UNIVERSIDAD DEl BIO BIO
FACULTAD DE CIENCIAS EMPRESARIALES
MAGISTER EN CIENCIAS DE LA COMPUTACIÓN

ESTUDIO DE TÉCNICAS DE PROGRAMACIÓN EN PARALELO BASADO EN MPI PARA APROXIMAR Y SIMULAR EL MODELO LWR DEL FLUJO DE TRÁFICO VEHICULAR

TESIS PARA OPTAR AL GRADO DE MAGISTER EN CIENCIAS DE LA COMPUTACIÓN

LIC. LENIN QUIÑONES HUATANGARI

PROFESORES GUÍA:
MG. LUIS DANIEL GAJARDO DÍAZ
DR. ANÍBAL CORONEL PÉREZ
DR. PEDRO ÁNGEL RODRÍGUEZ MORENO

MIEMBROS DE LA COMISIÓN:
MG. MARÍA ANTONIETA SOTO CHICO
DR. PATRICIO ALEJANDRO GALDAMES SEPÚLVEDA
DR. DINO ENZO RISSO ROCCO

CHILLAN
JUNIO 2013

Resumen

En esta tesis se estudia el problema de la solución numérica de leyes de conservación escalares, que modelan el flujo vehicular en autopistas, aplicando los principios de computación paralela y volumenes finitos. Se presentan seis algoritmos secuenciales para resolver el problema de cauchy bajo distintos tipos de flujo y basados en los denominados esquemas upwind, Godunov e híbrido. Luego, se introduce la paralelización de cada uno de estos seis algoritmos. En el caso de los algoritmos correspondientes a los esquemas upwind y Godunov la paralelización se realiza siguiendo la técnica de la descomposición de dominios. En tanto que para el esquema híbrido el principio básico es la descomposición del operador de diferencias finitas asociado, en tres operadores que evolucionan independientemente y de los cuales dos de ellos realizan el transporte de la solución y uno de ellos modela la evolución de la discontinuidad. Los algoritmos paralelos basados en descomposición de dominios resultados ser mejores en tiempos de ejecución y los basados en la descomposición de operadores en aproximación de las discontinuidades. Para validar los algoritmos paralelizados se realizaron pruebas numéricas para las ecuaciones de advección, Burger y el modelo de tráfico vehicular de Lighthill-Whitham-Richards, miediéndose en cada uno de estos cados con las métricas: tiempo de ejecución, speedup y eficiencia. Los resultados para los tiempos de ejecución son decrecientes cuando se utiliza tres procesadores y muestran un comportamiento no monótono para más procesadores, debido al incremento del paso de mensajes. En la mayor parte de los experimentos numéricos el speedup está en el intervalo $[1, 2]$. La eficiencia de los algoritmos paralelizados varía entre el 49 % y el 92 %.

A todos mis familiares que me han apoyado durante todo este tiempo al margen de las limitaciones y las diversas circunstancias adversas que se han ido presentando en el camino. Gracias por la paciencia y siempre confiar en mí.

Agradecimientos

Agradezco a todos los integrantes del Magister en Ciencias de la Commputación - Chillán, tanto a alumnos, profesores y de manera especial a los asesores Luis Gajardo, Aníbal Coronel y Pedro Rodríguez, por su tiempo y apoyo académico oportuno.

Índice general

Índice de tablas

Índice de figuras

Lista de algoritmos

Introducción General

El modelamiento matemático de fenómenos naturales es una técnica frecuentemente utilizada en la ciencia y tecnología moderna. El avance de esta área está fuertemente vinculada al desarrollo de las ciencias computacionales. A modo general, se puede establecer que los modelos matemáticos existentes en la actualidad establecen relaciones funcionales no lineales entre las variables cuantitativas y, en consecuencia, muy pocas veces pueden ser resueltos de manera analítica. En este punto, existe la imperiosa necesidad de desarrollar algoritmos que finalmente se traduzcan en códigos computacionales. Es así que la ayuda de los computadores, para simular estos modelos, ha traído consigo un gran desarrollo científico y tecnológico.

Existen diversas investigaciones sobre el problema de la gestión y el control del tráfico vehicular basado en Sistemas de Transporte Inteligente (STI), (ver [2, 27, 28, 40, 43, 44]). A pesar de estos avances, aún existen problemas de distinta naturaleza que deben ser analizados. Entre estos problemas está por ejemplo el diseño de modelos generales que incluyan el comportamiento de los peatones y de los conductores, la simulación en tiempo real del tráfico vehicular y el desarrollo de tecnología en automóviles con decisiones autónomas tales como lo que suceda actualmente en el transporte aéreo. Es así que, el interés en este trabajo es investigar un caso muy específico, el cual principalmente tiene como objetivo proporcionar una herramienta para simular un modelo de un flujo continuo para el tráfico vehicular en las grandes avenidas de las urbes densamente pobladas. Es conocido que para abordar este problema existen dos grandes teorías, una tipo microscópico y otra de tipo macroscópico, cuya diferencia radica, principalmente, en que la primera considera un modelo discreto (vehículo por vehículo) y la otra considera la distribución de densidad de manera continua sobre la ruta. En este trabajo, el interés es considerar modelos continuos, dado que han demostrado ventajas muy claras, frente a los otros modelos, para mejorar la gestión de la red vial a través del uso de las tecnologías STI. Uno de los primeros estudios de la teoría del flujo de tráfico vehicular es el introducido por Greenshields [29], que establece como modelo una ley de conservación escalar con una relación lineal entre la velocidad y la densidad. Otras dos contribuciones iniciales relevantes, en este sentido, son los trabajos desarrollado por Lighthill-Whitham [30] y Richards [37]. Actualmente, estos modelos iniciales tienen mejoras substantivas, pero en esencia conservan el planteamiento matemático inicial, es decir, el modelo es una ley de conservación escalar o un sistema de leyes de conservación.

Es conocido que las leyes de conservación escalares no son solamente interesantes para el modelamiento de tráfico vehicular, si no que juegan un rol importante distitas aplicaciones físicas. En términos generales se puede establecer que aquellos fenómenos donde existe un comportamiento análogo al transporte de un fluido sin pérdida de masa, son de manera

natural modelados por leyes de conservación. En tal sentido, las aplicaciones más relevantes actualmente conocidas de esta teoría son el flujo sanguíneo, la separación de tipo sólido-fluido por fuerzas mecánicas, el flujo de contaminantes en un río, el flujo en acueductos y oleoductos, etc, ver [42]. Todas estas aplicaciones han motivado, entre otras, dos grandes áreas de la ciencia: (a) La Teoría de Leyes de Conservación, la que establece los principios matemáticos rigurosos para el buen planteamiento de los problemas y (b) La Teoría de Volúmenes Finitos, es un área que desarrolla algoritmos numéricos para las simulaciones computacionales, fundamentalmente de las leyes de conservación. En cuanto al punto (a), actualmente existe una teoría que se puede considerar como cerrada para el caso escalar. Sin embargo, en lo referente al punto (b), a pesar que se ha experimentado un significativo progreso en décadas recientes, con la introducción de métodos numéricos cada vez más robustos, es un área con muchos vacíos. En general, los esquemas conocidas necesitan tiempos de ejecución muy cercanos a los tiempos que toma realizar los experimentos físicos. Es así que existe una necesidad de investigar sobre la construcción de algoritmos que sean capaces de realizar las simulaciones en tiempo óptimo y a largo plazo, posiblemente, en tiempo real.

En esta tesis la meta fue estudiar el desarrollo de algoritmos basados en computación paralela para resolver y simular la solución de una Ley de Conservación Escalar. En particular, el estudio se enfocó en desarrollar tal teoría para la Ecuación de la Advección, la Ecuación de Burger y, finalmente, para el modelo de tráfico vehicular de tipo Lighthill-Whitham-Richards. Las mejoras significativas obtenidas son fundamentalmente dos: (i) los tiempos de ejecución de los métodos de volúmenes finitos paralelizados son mejores que los actualmente utilizados para resolver este problema de manera secuencial y (ii) se obtiene una mejor resolución para la aproximación de las discontinuidades que aparecen en las ondas de choque.

La tesis es organizada como sigue. En el capítulo 1 se presenta la problemática y los aspectos de diseño teórico tales como el problema y los objetivos. En el capítulo 2 se exponen las teorias que sustentan el trabajo, esto es la teoría de leyes de conservación, el método de volúmenes finitos y la computación paralela. Así mismo se exponen los algoritmos secuenciales existentes y los algoritmos propuestos en la tesis para resolver el problema utilizado paralelización en leyes de conservación. En el capítulo 3 se presentan los principales resultados. Finalmente, en el capítulo 4 se hace una síntesis de las conclusiones y perspectivas de la tesis.

Capítulo 1

Problemática y diseño teórico

En este capítulo se presenta la problemática que originó la presente tesis. Así mismo se establece el diseño teórico que incluye las preguntas de investigación, la presentación formal del problema y los objetivos de la investigación.

1.1. Justificación

El tráfico vehicular en las autopistas interurbanas y en las grandes avenidas de las urbes densamente pobladas se ha convertido en una situación problemática que está constantemente motivando cambios culturales, políticos y económicos. Entre estos cambios están por ejemplo la restricciones para utilizar los automóviles por determindos dias de la semana, la modificación de los horarios habituales de viaje entre el hogar y el trabajo o estudio y la generación de proyectos de ley relacionadas con la contaminación del aire y la contaminación acústica. Esta importancia social convierte, de manera natural, al tráfico vehicular es un problema que merece una atención especial desde el punto de vista científico.

El hecho que el tráfico vehicular en una autopista se comporte como un sistema análogo al flujo de un fluido en una tubería sugirió la idea de modelar este fenómeno utilizando las leyes de la Mecánica Clásica. En particular y a sugerencia de Lighthill, Whitham y Richards (ver [30, 37]) se utilizó el principio de conservación de la masa para deducir un modelo matemático basado en ecuaciones diferenciales parciales. Este modelo ideal sentó las bases de una línea de investigación que ha generado aportes a muchas áreas de la Ciencia e Ingeniería. Además, es conocido que este enfoque aún posee muchos problemas abiertos, para mayores detalles consultar el artículo de revisión del estado del arte sobre tráfico vehicular recientemente publicado por B. Piccoli y A. Tosin [36].

La simulación numérica del modelo de tráfico vehicular utilizando el modelo de Lighthill, Whitham y Richards requiere de una técnica conocida como "Método de Volúmenes Finitos", la cual reemplaza la ecuación diferencial por una ecuación en diferencias que satisface propiedades de consistencia, estabilidad y conservatividad discreta. Las propiedades de los métodos de volúmenes finitos implican convergencia de la solución numérica hacia la solución

continua dada por el modelo. Ahora, la condición de estabilidad, especialmente en el caso de los métodos explícitos implica una restricción en la partición utilizada para discretizar la ecuación. Dicha restricción establece, de manera genérica, que la cantidad de iteraciones en la variable que modela el tiempo es de un orden mayor que la cantidad de nodos utilizados para discretizar la variable espacial. Además, las iteraciones temporales incrementan cuanto más grande sea el intervalo de tiempo que se desea simular. Esta dificultad se agudiza cuando los modelos son sistemas de leyes de conservación y con mayor razón al incrementar la dimensión.

Existen al menos dos posibles vías de solución que aparecen naturalmente, en el análisis numérico y la teoría informática, para simular computacionalmente el fenómeno de tráfico vehicular en un tiempo considerablemente inferior a lo que podría hacerse mediante una simulación física del fenómeno. La primera de estas alternativas es estudiar la posible paralelización de los esquemas de volúmenes finitos existentes. En tanto que la segunda es explorar la posibilidad de resolver numéricamente la ecuación mediante otras técnicas numéricas tales como la de Elementos Finitos. La línea de investigación que se sigue en esta tesis es la primera. Además, se conoce que recientemente algunos grupos de investigación están desarrollando la segunda de estas alternativas, aunque no precisamente focalizados en la aplicación de tráfico vehicular, son un referente válido, dado que sus modelos son leyes de conservación, para mayores detalles consultar [12].

Por otro lado, centrar el objetivo en desarrollar una teoría de algoritmos paralelos para leyes de conservación escalares unidimensionales se justifica desde el punto de vista que tanto los sistemas como los problemas multimensionales se pueden resolver vía generalizaciones directas de esquemas escalares unidimensionales. Para el caso de sistema de leyes de conservación se puede hacer utilizando la técnica de componente por componente (Ver [33]) y para los problemas multidimensionales utilizando la descomposición dimensional (Ver [34]). En otras palabras es claro que el desarrollo de algoritmos escalares unidimensionales efiecientes se puede generalizar a sistemas multidimensionales, donde la paralelización es realmente necesaria.

1.2. Preguntas de investigación

Las preguntas fundamentales que originaron la investigación son las siguientes:

- Dadas las dificultades técnicas generales que existen para generalizar algoritmos paralelos diseñados en ecuaciones no evolutivas ¿Existe alguna posibilidad de adaptar consistentemente las ideas de los algoritmos paralelos ampliamente utilizados ecuaciones elípticas (o parabólicas) a las leyes de conservación?
- El comportamiento no lineal de las leyes de conservación que modelan el tráfico vehicular implican, en particular, que no se puede asegurar de antemano el comportamiento de la evolución del esquema, surge una interrogante natural con respecto a los algoritmos paralelos que se logren diseñar ¿Serán capaces de aproximar las interacciones de onda? y en particular ¿Aproximarán con igual o mejor calidad las ondas de choque o discontinuidades?

- En el caso de los métodos de descomposición de dominios el aspecto esencial que se explota es el hecho que los sistemas lineales asociados se resuelven utilizando una descomposición por bloques de matrices asociadas. Esto implica la siguiente inquietud ¿ Es posible desarrollar una idea análoga de este hecho para el método de volúmenes finitos aprovechando el hecho que la fórmulas iterativas o también denominados esquemas se pueden escribir equivalentemente en término de operadores?

1.3. Planteamiento del problema

En esta sección se define de manera genérica el problema que se abordó en la presente tesis. Para esto se parte definiendo de manera precisa y con la notación adecuada lo que se denomina una ley de conservación escalar (Ver [29]):

$$\left\{\begin{array}{l} \text{Dadas las funciones reales } u_0, f : \mathbb{R} \to \mathbb{R}, \\ \text{encontrar } u : \mathbb{R} \times \mathbb{R}_0^+ \to \mathbb{R} \text{ que sea solución de} \\ u(x,t)_t + (f(u(x,t)))_x = 0, \;\; (x,t) \in \mathbb{R} \times \mathbb{R}_0^+, \\ u(x,0) = u_0(x), \;\; x \in \mathbb{R}. \end{array}\right. \tag{1.1}$$

El análisis matemático básico para resolver (1.1) se basa en el método de las características, ver [9, 29]. Utilizando este método se puede obtener las soluciones analíticas en el caso que la función de flujo f sea lineal y algunos casos especiales cuando f es no lineal. En consecuencia, cuando (1.1) representa un modelo físico de utilidad, para resolverlo se recurre a los métodos numéricos. Es así, que el método más natural y más ampliamente utilizado es el de volúmenes finitos, el cual aproxima coherentemente todas las interacciones de onda posibles y fundamentalmente es conservativo. Un esquema de volúmenes finitos explícito que aproxima numéricamente el problema (1.1) se representa de manera genérica por

$$\left\{\begin{array}{l} \text{Dados } \{k, N\} \subset \mathbb{N}, \{\Delta x, \Delta t, t_f, \lambda\} \subset \mathbb{R}^+ \text{ y } g : \mathbb{R}^{2k} \to \mathbb{R} \text{ tal que} \\ \;\; t_f = N\Delta t, \;\; \lambda = \Delta t/\Delta x, \;\; g(u, \ldots, u) = f(u) \;\; \text{para todo } u \in \mathbb{R} \text{ y} \\ \;\; g^n_{j-1/2} = g(u^n_{j-k+1}, \ldots, u^n_{j+k}) \approx \int_{t_n}^{t_{n+1}} f(u((j-1/2)\Delta x, t))\mathrm{d}t. \\ \text{Para } n \in \{0, \ldots, N\}, \text{ calcular iterativamente } u_j^{n+1}, \text{ aproximación} \\ \text{de } u, \text{ solución de (1.1) en } (j\Delta x, (n+1)\Delta t), \text{ vía la relación:} \\ u_j^{n+1} = u_j^n - \lambda\Big(g^n_{j+1/2} - g^n_{j-1/2}\Big), \;\; j \in \mathbb{Z}, \\ \text{partiendo de } u_j^0 = \frac{1}{\Delta x}\displaystyle\int_{(j-1/2)\Delta x}^{(j+1/2)\Delta x} u_0(x)\mathrm{d}x, \;\; j \in \mathbb{Z}. \end{array}\right. \tag{1.2}$$

La función g se denomina flujo numérico y la propiedad $g(u, \ldots, u) = f(u)$ para todo $u \in \mathbb{R}$ se denomina consistencia.

El esquema explícito (1.2) se puede expresar en términos de operadores. En efecto, denotando por $\mathbf{u}^n$ el estado aproximado en tiempo $n\Delta t$, es decir $\mathbf{u}^n = (\ldots, u^n_{-1}, u^n_0, u^n_1, \ldots)$, y por ℓ^∞ el espacio de todas las sucesiones acotadas, se define el operador $H : \ell^\infty \to \ell^\infty$ tal que

$$[H(\mathbf{u}^n)]_j = u_j^n - \lambda\Big(g^n_{j+1/2} - g^n_{j-1/2}\Big), \;\; j \in \mathbb{Z}.$$

Luego, en esta nueva notación, el esquema (1.2) se escribe como

$$\begin{cases} \text{Dado } \mathbf{u}^0, \text{ para } n \in \{0, \ldots, N\}, \text{ calcular iterativamente } \mathbf{u}^n \\ \text{mediante la relación } \mathbf{u}^{n+1} = H(\mathbf{u}^n). \end{cases} \tag{1.3}$$

La formulación (1.3) del método de volúmenes finitos es útil para estudiar propiedades numéricas tales como estabilidad, convergencia y orden de aproximación. En el caso de esta tesis permitirá introducir la idea central de paralelización.

