio espectral a partir de N^2 valores de ω en espacio real, y viceversa.

Una vez se han reescrito las ecuaciones de Navier-Stokes en forma espectral, es posible obtener un sistema de ecuaciones diferenciales ordinarias (EDOs) para los modos $\hat{\omega}$ (sección 2.2). La discretización del término advectivo de las ecuaciones de Navier-Stokes resulta difícil de calcular directamente en espacio espectral, por lo que se introduce una técnica pseudo-espectral para calcular exactamente éstos términos (sección 2.3). Finalmente, el sistema de EDOs se integra numéricamente utilizando un esquema numéricamente estable y económico (sección 2.4).

2.1. DISCRETIZACIÓN ESPACIAL

2.1.1. Transformada continua de Fourier

Para una función u definida en $\Omega = [0, 2\pi) \times [0, 2\pi)$, se definen los coeficientes de Fourier como

$$\tilde{u}(\mathbf{k}) = \frac{1}{2\pi} \int_{\Omega} u(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}} \, d\mathbf{x}, \tag{2.1}$$

donde $\mathbf{k} = (k_1, k_2)^T$ y $k_1, k_2 = 0, \pm 1, \pm 2, \ldots$. Las componentes k_α del vector $\mathbf{k}$ se conocen como los *modos* de Fourier en cada dirección espacial.

Las funciones $e^{-i\mathbf{k}\cdot\mathbf{x}}$ forman un conjunto ortogonal en Ω, ésto es

$$\frac{1}{2\pi} \int_{\Omega} e^{-i\mathbf{k}_a\cdot\mathbf{x}} e^{i\mathbf{k}_b\cdot\mathbf{x}} \, d\mathbf{x} = \delta_{\mathbf{k}_a,\mathbf{k}_b} = \begin{cases} 1 & \text{si } \mathbf{k}_a = \mathbf{k}_b \\ 0 & \text{si no} \end{cases} . \tag{2.2}$$

La serie de Fourier de u se define como

$$Su(\mathbf{x}) = \sum_{k_1,k_2=-\infty}^{\infty} \tilde{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}},$$

cuya convergencia se verifica para funciones u cuadrado-integrables en Ω, ésto es, funciones cuya norma

$$\|u\|_2 = \left(\int_\Omega |u(\mathbf{x})|^2 \, d\mathbf{x} \right)^{1/2}$$

es finita (Canuto, Hussaini, Quarteroni, y Zang, 2007). Para funciones en éste espacio, se verifica la igualdad

$$u(\mathbf{x}) = \sum_{k_1,k_2=-\infty}^{\infty} \tilde{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}. \tag{2.3}$$

2.1.2. Transformada discreta de Fourier

La serie (2.3), aunque converge, usa una cantidad infinita de modos, por lo que no es de uso práctico para soluciones numéricas. Para aplicaciones numéricas, en lugar de utilizar el conjunto infinito de modos, se usará un subconjunto finito de ellos que se definirán a continuación.

Para tal efecto, se definen primero las coordenadas discretas en espacio real $\mathbf{x_n} = (2\pi/N)\mathbf{n}$, con $\mathbf{n} = (n_1, n_2)^\top$ y $n_1, n_2 = 0, 1, \ldots, N-1$. Éstas coordenadas constituyen una malla discreta cubriendo Ω uniformemente con N divisiones en cada dirección. Se introduce la *transformada discreta de Fourier* (DFT) de una función u, definida en las coordenadas discretas $\mathbf{x_n}$, como

$$\hat{u}(\mathbf{k}) = \frac{1}{N^2} \sum_{0 \le \mathbf{n} < N} u(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x_n}} \tag{2.4}$$

donde $\mathbf{k} = (k_1, k_2)^\top$, $-N/2 \le k_\alpha < N/2$ y $\alpha = 1, 2$. Esta operación es la equivalente discreta a la operación continua (2.1). La DFT esencialmente relaciona una cantidad discreta u definida en el *espacio discreto real* $\mathbf{x_n}$ con otra cantidad discreta $\hat{u}$ definida en el *espacio discreto de Fourier* C_K compuesto por los vectores $\mathbf{k}$ tales que $-K \le k_\alpha < K$ y $N = 2K$.

Adicionalmente se define la *transformada discreta inversa de Fourier* para calcular los valores u en el espacio discreto real a partir de sus coeficientes de Fourier $\hat{u}$ como

$$u^N(\mathbf{x}) = \sum_{\mathbf{k} \in C_K} \hat{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}. \tag{2.5}$$

Como se ve, la serie de Fourier se ha truncado y solo cubre el subconjunto de modos $\mathbf{k} \in C_K$. El costo de ésta discretización es que la igualdad $u^N = u$ sólo se verifica en los nodos $\mathbf{x_n}$; para cualquier otro valor de $\mathbf{x}$, u^N es una aproximación de u construida a partir del subconjunto C_K de modos $\hat{u}$.

2.1.3. Diferenciación

La expansión de u en una serie infinita (2.3) y su forma discreta (2.5) permiten calcular derivadas espaciales de u en términos de los coeficientes de Fourier de u. En concreto, tomando derivadas parciales de (2.5), se obtienen las relaciones

$$\frac{\partial u^N}{\partial x_j} = \sum_{\mathbf{k}\in C_K} ik_j \hat{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}, \quad \frac{\partial^2 u^N}{\partial x_j^2} = -\sum_{\mathbf{k}\in C_K} k_j^2 \hat{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}. \tag{2.6}$$

La segunda relación permite reescribir el operador Laplaciano como

$$\nabla^2 u^N = -\sum_{\mathbf{k}\in C_K} |\mathbf{k}|^2 \hat{u}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}.$$

2.2. FORMULACIÓN DÉBIL

La formulación (1.11)-(1.12) se conoce como *formulación clásica* o *fuerte* del problema porque su solución debe satisfacer estrictos requisitos de regularidad; en particular, se conoce que para condiciones iniciales suficientemente regulares, la solución de (1.11)-(1.12) es suave (es decir, infinitamente derivable, o $\omega \in C^\infty$) (Ladyzhenskaya, 1969). Si se está interesado en obtener una aproximación a la solución del problema, resulta mas práctico resolver un problema diferente, con requerimientos mas débiles, cuya solución converja en el límite a la solución clásica.

Se utilizará un procedimiento de tipo Fourier-Galerkin (Canuto y cols., 2007) para obtener un forma espectral débil del problema (1.11)-(1.12). Considérese la ecuación (1.12), que puede reescribirse como

$$\nabla^2 \psi + \omega = 0.$$

En lugar de buscar dos funciones ψ y ω tales que satisfagan ésta relación estrictamente, se buscará calcular dos series de Fourier ψ^N y ω^N tales que satisfagan una relación mas débil. Denomínese por S_N el espacio de todas las funciones que se pueden escribir de la forma (2.5). Por definición, se tiene que ψ^N y ω^N hacen parte de S_N. Multiplicando la relación anterior por una función $v \in S_N$ e integrando sobre Ω, se obtiene

$$\int_\Omega (\nabla^2 \psi^N + \omega^N) v \, d\mathbf{x} = 0.$$

Si se toma v igual a cualquiera de las funciones base $e^{-i\mathbf{k}\cdot\mathbf{x}}$, obtenemos la siguiente forma débil de (1.12): Encuéntrense $\psi^N, \omega^N \in S_N$ tales que

$$\int_\Omega (\nabla^2 \psi^N + \omega^N) e^{-i\mathbf{k}\cdot\mathbf{x}} \, d\mathbf{x} = 0 \qquad \text{para todo } \mathbf{k} \in C_K.$$

En virtud de la propiedad de ortogonalidad (2.2), ésta condición es equivalente a

$$-|\mathbf{k}|^2\hat{\psi}(\mathbf{k},t)+\hat{\omega}(\mathbf{k},t)=0 \qquad \text{para todo } \mathbf{k}\in C_K; \tag{2.7}$$

ésto es, la ecuación (1.12) se reemplazó efectivamente por una condición en espacio de Fourier.

Procediendo de manera similar para (1.11), se obtiene la formulación débil

$$\int_\Omega \left(\frac{\partial\omega^N}{\partial t}+\frac{\partial\psi^N}{\partial y}\frac{\partial\omega^N}{\partial x}-\frac{\partial\psi^N}{\partial x}\frac{\partial\omega^N}{\partial y}-\frac{2\pi}{Re}\nabla^2\omega^N\right)e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}=0 \qquad \text{para todo } \mathbf{k}\in C_K.$$

Utilizando otra vez (2.2), se obtiene la relación en espacio de Fourier

$$\frac{d\hat{\omega}(\mathbf{k},t)}{dt}+\hat{h}(\mathbf{k},t)+\frac{2\pi}{Re}|\mathbf{k}|^2\hat{\omega}(\mathbf{k},t)=0 \qquad \text{para todo } \mathbf{k}\in C_K. \tag{2.8}$$

Las ecuaciones (2.7)-(2.8) gobiernan la evolución temporal de los coeficientes de Fourier de la vorticidad. Su ventaja radica en que se trata de un conjunto de ecuaciones diferenciales ordinarias en función exclusivamente del tiempo; un sistema de tal tipo puede integrarse fácilmente. En la ecuación (2.8), $\hat{h}(\mathbf{k},t)$ es el $\mathbf{k}$-ésimo coeficiente discreto de Fourier de

$$h^N=\frac{\partial\psi^N}{\partial y}\frac{\partial\omega^N}{\partial x}-\frac{\partial\psi^N}{\partial x}\frac{\partial\omega^N}{\partial y} \tag{2.9}$$

Como se trata de los coeficientes de Fourier de un producto, su cálculo a partir de los coeficientes de ψ^N y ω^N es más elaborado y se discutirá en la siguiente sección.

2.3. ELIMINACIÓN DEL SOLAPAMIENTO DE FRECUENCIAS

Si se tienen los valores de $\hat{\omega}$ en todo punto de la malla para un instante de tiempo determinado, los valores de $\hat{\psi}$ pueden ser calculados a partir de la relación (2.7), realizando esta operación en cada punto de la malla en espacio de Fourier, es decir, no se requiere la solución simultanea de un sistema de ecuaciones.

En la ecuación (2.8) se deben calcular los valores de $\hat{h}$. Para calcular dichos valores en espacio de Fourier, se debería realizar mediante el cálculo de dos convoluciones de la forma que se describirá a continuación.

Sean $u(\mathbf{x})$ y $v(\mathbf{x})$ dos funciones en espacio real, definidas en el dominio Ω que tienen sus correspondientes transformadas de Fourier $\hat{u}(\mathbf{m})$ $\hat{v}(\mathbf{n})$, con $m,n\in C_K$. Y sea

$$s(\mathbf{x})=u(\mathbf{x})v(\mathbf{x}). \tag{2.10}$$

Los coeficientes de Fourier de la función $s(\mathbf{x})$ se calculan mediante la relación

$$\hat{s}(\mathbf{k}) = \frac{1}{N^2} \sum_{\substack{\mathbf{k}=\mathbf{m}+\mathbf{n} \\ \mathbf{m},\mathbf{n}\in C_K}} \hat{u}(\mathbf{m})\hat{v}(\mathbf{n}) \qquad \text{para todo } \mathbf{k} \in C_K. \tag{2.11}$$

Sin embargo, el cálculo de este término como se presenta en (2.11) tiene un costo computacional de orden N^2 operaciones, es decir, si un producto se calcula de este modo en espacio de Fourier, no se tiene ninguna ventaja computacional respecto al uso de métodos algebraicos para la solución del problema.

En lugar de esto, se puede transformar los coeficientes de Fourier a espacio real usando la transformada inversa, realizar la multiplicación y finalmente transformar este resultado a espacio de Fourier, de modo que no se está usando un método puramente espectral para la solución del problema sino un método pseudo-espectral. Este procedimiento requiere tres transformadas de Fourier, requiriendo un numero de operaciones de orden $3N^2 \log_2 N^2$, representando una reducción en el costo computacional. Sin embargo, si esta operación se realiza directamente, se introducen errores de solapamiento de frecuencias (aliasing), como consecuencia del truncamiento de los modos de Fourier, es decir, es una consecuencia de la discretización del dominio.

La aplicación directa del método descrito resulta en que al transformar a espacio de Fourier el resultado del producto de las dos funciones en espacio real, no se obtienen los coeficientes $\hat{s}(\vec{k})$ sino

$$\tilde{s}(\mathbf{k}) = \hat{s}(\mathbf{k}) + \sum_{\substack{\mathbf{e}\neq(0,0) \\ e_\alpha=0,\pm1}} \left[\frac{1}{N^2}\right] \sum_{\substack{\mathbf{m}+\mathbf{n}=\mathbf{k}+N\mathbf{e} \\ \mathbf{m},\mathbf{n}\in C_K}} \hat{u}(\mathbf{m})\hat{v}(\mathbf{n}) \qquad \text{para todo } \mathbf{k} \in C_K. \tag{2.12}$$

Donde el segundo término de la derecha representa los errores por solapamiento de frecuencias.

El error de aliasing se eliminó mediante el método de desplazamiento de fase (Patterson y Orszag, 1971) que consiste en realizar la multiplicación en espacio real sobre mallas desplazadas por vectores constantes $\Delta\hat{\mathbf{e}}$, con $\hat{e}_\alpha = 0, 1$ y $\Delta = \pi/N$, entonces los nuevos puntos de la malla son $\mathbf{x} + \Delta\hat{\mathbf{e}}$, de modo que

$$u(\mathbf{x} + \Delta\hat{\mathbf{e}}) = \sum_{\mathbf{k}\in C_K} \hat{u}(\mathbf{k}) e^{i\mathbf{k}\cdot(\mathbf{x}+\Delta\hat{\mathbf{e}})} \qquad \mathbf{k} \in C_K.$$

Se procede de igual manera para encontrar $v(\mathbf{x} + \Delta\hat{\mathbf{e}})$ y se calcula el producto

$$s(\mathbf{x} + \Delta\hat{\mathbf{e}}) = u(\mathbf{x} + \Delta\hat{\mathbf{e}})v(\mathbf{x} + \Delta\hat{\mathbf{e}}),$$

permitiendo calcular sus coeficientes de Fourier como

$$\hat{s}^{\Delta}(\mathbf{k}) = \frac{1}{N^2} \sum_{0\leq\mathbf{n}<N} s(\mathbf{x} + \Delta\hat{\mathbf{e}}) e^{-i\mathbf{k}\cdot(\mathbf{x}+\Delta\hat{\mathbf{e}})},$$

donde se usó el superíndice Δ para indicar que los coeficientes corresponden a la malla desplazada. Esta operación es equivalente a

$$\hat{s}^{\Delta}(\mathbf{k}) = \hat{s}(\mathbf{k}) + \sum_{\substack{\mathbf{e}\neq(0,0)\\ e_\alpha=\pm 1,0}} e^{i\pi\mathbf{e}\cdot\hat{\mathbf{e}}} \times \frac{1}{N^2} \sum_{\substack{\mathbf{k}=\mathbf{m}+\mathbf{n}+N\hat{\mathbf{e}}\\ \mathbf{m},\mathbf{n}\in C_K}} u(\mathbf{m})v(\mathbf{n}) \qquad \mathbf{k}\in C_K.$$

Finalmente, si se efectúa la operación para los cuatro posibles vectores $\hat{\mathbf{e}}$, los coeficientes $\hat{s}$, sin error de solapamiento se obtienen como el promedio de los cuatro valores calculados,

$$\hat{s}(\mathbf{k}) = \frac{1}{4}\sum \hat{s}^{\Delta}(\mathbf{k})$$

El procedimiento se visualiza en el algoritmo 1.

Algoritmo 1 Eliminación del error por aliasing

$\hat{s}^{sum}_{l,k} \leftarrow 0$
para cada vector $\hat{\mathbf{e}}$ **hacer**
  para cada l **hacer**
    para cada k **hacer**
      $shift \leftarrow e^{\hat{e}_y\times\pi\times I\times l/N_y} \times e^{\hat{e}_x\times\pi\times I\times k/N_x}$
      $\hat{u}^{\Delta}_{l,k} \leftarrow \hat{u}_{l,k} \times shift$
      $\hat{v}^{\Delta}_{l,k} \leftarrow \hat{v}_{l,k} \times shift$
    fin para
  fin para
  $u^{\Delta}_{i,j} \leftarrow FFT^{-1}(\hat{u}^{\Delta}_{l,k})$
  $v^{\Delta}_{i,j} \leftarrow FFT^{-1}(\hat{v}^{\Delta}_{l,k})$
  para $i = 0 \rightarrow N_y$ **hacer**
    para $j = 0 \rightarrow N_x$ **hacer**
      $s^{\Delta}_{i,j} \leftarrow u^{\Delta}_{i,j} \times v^{\Delta}_{i,j}$
    fin para
  fin para
  $\hat{s}^{\Delta}_{l,k} \leftarrow FFT(s^{\Delta}_{i,j})$
  para cada l **hacer**
    para cada k **hacer**
      $shift \leftarrow e^{\hat{e}_y\times\pi\times I\times l_/N_y} \times e^{\hat{e}_x\times\pi\times I\times k/N_x}$
      $\hat{s}^{sum}_{l,k} \leftarrow \hat{s}^{sum}_{l,k} + \hat{s}^{\Delta}_{l,k}/shift$
    fin para
  fin para
fin para
$\hat{s}_{l,k} \leftarrow \hat{s}^{sum}_{l,k}/4$

2.4. INTEGRACIÓN TEMPORAL

La integración temporal se realiza mediante un método Runge-Kutta híbrido (implícito/explícito) de tercer orden (Spalart, Mosser, y Rogers, 1991).

El sistema de ecuaciones diferenciales parciales a resolver puede ser escrito como

$$\frac{\partial \mathbf{u}}{\partial t} = L_{\mathbf{u}}\mathbf{u} + N(\mathbf{u}), \tag{2.13}$$

donde $L_{\mathbf{u}}$ es un operador lineal y $N(\mathbf{u})$ es un operador no lineal. Ninguno depende del tiempo explícitamente. El término viscoso y el de presión son incluidos en $L_{\mathbf{u}}$, y $N(\mathbf{u})$ es el término convectivo, pero cualquier término lineal puede ser incluido en cualquier operador. $N(\mathbf{u})$ es difícil de linealizar en un método espectral, por lo que se requiere un esquema explícito para evitar iteraciones.

El avance temporal de $\mathbf{u}_n$ a $\mathbf{u}_{n+1}$ se realiza mediante tres subpasos:

$$\mathbf{u}' = \mathbf{u}_n + \Delta t\left[L(\alpha_1\mathbf{u}_n + \beta_1\mathbf{u}') + \gamma_1 N_n\right] \tag{2.14a}$$
$$\mathbf{u}'' = \mathbf{u}' + \Delta t\left[L(\alpha_2\mathbf{u}' + \beta_2\mathbf{u}'') + \gamma_2 N' + \zeta_1 N_n\right] \tag{2.14b}$$
$$\mathbf{u}_{n+1} = \mathbf{u}'' + \Delta t\left[L(\alpha_3\mathbf{u}'' + \beta_3\mathbf{u}_{n+1}) + \gamma_3 N'' + \zeta_2 N'\right] \tag{2.14c}$$

donde $N_n \equiv N(\mathbf{u}_n)$, $N' \equiv N(\mathbf{u}')$, y $N'' \equiv N(\mathbf{u}'')$. Los coeficientes utilizados en el método se escogen de modo que se obtenga precisión de orden 3 en el término convectivo, orden 2 en el término viscoso y que el método sea estable para cualquier valor de viscosidad(Spalart y cols., 1991).

$$\begin{aligned}
\alpha_1 &= \frac{29}{96}, & \alpha_2 &= -\frac{3}{40}, & \alpha_3 &= \frac{1}{6},\\
\beta_1 &= \frac{37}{160}, & \beta_2 &= \frac{5}{24}, & \beta_3 &= \frac{1}{6},\\
\gamma_1 &= \frac{8}{15}, & \gamma_2 &= \frac{5}{12}, & \gamma_3 &= \frac{3}{4},\\
\zeta_1 &= -\frac{17}{60}, & \zeta_2 &= -\frac{5}{12}. &&
\end{aligned}$$

Para determinar Δt se utiliza la condición de *Courant–Friedrichs–Lewy* (CFL) definida como

$$CFL \equiv \Delta t\left[\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y}\right]. \tag{2.15}$$

Una condición típica es utilizar $CFL = 1$, pero en este caso se usará una condición con $CFL = 0{,}3$.

El método se implementó utilizando

$$L_{\hat{\omega}}\hat{\omega} = -\frac{2\pi}{Re}|\mathbf{k}|^2\hat{\omega}(\mathbf{k},t) \quad \text{y} \quad N(\hat{\omega}) = -\hat{h}(\mathbf{k},t).$$

3. IMPLEMENTACIÓN EN PARALELO

En este capítulo se describen los detalles mas importantes de la implementación del algoritmo de solución de manera general y no se entrará en detalle en conceptos básicos de programación. Se recomienda consultar Kernighan y Ritchie (1988) como referencia del lenguaje C y Pacheco (2011) como guía de programación en paralelo.

El algoritmo para la simulación del problema se implementó en el lenguaje `C` usando programación en paralelo con MPI (Message Passing Interface). Adicionalmente se utilizó la librería de alto rendimiento FFTW (Fastest Fourier Transform of the West) que contiene funciones para calcular de forma eficiente la transformada discreta de Fourier y su transformada inversa en paralelo con MPI. La simulación parte de un estado de vorticidad aleatoria que se asignó utilizando la librería GSL (GNU Scientific Library).

Para realización de operaciones con números complejos se utilizaron las funciones contenidas en `complex.h`, que hace parte de la librería estándar de C.

3.1. PARALELIZACIÓN CON MPI

El programa fue paralelizado mediante MPI, que permite la escritura, compilación y ejecución de programas en sistemas de cómputo paralelo de memoria distribuida (clusters de computadores) o compartida (con procesadores multinucleo).

Un cluster de computadores es un sistema de memoria distribuida que consta de varios equipos de cómputo (o nodos de cómputo), con procesador y memoria independiente, que se encuentran interconectados mediante una interfaz de red que permite la comunicación entre ellos.

3.1.1. Distribución de trabajo en paralelo

En la presente implementación, se utilizó el modelo de programación en paralelo SPMD (single program, multiple data), es decir, todos los nodos de cómputo disponibles ejecutan simultaneamente una instancia, o *proceso*, del mismo programa, pero en la memoria de cada uno se encuentra solo una porción de los datos sobre los cuales se va a realizar operaciones. Si a su vez cada nodo de cómputo cuenta con varios procesadores (o procesador multi-núcleo), es posible ejecutar varios procesos por nodo (Pacheco, 2011).