Por requerimientos de estabilidad y convergencia de la aproximación (1.2) hacia las soluciones débiles de (1.1) se necesita que tanto Δx y Δt cumplan con la siguiente condición CFL[1]:

$$\frac{\Delta t}{\Delta x} \max_u |f'(u)| < 1.$$

La condición CFL limita el tiempo de ejecución, dado que implica que la cantidad de pasos de evolución se incrementa significativamente al aumentar en pocas unidades la cantidad de pasos espaciales. Una de las alternativas de solución para esta dificultad es considerar esquemas implícitos, los cuales generalmente convergen sin la condición CFL o con una condición CFL mucho menos restrictiva, ver [4]. Sin embargo, otra alternativa posible, aún muy débilmente investigada es paralelizar los esquemas de volúmenes finitos, ver por ejemplo [5, 22, 23, 26]. Es conocido que los esquemas implícitos pierden la aproximación de las discontinuidades debido a que desarrollan lo que es conocido como difusión numérica. Es así que la investigación de la paralelización de los esquemas explicitos ayudarían a simular el problema sin incorporar difusión, lo cual es esperable en muchos problemas físicos donde se desea una buena aproximación de las discontinuidades. En general, el problema de paralelización de los esquemas de volúmenes finitos explícitos (1.2), utilizando la notación (1.3), se define de la siguiente manera:

> *¿Es posible encontrar una descomposición del operador H en términos de una cantidad finita de operadores $\mathcal{H}^{\mathrm{i}} : A_{\mathrm{i}} \to B_{\mathrm{i}}$ for $\mathrm{i} \in \{1, \ldots, Pr\}$ con $A_{\mathrm{i}}, B_{\mathrm{i}} \subset \ell^\infty$ tal que $\bigcup_{\mathrm{i}=1}^{Pr} A_{\mathrm{i}} = \ell^\infty$ y $H(\mathbf{u}^n)$ se puede reconstruir en términos $\mathcal{H}^{\mathrm{i}}(\mathbf{u}^n)$?*

En el caso de la descomposición de dominios en ecuaciones elípticas, los conjuntos A_{i} son los subdominios y los operadores $\mathcal{H}^{\mathrm{i}}$ son los operadores para las discretizaciones sobre cada uno de estos dominios. Esto no es directamente generalizable al caso de leyes de conservación y para el caso lineal ha sido recientemente generalizado en [12].

[1]Esta abreviatura es debida a Courant, Frederich y Levy, quienes lo introdujeron por primera vez en [8]

1.4. Objetivos

En base al problema presntado en la sección 1.3, en este trabajo se define como objetivo general y objetivos específicos, los que se describen en las siguientes subsecciones.

1.4.1. Objetivo general

Construir y analizar algoritmos basados en computación paralela para aproximar Leyes de Conservación Escalar que modelan el flujo vehicular en autopistas.

1.4.2. Objetivos específicos

(O_1) Construir un algoritmo paralelo basado en MPI para la ecuación de la advección con coeficientes constantes.

(O_2) Extender el algoritmo anterior para los casos en que los coeficientes son variables.

(O_3) Construir un algoritmo paralelos para la ecuación de Burger y basado en el método numérico de Volúmenes Finitos.

(O_4) Aplicar el algoritmo diseñado para la Ecuación de Burger en (O_3) al caso del modelo LWR.

(O_5) Analizar la complejidad computacional de los algoritmos propuestos.

Capítulo 2

Marco teórico

En este capítulo se presentan los principales aspectos teóricos en los cules se fundamenta la solución al problema propuesto. El capítulo parte presentando el modelo de tráfico vehicular en autopistas según Lighthill, Whitham y Richards [30, 37] y continua con la descripción de algunos conceptos de Computación Paralela. Luego, se presentan propiedades fundamentales de la teoría de leyes de conservación. A continuación de esto se presentan los algoritmos secuenciales de volúmenes finitos. Se finaliza el capítulo presentando los algoritmos paralelos propuestos en la tesis.

2.1. Modelo de tráfico vehicular de Lighthill-Whitham-Richards (LWR)

En esta sección se presentan los argumentos generales que permiten formular el modelo matemático continuo propuesto por Lighthill - Whitham y Richards, el cual en muchas partes de la tesis es referenciado como el modelo LWR.

En modelos matemáticos continuos, el tráfico de vehículos se considera como un flujo de fluidos compresibles, es decir, los autos se asumen como las partículas de líquido, para los cuales se pueden escribir el equilibrio adecuado o las leyes de conservación. Tales modelos macroscópicos son a menudo llamados modelos hidrodinámicos o modelos de onda cinemática. Las principales variables dependientes utilizadas para describir matemáticamente la dinámica del tráfico en los tramos de las carreteras son la densidad de autos $\rho = \rho(x,t)$ y la velocidad media $v = v(x,t)$, los cuales toman en cuenta la ubicación x y el tiempo t. A partir de estas cantidades, se deriva otro parámetro importante del tráfico: el caudal o flujo del tráfico $q = q(x,t)$ dada por ρv, que es de gran interés tanto para fines teóricos como experimentales. De hecho, muchos de los datos reales utilizados para construir y validar modelos macroscópicos se refieren precisamente a las mediciones del caudal, debido a que estas se pueden realizar con mayor precisión que las medidas de las otras variables macroscópicas (es decir, la densidad y velocidad). El flujo o caudal del tráfico, también conocido como el volumen de tráfico, indica el número de vehículos que pasan en un lugar determinado de una

carretera durante un intervalo de tiempo específico. Los movimientos de vehículos en una red de carreteras son considerados como combinaciones de las ondas cinemáticas en cualquiera de las cantidades antes mencionadas. Tal como se ha señalado anteriormente, los modelos matemáticos continuos se basan en el principio de conservación de los vehículos. En otras palabras, la variación temporal en la cantidad de vehículos dentro de cualquier tramo de carretera, comprendido entre dos puntos x_1 y x_2, donde $x_1 < x_2$, sólo se debe a la diferencia entre el flujo entrante de cambio en x_1 y el flujo saliente en x_2, es decir:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{x_1}^{x_2}\rho(x,t)\mathrm{d}x = \rho(x_1,t)v(x_1,t) - \rho(x_2,t)v(x_2,t). \tag{2.1}$$

Ahora, suponiendo que ρ y v son funciones suficiente regulares, en el sentido que satisfacen las hipótesis del Teorema Fundamental del Cálculo, entonces las siguientes identidades son válidas

$$\begin{aligned}\rho(x_1,t)v(x_1,t) - \rho(x_2,t)v(x_2,t) &= \int_{x_2}^{x_1}(\rho v)_x(x,t)\mathrm{d}x \quad \text{y}\\ \frac{\mathrm{d}}{\mathrm{d}t}\int_{x_1}^{x_2}\rho(x,t)\mathrm{d}x &= \int_{x_1}^{x_2}\rho_t(x,t)\mathrm{d}x.\end{aligned}$$

Note que las derivadas en las funciones integrando de los segundos miembros son derivadas parciales con respecto a la variable espacial y temporal, respectivamente. Luego, se deduce que

$$\begin{aligned}\int_{x_1}^{x_2}\rho_t(x,t)\mathrm{d}x &= \rho(x_1,t)v(x_1,t) - \rho(x_2,t)v(x_2,t)\\ &= \int_{x_2}^{x_1}(\rho v)_x(x,t)\mathrm{d}x\\ &= -\int_{x_1}^{x_2}(\rho v)_x(x,t)\mathrm{d}x\end{aligned}$$

es decir

$$\int_{x_1}^{x_2}\Big(\rho_t + (\rho v)_x\Big)(x,t)\mathrm{d}x = 0.$$

Luego, debido a la arbitrariedad de x_1 y x_2, por el teorema de localización[1], se deduce que

$$\rho_t + (\rho v)_x = 0, \quad \text{para} \quad (x,t) \in \mathbb{R}\times\mathbb{R}^+. \tag{2.2}$$

Sin embargo, (2.2) no genera por sí misma una ecuación consistente, ya que implica encontrar simultáneamente dos variables, ρ y v, de una sola ecuación. Hay dos alternativas, para superar esta dificultad. La primera es considerar hipótesis adicionales y formular una ecuación para el momento lineal y así tener un sistema. Una segunda posibilidad es diseñar las relaciones adecuados de cierre que expresen la velocidad v, o equivalentemente el flujo f, en función de la densidad. En la segunda opción, la ecuación (2.2) se convierte en un modelo basado en una ecuación difrencial parcial de evolución en términos de la densidad. Este tipo de ecuación diferencial es llamada ley de conservación escalar y se escribe más precisamente como sigue:

$$\rho_t + f(\rho)_x = 0, \quad f(\rho) = \rho v(\rho). \tag{2.3}$$

[1] **Teorema**[13]. Sea f una función integrable. Si $\int_\Omega f(x)\mathrm{d}x = 0$ para cada $\Omega \subset \mathbb{R}$, entonces $f = 0$ sobre Ω.

Recordar que ρ modela la densidad, v la velocidad, f es la denominada función de flujo y tanto x como t denotan la posición espacial y el tiempo, respectivamente.

Las ecuaciones diferenciales parciales de la forma (2.3) generalmente tienen soluciones discontinuas, lo cual en el tráfico vehicular significa un cambio inesperado en la densidad vehicular, tal como lo que sucede con el cambio de luz verde a luz roja en un semáforo, ver [9, 29]. Este fenómeno no ocurre muy a menudo en autopistas interregionales de alto tráfico vehicular, dado que usualmente se evita utilizar semáforos. Sin embargo, para modelar este hecho en la teoría de modelos macroscópicos se incorpora en la ecuación un término de segundo orden que limita estos cambios bruscos de la densidad al incluir aspectos que imitan el efecto de la viscosidad en los problemas de mecánicas de fluidos, para mayores detalles consultar por ejemplo [16].

Modelo	$\mathbf{v}(\rho)$
Greenshield	$v_{max}(1-\rho/\rho_{max})$
Greenberg	$v_{max}\ln(\rho_{max}/\rho)$
Underwod	$v_{max}\exp(-\rho/\rho_{max})$
Northwestern	$v_{max}\exp(-0{,}5\rho^2/\rho_0^2)$
Drew	$v_{max}(1-(\rho/\rho_{max})^{(n+1)/2})$
Leveque	$\begin{cases} 1, & \text{si } \rho \in]0, 0.1], \\ 10\rho, & \text{si } \rho \in]0.1, 0.3], \\ 4.92(1-\rho), & \text{si } \rho \in]0.3, 1]. \end{cases}$

Tabla 2.1: Modelos de velocidad construidos bajo las hipótesis (H_1)-(H_3) del modelo LWR. Note que el modelo de Leveque no satisface (H_1), pero si es físicamente posible y matemáticamente coherente en el sentido de la la teoría presentada por ejemplo en [15].

Basándose en el principio de conservación de masa, que fue descrito anteriormente, y focalizándose en las características dinámicas del tráfico vehicular, Lighthill conjuntamente con Whitham en [30] e independientemente Richards en [37] propusieron un modelo macroscópico del tráfico vehicular en autopistas, el cual ahora es conocido, en la literatura especializada en el tema, como el modelo LWR. En este modelo se considera, idealmente una autopista infinita donde ρ es la densidad del tráfico vehicular medida en autos por unidad de área. El modelo consiste en representar el problema de transporte en una ley de conservación escalar [6, 16] de la forma (2.3), la cual se obtiene bajo la hipótesis razonable de suponer que la velocidad $v(\rho)$ es una función decreciente de la densidad que comienza con un valor ρ_{max} en $\rho = 0$ y disminuye hasta cero cuándo ρ alcanza el valor máximo ρ_{max}, en el sentido de que no hay espacios entre los vehículos. Más precisamente, las hipótesis matemáticas del modelo LWR son las siguientes:

(H_1) f es una función de clase $C^2([0, \rho_{\text{máx}}])$, es decir tanto la primera como la segunda derivada de f son funciones continuas en el intervalo $[0, \rho_{\text{máx}}]$.

(H_2) f es una función estrictamente cóncava en el intervalo $]0, \rho_{\text{máx}}[$, es decir $f''(\rho) < 0$ para todo $\rho \in]0, \rho_{\text{máx}}[$.

(H_3) $f(0) = f(\rho_{\text{máx}}) = 0$.

Notando, en (2.3) que $f(\rho) = \rho v(\rho)$, en la práctica en los modelos se proponen expresiones

para $v(\rho)$. Ahora, en el contexto de las hipótesis (H_1)-(H_3) hay varias propuestas para la función $v(\rho)$, como se puede observar en los primeros cinco modelos dados en la Tabla 2.1. En tanto que, en el último modelo no se cumple la hipótesis (H_1). Sin embargo, actualmente se conoce que esta hipótesis es una condición útil para el buen planteamiento matemático del problema de valores iniciales asociado a (2.3) y estudiado vía la teoría clásica de leyes de conservación introducida por Kružkov [24]. Además, se conoce que esta condición puede ser generalizada para incluir modelos donde la función de flujo es una función continua tal como es el caso del último modelo de la Tabla 2.1 e inclusive a modelos donde el flujo es discontinuo, ver [15].

2.2. Conceptos de computación paralela

En esta sección se describe los conceptos generales de la Teoría de Computación Paralela que son utilizados en la tesis. Algunos de los aspectos que no se consideren en esta sección serán presentados oportunamente a medida que se avance en el informe.

2.2.1. Herramientas de hardware y software de entornos paralelos

La implementación secuencial y paralela de los algoritmos para el estudio y observación experimental de los métodos que resuelven las Leyes de Conservación, se ha llevado a cabo dentro del entorno de programación que se describe en esta sección.

En general, los problemas que requieren un alto grado de cómputo para su resolución, como sucede en este caso, necesitan de potentes entornos de programación que garanticen una realización eficiente. Ahora, es conocido también que los entornos de programación secuencial pueden ofrecer estas garantías, siempre que se cuente con los recursos suficientes para acceder a computadoras de alto desempeño y con los avances tecnológicos de vanguardia. Sin embargo, en la programación secuencial, siempre se encontrará una limitación física que impida mayor beneficio. Es así que aparece, el entorno de programación alternativo que integra computadoras de bajo costo unidos en una red de procesadores. Con este tipo de entornos, se logra soluciones eficientes sin el problema de las limitaciones físicas ni de los altos costos de un supercomputador paralelo, al tener la posibilidad de aumentar el número de procesadores que se integran en la denominada red de procesadores.

Para las simulaciones numéricas realizadas en este trabajo, se han utilizado esencialmente multiprocesadores de memoria distribuida, cuyos componentes son computadores personales o estaciones de trabajo que funciona bajo el esquema de paso de mensaje. Esta arquitectura ha permitido diseñar algoritmos donde el control de la distribución y manipulación de los datos es administrada por el programador, utilizando mecanismos de paso de mensajes como las ofrecidas por la herramienta MPI. Una breve descripción técnica del cluster utilizado es la siguiente:

> El cluster será citado como "*Cluster Ciencias Básicas.*" Este cluster de memoria distribuida está constituido por 3 nodos físicos: 1 frontend y 2 nodos de cómputo. El equipo

que hace de frontend tiene las siguientes características: 2 CPU físicas con 2 núcleos cada uno del tipo Intel(R) Xeon(R) CPU X3430 2.40GHz con 4 GB de RAM y 2 discos duros SAS de 300 GB. Cada uno de los nodos de cómputo está conformado por: 2 CPU físicas de 4 núcleos del tipo Intel(R) Xeon(R) CPU X5365 @ 3.00GHz con 8 GB de RAM y discos duros (SAS) de 600 GB.

Es necesario notar que para el tipo de hardware utilizado, existe software adecuado que explota las características de la arquitectura y permite obtener el máximo rendimiento. Uno de estos es el lenguaje de programación C y en particular la biblioteca de paso de mensajes MPI. La abreviatura MPI es debido a su denominación en inglés "Message Passing Interface" y es una biblioteca de comunicaciones que implementa el mecanismo de paso de mensajes entre nodos de cómputo que se comunican mediante una red de interconexiones. Las funciones y ejemplos de la biblioteca MPI que se emplearon han resultado de las sugerencias dadas en [1, 38].

2.2.2. Metodología de Foster

La mayoría de los problemas de programación tiene varias soluciones paralelas. La mejor solución puede ser diferente a la que sugieren los algoritmos secuenciales. Las metodologías de diseño que se describen tienen por objeto promover un enfoque exploratorio del diseño en los algoritmos paralelos. Una de estas metodologías es la denominada metodología de Foster. Esta metodología del proceso de diseño consta de cuatro fases distintas: la partición, la comunicación, la aglomeración y el mapeo. En las dos primeras etapas, el foco está en la concurrencia y la escalabilidad, tratando de descubrir los algoritmos con estas cualidades. En la tercera y cuarta etapa, la atención se desplaza a la localidad y otros aspectos relacionados con el rendimiento. La Figura 2.1 ilustra gráficamente las cuatro etapas, cuyo resumen, descrito en [14], se comenta a continuación:

(i) **Particionamiento**. Se descompone el cálculo y los datos se distribuyen en pequeñas tareas. Cuestiones prácticas tales como el número de procesadores en el equipo de destino se ignora y se coloca más atención en el reconocimiento de las oportunidades de ejecución en paralelo.

(ii) **Comunicación**. Se determina la comunicación necesaria para coordinar la ejecución de las tareas, además de definir algoritmos y estructuras adecuadas de comunicación.

(iii) **Aglomeración**. Las estructuras de comunicación y el trabajo respectivo se definen en las dos primeras etapas, en esta se evalúa los requisitos de rendimiento y los costos de implementación. Si es necesario, las tareas se combinan en grandes tareas para mejorar el rendimiento o para reducir los costes de implementación.