Cada proceso MPI es una tarea independiente, de modo que cada uno tiene su espacio de memoria que no puede ser accedido por ningún otro, aún si el programa es ejecutado en un sistema con memoria compartida. Cuando un proceso requiere

datos que se encuentran en la memoria de otro, se debe establecer una comunicación entre ellos e intercambiar los datos necesarios mediante mensajes, esto quiere decir que los datos se copian del espacio de memoria de un proceso al espacio de memoria de otro (Pacheco, 2011). Las comunicaciones entre procesos pueden ser punto a punto o colectivas. En las comunicaciones punto a punto, se envían mensajes de un proceso a otro, esto es, solo dos procesos participan en la comunicación y esencialmente son usadas para enviar y recibir datos. Por otra parte, las comunicaciones colectivas permiten la participación en la comunicación de todo un grupo de procesos, y pueden ser usadas para sincronización, intercambio de datos o cálculos colectivos (Barney, 2012).

En el programa realizado se utilizaron solo funciones colectivas, ya que en todo momento que se requieren comunicaciones, todos los procesos ejecutados están involucrados y se puede aprovechar las funciones para cálculos colectivos.

Es necesario distribuir los datos de manera correcta en los procesos, ya que una inadecuada distribución de los mismos puede impactar de manera negativa el rendimiento del programa. En el caso de un cluster homogéneo, en el que todos los nodos de cómputo tienen las mismas características de hardware, es deseable que la cantidad de datos asignados sea la misma para cada uno de los procesos. Por otra parte, si las características de hardware son diferentes en los nodos de cómputo (cluster heterogéneo), será preferible una distribución proporcional a la capacidad de cada uno, ya sea asignando diferente numero de datos por proceso o asignando diferente numero de procesos por nodo. En el programa implementado, se asume que se va a ejecutar en un cluster homogeneo. La distribución de datos se realizó mediante una función de la librería FFTW y se discutirá en la sección 3.2.

3.1.2. Intercambio de datos entre procesos

Antes de hacer cualquier llamado a funciones MPI, se debe ejecutar la función `MPI_Init`, que inicializa el entorno de cómputo en paralelo en todos los procesos, permitiendo que estos puedan comunicarse entre si. Igualmente al final del programa, se debe finalizar con la función `MPI_Finalize` para declarar que no se van a realizar mas comunicaciones.

Cada proceso tiene una identificación que se crea en el momento en que se ejecuta la aplicación con MPI. Esta identificación (rank) es un entero positivo asignado de forma secuencial partiendo cero, y se puede obtener mediante la función `MPI_Comm_rank`, que al ejecutarse en cada proceso, almacena la identificación del mismo en una variable para que esté disponible durante la ejecución del programa.

Otro dato de importancia es el número de procesos efectivamente ejecutados. Dicha cantidad se puede obtener llamando la función `MPI_Comm_size` que igualmente almacena en la variable deseada el valor devuelto.

MPI provee las funciones necesarias para el intercambio de datos entre los procesos. Como se dijo anteriormente, solo se utilizaron rutinas que utilizan comunicación colectiva. A continuación se hace una breve descripción de las funciones utilizadas para las comunicaciones que no están implícitas en el cálculo de la transformada de Fourier. Para una descripción detallada del uso de las funciones, se recomienda consultar Barney (2012).

`MPI_Gather` recolecta arrays de varios procesos en uno solo array, de manera contigua, en el proceso especificado. Se requiere que todos los arrays sean del mismo tamaño.

`MPI_Gatherv` igual que `MPI_Gather` pero permite que los arrays sean de tamaños diferentes.

`MPI_Bcast` envía datos desde un proceso a todos los demás.

`MPI_Barrier` detiene la ejecución del programa y solo continúa cuando todos los procesos han llamado la función. Es útil para la sincronización de los procesos en un punto del programa.

`MPI_Scatter` divide un array de un proceso en partes iguales y envía cada parte a un proceso.

`MPI_Scatterv` es igual igual a `MPI_Scatter`, pero permite enviar porciones de diferentes tamaños.

`MPI_Reduce` realiza una operación de reducción, escribiendo el resultado en uno de los procesos.

`MPI_Allreduce` realiza una operación de reducción, enviando el resultado a todos los procesos.

En el programa se usaron las operaciones de reducción `MPI_SUM` para obtener la sumatoria de un grupo de valores y `MPI_MIN` para obtener valor mínimo entre un grupo de datos.

En el algoritmo 2 se muestra un esquema general del programa realizado para indicar el modo en que fueron usadas las funciones MPI.

3.1.3. Tipos de datos

Las funciones MPI para el envío de mensajes requieren como parámetros el tipo de dato y la cantidad de datos que van a ser enviados. MPI define los tipos de datos más comunes. En el programa realizado se usaron datos de tipo `MPI_INT` y `MPI_DOUBLE`, equivalentes a los tipos `int` y `double` en C. Algunas comunicaciones requirieron el envío de numeros complejos, sin embargo MPI no define el tipo de dato análogo a `double complex` de C, por lo que es necesario usar un tipo de dato derivado. Se

Algoritmo 2 Estructura general del programa

1: Inicializar MPI con `MPI_Init`
2: $rank \leftarrow$`MPI_Comm_rank`
3: $size \leftarrow$`MPI_Comm_size`
4: Declarar variables de tipo `MPI_Datatype` para crear nuevos tipos de datos.
5: Asignar el tamaño del nuevo tipo de dato con `MPI_Type_contiguous`
6: Validar el tipo de dato con `MPI_Type_commit`
7: Definición de la distribución de datos con FFTW.
8: Recolectar información de la distribución de datos en el proceso con $rank = 0$ con `MPI_Gather`.
9: Asignar el estado inicial de la simulación.
10: Recolectar el estado inicial en el proceso con $rank = 0$, con `MPI_Gatherv`
11: Calcular resultados parciales energia y enstropía del estado inicial en cada proceso.
12: Sumar los valores parciales de energía y enstropía en el proceso con $rank = 0$ con `MPI_Reduce` con la operación `MPI_SUM`.
13: Guardar estado inicial al disco duro en el proceso con $rank = 0$.
14: **para** Bucle de solución **hacer**
15: Calcular el incremento de tiempo dt en cada proceso.
16: Encontrar el valor mas bajo de dt entre todos los procesos y comunicarlo a todos ellos con `MPI_Allreduce` con la operación `MPI_MIN`.
17: Hacer paso temporal (algoritmo 5).
18: Unir datos en el proceso con $rank = 0$ con `MPI_Gatherv` y escribir al disco.
19: **fin para**
20: Finalizar MPI con `MPI_Finalize`

definió el tipo de dato `mpiComplex` mediante la función `MPI_Type_contiguous`, que permite establecer un tipo de dato a partir de un tipo ya definido. En el caso del tipo de dato `double complex`, su tamaño en memoria es equivalente a tener dos de tipo `double` contiguos, de modo que se puede usar esta función partiendo del tipo de dato `MPI_DOUBLE`, especificando que el nuevo tipo de dato es de un tamaño equivalente a dos datos de este tipo. Adicionalmente, la implementación de FFTW en MPI no admite el tipo de dato `int` como argumento de sus funciones, requiriendo en su lugar el uso de variables del tipo `ptrdiff_t`, que tampoco se encuentra definido como tipo de dato de MPI, por lo que también se definió el tipo `mpiptrdiff_t`, con un tamaño equivalente a dos datos de tipo `MPI_INT`.

3.2. TRANSFORMADA RÁPIDA DE FOURIER EN PARALELO CON FFTW

FFTW es una librería de alto rendimiento para el cómputo de la tranformada rápida de Fourier (FFT). Estas librerías no usan solo un algoritmo para el cálculo de la transformada, en cambio, consiste en un grupo de algoritmos que presentan diferente rendimiento dependiendo del tamaño de la transformada y del hardware del equipo (Frigo y Johnson, 2005).

3.2.1. Planes

Para realizar el cálculo de transformadas, FFTW crea un *plan*. Un plan es una variable de tipo `fftw_plan` que contiene información del algoritmo a utilizar para calcular transformadas durante la ejecución del programa. La creación del plan consiste en la ejecución de un benchmark antes de inicializar los datos, con el que se compara el rendimiento de los diferentes algoritmos y se escoge el mas adecuado para los requerimientos actuales. El plan es creado con las funciones `fftw_mpi_plan_dft_r2c_2d` para la transformada y `fftw_mpi_plan_dft_c2r_2d` para la inversa. Uno de los parámetros de dichas funciones indica que tan exhaustiva debe ser la búsqueda del algoritmo adecuado, de lo cual depende el tiempo que se invierte en la creación del plan. En el programa implementado, se utilizó el parámetro `FFTW_EXHAUSTIVE` que produce el resultado mas óptimo, utilizando un tiempo de inicialización mayor, pero esto no presenta inconveniente ya que la cantidad de transformadas y el tiempo de cómputo ameritan la búsqueda del mejor rendimiento.

Con MPI, la creación del plan depende también de la cantidad de procesos disponibles para el cálculo, que afecta en la división de los datos. Previo a la creación del plan, se debe establecer la cantidad de datos que le corresponde manejar a cada proceso MPI. Dicha distribución se realiza asumiendo que todos los nodos tienen las mismas capacidades de cómputo, de modo que la cantidad de datos presente en cada proceso es igual, o lo más equilibrada si la división exacta no es posible.

3.2.2. Distribución de datos y cálculo de transformadas

La división de los datos está soportada solo en una dimensión del array, de modo que si se tiene un grupo de datos de N_y filas y N_x columnas, la división se hará en las filas. Dicho de otra forma, si se ejecuta el programa en np procesos MPI, cada proceso tendrá una cantidad de datos correspondiente a N_y/np filas y N_x columnas.

La función `fftw_mpi_local_size_2d` permite obtener la cantidad de datos (de tipo complejo), el índice de la primera fila (del array original), y la cantidad de filas que va a estar presente en la memoria de cada proceso. A pesar de distribuir los datos de forma equitativa, FFTW requiere espacio en memoria adicional para la realización de operaciones intermedias, por lo que, aún cuando todos los procesos tienen la misma cantidad de filas asignadas, el proceso con $rank$ mas alto, tendrá una cantidad total de datos mayor. Estos datos solo tienen importancia para las funciones FFTW, de modo que no son de mayor interés para el usuario (Frigo y Johnson, 2012).

Las funciones `fftw_alloc_real` y `fftw_alloc_complex` permiten la asignación de memoria dinámica, utilizando como argumento el número de datos obtenido con la función `fftw_mpi_local_size_2d`. Estas funciones realizan la misma operación que la función `malloc` de C, pero garantizan que la memoria se asigne de forma tal que es posible utilizar las capacidades de paralelización de nivel de instrucción de los procesadores (Frigo y Johnson, 2012).

A pesar de que se está hablando de arreglos de datos bidimensionales, la memoria es asignada como un arreglo unidimensional y debe ser accedida en orden por filas, es decir, en al arreglo real, los primeros N_x datos corresponden a la fila 0, los siguientes N_x corresponden a la fila 1, y así sucesivamente.

La transformada de Fourier de una función real es simétrica de modo que $\hat{u}_{\mathbf{k}} = \hat{u}^*_{-\mathbf{k}}$, donde el asterisco denota el complejo conjugado. Esto permite que sólo sea necesario almacenar la mitad de los coeficientes de Fourier. En este caso se almacenan los coeficientes correspondientes a las $N_x/2 + 1$ frecuencias positivas, y la frecuencia $-N_x/2$, en la dimensión de las columnas y los coeficientes correspondientes a todas las N_y frecuencias en la dirección de las filas.

Las subrutinas `fftw_mpi_execute_dft_c2r` y `fftw_mpi_execute_dft_r2c` se encargan de ejecutar los planes previamente creados para calcular la tranformada e inversa, respectivamente. Estas funciones toman los $N_y \times N_x$ datos reales y obtienen los $N_y \times (N_x/2 + 1)$ coeficientes de Fourier.

La transformada se calcula a partir de un conjunto de números reales para obtener un conjunto de números complejos y viceversa. FFTW requiere que el array lógico de números reales sea ligeramente superior en tamaño al array físico en la dimensión de las columnas. Como se dijo anteriormente, el arreglo complejo se almacena en un array de N_y filas y $N_x/2+1$ columnas, con la división redondeada hacia abajo. El requerimiento es que en el arreglo lógico de números reales existan $2*(N_x/2+1)$

columnas, es decir, se requieren dos columnas adicionales si N_x es par y una columna adicional si es impar. Estas columnas adicionales solo son de importancia para la librería y el usuario solo debe omitirlas en cualquier operación o escritura de datos en los arreglos (Frigo y Johnson, 2012). Los índices usados para acceder al elemento (i, j) en un array con datos de espacio real será entonces `[i*2*(Nx/2+1)+j]` y en en el array de números complejos será `[i*(Nx/2+1)+j]`.

3.3. CONDICIÓN INICIAL

La condición inicial corresponde a un campo de velocidad aleatorio tal que el factor de adimensionalización $u' = 1$.

Se utilizó la librería GSL para obtener un conjunto uniforme de números pseudo-aleatorios mediante la función `gsl_rng_uniform`, con la que inicialmente se obtiene un conjunto de números entre -1 y 1 que se usan como valores de la vorticidad ω. Obtenidos dichos valores se calcula el valor equivalente al RMS de u y se dividen los valores de vorticidad para obtener un campo de vorticidad que cumpla la condición $u' = u_{RMS} = 1$.

La asignación inicial de números aleatorios se implementó sin comunicaciones entre los procesos, es decir que en cada proceso se generan los $N_y \times N_x$ números aleatorios, pero solo se almacenan los $N_y/np \times N_x$ correspondientes al proceso, como lo muestra el algoritmo 3, que también sirve de ejemplo de como se manejan los índices en el programa en paralelo.

Algoritmo 3 Inicialización de w

```
1: para cada i desde 0 hasta N_y hacer
2:    para cada j desde 0 hasta N_x hacer
3:       rndm ← random_number
4:       si i ≥ i_{local,inicial} y i < i_{local,inicial} + N_{y,local} entonces
5:          w_{i,j} ← rndm
6:       fin si
7:    fin para
8: fin para
```

Tras calcular los correspondientes coeficientes de Fourier del estado inicial, se requiere calcular el valor correspondiente a la RMS de u con los valores de ω obtenidos anteriormente. El valor RMS de este conjunto de números se puede evaluar mediante la expresión:

$$
\begin{aligned}
u_{RMS}^2 =& \frac{1}{N_x N_y} \sum_{i=0}^{N_y-1} \left[\sum_{j=0}^{Nx-1} |\mathbf{u}_{i,j}|^2 \right] \\
=& \frac{1}{N_x N_y} \sum_{i=0}^{N_y-1} \left[\sum_{j=0}^{Nx-1} (u_{i,j}^2 + v_{i,j}^2) \right].
\end{aligned}
\tag{3.1}
$$

El valor u_{RMS} puede ser calculado usando los coeficientes de Fourier $\hat{\omega}$ mediante el teorema de Parseval (Canuto, Hussaini, Quarteroni, y Zang, 2006), que relaciona la suma de los valores absolutos de una función elevados al cuadrado con la suma de los valores absolutos de sus coeficientes de Fourier elevados al cuadrado, que en forma discreta se escribe,

$$
\sum_{n=0}^{N-1} |u(n)|^2 = \frac{1}{N} \sum_{k=-N/2}^{N/2-1} |\hat{u}(k)|^2. \tag{3.2}
$$

Usando (1.8) sobre la ecuación (3.1), se tiene

$$
u_{RMS}^2 = \frac{1}{N_x N_y} \sum_{i=0}^{N_y-1} \left[\sum_{j=0}^{Nx-1} \left[\left(\frac{\partial \psi_{i,j}}{\partial y} \right)^2 + \left(\frac{\partial \psi_{i,j}}{\partial x} \right)^2 \right] \right].
$$

La relación (2.6) permite escribir, con con $\mathbf{k} = (k, l)^T$,

$$
\widehat{\frac{\partial \psi_{i,j}}{\partial x}} = ik\hat{\psi}_{l,k} \qquad y \qquad \widehat{\frac{\partial \psi_{i,j}}{\partial y}} = il\hat{\psi}_{l,k}
$$

Mediante (3.2) se puede obtiene la relación

$$
\begin{aligned}
u_{RMS}^2 =& \frac{1}{N_x^2 N_y^2} \sum_{l=-N_y/2}^{N_y/2-1} \left[\sum_{k=-N_x/2}^{Nx/2-1} \left| il\hat{\psi}_{l,k} \right|^2 + \left| ik\hat{\psi}_{l,k} \right|^2 \right] \\
=& \frac{1}{N_x^2 N_y^2} \sum_{l=-N_y/2}^{N_y/2-1} \left[\sum_{k=-N_x/2}^{Nx/2-1} |\hat{\psi}_{l,k}|^2 \left(l^2 + k^2 \right) \right].
\end{aligned}
$$

La ecuación de continuidad en espacio de Fourier (2.7) puede escribirse como

$$\hat{\psi}_{l,k} = \frac{\hat{\omega}_{l,k}}{l^2 + k^2},$$

con lo que finalmente se obtiene la expresión

$$u_{RMS}^2 = \frac{1}{N_x^2 N_y^2} \sum_{l=-N_y/2}^{N_y/2-1} \left[\sum_{k=-N_x/2}^{Nx/2-1} \frac{|\hat{\omega}_{l,k}|^2}{l^2 + k^2} \right], \tag{3.3}$$

que permite calcular el valor u_{RMS} como función de los coeficientes de Fourier de la vorticidad $\hat{\omega}$.

Como se ha dicho anteriormente los datos se encuentran divididos, de modo que cada proceso ejecutado calcula un valor parcial de la RMS de acuerdo a (3.3), por lo que se aplica una operación de reducción con `MPI_Allreduce` para sumar los valores parciales calculados en cada proceso y entregar el resultado total a todos los procesos. El algoritmo 4 muestra como se realiza el cálculo de los valores parciales de la RMS. Al calcular dichos valores parciales se omite el valor correspondiente al modo $\mathbf{k} = (0, 0)$ para evitar la división por cero y los coeficientes de las filas correspondientes a $\mathbf{k} = (l, 0)$ y $\mathbf{k} = (l, -N_x/2)$ se cuentan una vez debido a que estas columnas son simétricas respecto a ellas mismas, mientras que los demás coeficientes se deben contar dos veces por no encontrarse sus conjugados presentes en el array.

Algoritmo 4 Cálculo de la RMS de u

1: $u_{RMS} \leftarrow 0$
2: **para** cada l_{local} **hacer**
3: **para** cada k **hacer**
4: **si** $k = 0$ y $l_{local} = 0$ **entonces**
5: $u_{RMS} \leftarrow u_{RMS} + 0$
6: **si no**, **si** $k = -N/2$ o $k = 0$ **entonces**
7: $u_{RMS} \leftarrow u_{RMS} + |\hat{w}_{l,k}|^2/(k_j^2 + l_i^2)$
8: **si no**
9: $u_{RMS} \leftarrow u_{RMS} + 2 \times |\hat{w}_{l,k}|^2/(k_j^2 + l_i^2)$
10: **fin si**
11: **fin para**
12: **fin para**
13: $u_{RMS} \leftarrow \sqrt{u_{RMS}/(N_x^2 \times N_y^2)}$
14: Obtener suma total en todos los procesos con `MPI_Allreduce`

Dividiendo cada uno de los coeficientes de Fourier por u_{RMS} se obtienen los coeficientes de Fourier de la vorticidad adimensional correspondiente al campo de velocidad anteriormente descrito.

3.4. DETALLES DE LA IMPLEMENTACIÓN

3.4.1. Estructuras de datos

En el programa fueron utilizadas variables de tipo `struct` para agrupar los datos de manera que el código sea fácilmente relacionado con los métodos y formulaciones anteriormente expuestas. En los contadores de los bucles y variables relacionadas con las dimensiones del problema, se utilizaron variables de tipo `ptrdiff_t` en lugar de `int` ya que, como se dijo anteriormente, las funciones de la librería FFTW para MPI solo soportan este tipo de dato como parámetro. A continuación se describen las estructuras de datos definidas en el programa y las variables contenidas en ellas.

- `dminfo` contiene las variables relacionadas con el tamaño del problema.
  - `const ptrdiff_t` Nx, Ny. Son el número de divisiones del dominio en x y y, respectivamente.
  - `ptrdiff_t` $Nx21$, $Ny21$. Almacenan a los valores $N_x/2+1$ y $N_y/2+1$, respectivamente.
  - `ptrdiff_t` $alloc\_local$, $local\_Ny$, $local\_y\_start$. Son las variables relacionadas con la división de los datos en los procesos, siendo respectivamente, el número de datos de tipo `fftw_complex`, el número de filas y el índice de la primera fila presente en el proceso.
  - `int` *$modesX$, *$modesY$. Son punteros a los arreglos que contienen los índices de los modos de Fourier. El primero contiene los valores $0, 1, 2...-(N_x/2+1)$ que corresponde a las frecuencias negativas y la frecuencia $-(Nx/2+1)$ en la dirección de las columnas, y la segunda contiene los valores $0, 1, 2, ...-(N_y/2+1), -(N_y/2+1)+1, -(N_y/2+1)+2, ...-1$, que corresponde a todas las Ny frecuencias en la dirección de las filas.
  - `const double` Re. Es el número de Reynolds del flujo.
  - `double` *$kkabs$. En un puntero al array que contiene los valores de la operación $|\mathbf{k}|^2$ para cada uno de los modos de Fourier. Esta operación se repite con frecuencia en la simulación por lo que se decidió almacenarla.
  - `double` *$linop$. Es un puntero al array que contiene el operador lineal $2\pi|\mathbf{k}|^2/Re$ para cada uno de los modos de Fourier.
- `funx` se utilizó para agrupar los punteros a los arrays que contienen valores en espacio real y de Fourier de los términos $\hat{\omega}$ y $\hat{\psi}$.
  - `fftw_complex` *$four$. Coeficientes de Fourier de la función.
  - `fftw_complex` *dxf, *dyf. Coeficientes de Fourier de las derivadas parciales respecto a x y y, respectivamente.