(iv) **Mapeo**. Cada tarea se asigna a un procesador de manera de satisfacer las metas, maximizando la utilización del procesador y minimizando los costos de comunicación. El mapeo se puede especificar de forma estática o dinámica en tiempo de ejecución con algoritmos de balanceo de carga.

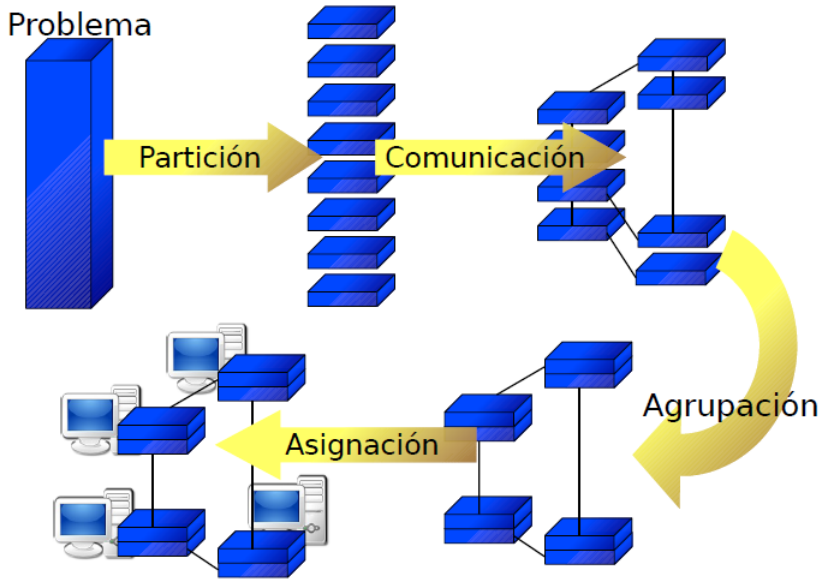

Figura 2.1: Diseño de Foster de cuatro etapas para un Algoritmo Paralelo

2.2.3. Descomposición de dominios

El problema de paralelismo de los métodos numéricos utilizados para resolver ecuaciones diferenciales parciales está vinculado a una teoría denominada: Método de Descomposición de Dominios. En líneas generales los algoritmos, en el marco de esta teoría, tienen tres grandes etapas:

(a) Definen una partición del dominio global donde se necesita resolver la ecuación diferencial. La partición debe ser tal que la ecuación se puede resolver independientemente sobre cada uno de estos subdominios.

(b) Resolver en paralelo la ecuación diferencial sobre cada subdominio.

(c) Ensamblar las soluciones locales con el objetivo de construir la solución global.

Para detalles específicos, del método, consultar [39, 41]. Esta idea está indirectamente vinculada al problema de paralelización de la multiplicación de matrices y de la solución de sistemas lineales. En este sentido su aplicación a algoritmos basados en Elementos Finitos ha sido exitosa y de alguna manera bastante natural, dado que en la mayor parte de los casos la aproximación de ecuaciones diferenciales parciales con esta metodología conduce a la solución de sistemas lineales, ver [39]. En contraste con el método de Elementos Finitos, el método de Volúmenes Finitos no necesariamente requiere resolver un sistema lineal. Sin embargo, la evolución temporal de los esquemas de Volúmenes Finitos utilizan operadores en dimensión finita, es decir, las fórmulas iterativas se pueden expresar como multiplicación matriz-vector. En consecuencia, es natural esperar que la paralelización de los esquemas de Volúmenes Finitos superen una de las grandes dificultades de los esquemas explícitos, los cuales por razones de estabilidad y convergencia necesitan pasos de tiempo demasiado pequeños para alcanzar alta resolución y alto orden de aproximación. Específicamente, en el caso de leyes de conservación algunos avances en este sentido fueron dados por el trabajo de Perkins-Rodrigue [35]. La desventaja del método introducido en [35] es que se agrega un

término de difusión, lo cual lo convierte en un método que tiene desventajas para aproximar las discontinuidades de la solución.

2.2.4. Evaluación de algoritmos paralelos

El objetivo del paralelismo es aumentar el rendimiento de los programas, esto depende de las capacidades del sistema y del comportamiento del programa. Sin embargo, se pueden establecer criterios que son unificados y no sensibles a los sistemas, estos indicadores se denominan métricas. Las métricas que se utilizan típicamente, para la evaluación de los algoritmos paralelos diseñados para un problema de tamaño m y disponiendo de una cantidad de P_{rc} procesadores son [25]:

- **Tiempo de Ejecución**. Se denomina de esta manera a la función definida como

$$T(m, P_{rc}) = T_A(m, P_{rc})t_f + T_C(m, P_{rc}),$$

  donde T_A es la función que mide el tiempo empleado por el algoritmo para realizar operaciones aritméticas, t_f es el tiempo de ejecución de una operación en punto flotante y T_C es la función que mide el tiempo empleado por el algoritmo para realizar comunicaciones entre procesadores. En el modelo de paso de mensajes se conoce que [11]:

$$T_C(m, P_{rc}) = t_m + mt_v,$$

  donde t_m es el tiempo de establecimiento de comunicación entre los nodos (latencia) y t_v es el tiempo de transferencia de un mensaje de longitud 1 (inverso del ancho de banda).
- **Aceleración de Ejecución (speed-up)**. Esta métrica mide la ganancia de velocidad de ejecución del algoritmo paralelo con respecto al mejor algoritmo secuencial que resuelve el mismo problema y se define como el siguiente cociente

$$S(m, P_{rc}) = \frac{T(m, 1)}{T(m, P_{rc})}.$$

  Idealmente se espera que el speed-up sea igual al número de procesadores empleados [25].
- **Eficiencia**. Es una métrica que mide el porcentaje de tiempo de ejecución en el cual los procesadores se mantienen útilmente empleados y se calcula mediante la siguiente expresión

$$E(m, P_{rc}) = \frac{S(m, P_{rc})}{P_{rc}}.$$

  El valor ideal que puede alcanzar la eficiencia es igual a 1, ver [25].

Es conveniente notar que en la práctica, para evaluar un algoritmo mediante las métricas definidas anteriormente, se necesita estimar el valor de los parámetros t_f, t_m y t_v.

2.2.5. Complejidad computacional

La complejidad computacional estudia la eficiencia de los algoritmos estableciendo su efectividad de acuerdo al tiempo de ejecución y al espacio requerido en la computadora para el almacenamiento de los datos, ayudando a evaluar la viabilidad de la implementación práctica en tiempo y costo.

Notación asintótica

Las notaciones para el tiempo de ejecución asintótico de un algoritmo se definen en términos de funciones cuyo dominio es el conjunto de los números naturales. Las notación asintótica se utiliza para describir la función del peor caso del tiempo de ejecución, $T(n)$, cuando n es creciente y toma valores grandes, para mayores detalles consultar [10].

(i) Notación $O(g(n))$: Representa una cota asintótica superior, tal como se aprecia en la Figura 2.2, en la cual f está acotada superiormente por cg a partir de $n = n_0$. Sea una función $g(n)$, se denota por $O(g(n))$ al conjunto de funciones:

$$O(g(n)) = \Big\{ f(n) : \exists \ c, \ \ n_0 \in \mathbb{N} \text{ tal que } 0 \leq f(n) \leq cg(n), \ \ \forall n \geq n_0 \Big\}.$$

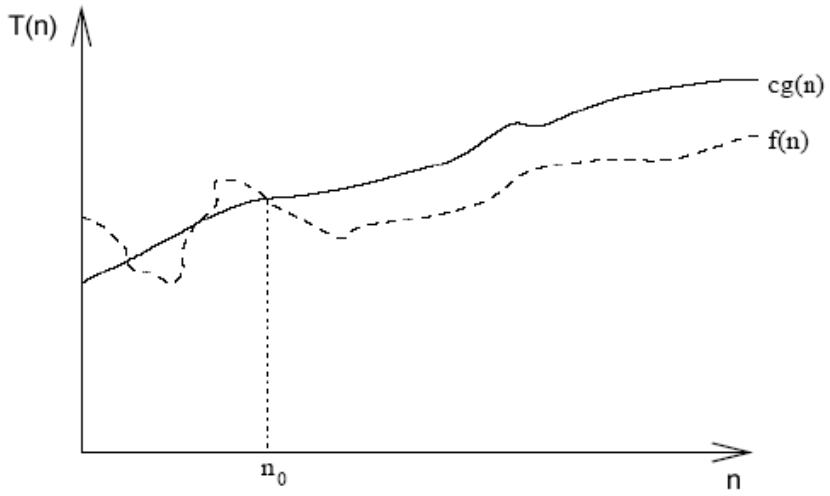

Figura 2.2: $O(g(n))$, representa una cota asintótica superior

(ii) Notación Ω: Representa una cota asintótica inferior, tal como se aprecia en la Figura 2.3, en la cual f está acotada superiormente por cg a partir de $n = n_0$. Sea una función $g(n)$, se denota por $\Omega(g(n))$ al conjunto de funciones:

$$\Omega(g(n)) = \Big\{ f(n) : \exists \ c, \ \ n_0 \in \mathbb{N} \text{ tal que } 0 \leq cg(n) \leq f(n), \ \ \forall n \geq n_0 \Big\}.$$

(iii) Notación Θ: Representa la existencia de cota superior e inferior, tal como se aprecia en la Figura 2.4, en la cual f está acotada superiormente por $c_2 g$ e inferiormente por

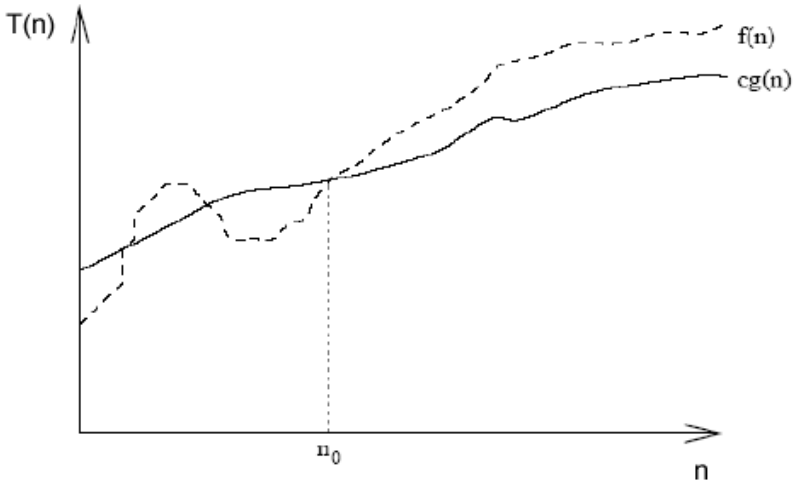

Figura 2.3: $\Omega g(n)$, representa una cota asintótica inferior

$c_1 g$, a partir de $n = n_0$. Dada una función $g(n)$ se denota por $\Theta(g(n))$ al conjunto de funciones:

$$\Theta(g(n)) = \Big\{ f(n) : \exists \ c_1, \ \ c_2, \ \ n_0 \in \mathbb{N} \text{ tal que } c_1 g(n) \leq f(n) \leq c_2 g(n), \ \ \forall n \geq n_0 \Big\}.$$

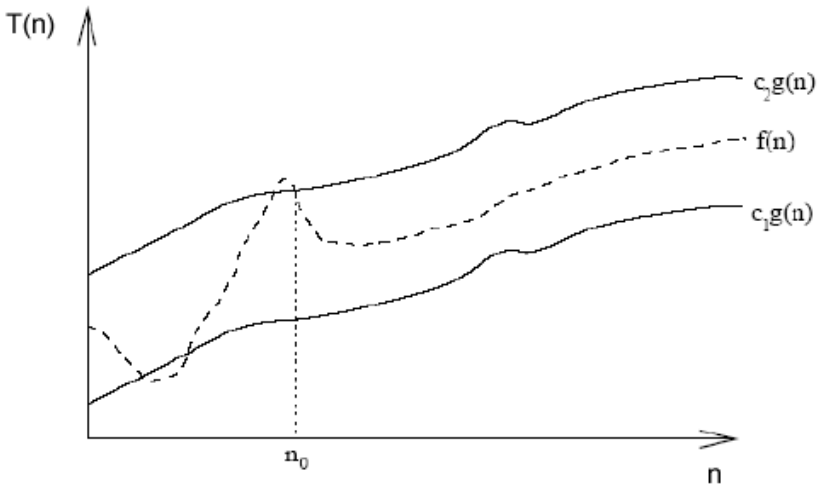

Figura 2.4: $\Theta(g(n))$, representa la existencia de cota superior e inferior

Clasificación

La teoría de la complejidad computacional trata de clasificar los problemas que pueden, o no pueden ser resueltos con una cantidad determinada de recursos, según [10] esta clasificación corresponde a:

(i) **Problemas Indecibles**. Son problemas para los cuales no se puede escribir un algoritmo para su solución, por lo tanto, son los problemas de complejidad más alta.

(ii) **Problemas Intratables**. Son los problemas para los cuales no se puede desarrollar un algoritmo de tiempo polinomial, únicamente algoritmos exponenciales.

(iii) **Problemas NP (Polinomial no deterministico)**. Son los problemas para los cuales la factibilidad del problema utilizando el correspondiente problema de decisión, puede ser verificada en tiempo polinomial. Sin embargo, el problema solo puede resolverse con algoritmos no deterministicos.

(vi) **Problemas P**. Un problema se dice que es polinomial o que está en clase P si existe un algoritmo de tiempo polinomial para su solución.

2.3. Teoría de leyes de conservación

El modelo (2.3) es un caso especial de lo que en la teoría matemática se denomina ley de conservación escalar. Más precisamente, a continuación se recuerda los conceptos y terminología esencial en la teoría de leyes de conservación. Se recuerda que una ley de conservación se define de manera genérica como la ecuación diferencial parcial

$$\mathbb{U}_t + \sum_{\mathrm{i}=1}^{\mathrm{d}} [\mathbb{F}(\mathbb{U})]_{x_\mathrm{i}} = 0, \quad \mathbb{U} = \mathbb{U}(\mathbf{x}, t) \in \mathbb{R}^m, \quad \mathbb{F} : \mathbb{R}^m \to \mathbb{R}^m, \quad \text{y} \quad (\mathbf{x}, t) \in \mathbb{R}^{\mathrm{d}} \times \mathbb{R}_0^+. \quad (2.4)$$

En la ecuación (2.4) se utiliza la siguiente términología dependiendo de d y m:

- Si $m = \mathrm{d} = 1$, la ecuación (2.4) se llama ecuación escalar unidimensional,
- Si $m = 1$ y $\mathrm{d} > 1$, la ecuación (2.4) se llama ecuación escalar multidimensional,
- Si $m > 1$ y $\mathrm{d} = 1$, la ecuación (2.4) se llama sistema unidimensional y
- Si $m > 1$ y $\mathrm{d} > 1$, la ecuación (2.4) se llama sistema multidimensional.

En síntesis el caso escalar o sistema depende del valor de m y el caso unidimensional o multidimensional del valor de d.

Para resolver la ecuación diferencial parcial (2.4) es necesario considerar condiciones iniciales y en algunos casos condiciones de frontera, dependiendo de la información física disponible para el problema en específico. En el caso del modelo (2.4), se considera que la autopista es infinita, es decir, no se tiene condiciones de frontera y tan sólo se tiene condiciones iniciales con las cuales hay que resolver el problema. Es así que en el trabajo, el problema con condiciones iniciales o problema de Cauchy que se estudiará es el siguiente

$$u_t + f(u)_x = 0, \qquad (x, t) \in \mathbb{R} \times \mathbb{R}^+, \quad (2.5)$$

$$u(x, u) = u_0(x), \qquad x \in \mathbb{R}. \quad (2.6)$$

En algunos casos es posible encontrar la solución analítica de (2.5)-(2.6) pero por lo general esto no es posible y sobre todo cuando f es no lineal.

En el problema (2.5)-(2.6), el caso más simple de resolver es cuando f es lineal de la forma $f(u) = au$ con a constante, tal ecuación es denominada **ecuación de advección con**

coeficientes constantes y su solución se puede obtener analíticamente por el método de las características y está dada por $u(x,t) = u_0(x - at)$. Luego, en dificultad creciente, se encuentra el caso de la **ecuación de advección con coeficientes variables**, donde el flujo es de la forma $f(u) = a(x,t)u$, para el cual, en general, no existe soluciones analíticas. Las dos ecuaciones anteriores son lineales y su análisis se basa en el método de las características, ver [9, 13, 29]. Ahora, al aplicar este mismo método en el caso de las ecuaciones no lineales se descubre que tan sólo puede ser utilizado de manera local. El prototipo de las ecuaciones no lineles es la denominada **ecuación de Burger**, donde $f(u) = u^2/2$, introducida en [21] y largamente estudiada. La contribución del estudio de esta ecuación a la teoría de leyes de conservación es la formalización de los conceptos de la dinámica de gases denominados ondas de choque y rarefacción. De manera genérica, las ondas de choque son soluciones discontinuas y las ondas de rarefacción son soluciones continuas pero no diferenciables, para mayor detalle consultar [29]. En el caso de la ecuación de Burger, en general, no existen soluciones analíticas, salvo para el caso de que la condición inicial es un salto o lo que se denomina **problema de Riemann**. Para fijar ideas, si u_0 es de la siguiente forma

$$u_0(x) = \begin{cases} u_L, & x \leq 0, \\ u_R, & x > 0, \end{cases} \quad \text{con} \quad (u_L, u_R) \in \mathbb{R}^2, \tag{2.7}$$

entonces las solución analítica es

$$u(x,t) = \begin{cases} \begin{cases} u_L, & x \leq st, \\ u_R, & x > st, \end{cases} & s = \dfrac{u_L + u_R}{2}, \qquad \text{si } u_L > u_R, \\ \\ \begin{cases} u_L, & x < u_L\, t, \\ x/t, & u_L\, t \leq x \leq u_R\, t, \\ u_R, & x > u_R\, t, \end{cases} & \qquad\qquad\qquad\qquad \text{si } u_L \leq u_R. \end{cases} \tag{2.8}$$

La primera posibilidad de estas soluciones, es decir si $u_L > u_R$, se llama onda de choque y la segunda, es decir si $u_L \leq u_R$, se llama onda de rarefacción. La teoría desarrollada para la ecuación de Burger se puede extender al caso de leyes de conservación con flujo convexo, es decir cuando $f''(u) > 0$ para $u \in \mathbb{R}$. En el caso del modelo LWR, por la hipótesis (H_2), se sabe que la función f es cóncava. En este caso, la solución del problema de Riemann, también se puede obtener adaptando el caso convexo y más precisamente, la solución de (2.3)-(2.7) es

$$u(x,t) = \begin{cases} \begin{cases} u_L, & x \leq st, \\ u_R, & x > st, \end{cases} & s = \dfrac{f(u_L) - f(u_R)}{u_L - u_R}, \qquad \text{si } u_L < u_R, \\ \\ \begin{cases} u_L, & x < f'(u_L)\, t, \\ (f')^{-1}(x/t), & f'(u_L)\, t \leq x \leq f'(u_R)\, t, \\ u_R, & x > f'(u_R)\, t, \end{cases} & \qquad\qquad\qquad\qquad \text{si } u_L \geq u_R. \end{cases} \tag{2.9}$$

Esta solución analítica de (2.3)-(2.7), en la práctica, es muchas veces limitada por el hecho que no necesariamente es posible encontrar $(f')^{-1}$ de manera explícita.