  - `double` *$real$. Valores de la función en espacio real.
  - `double` *$real1$. Almacenamiento auxiliar de valores en espacio real de la función.
- `fshifts` agrupa los punteros a los arrays relacionados con el cálculo de los valores de $\hat{h}$ como fue descrito en la sección 2.3.
  - `fftw_complex` *$four1s$, *$four2s$. Valores de las coeficientes de Fourier correspondientes a la malla desplazada de las dos funciones a multiplicar.
  - `fftw_complex` *$four$. Valores resultantes de la multiplicación sin error de solapamiento.
  - `fftw_complex` *$four1$. Almacenamiento auxiliar de valores en espacio de Fourier.
  - `double` *$real$. Valores de la función en espacio real.
  - `double` *$real1$. Almacenamiento auxiliar de valores en espacio real.
- `rkvar` agrupa las variables relacionadas con el método de integración temporal.
  - `double` *$alpha$, *$beta$, *$gamma$, *$zeta$. Son punteros a los arrays de los coeficientes utilizados en el método.
  - `double` t, dt, CFL. Son el tiempo total, el incremento de tiempo y el número de *Courant–Friedrichs–Lewy*.
  - `fftw_complex` *$wp1$, *$hm1$. Punteros a los arrays que almacenan los coeficientes de Fourier de $\hat{\omega}$ del tiempo siguiente y $\hat{h}$ del tiempo anterior, necesarios en el método.

3.4.2. Funciones

El cálculo de los coeficientes $\hat{\psi}$ de acuerdo a la ecuación (2.7) que puede ser escrita para cada modo como $\hat{\psi}_{l,k} = \hat{\omega}_{l,k}/(l^2 + k^2)$. La función

```
void psiCalc(double complex *in, double complex *out, diminfo dims);
```

toma los valores de `in`, aplica la operación en cada uno de los elementos y los almacena en `out`.

Una de las ventajas de transformar el dominio a espacio de Fourier es la facilidad para expresar las derivadas parciales en función de los coeficientes de Fourier. De acuerdo a la primera relación de (2.6), los coeficientes de Fourier de la derivadas parciales de una función se pueden calcular como

$$\left(\widehat{\frac{\partial u}{\partial x}}\right)_{l,k} = ik\hat{u}_{l,k} \quad \text{y} \quad \left(\widehat{\frac{\partial u}{\partial y}}\right)_{l,k} = il\hat{u}_{l,k}; \quad \mathbf{k} = [l,k]^T, \quad i = \sqrt{-1}.$$

La función

```
void fourierDiff(double complex *in, double complex *outdx,
                            double complex *outdy, diminfo dims);
```

realiza este procedimiento, tomando los valores de `in`, calcula los coeficientes de Fourier de las derivadas parciales respecto a x y y, y los almacena en `outdx` y `outdx`, respectivamente. Sin embargo, aplicar esta operación resulta en una transformada que pierde la simetría, lo que produce una transfromada inversa no real. Para corregir esto, se utiliza la función

```
void nyquisteraser(double complex *in, diminfo dims);
```

que toma el array `in` y hace cero los valores correspondientes a los modos tales que para todo $\mathbf{k}$, $l = -(Ny/2+1)$ o $k = -(N_x/2+1)$, o dicho de otra manera la fila de modos $-(N_y/2+1)$ y la columna de modos $-(N_x/2+1)$, que en el array lógico de coeficientes de Fourier corresponde a la fila $N_y/2$ y la columna $N_x/2$. Con este procedimiento se corrige la simetría, haciendo posible obtener una transformada inversa sin parte imaginaria.

El cálculo del término no lineal con eliminación del aliasing se implementó en la función

```
void shiftDaCalc(double complex *in1, double complex *in2,
                  double complex *out,fshifts h, diminfo dims,
                                fftw_plan p_c2r, fftw_plan p_r2c);
```

que permite calcular un producto entre funciones utilizando el método pseudo espectral, donde los parámetros `in1` e `in2` son los datos correspondientes a cada uno de los términos de la multiplicación y `out` es el resultado sin aliasing. Para calcular el término $\hat{h}$, se ejecuta la función dos veces, una con los coeficientes de $\partial\psi/\partial y$ y $\partial\omega/\partial x$, y otra con los coeficientes de $\partial\psi/\partial x$ y $\partial\omega/\partial y$, y posteriormente se realiza la resta punto a punto de los dos resultados en espacio de Fourier. En el procedimiento se deben computar transformadas y transformadas inversas por lo que es necesario pasar los planes de FFTW como parámetro.

La integración temporal se realiza mediante la función

```
void compute(funx w, funx p, fshifts h, diminfo dims, rkvar rk,
                                fftw_plan p_r2c, fftw_plan p_c2r);
```

en la que se utilizan las funciones mencionadas y se realiza la integración temporal. Cada vez que se llama esta función se calculan los valores de $\hat{\omega}$ en el tiempo $t_{n+1} = t_n + \Delta t$. El bucle de solución consiste en llamar esta función cuantas veces

se requiera entre estados a guardar. El algoritmo 5 muestra la estructura de la dicha función, mostrando el uso de las otras funciones anteriormente descritas. El valor de $L = -2\pi|\mathbf{k}|^2/Re$ mostrado en el algoritmo es precalculado en el programa y es almacenado en la variable $linop$.

Algoritmo 5 Cálculo de un paso temporal

1: $\beta_0 \leftarrow 0$
2: **para** $i = 0$ **hasta** $i < 3$ **hacer**
3: $\hat{\psi} \leftarrow$`psiCalc`$(\hat{\omega})$
4: $\partial_x\hat{\omega}, \partial_y\hat{\omega} \leftarrow$`fourierDiff`$(\hat{\omega})$
5: $\partial_x\hat{\psi}, \partial_y\hat{\psi} \leftarrow$`fourierDiff`$(\hat{\psi})$
6: $\hat{h}_1 \leftarrow$`shiftDaCalc`$(\partial_x\hat{\psi}, \partial_y\hat{\omega})$
7: $\hat{h}_2 \leftarrow$`shiftDaCalc`$(\partial_x\hat{\omega}, \partial_y\hat{\psi})$
8: $\hat{h} \leftarrow \hat{h}_2 - \hat{h}_1$
9: `nyquisteraser`$(\hat{h})$
10: $\hat{\omega}_{n+1} \leftarrow \left[\hat{\omega}_n(1 + dtL\alpha_i) + dt(\gamma_i\hat{h}_n + \zeta_i\hat{h}_{n-1})\right] / (1 - dlL\beta_i)$
11: $\hat{\omega}_n \leftarrow \hat{\omega}_{n+1}$
12: **fin para**

La integración temporal se implementó usando un paso temporal variable que se recalcula con la función

```
void setdt(funx p,diminfo dims, rkvar *rk, fftw_plan p_c2r);
```

cada vez que se guarda un estado de la vorticidad. Dicho incremento se calcula como se detalló en la sección 2.4, mediante la expresión (2.15). El cálculo del nuevo Δt requiere obtener las componentes x y y de la velocidad máxima, por lo que se requiere el cálculo de las transformadas inversas de las derivadas parciales de $\hat{\psi}$, por lo que se debe pasar como argumento el correspondiente plan de FFTW.

3.4.3. Almacenamiento de resultados

El programa escribe los resultados al disco duro en archivos de *valores separados por comas* (CSV) con la función

```
void writeCSV(double *in, diminfo dims, double t, int filenum,
                                                    char *dirname);
```

que se ejecuta en el proceso MPI identificado con $rank = 0$, en el que se unieron previamente los resultados de todos los nodos con la función `MPI_Gatherv` de MPI. Los resultados fueron importados al programa *ParaView* para su su visualización. Estos se visualizan aplicando el filtro “Tabla a malla estructurada”. Cada estado corresponde a un archivo CSV que, para cada valor de vorticidad en la malla, contiene

las coordenadas x, y, z correspondientes, con $z = 0$ por tratarse de un dominio en dos dimensiones.

Una característica adicional implementada en el programa es la capacidad de retomar la simulación en el ultimo estado guardado. Esto permite reanudar el programa después de haber sido interrumpido de forma voluntaria o accidental, o si se requiere ejecutar la simulación durante más pasos temporales que los planeados originalmente. Esto se logra guardando las variables importantes del problema en un archivo binario utilizando la función

```
void writeHstartFile(double *wRealTotal, double complex *wFourTotal,
             diminfo dims, rkvar rk, ptrdiff_t i, char *filename);.
```

Al inicio del programa se verifica si el archivo de respaldo existe en el directorio de salida y verifica la validez del mismo para las condiciones actuales de la simulación. Si el archivo resulta no válido, este será ignorado, la simulación empezará a correr desde el principio y el archivo de respaldo será sobreescrito.

Los valores de métricas globales de energía y enstropía (que se discutirá en la sección 4.2), se calculan cada vez que es guardado un estado de vorticidad por la función

```
void energyCalc(double complex *wFour, double *kenergy,
                                    double *enstrophy, diminfo dims);
```

que es ejecutada en cada proceso, en los que se obtiene un valor parcial para luego ser aplicada una operación de reducción `MPI_Reduce`, con el parámetro `MPI_SUM`. Estos valores se escriben en archivos CSV directamente en la función `main`.

Si los directorios y sub directorios necesarios para el almacenamiento de datos no existen, serán creados por el programa en la ejecución, mediante la función

```
int makeDir(char *dirname);
```

que retorna un valor de tipo `int` para verificar si la creación fue exitosa o no. En caso de no poder crear los directorios, el programa es finalizado.

3.5. COMPILACIÓN Y EJECUCIÓN

La compilación del programa se realiza con el comando `mpicc` (que se incluye en la instalación de MPI). Para la compilación de el programa realizado se escribió un Makefile que se utiliza con el comando `make` para la compilación automática del programa.

En el momento de la compilación se deben vincular las librerías de FFTW con las banderas `-lfftw3_mpi -lfftw3`, la librería GSL con `-lgsl -lgslcblas` y las li-

brerías matemáticas de C con `-lm`. Las vinculaciones se encuentran en el archivo Makefile y solo hace falta ejecutar el comando `make` en la linea de comandos, en el directorio que contiene el makefile, y se genera el binario `dns2d` de forma automática en el mismo directorio.

El programa se ejecuta con el comando `mpirun` (también incluido en la instalación de MPI), y este se encarga de ejecutar los procesos en los nodos y permitir las comunicaciones entre los procesos ejecutados.

En la ejecución se requiere especificar el número de procesos a ejecutar con el parámetro `-np`, y se debe especificar un archivo que contiene una lista de los nombres host de los equipos disponibles en el cluster y los slots disponibles en cada equipo. Los slots especifican la cantidad de procesos que pueden ser ejecutados en un equipo del cluster.

Además de los parámetros que requiere `mpirun`, el programa realizado requiere una serie de parámetros obligatorios y otros opcionales. El programa se ejecuta de la siguiente forma

```
mpirun -np NP -machinefile machines dns2d Nx Ny Re outDir
```

en donde `NP` (número de procesos) y `machines` (nombre del archivo de host) son parámetros para MPI. `Nx` (divisiones de la malla en la dirección x), `Ny` (divisiones de la malla en dirección y) y `Re` (número de Reynolds del flujo) son parámetros obligatorios del programa. El parámetro `outDir` es un parámetro opcional que indica el subdirectorio en que el programa va a guardar los resultados obtenidos. Si el parámetro `outDir` no se especifica, los resultados se almacenarán en el subdirectorio `output`.

El programa puede ser compilado y ejecutado con un solo comando la llamada a un script *BASH*, que simplemente consta del comando make y el comando de ejecución anteriormente mostrado. El script se encuentra en el archivo *runmpi.sh* y se puede ejecutar en la linea de comandos mediante `./runmpi.sh`.