Para finalizar, esta sección se debe mencionar que las soluciones discontinuas tienen sentido matemático al estudiarlo desde el punto de vista de las denominadas soluciones entrópicas introducida por Kružkov [24]. Esta teoría es cerrada para el caso escalar (unidimensional y multidimensional) pero aún no existe una teoría para el caso de sistemas.

2.4. Métodos numéricos y algoritmos secuenciales

En este capítulo se estudian los métodos numéricos upwind y Godunov correspondientes a los algoritmos secuenciales que dan solución a las siguientes leyes de conservación: la ecuación de la advección constante, la ecuación de la advección Variable, la ecuación de Burger y el modelo de transporte LWR. Todos los algoritmos son de naturaleza iterativa y calculan en cada iteración una aproximación a la solución $u(x,t)$, partiendo de una condición inicial $u_0(x)$.

2.4.1. Métodos numéricos

Existen varios métodos numéricos que nos permiten simular computacionalmente las soluciones de leyes de Conservación, consultar [17, 20, 29]. La bibliografía acerca de la aproximación numérica es muy extensa, ya que se trata de un tema de gran importancia en problemas de Ingeniería. En esta sección, se focaliza en tres métodos que se denominan upwind, Godunov e híbrido, dado que serán utilizados posteriormente en las aplicaciones y simulaciones realizadas en la tesis.

Nociones y generalidades de volúmenes finitos

Para discretizar una ecuación diferencial se requieren dos etapas: (i) la discretización del dominio computacional y (ii) la discretización de las ecuaciones. En esta tesis, para la etapa (i) se elige una discretización homogénea y para la etapa (ii) el método de volúmenes finitos. Con mayor precisión

(i) **Discretización del dominio.** Para fijar ideas, se supone que el dominio computacional está dado por el conjunto $Q_T = [L_0, L_1] \times [T_0, T_1]$. Se considera $M \in \mathbb{N}$, $CFL \in]0,1[$ y M_f el máximo de las velocidades de onda, es decir

$$M_f = \max_{u\in[u_m,u_M]} |f'(u)| \quad \text{con} \quad u_m = \min_{x\in\mathbb{R}} u_0(x), \quad u_M = \max_{x\in\mathbb{R}} u_0(x),$$

se define

$$\Delta x = \frac{L_1 - L_0}{M}, \quad N = \frac{M_f(T_1 - T_0)}{\Delta x\, CFL}, \quad \Delta t = \frac{T_1 - T_0}{N} \quad \text{y} \quad \lambda = \frac{\Delta t}{\Delta x}. \tag{2.10}$$

Luego, la discretización de Q_T es dada por

$$Q_T = \bigcup_{j=0}^{M} \bigcup_{n=0}^{N} Q_j^n \quad \text{con } Q_j^n = [x_{j-1/2}, x_{j+1/2}[\times[t_n, t_{n+1}[, \tag{2.11}$$

donde

$$x_{j+1/2} = L_0 + (j+1/2)\Delta x \quad \text{y} \quad t_n = n\Delta t. \tag{2.12}$$

Los intervalos $[x_{j-1/2}, x_{j+1/2}]$ se llaman volúmenes de control y a los intervalos $[t_n, t_{n+1}]$ se llama n-ésimo intervalo de evolución.

(ii) **Discretización de la ecuación.** Para discretizar la ecuación (2.5)-(2.6), mediante volúmenes finitos se integra (2.5) sobre Q_j^n y se obtiene

$$\int_{x_{j-1/2}}^{x_{j+1/2}} u(x,\tau)\Big|_{\tau=t_n}^{\tau=t_{n+1}} \mathrm{d}x + \int_{t_n}^{t_{n+1}} f(u(\xi,t))\Big|_{\xi=x_{j-1/2}}^{\xi=x_{j+1/2}} \mathrm{d}t = 0. \tag{2.13}$$

Luego, introduciendo la notación

$$u_j^n = \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} u(x,t_n)\mathrm{d}x \quad \text{y} \quad g_{j+1/2}^n = \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f(u(x_{j+1/2},t))\mathrm{d}t,$$

se deduce que (2.13) es dada por

$$u_j^{n+1} = u_j^n - \lambda(g_{j+1/2}^n - g_{j-1/2}^n). \tag{2.14}$$

El esquema (2.14) parte de la condición inicial discretizada de la siguiente manera

$$u_j^0 = \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} u_0(x)\mathrm{d}x,$$

donde u_0 es definida por (2.6).

La expresión (2.14) es convergente asumiendo ciertas condiciones para el flujo numérico, para mayores detalles consultar [29].

Esquemas upwind

Para ilustrar el método, se considera la ecuación de onda en una dimensión.

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0.$$

Tal como se señalo en la sección 2.3, esta ecuación describe una onda que se propaga en la dirección x con una velocidad a y además es conocido que es un modelo matemático unidimensional de advección lineal. Considere un punto de la cuadrícula típica j en el dominio, ver Figura 2.5. En un dominio unidimensional, sólo hay dos direcciones asociadas con el punto i: izquierda y derecha. Si a es positiva el lado izquierdo se llama lado del viento y el lado derecho es el lado de sotavento. Del mismo modo, si a es negativo el lado izquierdo se llama lado de sotavento y el lado derecho es el lado del viento. Si el esquema de diferencias finitas para las derivadas espaciales contiene más puntos en el lado del viento, el esquema que se llama un sesgo contra el viento o, simplemente, un sistema de dirección del viento. El esquema de volúmenes finitos más simple posible es el esquema upwind de primer orden, el cual está dada por

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0, \quad \text{para} \quad a > 0, \tag{2.15}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0, \quad \text{para} \quad a < 0. \tag{2.16}$$

Ahora, introduciendo la notación

$$a^+ = \text{máx}\{a, 0\} \quad a^- = \text{mín}\{a, 0\} \quad u_x^- = \frac{u_i^n - u_{i-1}^n}{\Delta x} \quad u_x^+ = \frac{u_{i+1}^n - u_i^n}{\Delta x}, \tag{2.17}$$

es posible unificar las ecuaciones (2.16) y (2.15) en la forma compacta dada por:

$$u_i^{n+1} = u_i^n - \Delta t[a^+ u_x^- + a^- u_x^+]. \tag{2.18}$$

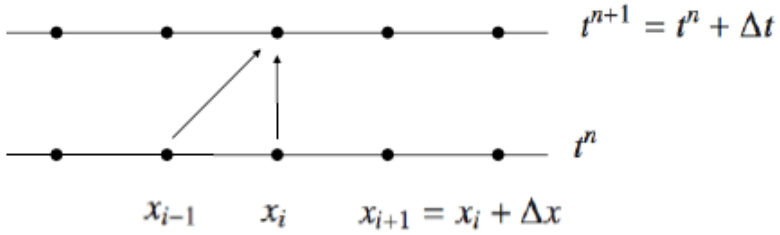

Figura 2.5: Participación de los puntos en el Esquema upwind, para $a > 0$.

Esquema de Godunov

Como es conocido, cuándo f es no lineal, las soluciones de la ecuación diferencial (1.1) pueden tener discontinuidades por más regular que sea la condición inicial. Para resolver esta ecuación es necesario apelar a métodos numéricos, pues salvo casos especiales, no existen soluciones explícitas. Entre los métodos numéricos usualmente empleados, interesan en particular aquellos que refleja con exactitud la conservación de la densidad de la masa (o de la que la cantidad conservada que se modele) por un lado, y que detecten con la mayor precisión posible la ubicación de las ondas de choque en cada instante de discretización. Entre estos métodos, el método de Godunov, es de lo más difundidos. Una descripción del método de Godunov y sus generalizaciones pueden verse en [29, 19].

El método de Godunov es un método de volúmenes finitos explícito que generaliza el método upwind para flujo lineal al caso de flujo no lineal. Recordar que u_i^n denota el promedio de u sobre el intervalo $]x_{i-1/2}\ ,\ x_{i+1/2}[$ en el instante $n\Delta t$. Dados los valores u_i^n en el instante conocido $n\Delta t$, los valores en el instante siguiente $(n+1)\Delta t$ se obtienen mediante un procedimiento en dos etapas:

(I) **Resolver problemas de Riemann Locales**. En esta primera etapa se obtienen los valores provisorios $\bar{u}_{i+1/2}^n$ como soluciones en el instante Δt del problema de Riemann con condición inicial $u = u_i^n$, para $x < i\Delta x$, $u = u_{i+1}^n$ para $x > i\Delta x$.

(II) **Proyectar la solución**. En esta segunda etapa se obtienen los valores definidos de u_i^{n+1} como soluciones de la ecuación en diferencias

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(f(\bar{u}_{i-1/2}^n) - f(\bar{u}_{i+1/2}^n)). \tag{2.19}$$

Conviene hacer notar que el cálculo de $\bar{u}_{i+1/2}^n$ es obtenido mediante la solución analítica del problema de Riemann, lo cual se hace tal como fue indicado al final de la sección 2.3.

Método híbrido

Para evitar la alta difusión numérica introducida, a lo largo de las discontinuidades, en la discretización usual por volúmenes finitos, se representa la solución u por medio de tres funciones del modo siguiente: u se divide en dos estados continuos, u^1 y u^2, los cuales se pegan justamente en las discontinuidades de u, por medio de la función conjunto de nivel ψ, de forma que la solución u en un punto dado es igual a u^1 o bien a u^2 dependiendo del signo de la función conjunto de nivel ψ en dicho punto, o sea:

$$u(x,t) = \begin{cases} u^1(x,t), & \text{si } \psi(x,t) > 0, \\ u^2(x,t), & \text{si } \psi(x,t) \leq 0. \end{cases} \tag{2.20}$$

Para más detalles técnicos del método ver [7].

Desde el punto de vista algorítmico, el método híbrido consiste de tres etapas. Para fijar ideas, si se considera $a > 0$ se tiene que estas etapas son

(i) **Inicialización.** Considerar ψ_i^0 tal que $\psi_{\bar{i}}^0 = 0$ para todos los $\bar{i}$ tal que u_0 es dicontinua sobre el volumen de control $[x_{\bar{i}-1/2}, x_{\bar{i}+1/2}]$. Además, se considera la descomposición inicial de la forma (2.20), es decir se elige $u1_i^0$ y $u2_i^0$ tal que

$$u_i^0 = \begin{cases} u1_i^0, & \text{si } \psi_i^0 > 0, \\ u2_i^0, & \text{si } \psi_i^0 \leq 0. \end{cases}$$

La extensión de $u1_i^0$ a los nodos, donde $\psi_i^0 \leq 0$ se hace de manera arbitraria tratando de conservar la continuidad y similarmente $u2_i^0$ se extiende de manera continua a los nodos donde $\psi_i^0 > 0$.

(ii) **Evolución.** Para $n \in \{0, \ldots, N-1\}$ se calculan $u1_i^{n+1}, u2_i^{n+1}$ y ψ_i^{n+1}, mediante los esquemas

$$\begin{aligned} u1_i^{n+1} &= u1_i^n - \lambda a(u1_i^n - u1_{i-1}^n) \\ u2_i^{n+1} &= u2_i^n - \lambda a(u2_i^n - u2_{i-1}^n) \\ \psi_i^{n+1} &= \psi_i^n - \lambda a(\psi_i^n - \psi_{i-1}^n) \end{aligned}$$

(iii) **Reconstrucción final.** El perfil de solución se reconstruye de la siguiente manera

$$u_i^N = \begin{cases} u1_i^N, & \text{si } \psi_i^N > 0, \\ u2_i^N, & \text{si } \psi_i^N \leq 0. \end{cases}$$

La extensión de este algoritmo al caso $a < 0$ es directa y básicamente consiste en modificar los esquemas de volúmenes finitos de la etapa (ii). Sin embargo, la extensión no es directa para el caso no lineal, en el cual hay que considerar una etapa adicional de verificación de la condición de entropía, ver [7].

2.4.2. Algoritmos secuenciales para las leyes de conservación

En esta sección se presentan seis algoritmos secuenciales que fueron implementados en la tesis para aproximar el problema de Cauchy (2.5)-(2.6), mediante volúmenes finitos bajo

Algoritmo secuencial	Algoritmo paralelizado	Método numérico Método Numérico	flujo flujo
1	7	upwind	$f(u) = au$ con $a > 0$
2	7	upwind	$f(u) = au$ con $a \in \mathbb{R}$
3	8	híbrido	$f(u) = au$ con $a \in \mathbb{R}$
4	9	upwind	$f(u) = a(x,t)u$
5	10	Godunov	$f(u) = u^2/2$
6	11	Godunov	f de tipo LWR

Tabla 2.2: Leyes de conservación y su método numérico utilizado.

distintas consideraciones para la función de flujo. El primer de estos es el Algoritmo 1, el cual muestra el algoritmo para el esquema upwind cuando $a > 0$, es decir, este algoritmo resuelve el problema (2.5)-(2.6) cuando $f(u) = au$, denominada ecuación de advección con velocidad constante. El segundo algoritmo, denominado Algoritmo 2, muestra el algoritmo para el esquema upwind cuando a puede cambiar de signo, es decir, este algoritmo resuelve el problema (2.5)-(2.6) cuando $f(u) = au$ y a una cosntante que puede ser positiva o negativa. En tercer lugar se presenta el Algoritmo 3, el cual resuelve el problema (2.5)-(2.6) aplicando el método híbrido y el caso cuando $f(u) = au$ y a una constante que puede ser positiva o negativa. El cuarto de estos algoritmos es el denominado Algoritmo 4, el cual resuelve el problema (2.5)-(2.6) aplicando el método upwind y el caso que $f(u) = a(x,t)u$, es decir, la denominada ecuación de advección con velocidad variable. En quinto lugar se presenta el Algoritmo 5, que resuelve el problema (2.5)-(2.6) aplicando el método de Godunov y en el caso que $f(u) = u^2/2$, es decir, la denominada ecuación de Burger. Finalmente, se presenta el Algoritmo 6, que resuelve el problema (2.5)-(2.6) aplicando el método de Godunov y en el caso que cuando f sea el flujo del tráfico vehicular según el modelo LWR. Una síntesis de toda esta descripción de los algoritmos secuenciales y sus respectivos algoritmos paralelizados se encuentra en la Tabla 2.2. Los algoritmos paralelizados se presentarán en la sección 2.5.

Para finalizar la sección se hacen dos comentarios. En primer lugar, uno de los aportes importantes de esta tesis es el Algoritmo 3, que se basa en la implementación del método numérico propuesto en [7]. La idea central presentada en [7] consiste en asumir que un cambio de variable adecuado transforma el problema (2.5)-(2.6) en tres subproblemas totalmente desacoplados, generando de esta manera una forma natural de programar en paralelo. Además, dicho método numérico garantiza mejores resultados de aproximación de las discontinuidades en contraste con el método upwind, dado que este último presenta alta difusión numérica en las discontinuidades como se observará en las simulaciones. En segundo lugar, se comenta que la base para escribir los algoritmos de Godunov (Algoritmos 5 y 6) fueron los códigos MATLAB publicados en [18].

Algoritmo 1 Calcula la solución de (2.5)-(2.6) cuando $f(u) = au$ con $a > 0$ y utilizando el esquema upwind.

Entrada: $\{L_0, L_1, T_0, T_1\} \subset \mathbb{R}$, $CFL \in]0,1[$, $a \in \mathbb{R}^+$, $M \in \mathbb{N}$, $u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
Calcular $N = \dfrac{a(T_1 - T_0)}{\Delta x \; CFL}$
Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
para $j = 0$ **hasta** M **hacer**
    Calcular $u_j^0 = u_0(j\Delta x)$
fin para
para $n = 0$ **hasta** N **hacer**
    para $j = 1$ **hasta** M **hacer**
        Calcular $u_j^{n+1} = u_j^n - \lambda a(u_j^n - u_{j-1}^n)$
    fin para
    $u_0^{n+1} = u_1^{n+1}$
fin para

Algoritmo 2 Calcula la solución de (2.5)-(2.6) cuando $f(u) = au$ con $a \in \mathbb{R}$ y utilizando el esquema upwind.

Entrada: $\{L_0, L_1, T_0, T_1, a\} \subset \mathbb{R}$, $CFL \in]0,1[$, $M \in \mathbb{N}$, $u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
Calcular $N = \dfrac{|a|(T_1 - T_0)}{\Delta x \;\; CFL}$
Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
Calcular $a^+ = \max\{a, 0\}$
Calcular $a^- = \min\{a, 0\}$
para $j = 0$ **hasta** $M - 1$ **hacer**
    Calcular $u_j^0 = u_0(j\Delta x)$
fin para
para $n = 0$ **hasta** N **hacer**
    para $j = 1$ **hasta** M **hacer**
        Calcular $u_x^- = \dfrac{u_j^n - u_{j-1}^n}{\Delta x}$
        Calcular $u_x^+ = \dfrac{u_{j+1}^n - u_j^n}{\Delta x}$
        Calcular $u_j^{n+1} = u_j^n - \Delta t[a^+ u_x^- + a^- u_x^+]$
    fin para
    $u_0^{n+1} = u_1^{n+1}$
    $u_M^{n+1} = u_{M-1}^{n+1}$
fin para

Algoritmo 3 Calcula la solución de (2.5)-(2.6) cuando $f(u) = au$ con $a \in \mathbb{R}$ y utilizando el esquema híbrido.

Entrada: $\{L_0, L_1, T_0, T_1, a\} \subset \mathbb{R}$, $CFL \in]0,1[$, $M \in \mathbb{N}$, $u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Descomponer $u_0(x)$ en términos de $u_0^1(x)$, $u_0^2(x)$, $u_0^3(x)$, talque:
$u_0(x) = \begin{cases} u_0^3(x) & \text{si } u_0^1(x) \leq 0 \\ u_0^2(x) & \text{si } u_0^1(x) > 0 \end{cases}$
$u_0^3 = 0$ sobre las discontinuidades de u_0.
$u^1(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^1$
$u^2(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^2$
$u^3(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^3$
para $n = 0$ **hasta** N **hacer**
  para $j = 0$ **hasta** M **hacer**
    si $u^1(n\Delta t, j\Delta x) < 0$ **entonces**
      $u_j^n = u^3(n\Delta t, j\Delta x)$
    si no
      $u_j^n = u^2(n\Delta t, j\Delta x)$
    fin si
  fin para
fin para

Algoritmo 4 Calcula la solución de (2.5)-(2.6) cuando $f(u) = a(x,t)u$ y utilizando el esquema upwind.

Entrada: $\{L_0, L_1, T_0, T_1\} \subset \mathbb{R}$, $M \in \mathbb{N}$, $u_0 : \mathbb{R} \to \mathbb{R}$, $a : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$
Salida: $u(x,t)$

Calcular $\Delta x = \frac{L_1 - L_0}{M}$
Calcular $N = \frac{Max|a|(T_1 - T_0)}{\Delta x CFL}$
para $j = 0$ **hasta** M **hacer**
  Calcular $u_j^0 = u_0(j\Delta x)$
fin para
para $n = 0$ **hasta** N **hacer**
  para $j = 1$ **hasta** $M - 1$ **hacer**
    Calcular $u_x^- = \dfrac{u_j^n - u_{j-1}^n}{\Delta x}$
    Calcular $u_x^+ = \dfrac{u_{j+1}^n - u_j^n}{\Delta x}$
    $u_j^{n+1} = u_j^n - \Delta t[\text{máx}(a(j\Delta x), 0)u_x^- + \text{mín}(a((j+1)\Delta x), 0)u_x^+]$
  fin para
  $u_0^{n+1} = u_1^{n+1}$
  $u_M^{n+1} = u_{M-1}^{n+1}$
fin para

Algoritmo 5 Calcula la solución de (2.5)-(2.6) cuando $f(u) = u^2/2$ y utilizando el método de Godunov

Entrada: $\{L_0, L_1, T_0, T_1\} \subset \mathbb{R}$, $M \in \mathbb{N}$, $u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Calcular $\Delta x = \frac{L_1 - L_0}{M}$
Calcular $N = \frac{Max|u_0(x)|(T_1 - T_0)}{\Delta x CFL}$
Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
para $j = 0$ **hasta** M **hacer**
    Calcular $u_j^0 = u_0(j\Delta x)$
fin para
para $n = 0$ **hasta** N **hacer**
    para $j = 1$ **hasta** $M - 1$ **hacer**
        si $u_j^n > u_{j+1}^n$ **entonces**
            Calcular $\sigma = \dfrac{u_{j+1} + u_j}{2}$
            si $\sigma < 0$ **entonces**
                $f_j = 0{,}5(u_{j+1}^n)^2$
            fin si
        si no, **si** $u_j^n < u_{j+1}^n$ **entonces**
            si $u_{j+1}^n < 0$ **entonces**
                $f_j = 0{,}5(u_{j+1}^n)^2$
            si no, **si** $u_j^n > 0$ **entonces**
                $f_j = 0{,}5(u_j^n)^2$
            si no
                $f_j = 0$
            fin si
        fin si
        Calcular $u_j^n = u_j^n - \lambda(f_j - f_{j-1})$
    fin para
    $u_0^{n+1} = u_1^{n+1}$
    $u_M^{n+1} = u_{M-1}^{n+1}$
fin para

Algoritmo 6 Calcula la solución de (2.5)-(2.6) cuando f de tipo LWR y utilizando el método de Godunov

Entrada: $\{L_0, L_1, T_0, T_1, M\} \subset \mathbb{R}, \quad u_0 : \mathbb{R} \to \mathbb{R}, \quad f : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Calcular $\Delta x = \frac{L_1 - L_0}{M}$
Calcular $N = \frac{Max|f'(u)|(T_1 - T_0)}{\Delta x CFL}$
Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
para $j = 0$ **hasta** M **hacer**
    Calcular $u_j^0 = u_0(j\Delta x)$
fin para
para $n = 0$ **hasta** N **hacer**
    para $j = 1$ **hasta** $M - 1$ **hacer**
        si $u_j^n < u_{j+1}^n$ **entonces**
            Calcular $\sigma = \dfrac{f(u_{j+1}^n) - f(u_j^n)}{u_{j+1}^n - u_j^n}$
            si $\sigma < 0$ **entonces**
                $f_j = f(u_{j+1}^n)$
            fin si
        si no, **si** $u_j^n > u_{j+1}^n$ **entonces**
            si $u_{j+1}^n > 0$ **entonces**
                $f_j = f(u_{j+1}^n)$
            si no, **si** $u_j^n < 0$ **entonces**
                $f_j = f(u_j^n)$
            si no
                $f_j = 0$
            fin si
        fin si
        Calcular $u_j^n = u_j^n - \lambda(f_j - f_{j-1})$
    fin para
    $u_0^{n+1} = u_1^{n+1}$
    $u_M^{n+1} = u_{M-1}^{n+1}$
fin para

2.5. Algoritmos paralelos propuestos

En esta sección se presentan los algoritmos paralelos propuestos para los algoritmos secuenciales definidos en la sección 2.4. La paralelización de estos métodos se fundamenta en una adaptación de las ideas de descomposición de dominios, en el sentido que se distribuye el trabajo de tamaño M, que representa la cantidad de nodos espaciales, entre los procesadores disponibles, permitiendo de manera natural disminuir los costos de ejecución y almacenamiento. La paralelización fue realizada utilizando el entorno paralelo MPI, consistente en la distribución de trabajo para cada uno de los procesadores, ejecución de un mismo conjunto de

instrucciones sobre diferentes conjuntos de datos en las unidades de procesamiento y empleo del modelo de paso de mensajes entre las unidades de procesamiento, tal como se muestra a continuación:

(i) *Inicialización.* Se replican todos los datos iniciales en todos los procesadores.

(ii) *Solución local.* Cada procesador calcula el intervalo en que se debe desarrollar la ley de Conservación y calcula la solución aproximada en la iteración.

(iii) *Ensamble.* Envío de datos necesarios, mediante mensajes a los procesadores para el ensamble de la solución en esa iteración.

(iv) *Evolución.* Calcular la segunda solución aproximada y repetir el paso de mensajes, es decir, los pasos (ii) y (iii). Además del cálculo aproximado sucesivamente hasta la última iteración

(v) *Solución final.* En la iteración final, cada procesador distinto del procesador Maestro (0) envía el resultado al proceso Maestro y este los escribe en un archivo.

Los algoritmos paralelos propuestos trabajan con estructuras de datos cuya distribución sigue la convención establecida por el lenguaje de Programación C. Así, todas las estructuras de datos se han particionado en bloques de dimensión de $Subarreglo = (M+1)/P_{rc}$ y distribuido en una malla lógica de procesadores P_{rc} y un identificador para cada procesador id, para nuestro caso de 0 hasta $P_{rc}-1$. Para evitar muchas comunicaciones entre nodos, u_0 se ha replicado en cada procesador. Por lo tanto, en cada nodo de procesamiento se manejan matrices y vectores con los tamaños locales (resultado de dividir la dimensión global de la estructura por el número de procesadores), presentados en la Tabla 2.3.

Estructura	**Dimensión Global**	**Dimensión Local**
u_0	$M+1$	$(M+1)(P_{rc})^{-1}$
$uAprox$	$M+1$	$(M+1)(P_{rc})^{-1}$

Tabla 2.3: Dimensiones de las estructuras globales y locales de la paralelización.

Las operaciones de los algoritmos para Leyes de Conservación Escalar, pueden paralelizarse utilizando directamente rutinas de MPI, donde las rutinas de comunicaciones y redistribución de datos de MPI utilizadas se detallan en la Tabla 2.4.

Algoritmo 7 Paralelización del Algoritmo 2. Esta paralelización también es válida como paralelización del Algoritmo 1, basta considerar $a \in \mathbb{R}^+$ y actualizar el esquema

Entrada: $\{L_0, L_1, T_0, T_1, \mathrm{id}, a\} \subset \mathbb{R}$, $\{M, P_{rc}\} \subset \mathbb{N}$, $CFL \in]0,1[$, $u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

1: Replicar todos los datos en los P_{rc}
2: Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
3: Calcular $N = \dfrac{|a|(T_1 - T_0)}{\Delta x CFL}$
4: Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
5: Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
6: Calcular $a^+ = \max\{a, 0\}$
7: Calcular $a^- = \min\{a, 0\}$
8: Calcular el intervalo de trabajo.
9: Calcular $Subarreglo = \dfrac{M}{P_{rc}} + 1$
10: **para** $j = 0$ **hasta** $Subarreglo$ **hacer**
11: Calcular $u_j^0 = u_0(j\Delta x)$
12: **fin para**
13: **para** $n = 0$ **hasta** N **hacer**
14: Enviar datos al siguiente procesador.
15: **si** id $< P_{rc} - 1$ **entonces**
16: Enviar $u[Subarreglo - 1]$ a id $+ 1$
17: **fin si**
18: Recibiendo datos del anterior procesador.
19: **si** id > 0 **entonces**
20: Recibir $u[0]$ de id $- 1$
21: **fin si**
22: **si** id $= 0$ **entonces**
23: $u[0] = u[1]$
24: **fin si**
25: **para** $j = 1$ **hasta** $Subarreglo$ **hacer**
26: Calcular $u_x^- = \dfrac{u_j^n - u_{j-1}^n}{\Delta x}$
27: Calcular $u_x^+ = \dfrac{u_{j+1}^n - u_j^n}{\Delta x}$
28: Calcular $u_j^{n+1} = u_j^n - \Delta t[a^+ u_x^- + a^- u_x^+]$
29: **fin para**
30: $u_0^{n+1} = u_1^{n+1}$
31: $u_{Subarreglo}^{n+1} = u_{Subarreglo-1}^{n+1}$
32: **fin para**
33: Recojo los datos al $P_{rc} = 0$
34: El $P_{rc} = 0$ Escribe la solución en un archivo.

Algoritmo 8 Paralelización del Algoritmo 3.

Entrada: $\{L_0, L_1, T_0, T_1, \mathrm{id}\} \subset \mathbb{R},\ \ a \in \mathbb{R}^+,\ \ \{M, P_{rc}\} \subset \mathbb{N},\ \ u_0 : \mathbb{R} \to \mathbb{R}$
Salida: $u(x,t)$

Replicar todos los datos en los P_{rc}
Descomponer $u_0(x)$ en términos de $u_0^1(x)$, $u_0^2(x)$, $u_0^3(x)$, talque:
$u_0(x) = \begin{cases} u_0^3(x) & \text{si } u_0^1(x) \leq 0, \\ u_0^2(x) & \text{si } u_0^1(x) > 0. \end{cases}$
$u_0^3 = 0$ sobre las discontinuidades de u_0.
si $\mathrm{id} = 0$ **entonces**
    $u^1(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^1$
fin si
si $\mathrm{id} = 1$ **entonces**
    $u^2(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^2$
fin si
si $\mathrm{id} = 2$ **entonces**
    $u^3(x,t) \leftarrow$ Utilizar **Algoritmo 2** con $u_0 = u_0^3$
fin si
Recojo los datos en el $P_{rc} = 0$
si $P_{rc} = 0$ **entonces**
    para $n = 0$ **hasta** N **hacer**
        para $j = 0$ **hasta** M **hacer**
            si $u^1(n\Delta t, j\Delta x) < 0$ **entonces**
                $u_j^n = u^3(n\Delta t, j\Delta x)$
            si no
                $u_j^n = u^2(n\Delta t, j\Delta x)$
            fin si
        fin para
    fin para
fin si

Algoritmo 9 Paralelización del Algoritmo 4

Entrada: $\{L_0, L_1, T_0, T_1, \mathrm{id}\} \subset \mathbb{R}$, $\{M, P_{rc}\} \subset \mathbb{N}$, $CFL \in]0,1[$, $u_0 : \mathbb{R} \to \mathbb{R}$, $a : \mathbb{R} \times \mathbb{R}_0^+ \to \mathbb{R}$
Salida: $u(x,t)$

1: Replicar todos los datos en los P_{rc}
2: Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
3: Calcular $N = \dfrac{\max |a(x,t)|(T_1 - T_0)}{\Delta x CFL}$
4: Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
5: Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
6: Calcular $a^+[x] = \max\{a[x], 0\}$
7: Calcular $a^-[x] = \min\{a[x], 0\}$
8: Calcular el intervalo de trabajo.
9: Calcular $Subarreglo = \dfrac{M}{P_{rc}} + 1$
10: **para** $j = 0$ **hasta** $Subarreglo$ **hacer**
11: Calcular $u_j^0 = u_0(j\Delta x)$
12: **fin para**
13: **para** $n = 0$ **hasta** N **hacer**
14: Enviar el dato $u[1]$ al anterior procesador
15: **si** $0 <$ id **entonces**
16: Enviar $u[1]$ a id $- 1$
17: **fin si**
18: Recibir el dato $u[Subarreglo + 1]$ del siguiente procesador
19: **si** id $< P_{rc} - 1$ **entonces**
20: Recibir $u[n+1]$ de id $+ 1$
21: **fin si**
22: Enviar datos al siguiente procesador.
23: **si** id $< P_{rc} - 1$ **entonces**
24: Enviar $u[Subarreglo - 1]$ a id $+ 1$
25: **fin si**
26: Recibir datos del anterior procesador.
27: **si** id > 0 **entonces**
28: Recibir $u[0]$ de id $- 1$
29: **fin si**
30: **si** id $= 0$ **entonces**
31: $u[0] = u[1]$
32: **fin si**
33: **si** id $= p - 1$ **entonces**
34: $u[subarreglo] = [subarreglo + 1]$
35: **fin si**
36: **para** $j = 1$ **hasta** $Subarreglo$ **hacer**
37: Calcular $u_j^{n+1} = u_j^n - \Delta t[a^+[x[j]+0{,}5\Delta x]u_j^n + a^-[x[j]+0{,}5\Delta x]u_{j+1}^n - [a^+[x[j]-0{,}5\Delta x]u_{j-1}^n - a^-[x[j]-0{,}5\Delta x]u_j^n$
38: **fin para**
39: $u_0^{n+1} = u_1^{n+1}$
40: $u_{Subarreglo}^{n+1} = u_{Subarreglo-1}^{n+1}$
41: **fin para**
42: Recojo los datos al $P_{rc} = 0$
43: El $P_{rc} = 0$ Escribe la solución en un archivo.

Algoritmo 10 Paralelización del Algoritmo 5

Entrada: $\{L_0, L_1, T_0, T_1, \mathrm{id}\} \subset \mathbb{R}$, $\{M, P_{rc}\} \subset \mathbb{N}$, $CFL \in]0,1[$, $u_0 : \mathbb{R} \to \mathbb{R}$

Salida: $u(x,t)$

1: Replicar todos los datos en los P_{rc}
2: Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
3: Calcular $N = \dfrac{\max |u_0(x)|(T_1 - T_0)}{\Delta x CFL}$
4: Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
5: Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
6: Calcula el intervalo de trabajo.
7: $Subarreglo = \dfrac{M+1}{P_{rc}}$
8: **para** $j = 1$ **hasta** $Subarreglo$ **hacer**
9: Calcular $u_j^0 = u_0(j\Delta x)$
10: Calcular $x[j] = L_0 + \mathrm{id} Subarreglo \Delta x + (\mathrm{i} - 1)\Delta x$
11: **fin para**
12: **para** $n = 0$ **hasta** N **hacer**
13: Enviar el dato $u[1]$ al anterior procesador
14: **si** $0 <$ id **entonces**
15: Enviar $u[1]$ a id $- 1$
16: **fin si**
17: Recibir el dato $u[Subarreglo + 1]$ del siguiente procesador
18: **si** id $< P_{rc} - 1$ **entonces**
19: Recibir $u[n+1]$ de id $+ 1$
20: **fin si**
21: Enviar datos al siguiente procesador.
22: **si** id $< P_{rc} - 1$ **entonces**
23: Envio $u[Subarreglo - 1]$ a id $+ 1$
24: **fin si**
25: Recibir datos del anterior procesador.
26: **si** id > 0 **entonces**
27: Recibir $u[0]$ de id $- 1$
28: **fin si**
29: **si** id $= 0$ **entonces**
30: $u[0] = u[1]$
31: **fin si**
32: **si** id $= p - 1$ **entonces**
33: $u[subarreglo] = [subarreglo + 1]$
34: **fin si**
35: Usando el **Algoritmo 5**.
36: Calcular $u_j^n = u_j^n - \lambda(f_j - f_{j-1})$
37: **fin para**
38: $u_0^{n+1} = u_1^{n+1}$
39: $u_{Subarreglo}^{n+1} = u_{Subarreglo-1}^{n+1}$
40: Recojo los datos al $P_{rc} = 0$

Algoritmo 11 Paralelización del Algoritmo 6.