Las pruebas de tiempo de cálculo se ejecutaron mediante un script BASH que tienen dos bucles anidados, uno varía el tamaño del problema y el otro varía el número de procesos a usar. La prueba de rendimiento se puede ejecutar mediante `./performanceTest.sh`.

4. RESULTADOS

4.1. VORTICIDAD

El programa fue ejecutado en una malla de 1024×1024 puntos con $Re = 50000$. Los resultados fueron almacenados cada 100 pasos temporales calculados. Se almacenaron 3000 resultados, que corresponde a un total de 300000 pasos temporales.

En la figura Figura 1 se muestran algunos de los resultados de la vorticidad. En un primer instante, la distribución de velocidades es aleatoria, por lo que no se identifican vórtices a simple vista. Al cabo de 400 pasos temporales, se evidencia la formación de pequeños vórtices, y después de 1500 pasos temporales desde el inicio de la simulación, son fácilmente identificables. Al pasar el tiempo se evidencia la forma cómo los vórtices pequeños se unen para formar vórtices más grandes y los vórtices pequeños se siguen sumando a dichos vórtices grandes, dejando en evidencia que se tiene una cascada inversa de energía (Yin, 2003), es decir, la energía de los vórtices pequeños pasa a los vórtices mas grandes. Después de 300000 pasos temporales, solo quedan dos vórtices restantes, uno en sentido positivo y otro en sentido negativo, presentando un una reducción lenta en la energía cinética total del sistema.

En la figura Figura 2 muestra el valor absoluto de los coeficientes de Fourier de la vorticidad en los primeros 200 pasos temporales, correspondientes a los $N_x/2 + 1$ modos positivos en la dirección horizontal y todos los N_y modos en la dirección vertical de la figura. En la figura se aprecia una acumulación de energía en los modos interiores (las frecuencias mas bajas), mientras que el valor en los modos exteriores (frecuencias mas altas) tienden a volverse cero con el paso del tiempo, siendo consistente con la cascada inversa de energía.

4.2. VALIDACIÓN

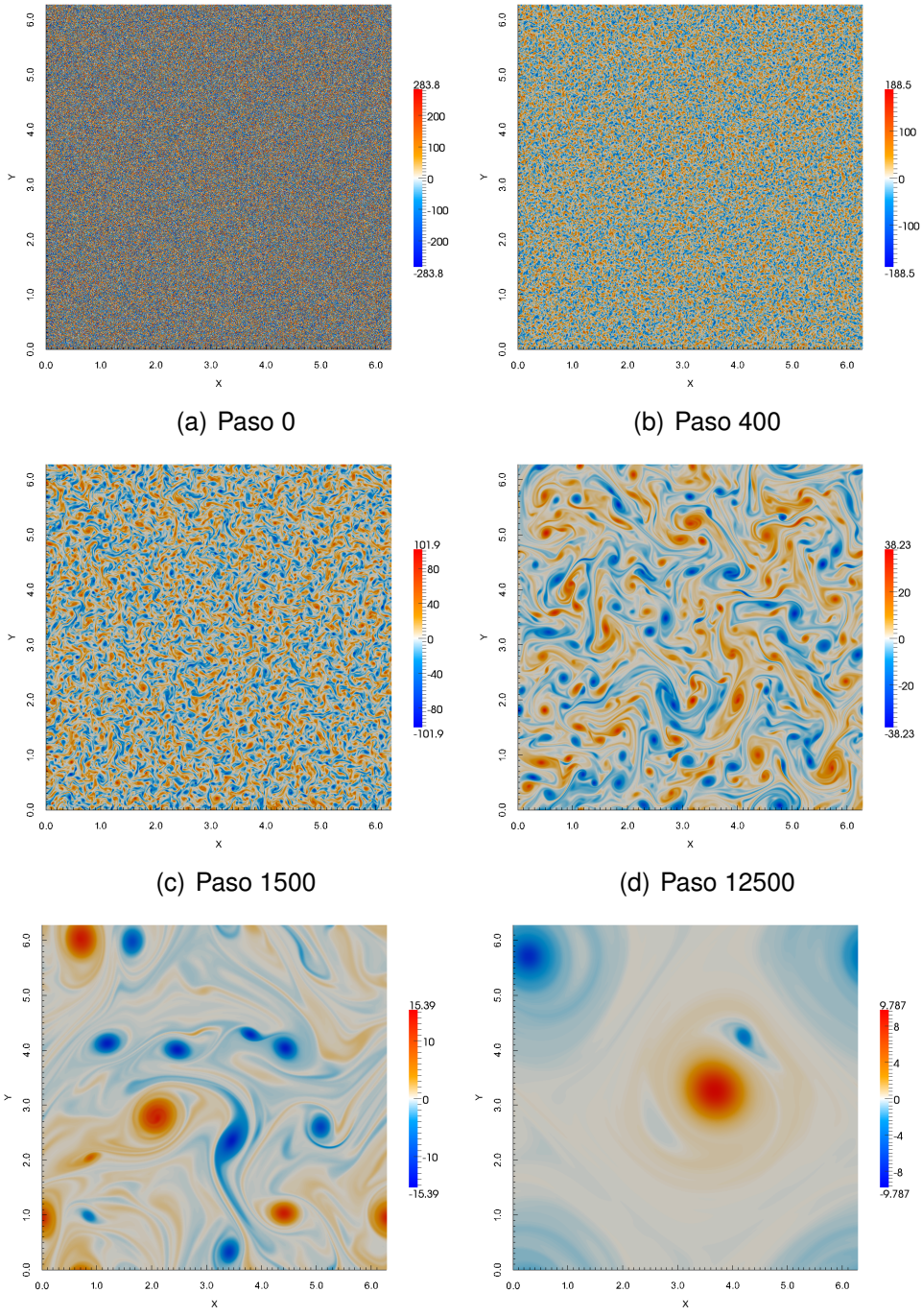
La validez de los resultados obtenidos se verificó mediante mediciones globales de energía cinética y enstropía. Al no existir fuente externa de energía, dichas cantidades deben disminuir al pasar el tiempo pues la energía se disipa por efecto de la viscosidad.

La energía cinética E y la enstropía $\mathcal{E}$ se pueden evaluar, respectivamente, mediante las expresiones

$$E(t) = \frac{1}{2}\int_{\omega}(u^2 + v^2)d\Omega, \quad \mathcal{E}(t) = \frac{1}{2}\int_{\Omega}\omega^2 d\Omega.$$

Mediante un procedimiento similar al utilizado para obtener (3.3), usando el teorema de Parseval, dichas cantidades pueden ser aproximadas utilizando los coeficientes

Figura 1: Evolución de la vorticidad.



(a) Paso 0

(b) Paso 400

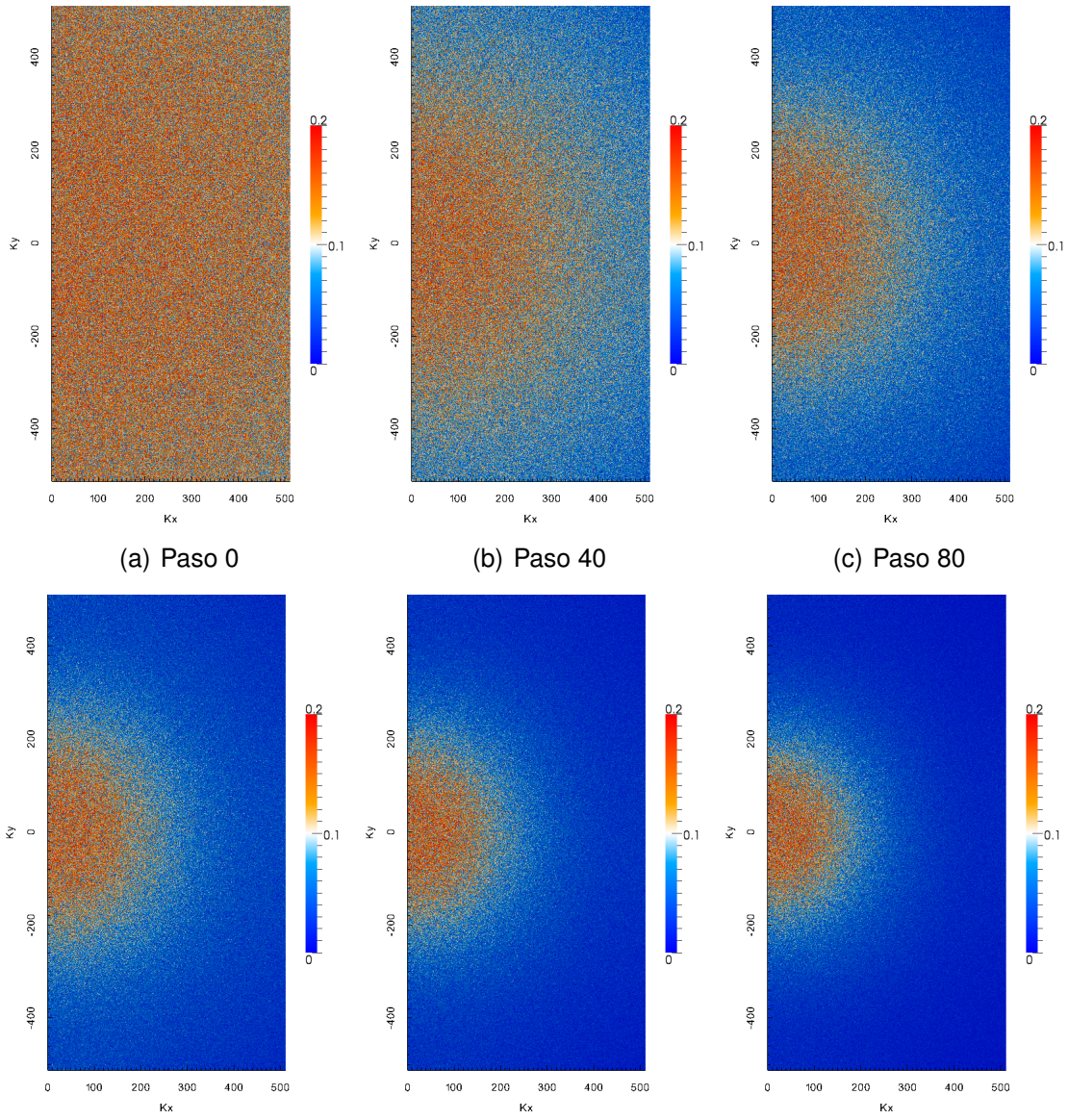
(c) Paso 1500

(d) Paso 12500

(e) Paso 80000

(f) Paso 300000

Figura 2: Evolución de los coeficientes de Fourier de la vorticidad. Se muestra el valor absoluto de los coeficientes para $(N_x/2+1) \times N_y$ modos.