Entrada: $\{L_0, L_1, T_0, T_1, \mathrm{id}\} \subset \mathbb{R}$, $\{M, P_{rc}\} \subset \mathbb{N}$, $CFL \in]0,1[$, $u_0 : \mathbb{R} \to \mathbb{R}$, $f : \mathbb{R} \to \mathbb{R}$,
Salida: $u(x,t)$

1: Replicar todos los datos en los P_{rc}
2: Calcular $\Delta x = \dfrac{L_1 - L_0}{M}$
3: Calcular $N = \dfrac{\max|f'(u)|(T_1 - T_0)}{\Delta x CFL}$
4: Calcular $\Delta t = \dfrac{T_1 - T_0}{N}$
5: Calcular $\lambda = \dfrac{\Delta t}{\Delta x}$
6: Calcula el intervalo de trabajo.
7: $Subarreglo = \dfrac{M+1}{P_{rc}}$
8: **para** $j = 1$ **hasta** $Subarreglo$ **hacer**
9: Calcular $u_j^0 = u_0(j\Delta x)$
10: Calcular $x[j] = L_0 + \mathrm{id} Subarreglo \Delta x + (\mathrm{i} - 1)\Delta x$
11: **fin para**
12: **para** $n = 0$ **hasta** N **hacer**
13: Enviar el dato $u[1]$ al anterior procesador
14: **si** $0 <$ id **entonces**
15: Enviar $u[1]$ a id $- 1$
16: **fin si**
17: Recibir el dato $u[Subarreglo + 1]$ del siguiente procesador
18: **si** id $< P_{rc} - 1$ **entonces**
19: Recibir $u[n+1]$ de id $+ 1$
20: **fin si**
21: Enviar datos al siguiente procesador.
22: **si** id $< P_{rc} - 1$ **entonces**
23: Enviar $u[Subarreglo - 1]$ a id $+ 1$
24: **fin si**
25: Recibir datos del anterior procesador.
26: **si** id > 0 **entonces**
27: Recibir $u[0]$ de id $- 1$
28: **fin si**
29: **si** id $= 0$ **entonces**
30: $u[0] = u[1]$
31: **fin si**
32: **si** id $= p - 1$ **entonces**
33: $u[subarreglo] = [subarreglo + 1]$
34: **fin si**
35: Usando el **Algoritmo 6**.
36: Calcular $u_j^n = u_j^n - \lambda.(f_j - f_{j-1})$
37: **fin para**
38: $u_0^{n+1} = u_1^{n+1}$
39: $u_{Subarreglo}^{n+1} = u_{Subarreglo-1}^{n+1}$
40: Recojo los datos al $P_{rc} = 0$

MPI_Init	Esta subrutina inicializa MPI. Todos los programas MPI deben llamar a *MPI_Init* antes que cualquier otra rutina MPI . Más de una llamada a *MPI_Init* por cualquier tarea es errónea.
MPI_Finalize	Termina todos los procesos de MPI.
MPI_COMM_WORLD	Describe el conjunto de procesadores (comunicador).
MPI_Comm_rank	Devuelve el rango o identificador de la tarea local en el grupo asociado a un comunicador.
MPI_Comm_size	Devuelve el tamaño del grupo asociado con un comunicador.
MPI_Float	Tipo de dato punto flotante.
MPI_Barrier	Es una subrutina puramente de sincronización, que bloquea a los procesos de un comunicador hasta que todos ellos han pasado por la barrera.
MPI_Wtime	Devuelve el valor actual del tiempo como un valor de punto flotante.
MPI_Gather	Realiza una recolección de datos de todos los procesos del comunicador en el proceso raíz.
MPI_Status	Estado del mensaje que indica cómo fue recibido el mensaje.
MPI_Send	Realiza el envío de datos, punto a punto (con bloqueo).
MPI_Recv	Realiza una operación de recepción de datos, punto a punto (con bloqueo).

Tabla 2.4: Rutinas de comunicación y redistribución de MPI.

Capítulo 3

Resultados

En este capítulo se reportan los principales resultados obtenidos en la tesis. Se comienza presentando el análisis de la complejidad computacional de los algoritmos. Luego se presentan las simulaciones correspondientes los algoritmos secuenciales y se finaliza el capítulo con los resultados para los algoritmos paralelos. En este último caso, se muestran los resultados para las métricas introducidas en el capítulo 2.

3.1. Notación y consideraciones

Una notación común que se utiliza en el capítulo es la siguiente:

L_0	Límite inferior para la variable espacial "x".
L_1	Límite superior para la variable espacial "x".
T_0	Límite inferior para la variable temporal "t".
T_1	Límite superior para la variable temporal "t".
M	Número de particiones del intervalo $[L_0, L_1]$.
N	Número de particiones del intervalo $[T_0, T_1]$.
$u_0(x)$	Condición inicial del problema.
$u(x,t)$	Solución del problema de Cauchy.

Además, una consideración general es que las pruebas numéricas para este capítulo se realizaron ejecutando implementaciones en Lenguaje C (Ver Apéndice A) y se utilizó un nodo del "Cluster Ciencias Básicas", cuyas características técnicas fueron descritas en la subsección 2.2.1.

3.2. Estimación de la complejidad computacional

La estimación de la complejidad de tiempo está en función del tamaño de las particiones del intervalo espacial o nodos espaciales que es denotado por M y del número de iteraciones

temporales que es denotado por N. Para medir el costo de los algoritmos que se presentan en este trabajo, se procederá a determinar cuántas instrucciones de cada tipo se ejecutan, se multiplica este número por el tiempo que emplea la instrucción en ejecutarse y se realiza la suma para los diferentes tipos de tiempo que se consideran:

- t_a es el tiempo de asignación.
- t_c es el tiempo de comparación.
- t_{i} es el tiempo de incremento.
- t_v es el tiempo de acceso a un vector.

Para el análisis de complejidad se toma de manera referencial el Algoritmo 1. La complejidad de los otros algorimos es de un comportamiento similar, puesto que tienen una estructura algebraica análoga. En efecto, analizadas cada una de las líneas que conforman el Algoritmo 1, se determinó que los tiempos involucrados son los que se indican a continuación:

- Control de las primeras 4 asignaciones (4 primeras líneas): $4t_a$.
- Control del primer bucle (líneas 5 y 6): Mt_a.
- Control del bucle interno, para cada n (líneas 9, 10 y 11): $(M-1)t_a + 2(M-1)t_{\mathrm{i}}$.
- Control de las asignaciones (línea 12): t_a.

Luego en total se tiene

$$\begin{aligned} Total &= 4t_a + Mt_a + \sum_{n=1}^{N} \Big[(M-1)t_a + 2(M-1)t_{\mathrm{i}} + t_a\Big] \\ &= 4t_a + Mt_a + \Big[(M-1)t_a + 2(M-1)t_{\mathrm{i}} + t_a\Big]N \\ &= (M-1)N(t_a + 2t_{\mathrm{i}}) + Mt_a + Nt_a + 4t_a \\ &= MN(t_a + 2t_{\mathrm{i}}) + Mt_a - 2Nt_{\mathrm{i}} + Mt_a + 4t_a. \end{aligned}$$

Luego, denotando por $n = \max\{M, N\}$, se tiene que en el caso más desfavorable

$$T(n) = (t_a + 2t_{\mathrm{i}})n^2 + t_a n + 4t_a.$$

Por consiguiente, los algoritmos son de orden $O(n^2)$ donde $n = \max\{M, N\}$. Además, es calaro que es de clase P. Es necesario notar que para los esquemas explícitos es necesario seleccionar M muy grande par evitar la difusión numérica. Esto, debido a la condición CFL, tiene dos implicaciones inmediatas. Por un lado, si el tiempo final de simulación es pequeño, típicamente se tiene que $N \approx M$ y por ende $T(n)$ queda determinado por M. En cambio, si los tiempos son grandes se tiene que $N = CM$ con $C > 0$ y grande. Para fijar ideas, por ejemplo, si $L_0 = T_0 = 0$, $L_1 = 1$, $M = 100$, $a = 1$ y $CFL = 0.1$, entonces

$$N = \frac{(T_1 - T_0)M\ a}{(L_1 - L_0)\ CFL} = T_1 M = \begin{cases} < M, & \text{para } T_1 < 1, \\ = M, & \text{para } T_1 = 1, \\ > M, & \text{para } T_1 > 1. \end{cases}$$

Luego, si se supone que $T_1 = 100 = M$ se deduce que $n = \max N, M = N\ N = T_1 M = M^2$, entonces $O(n^2) = O(N^2) = O(M^4)$. Este ejemplo muestra que al realizar una simulación en tiempos grandes, con un algoritmo secuencial, se puede alcanzar la convergencia en tiempos superiores a los que se podría demorar en realizar un experimento físico. Esto, naturalmente, conduce a utilizar herramientas tales como la parelización.

3.3. Simulaciones numéricas con los algoritmos secuenciales

En esta sección se muestran los resultados numéricos de las implementaciones de los Algoritmos secuenciales presentados en el Capítulo 2. Es necesario notar que en el caso práctico de la implementación, la salida para el Algoritmo para la ecuación de la edvección con velocidad constante y el algoritmo para la ecuación de advección con velocidad variable es un archivo de datos con los siguientes parámetros: nodos del intervalo espacial, condición inicial, solución aproximada y solución exacta. Note que la solución exacta existe puesto que por el método de las características es posible obtener una solución analítica para dichos problemas, ver sección 2.3. En tanto que para la ecuación de Burger y el Modelo de tráfico vehicular LWR no es siempre posible obtener la solución exacta. En todos los casos, para la gráfica de dichos datos se ha utilizado GNUPLOT 4.4.

3.3.1. Algoritmo 1. Ecuación de advección con velocidad constante.

En esta simulación se considera la ecuación de la advección lineal con una condición inicial compuesta de dos estados constantes que generan dos discontinuidades y la velocidad de transporte positiva. En este ejemplo la condición inicial es dada por:

$$u_0(x) = \begin{cases} 1, & -0{,}5 \leq x \leq 0{,}5, \\ 0, & \text{en otro caso.} \end{cases} \tag{3.1}$$

Además, la velocidad de transporte es $a = 2$. Se consideran 200 nodos espaciales, es decir, $M = 200$. La diferencia que visualmente se observa entre la solución simulada y la solución analítica es debido a que el método upwind es de primer orden y en consecuencia tiene una alta difusión numérica. En la práctica este hecho se mejora aumentando la cantidad de nodos espaciales, dado que es conocido que aumentando la cantidad de nodos espaciales la solución numérica converge a la solución exacta. Además, en la Figura 3.1, se observa que el perfil inicial u_0 se transporta con una velocidad $a = 2$, manteniendo su forma hasta el tiempo final de simulación $t = 1$.

3.3.2. Algoritmo 4. Ecuación de advección con velocidad variable

En esta simulación se considera la ecuación de la advección variable y con término fuente. Más precisamente se considera la siguiente ecuación

$$u_t + a(x)u_x = 0, \tag{3.2}$$

la cual se puede escribir equivalentemente de manera conservativa como

$$u_t + \Big(a(x)u\Big)_x = -a(x)u_x.$$

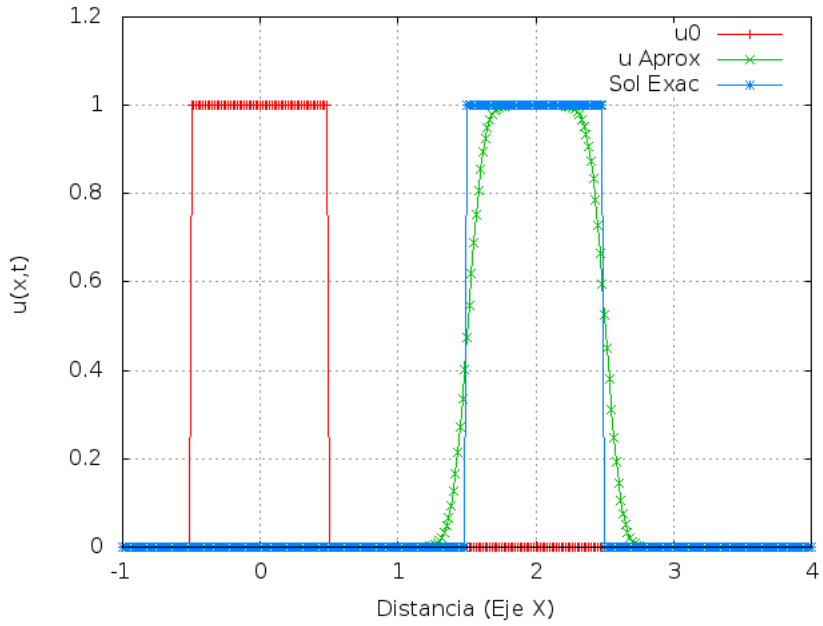

Figura 3.1: Resultado de simulación para el flujo $f(u) = 2u$ y codición inicial (3.1) aplicando el esquema upwind detallado en el Algoritmo 1.

En esta última ecuación el flujo es $a(x)u$ y el término fuente $-a(x)u$. En este ejemplo, se resuelve numéricamente la ecuación (3.2) dotado de la condición inicial y velocidades dadas por

$$u_0(x) = \frac{1}{1+x^2} \qquad \text{y} \qquad a(x) = x, \tag{3.3}$$

respectivamente. En este caso se escogió $M = 300$ para obtener los perfiles de las simulaciones reportadas en la Figura 3.2, donde se muestra el perfil inicial u_0 y el perfil final de la solución u del problema en el tiempo $t = 1$. Es conveniente notar que el perfil de la solución aproximada obtenida por el método upwind no se transporta sin deformación tal como lo hace en el caso lineal de velocidad constante, si no que la condición inicial se va deformando. Con mayor precisión, y en términos generales, conviene comentar que la velocidad variable implica que la condición inicial se deforme. Para remarcar, un aspecto que se puede notar en la Figura 3.2 es el que el perfil final no se ha transportado y para esto conviene observar que el punto máximo y los puntos más bajos aparecen en el mismo lugar. El punto máximo está en $x = 0$ y los puntos más bajos en $x = -4$ y $x = 4$. Este hecho es debido a que el término fuente compensa el transporte o en otras palabras actúa como una fuerza de resistencia.

3.3.3. Algoritmo 5. Ecuación de Burger

En esta simulación se considera la ecuación de Burger con una condición inicial compuesta de dos estados constantes que generan una discontinuidad. Es conveniente recordar que el problema de condiciones iniciales en el cual la condición inicial es una discontinuidad de salto para dos estados constantes se llama *Problema de Riemann*, ver la sección 2.3 y para mayores detalles consultar [9, 17, 20, 29]. En consecuencia, en esta sección se resolverán dos problemas

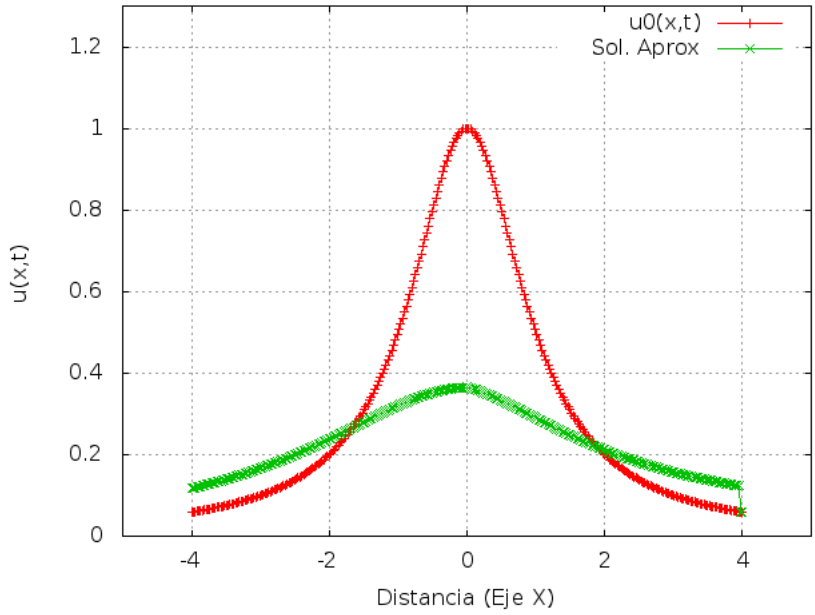

Figura 3.2: Resultado de simulación para (3.2)-(3.3) aplicando el esquema upwind detallado en el Algoritmo 4.

de Riemann, el primero con condición inicial

$$u_0(x) = \begin{cases} 5, & x \leq 0, \\ 0, & x > 0. \end{cases} \tag{3.4}$$

y el segundo con condición inicial

$$u_1(x) = \begin{cases} 0, & x \leq 0, \\ 5, & x > 0. \end{cases} \tag{3.5}$$

En las simulaciones de ambos problemas de Riemann se ha considerado $M = 300$, $L_0 = -20$, $L_1 = 20$ y el tiempo final $t = 1$. En la Figura 3.3 se puede observar la simulación para la condición inicial (3.4) y en la Figura 3.4, para la condición inicial (3.5). Teóricamente se conoce que la condición inicial (3.4) genera una onda de choque, es decir, una discontinuidad que se propaga en el tiempo a una velocidad determinada en función del salto. En tanto que la condición inicial (3.5) genera una onda de rarefacción, es decir, una curva continua. Para mayores detalles sobre este tipo de ondas consultar [9, 17, 20, 29].