(a) Paso 0 (b) Paso 40 (c) Paso 80

(d) Paso 120 (e) Paso 160 (f) Paso 200

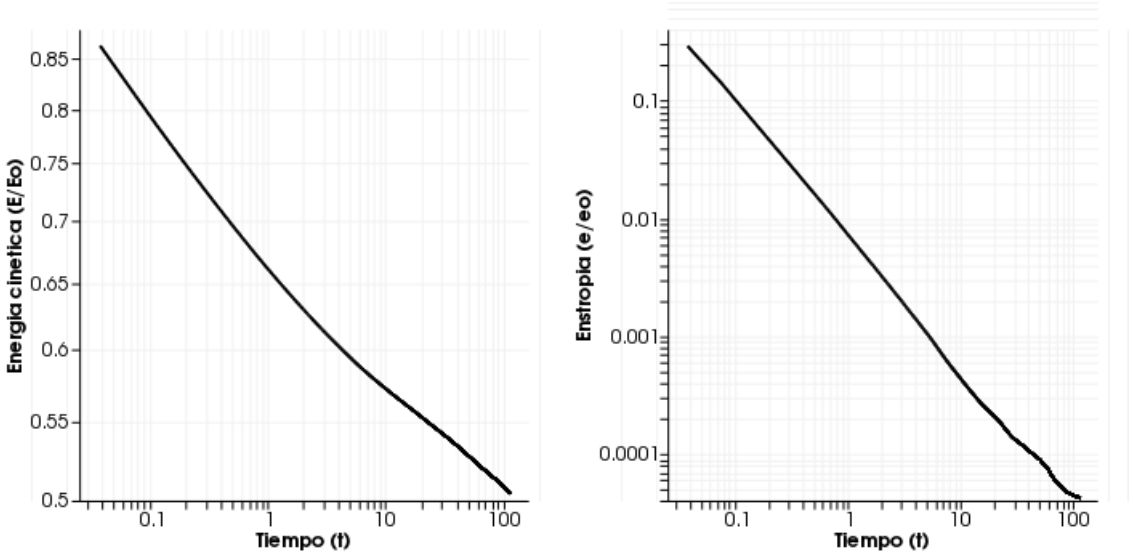
de Fourier de la vorticidad $\hat{\omega}$, mediante las expresiones

$$E(t) \approx \frac{2\pi^2}{N_x^2 N_y^2} \sum_{l=-N_y/2}^{N_y/2-1} \left[\sum_{k=-N_x/2}^{N_x/2-1} \frac{|\hat{w}_{k,l}|^2}{k^2+l^2} \right],$$

$$\mathcal{E}(t) \approx \frac{2\pi^2}{N_x^2 N_y^2} \sum_{l=-N_y/2}^{N_y/2-1} \left[\sum_{k=-N_x/2}^{N_x/2-1} |\hat{w}_{k,l}|^2 \right].$$

La Figura Figura 3 muestra la evolución en el tiempo de dichas cantidades.

Figura 3: Evolución en el tiempo de la energía cinética y enstropía del sistema.



(a) Energía (b) Enstropía

Se observa una caída rápida de la energía cinética en el instante inicial, llegando a aproximadamente un 86 % de la energía inicial en los primeros 100 pasos temporales, llegando a 70 % en el paso 1400, a 60 % en el paso 11300, y continúa decayendo lentamente hasta un valor cercano al 50 % en el paso 300000.

Por otra parte, se puede demostrar (Yin, 2003) que las cantidades E y $\mathcal{E}$ se relacionan mediante la expresión

$$\frac{dE}{dt} = -2\nu\mathcal{E}.$$

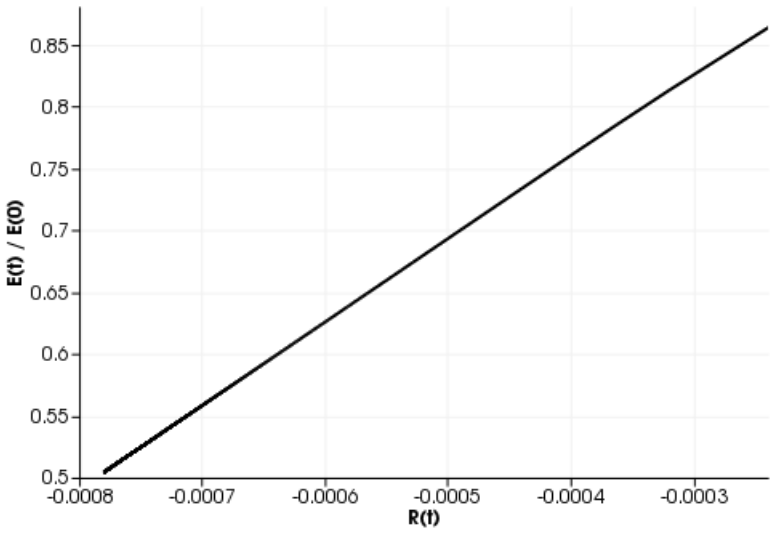
Intengrando dicha expresión, se obtiene

$$E(t) = CR(t), \quad \text{con} \quad R(t) = -\int_0^t \mathcal{E} dt \tag{4.1}$$

La figura Figura 4 muestra la relación entre las cantidades

$$\frac{E(t)}{E(0)} \qquad y \qquad \frac{R(t)}{\mathcal{E}(0)},$$

Figura 4: Relación entre la energía cinética y la enstropía.



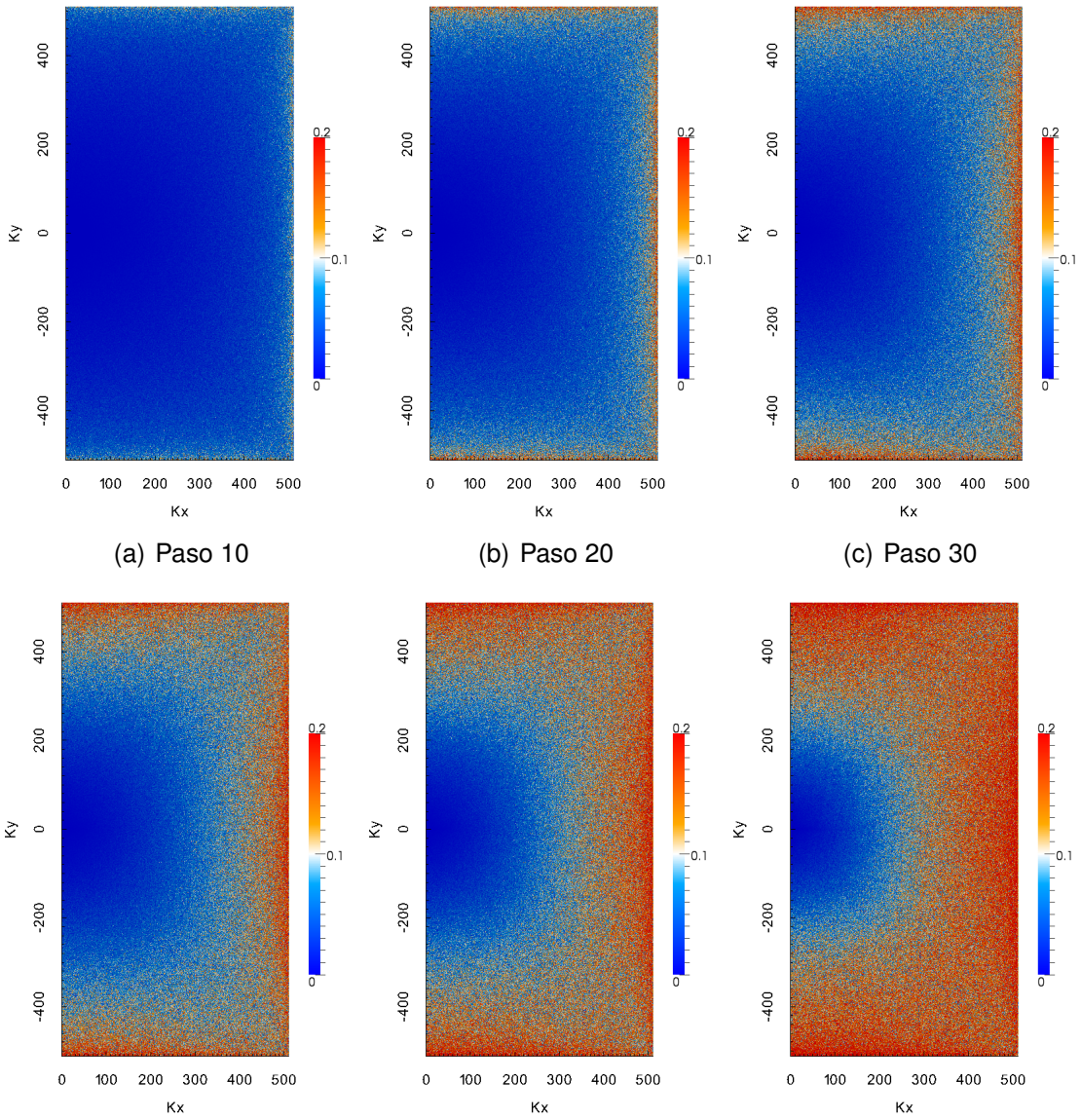
que como es de esperarse según (4.1), mantienen una relación prácticamente lineal.

Los anteriores resultados muestran que los resultados que arroja el programa tienen un comportamiento físico correcto, y junto con los resultados de la vorticidad, se puede asumir que el programa está funcionando de manera correcta.

4.3. EFECTO DEL SOLAPAMIENTO DE FRECUENCIAS

La figura Figura 5 muestra la acumulación del error de aliasing cada 10 pasos temporales. Partiendo de la misma condición inicial, se calculó cada paso temporal con y sin aliasing y calculando la diferencia entre los coeficientes de Fourier. Se ha graficado el valor absoluto de dicha diferencia, mostrando solo los modos 0 a $N_x/2$ en en la dirección horizontal. En la figura Figura 5 se observa un rápido incremento en el valor absoluto de los coeficientes de Fourier en los modos exteriores, mostrando una tendencia incorrecta, ya que, en general, el valor de los coeficientes tiende a disminuir, y como se mostró en la figura Figura 2, los coeficientes cuyos valores llegan a cero mas rápidamente son los correspondientes a los modos exteriores. Lo expuesto anteriormente físicamente implicaría que se le está suministrando energía al sistema, lo cual no es correcto.

Figura 5: Efecto del solapamiento de frecuencias en los modos de Fourier.



(a) Paso 10 (b) Paso 20 (c) Paso 30

(d) Paso 40 (e) Paso 50 (f) Paso 60

4.4. RENDIMIENTO DEL PROGRAMA

Siempre que se hable de cómputo en paralelo, se debe tener presente el rendimiento del programa. Obtener el menor tiempo de cómputo posible es tan importante como obtener resultados correctos, ya que la programación implica un esfuerzo extra y esto debe ser retribuido con menores tiempos de cómputo.

Si se ejecuta un programa en paralelo en np procesos, idealmente su tiempo de cómputo sería

$$t_{np}^{i} = \frac{t_s}{np},$$

donde t_s es el tiempo empleado por el programa secuencial. Sin embargo, esta relación no se cumple debido a que la comunicación entre procesos introduce tiempos adicionales donde los procesos están enviando o esperando recibir un mensaje, entonces t_{np}^{i} es el menor tiempo que se puede esperar para un programa paralelo ejecutado en np procesos. Una forma de medir los resultados de la paralelización de un programa es midiendo los tiempos de cómputo para diferentes valores de np y obtener la relación

$$S_{np} = \frac{t_s}{t_{np}},$$

cuyo valor es la ganancia de velocidad obtenida. En el caso que $S_{np} = np$, se dice que el programa tiene una ganancia de velocidad lineal, que es un caso ideal. La figura Figura 6 muestra los resultados obtenidos para diferentes valores de np y diferentes tamaños de problema. Las lineas no solidas corresponden a tamaños de problema muy pequeños, que no tendrían interés físico, pero se incluyeron para observar el comportamiento del programa.

Se observa que la ganancia de velocidad nunca se acerca a la ganancia lineal, y a mayor número de procesos, la diferencia se hace cada vez mayor.

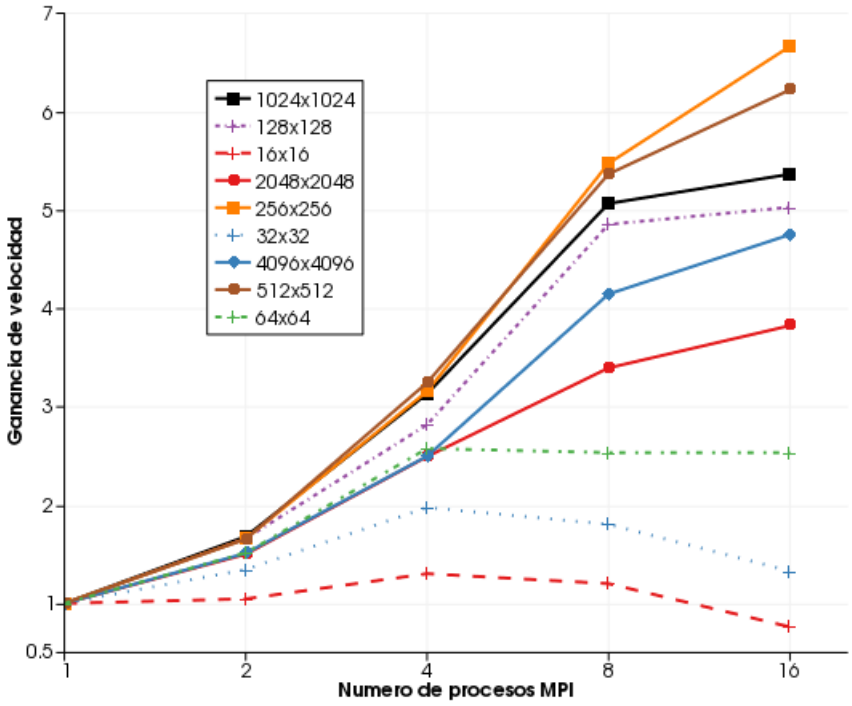
Otra forma de estimar el costo de la paralelización es analizar que tan cerca se encuentra la ganancia real del programa a la ganancia lineal o ideal. Esta comparación se conoce como la eficiencia del programa en paralelo y se define como

$$\eta_{np} = \frac{S_{np}}{np} = \frac{t_s}{t_{np} \cdot np},$$

que se puede ver como una medida del costo de las comunicaciones entre procesos. La figura Figura 7 muestra los valores de η_{np} del programa para diferentes tamaños de problema.

Los resultados mostrados en las figuras Figura 6 y Figura 7 se obtuvieron ejecutando el programa para los diferentes tamaños de problema para diferentes números de procesos, guardando 10 medidas de tiempo, tomadas cada 100 pasos temporales, para cada combinación de tamaño y número de procesos. De las 10 medidas se utilizó la menos obtenida, que representa el rendimiento pico en la prueba. Los tiempos fueron medidos con la función `MPI_Wtime` de la librería de MPI.

Figura 6: Ganancia de velocidad de cómputo.



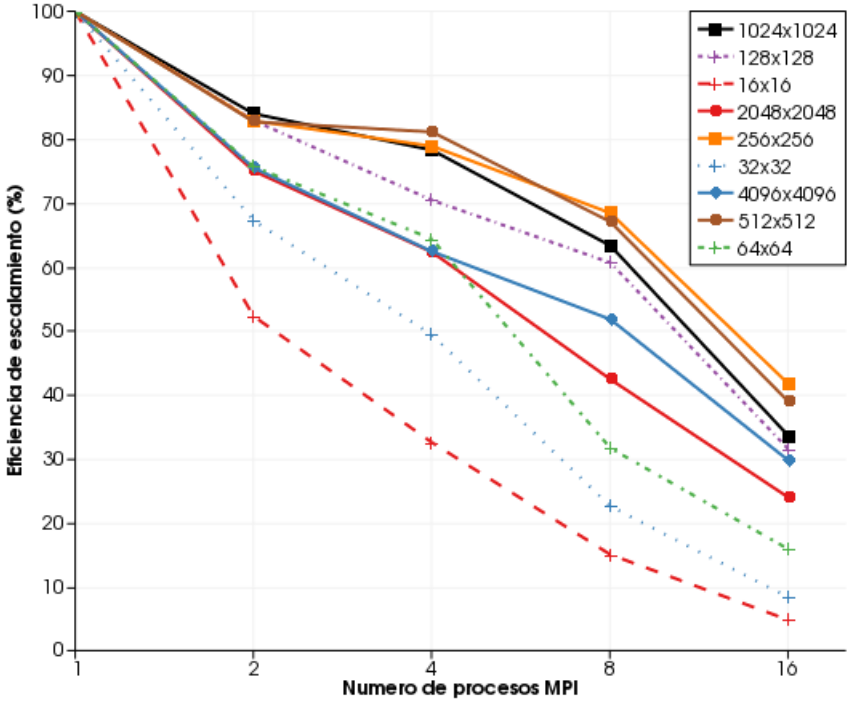
Un concepto ampliamente usado en la evaluación del rendimiento de un programa en paralelo es la escalabilidad que es una noción de como cambia la eficiencia respecto al número de procesos y tamaño del problema.

Se dice que un programa escala de forma fuerte si, dejando fijo el tamaño del problema y se aumenta el número de procesos, la eficiencia permanece invariable. Si se incrementa el número de procesos y el tamaño del problema en la misma proporción y la eficiencia permanece constante, se dice que el programa escala de forma débil (Pacheco, 2011).

En los resultados obtenidos no se observa que el programa escale de forma fuerte o débil. Sin embargo, es posible que se desarrolle una tendencia hacia alguno de los tipos de escalamiento a números de procesos mayores a 16 que fue el máximo que se pudo obtener en el momento de las pruebas.

El hecho de no presentar ninguna tendencia respecto a los tipos de escalamiento no implica que el programa sea incorrecto ya que la transformada rápida de Fourier es un algoritmo que requiere alto flujo de datos que produce altos costos de las comunicación y su implementación en sistemas de memoria distribuida sea un gran reto computacional. En las pruebas realizadas se obtienen tiempos de cómputo hasta

Figura 7: Eficiencia del programa para diferentes tamaños de problema.



6.7 veces menor que el caso secuencial, que es una ganancia satisfactoria.

5. CONCLUSIONES

Se estableció un modelo matemático para describir el flujo turbulento homogéneo incompresible en un dominio bidimensional cuadrado, discretizado en espacio de Fourier con condiciones de frontera periódicas, encontrando la solución temporal mediante un método Runge-Kutta hibrido (implícito/explícito) de tercer orden, que fue implementado utilizando código en paralelo, encontrando resultados de acuerdo a lo esperado, con un comportamiento físico satisfactorio bajo las condiciones formuladas.

En el método pseudo-espectral implementado, el error de solapamiento de frecuencias produce un comportamiento no físico en el sistema, por lo que es indispensable el uso de un método para eliminar sus efectos. Se implementó el método de desplazamiento de fase en dos dimensiones para la eliminación del solapamiento de frecuencias a partir de su formulación original en tres dimensiones. La formulación e implementación específica para el problema bidimensional, tal como se presenta en este trabajo de grado, no se encontró en la literatura consultada.

Mediante la implementación del método de solución en MPI, además del uso de la librería de alto rendimiento FFTW y la ejecución en la supercomputadora GUANE-1 de la plataforma GridUIS-2, se logró una reducción significativa del tiempo de cálculo, obteniendo una velocidad de cómputo alrededor de 14 veces mayor respecto a la ejecución en un computador portátil con procesador Intel Core i5 430m. Aunque se obtuvieron eficiencias de escalamiento por debajo de lo esperado, esta gran reducción de tiempo justifica la paralelización.

6. OBSERVACIONES Y RECOMENDACIONES

El presente trabajo de grado puede ser usado como punto de partida para el desarrollo de problemas más avanzados como la simulación numérica directa de turbulencia homogénea en tres dimensiones, la simulación de turbulencia en dos dimensiones con un dominio periódico en una dimensión y no periódico en la otra o la aplicación de los métodos numéricos espectrales a la solución de flujos no turbulentos. También puede ser usado como punto de comparación en la implementación de otros métodos de paralelización como OpenMP o CUDA.

El cluster GUANE-1 dispone de un gran poder de cómputo enfocado a implementaciones en CUDA, para lo cual, en gran cantidad de implementaciones es suficiente contar con un solo nodo de cómputo. En el momento de las pruebas realizadas presentó limitaciones en la velocidad de transferencia de datos entre nodos, por lo tanto, no fue posible poner a prueba el algoritmo en un sistema real de memoria distribuida y solo se obtuvieron resultados utilizando un solo nodo de cómputo.

7. BIBLIOGRAFÍA

Barney, B. (2012). *Message passing interface (MPI).* En *Lawrence Livermore National Laboratory*. Consultado Abril 30, 2013, en, https://computing.llnl.gov/tutorials/mpi/.

Batchelor, G. (1953). *The theory of homogeneous turbulence*. London, England: Cambridge University Press.

Canuto, C., Hussaini, M. Y., Quarteroni, A., y Zang, T. A. (2006). *Spectral methods, fundamentals in single domains* (1nd ed.). Berlin, Alemania: Springer.

Canuto, C., Hussaini, M. Y., Quarteroni, A., y Zang, T. A. (2007). *Spectral methods, evolution to complex geometries and applications to fluid dynamics* (1nd ed.). Berlin, Alemania: Springer.

Frigo, M., y Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, *93*(2), 216–231. (Special issue on "Program Generation, Optimization, and Platform Adaptation")

Frigo, M., y Johnson, S. G. (2012). FFTW [Manual de software informático]. Massachusetts Institute of Technology. Disponible en http://www.fftw.org/

Kernighan, B., y Ritchie, D. (1988). *The c programming language*. Finansy i statistika.

Ladyzhenskaya, O. A. (1969). *The mathematical theory of viscous incompressible flow* (Vol. 76). New York, USA: Gordon and Breach.

Mathieu, J., y Scott, J. (2000). *An introduction to turbulent flow*. Cambridge, Reino Unido: Cambridge University Press.

Moin, P., y Mahesh, K. (1998). Direct Numerical Simulation: A Tool in Turbulence Research. *Annual Review of Fluid Mechanics*, *30*, 539–578.

Pacheco, P. (2011). *An introduction to parallel programming*. Burlington, USA: Elsevier Science. Disponible en http://books.google.com.co/books?id=SEmfraJjvfwC

Patterson, G. S., y Orszag, S. A. (1971). Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions. *Physics of Fluids*, *14*, 2538-2541.

Pope, S. (2000). *Turbulent flows*. Cambridge, Reino Unido: Cambridge University Press.

Spalart, P. R., Mosser, R. D., y Rogers, M. M. (1991). Spectral methods for the navier-stokes equations with one finite and two periodic directions. *Journal of Computational Physics*, *96*, 297-324.

Taylor, G. I., y Green, A. E. (1937). Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *158*(895), pp. 499-521. Disponible en http://www.jstor.org/stable/96892

Yin, Z. (2003). *On statistical-mechanical descriptions of decaying two-dimensional turbulence*. Eindhoven, Paises Bajos: Technische Universiteit Eindhoven.