3.3.4. Algoritmo 6. Modelo de tráfico vehicular LWR

Se tiene conocimiento que existe diversas propuestas de la velocidad en el modelo de tráfico vehicular en autopistas siguiendo las hipótesis del modelo LWR (ver Tabla 2.1). En el presente trabajo de tesis se consideró la velocidad propuesta por **Greenshield**. En consecuencia, para las simulaciones de esta sección, se considera el siguiente flujo

$$f(u) = u\, v(u) = \begin{cases} 15u(1 - \dfrac{u}{2}), & 0 < u < 0{,}2, \\ 0, & \text{en otro caso.} \end{cases}$$

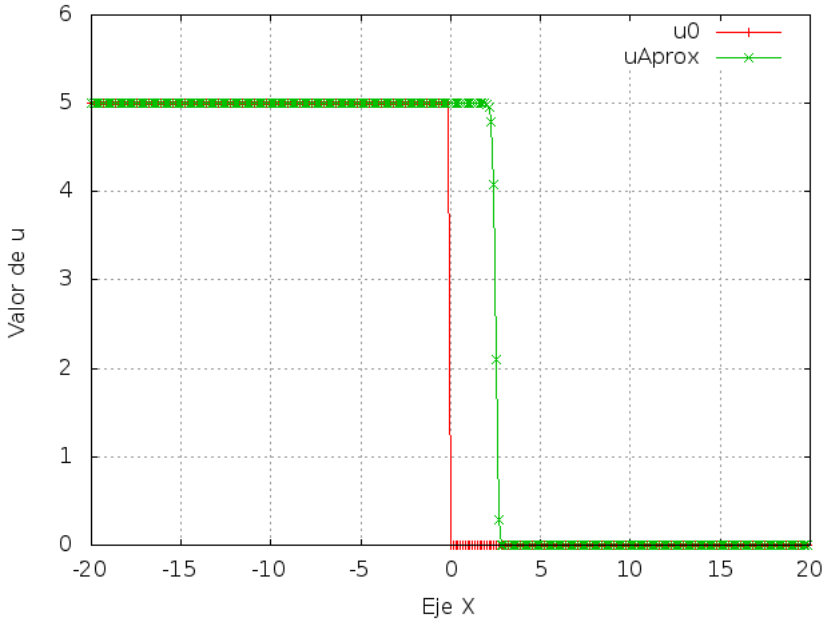

Figura 3.3: Resultado de simulación para la ecuación de Burger con condición inicial (3.4) aplicando el esquema de Godunov detallado en el Algoritmo 5.

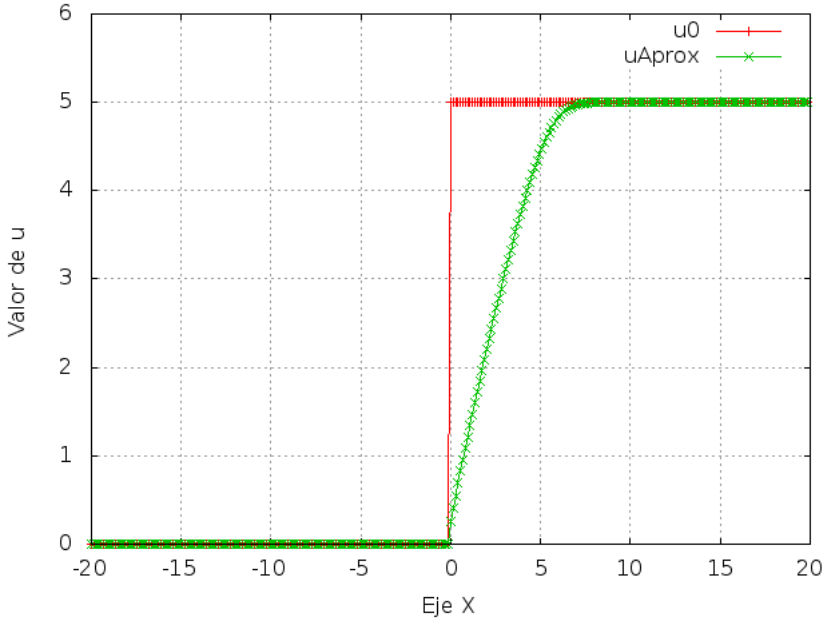

Figura 3.4: Resultado de simulación para la ecuación de Burger con condición inicial (3.5) aplicando el esquema de Godunov detallado en el Algoritmo 5.

Para las cuatro simulaciones se utilizó $M = 300$, $L_0 = -20$, $L_1 = 20$ y tiempo final $t = 1$. Las cuatro simulaciones que se consideran se diferencian en el hecho que la condición inicial es distinta para cada caso. Más precisamente las condiciones iniciales que se consideraron son las siguientes:

(a) En una primera aplicación se considera la siguiente condición inicial:

$$u_0(x) = \begin{cases} 0, & x \leq 0, \\ 0{,}09, & x > 0. \end{cases} \tag{3.6}$$

Esta condición inicial corresponde a un estado en el cual hay una densidad constante al lado derecho de $x = 0$ y al lado izquierdo la autopista está vacía.

(b) En la segunda aplicación se considera la siguiente condición inicial:

$$u_0(x) = \begin{cases} 0{,}09, & x \leq 0, \\ 0, & x > 0. \end{cases} \tag{3.7}$$

Esta condición inicial corresponde a un estado en el cual hay un semáforo en el instante que cambia de luz roja a luz verde.

(c) En la tercera simulación se considera la siguiente condición inicial:

$$u_0(x) = \begin{cases} 0{,}09, & -1 < x < 1, \\ 0, & \text{en otro caso.} \end{cases} \tag{3.8}$$

Esta condición inicial corresponde a un estado constante en un tramo de la autopista.

(d) En la cuarta simulación numérica se considera la siguiente condición inicial

$$u_0(x) = \begin{cases} 0, & -1 < x < 1, \\ 0{,}09, & \text{en otro caso.} \end{cases} \tag{3.9}$$

Esta condición inicial corresponde a un estado vacío en un tramo de la autopista.

Los resultados para estas cuatro simulaciones se muestran en las Figuras 3.5, 3.6, 3.7 y 3.8, respectivamente. En la Figura 3.5, se observa que el vacío que existe al lado izquierdo se propaga en el tiempo, manteniendo la onda choque inicial. En la Figura 3.6, se nota que el cambio a luz verde inicial genera un estado suave que se propaga a la izquierda a medida que los vehículos situados adelante aumentan su velocidad. En la Figura 3.7, se observa que el pequeño bloque inicial de vehículos que inicialmente están cercanos se tienden a dispersar a medida que transcurre el tiempo y preservan una distribución suave de la densidad. En la Figura 3.8, se nota que el pequeño espacio vacío desaparece en el transcurso del tiempo y los automóviles que están a la izquierda del espacio vacío alcanzan a los automóviles que están a la derecha de este y en el corto plazo este espacio vacío desaparece.

3.4. Métricas para los algoritmos paralelos

Como se especificó en el Capítulo 1, las métricas para analizar y evaluar los algoritmos paralelos son: el tiempo experimental, speed-up y eficiencia.

Para cada algoritmo paralelo propuesto se presentan las métricas respectivas, además se considera el número de particiones del eje X (M), obteniéndose de esta manera $M+1$ puntos espaciales. Además, estas particiones han sido consideradas, para que el total de puntos a desarrollar sean múltiplos de los números de procesadores a evaluar (1, 2, 4 y 8).

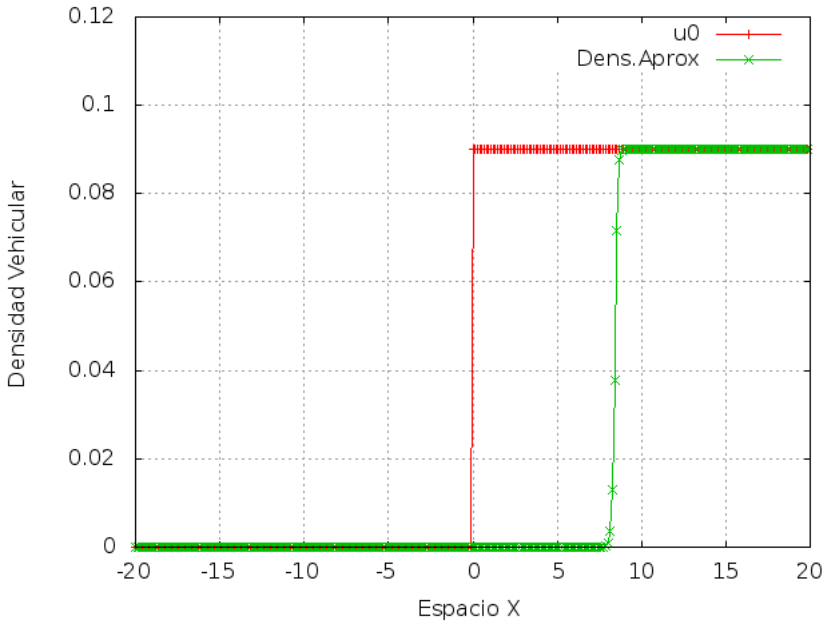

Figura 3.5: Simulación del modelo LWR con el esquema de Godunov dado en el Algoritmo 5 y condición inicial (a), simulando la propagación de un vacío en la autopista.

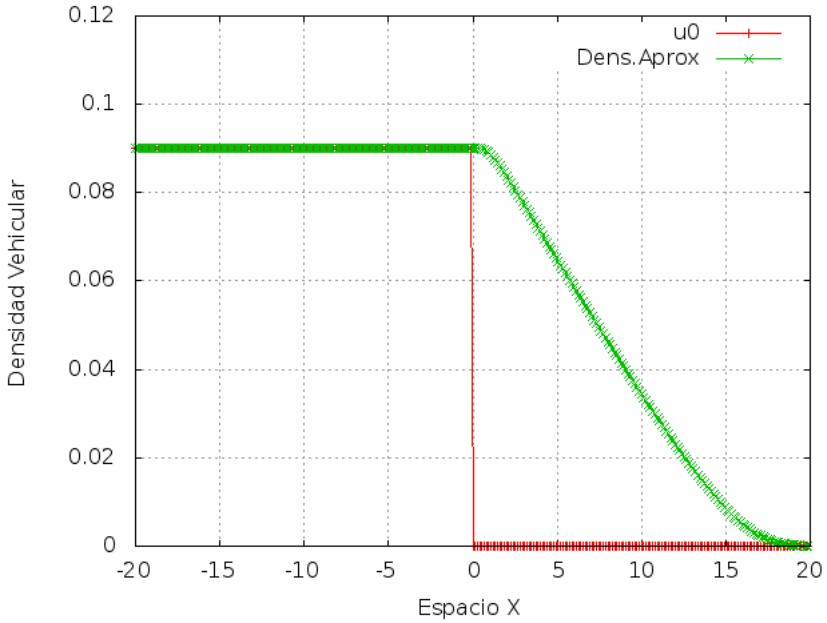

Figura 3.6: Simulación del modelo LWR con el esquema de Godunov dado en el Algoritmo 5 y condición inicial (b), simulando el cambio de luz roja a luz verde en un semáforo.

No es necesario encontrar los tiempos experimentales de los algoritmos secuenciales, ya que dichos algoritmos paralelos fueron ejecutados para un procesador y de esta manera se obtuvo el dato requerido. Los tiempos obtenidos son el promedio de los tiempos recogidos después de ejecutar diez (10) veces de ser aplicado el algoritmo en distintas condiciones de carga del sistema operativo, esto para depurar valores muy altos o muy bajos que suelen

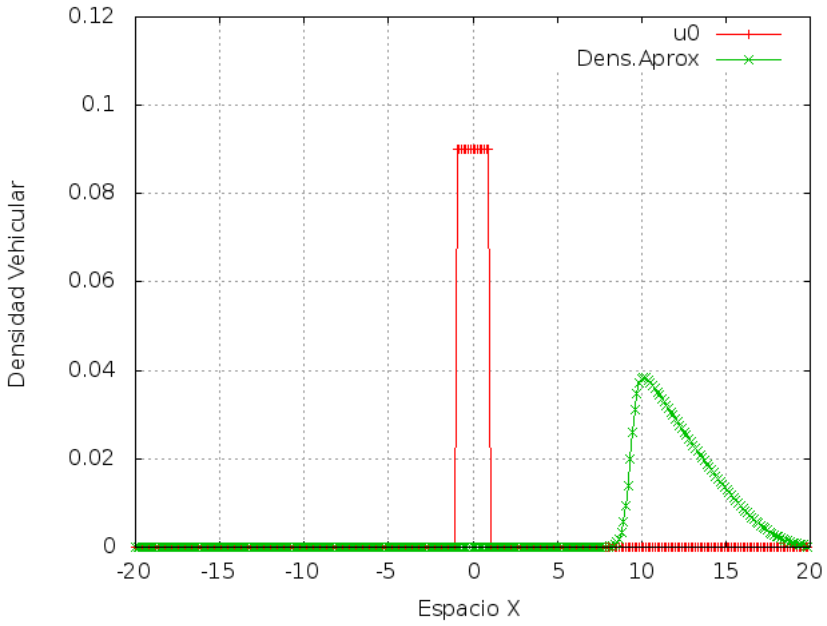

Figura 3.7: Simulación del modelo LWR con el esquema de Godunov dado en el Algoritmo 5 y condición inicial (c), simulando la evolución de un estado constante intermedio en la autopista.

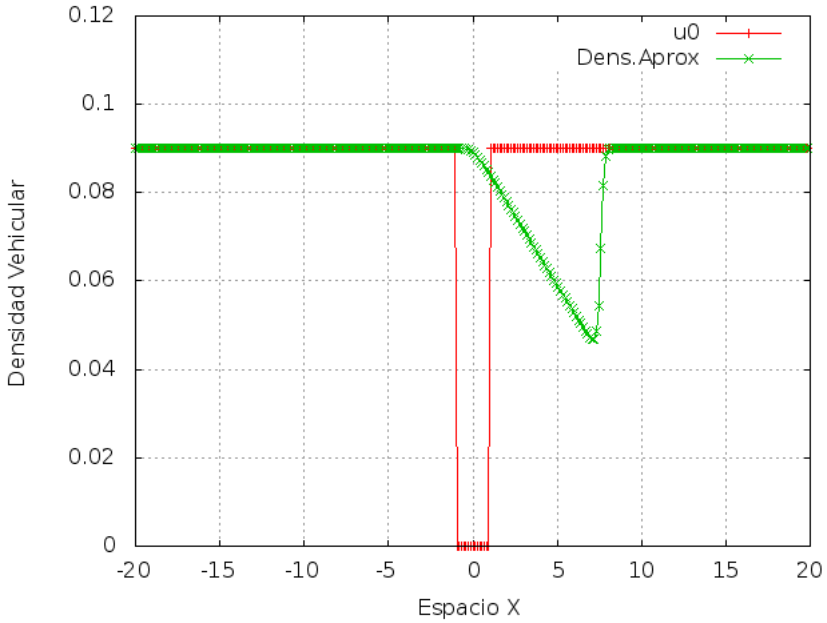

Figura 3.8: Simulación del modelo LWR con el esquema de Godunov dado en el Algoritmo 5 y condición inicial (d), simulando la evolución de un estado vacío en la autopista.

obtenerse en los tiempos experimentales.

3.4.1. Métricas para el Algoritmo 7

A continuación se presentan los tiempos experimentales para las siguientes particiones del Eje X: $M = \{1199; 3199; 6399; 8799; 13599; 53599; 107199\}$, los resultados se presentan en la Tabla 3.1. Se aprecia una reducción los tiempos de ejecución cuando se utilizan 2 procesadores, excepto en algunos casos donde $M = \{53599, 107199\}$, ver Figura 3.9. Este fenómeno tiende a conservarse cuando el tamaño de la partición se incrementa, por ejemplo, si se compara el rendimiento de los algoritmos paralelos con 4 procesadores y el elgoritmo secuencial, para $M = 53599$ y $M = 107199$, se observa una reducción de los tiempos experimentales de ejecución como se pueden observar en las Figuras 3.10 y 3.11 respectivamente, debido a que el costo relativo de paso de mensajes es menor.

Para el cálculo del speed-up sólo se han considerado 1, 2 y 4 procesadores, debido a que con mayor número de procesadores, el costo del tiempo experimental es demasiado alto, producto de un mayor paso de mensajes entre los procesadores. Considerando esta apreciación los speed-up correspondientes se puede observar en la Tabla 3.2. Con 2 procesadores el speed-up es aceptable para $M = \{1199; 3199; 6399; 8799; 13599\}$.

La eficiencia correspondiente a estos experimentos se muestran en la Tabla 3.3, donde se pueden observar porcentajes aceptables con tamaños de problema relativamente pequeños, $M = \{1199; 3199; 6399; 8799; 13599\}$.

P_{rc}	Tiempo(Seg)						
1	0.0033883	0.0154025	0.0506686	0.0911939	0.2082826	25.490652	121.070992
2	0.0033517	0.0146900	0.0478649	0.0859449	0.1957607	31.562746	124.342032
4	0.6939478	0.3319459	0.7270046	0.8930572	0.5272893	27.487774	92.537643
8	47.995830	84.622921	142.87452	212.07548	—	—	—
M	1 199	3 199	6 399	8 799	13 599	53 599	107 199

Tabla 3.1: Tiempos de ejecución experimentales para el Algoritmo 7.

P_{rc}	speed-up						
1	1	1	1	1	1	1	1
2	1.0109198	1.0485024	1.0585753	1.0610742	1.0639654	0.8076183	0.97369321
4	0.048826	0.0464006	0.0696951	0.1021143	0.3950063	0.9273451	1.30834316
M	1 199	3 199	6 399	8 799	13 599	53 599	107 199

Tabla 3.2: Speed-up experimentales para el Algoritmo 7.

3.4.2. Métricas para el Algoritmo 8

Los tiempos experimentales para $M = \{1199; 3199; 6399; 8799; 13599\}$, se pueden observar en la Tabla 3.4. Se aprecia que se reducen los tiempos de ejecución cuando se utilizan 3 procesadores, excepto si $M = \{1199, 3199\}$, ver Figura 3.12. Este fenómeno tiende a conservarse

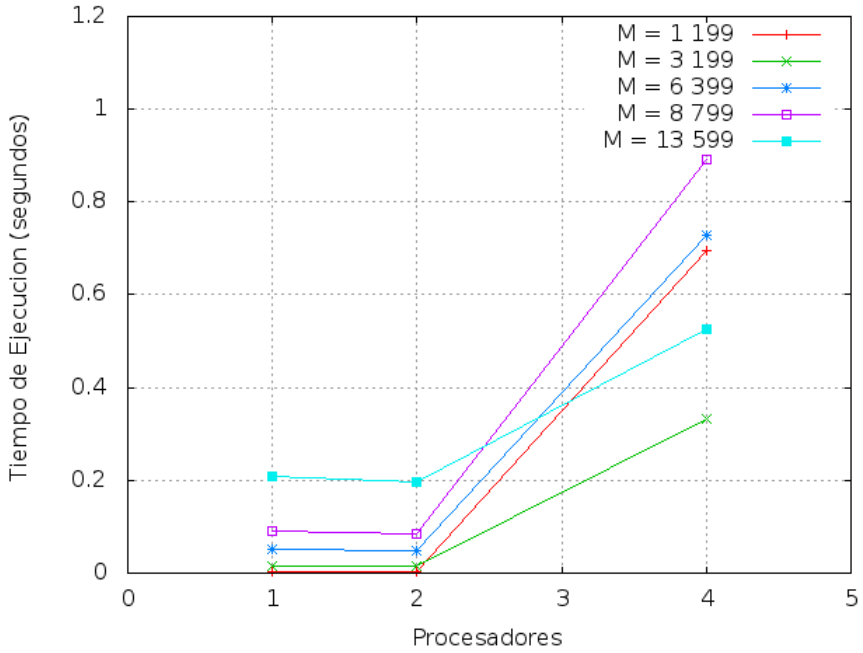

Figura 3.9: Tiempos de ejecución experimentales para el Algoritmo 7

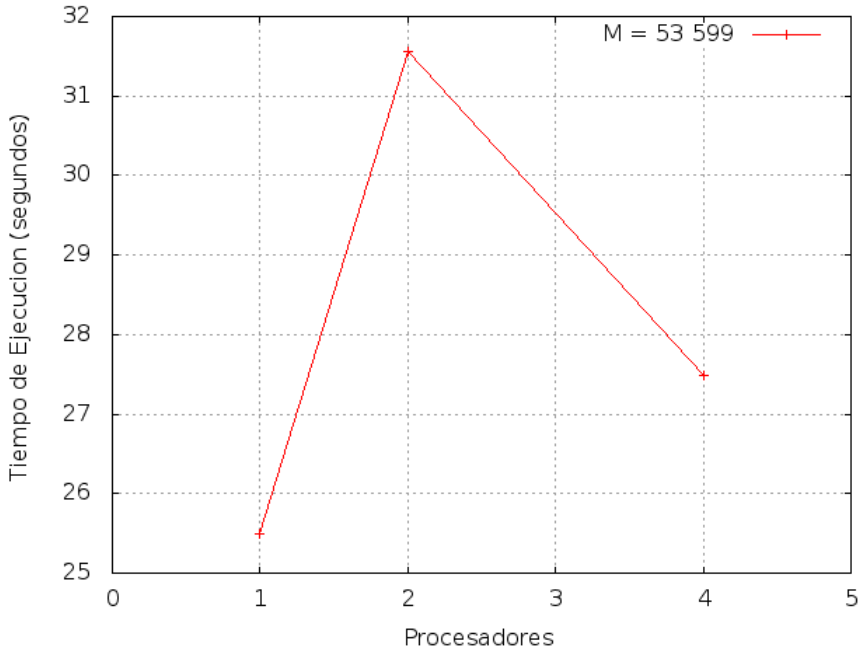

Figura 3.10: Tiempos de ejecución experimentales para el Algoritmo 7 con M = 53 599.

cuando el tamaño de la partición es pequeña, debido a que el costo de paso de mensajes entre los 3 procesadores es mayor por la sincronización y pocos datos a operar.

Para el cálculo del speed-up sólo se han considerado los resultados 1 y 3 procesadores, debido a la solución del método numérico, que subdivide el problema en 3 sub problemas, dicho speed-up se puede observar en la Tabla 3.5. Con 3 procesadores el speed-up es aceptable

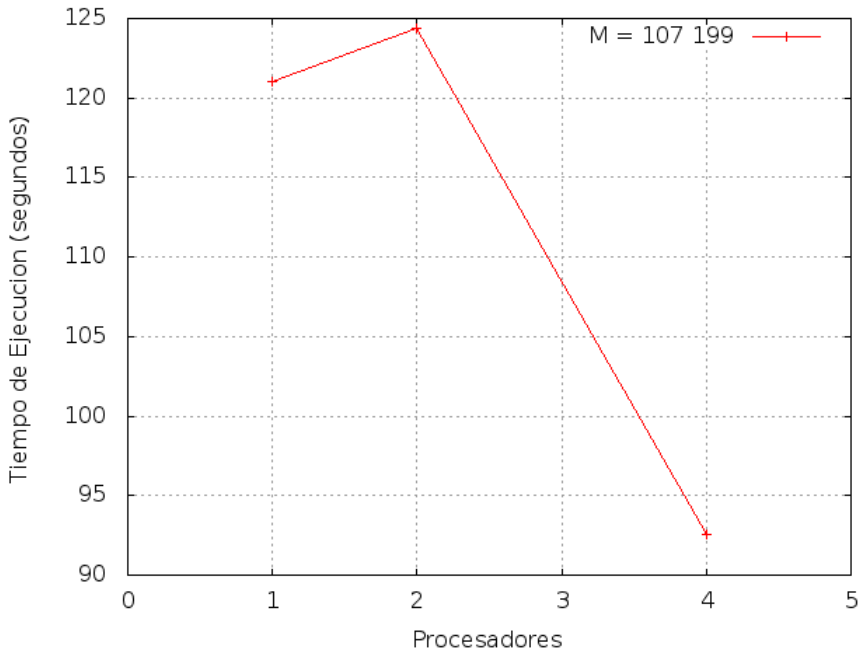

Figura 3.11: Tiempos de ejecución experimentales para el Algoritmo 7 con M = 107 199.

Tabla 3.3: Eficiencia experimental del Algoritmo Paralelo UPWIND, para el flujo Lineal Constante.

P_{rc}	eficiencia						
1	100 %	100 %	100 %	100 %	100 %	100 %	100 %
2	51 %	53 %	53 %	53 %	53 %	41 %	49 %
4	1.2 %	1.2 %	1.8 %	2.5 %	10 %	23 %	33 %
M	1 199	3 199	6 399	8 799	13 599	53 599	107 199

para $M = \{1199; 3199; 6399; 8799; 13599\}$.

La eficiencia de estos experimentos (Tabla 3.6) es aceptable con tamaños de problema grandes $M = \{1199; 3199; 6399; 8799; 13599\}$. La mejor eficiencia se obtiene en $M = \{1199\}$ con un 60 %.

P_{rc}	Tiempo(Seg)						
1	0.0066334	0.0216085	0.0451932	0.3583469	6.5319125	21.674598	52.9702064
3	0.2085543	0.25451	0.02513	0.2158404	3.9887103	14.558843	33.5897423
M	399	799	1 199	3 199	6 399	8 799	13 599

Tabla 3.4: Tiempos de ejecución experimentales para el Algoritmo 8.

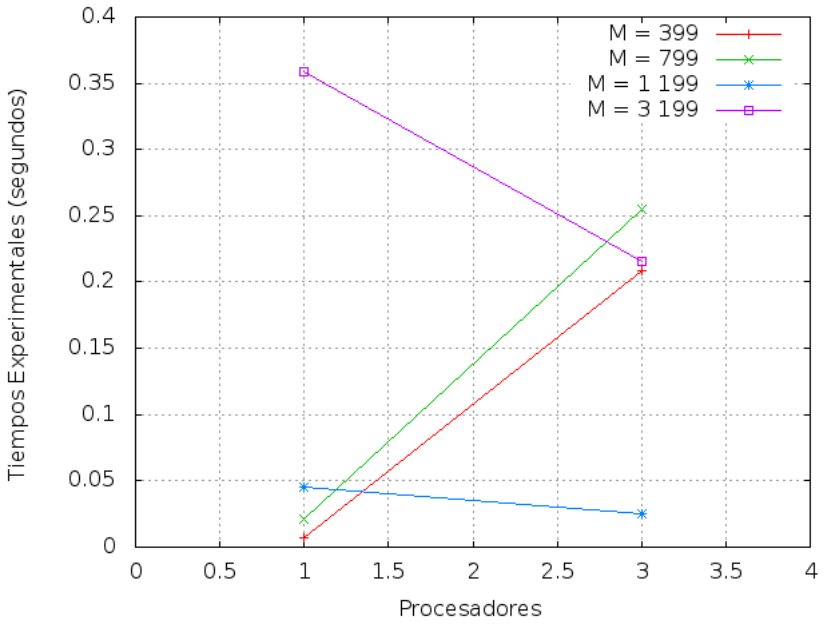

Figura 3.12: Tiempos experimentales para el Algoritmo 8.

P_{rc}	speed-up						
1	1	1	1	1	1	1	1
3	0.0318066	0.0849024	1.7983764	1.6602402	1.6376001	1.4887583	1.5769757
M	399	799	1 199	3 199	6 399	8 799	13 599

Tabla 3.5: Speed-up experimentales para el Algoritmo 8.

P_{rc}	eficiencia						
1	100 %	100 %	100 %	100 %	100 %	100 %	100 %
3	1 %	3 %	60 %	55 %	53 %	55 %	53 %
M	399	799	1 199	3 199	6 399	8 799	13 599

Tabla 3.6: Eficiencia experimental para el Algoritmo 8.

3.4.3. Métricas para el Algoritmo 9

Se presentan en la Tabla 3.7 los tiempos experimentales para las siguientes particiones del Eje X: $M = \{1199; 3199; 6399; 8799; 13599; 53599\}$. En la Figura 3.13, se puede observar que cuando se utilizan 2 procesadores, los tiempos experimentales se reducen para $M = \{1199, 3199\}$. Este fenómeno tiende a conservarse cuando el tamaño de la partición es pequeña.

No se muestra el cálculo del speed-up y eficiencia, debido a que:

1. Para varios procesadores ($M = \{6399; 8799; 13599; 53599\}$), el costo del tiempo experimental es demasiado alto.
2. Usando la definición de speed-up, esta nos garantiza que debe ser menor o igual que

el número de procesadores. Los tiempos experimentales recogidos no cumplen con tal requisito, excepto para $M = \{1199, 3199\}$. Teniendo en cuenta las métricas para el algoritmo paralelo, se puede decir, para este algoritmo que no es necesaria la paralelización, cuando las particiones son grandes.

P_{rc}	Tiempo(Seg)					
1	0.0030754	0.0095229	0.0250557	0.0406273	0.0819422	4.967283
2	0.0028651	0.0079896	0.0573804	0.1010364	0.2254329	165.0565
4	33.5094431	0.2727558	152.43977	213.66450	—	—
8	137.336589	—	—	—	—	—
M	1 199	3 199	6 399	8 799	13 599	53 599

Tabla 3.7: Tiempos de ejecución experimentales para el Algoritmo 9.

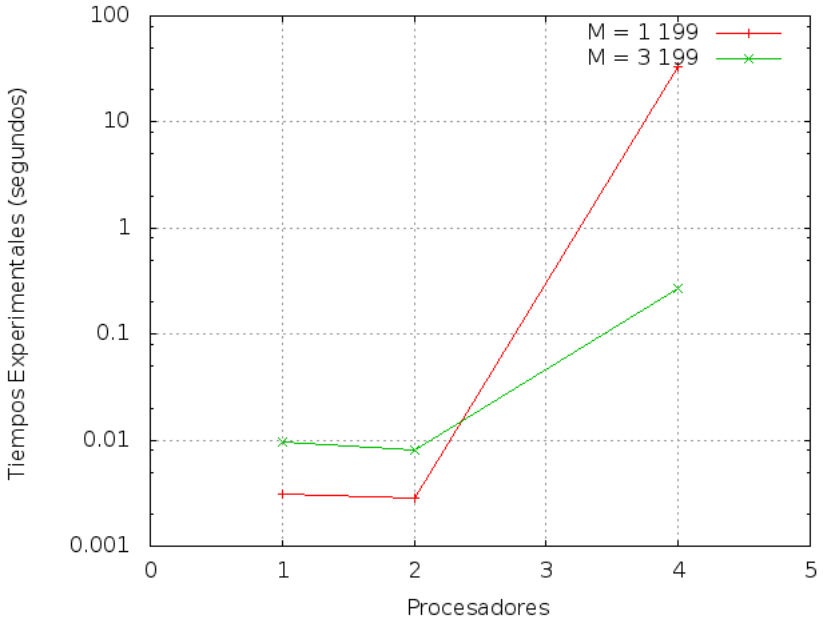

Figura 3.13: Tiempos de ejecución experimentales para el Algoritmo 9 con M = 1 199 y M = 3 199.

3.4.4. Métricas para el Algoritmo 10

Los tiempos experimentales obtenidos para $M = \{399; 799; 1199; 3199; 8799\}$, se pueden observar en la Tabla 3.8. Los tiempos experimentales se reducen cuando se utilizan 2 procesadores para todas las particiones definidas, ver Figura 3.14.

Para el cálculo del speed-up se han considerado 1 y 2 procesadores, debido a que los tiempos experimentales obtenidos en la Tabla 3.8 son demasido grandes para más procesadores. Para 2 procesadores el speed-up es aceptable para $M = \{6399; 8799; 13599\}$.

La eficiencia correspondiente a estos experimentos (Tabla 3.10) indican que el algoritmo

logra mejores resultados cuando las partición se van incrementando. La mejor eficiencia se obtiene para $M = 8799$ con un valor de 92 %.

P_{rc}	Tiempo(Seg)				
1	0.0033883	0.0068640	0.0137742	0.0856222	0.328734
2	0.0033517	0.0049772	0.0091815	0.049155	0.1790448
4	271.125441	543.01545	—	—	—
M	399	799	1 199	3 199	8 799

Tabla 3.8: Tiempos de ejecución experimentales para el Algoritmo 10.

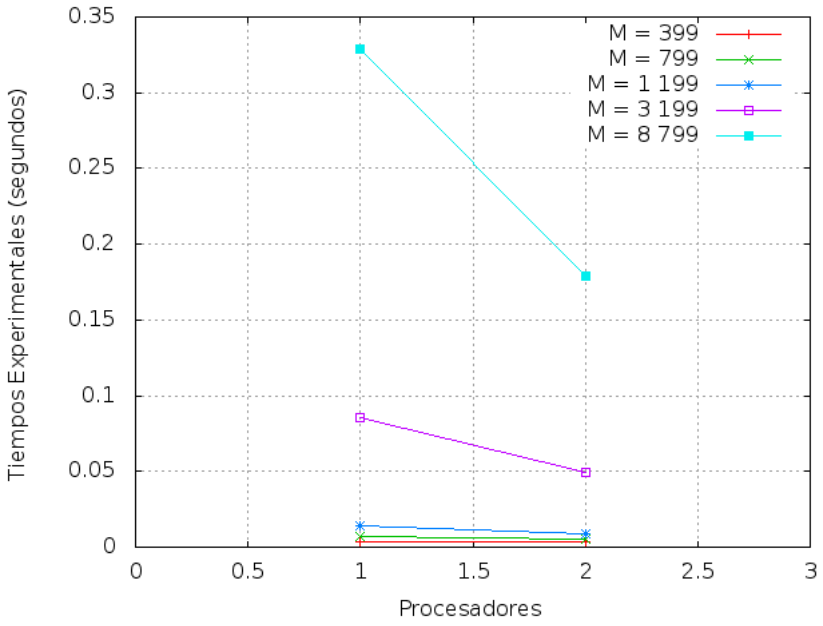

Figura 3.14: Tiempos experimentales para el Algoritmo 10.

P_{rc}	speed-up				
1	1	1	1	1	1
2	1.0109198	1.3790886	1.5002124	1.7418818	1.8360433
M	399	799	1 199	3 199	8 799

Tabla 3.9: speed-up para el Algoritmo 10.

P_{rc}	eficiencia				
1	100 %	100 %	100 %	100 %	100 %
2	51 %	69 %	75 %	87 %	92 %
M	399	799	1 199	3 199	8 799

Tabla 3.10: Eficiencia para el Algoritmo 10.

3.4.5. Métricas para el Algoritmo 11

Los tiempos experimentales para $M = \{399; 799; 1199; 3199; 8799\}$ se pueden observar en la tabla 3.11. Se aprecia que se reducen los tiempos de ejecución cuando se utilizan 2 procesadores para $M = \{399; 799; 1199; 3199\}$, ver Figura 3.15. En la Figura 3.16 se observa que el tiempo experimental es más costoso para $M = \{8799\}$.

Para el cálculo del speed-up se han considerado 1 y 2 procesadores, dado que com más procesadores los tiempos se incrementan demasiado, dicho speed-up se puede observar en la Tabla 3.12. Notar que para 2 procesadores el speed-up es aceptable para todas las particiones.

La eficiencia correspondiente a estos experimentos (Tabla 3.10) mejora cuando las particiones se van incrementando. La mejor eficiencia se obtiene en $M = \{3199\}$ con un 76 %.

P_{rc}	Tiempo(Seg)				
1	0.0068895	0.0199585	0.0452490	0.2759218	9.474446
2	0.0058181	0.0169663	0.0327413	0.1814459	125.712
4	—-	—	—	—	—
M	399	799	1 199	3 199	8 799

Tabla 3.11: Tiempos de ejecución experimentales para el Algoritmo 11.

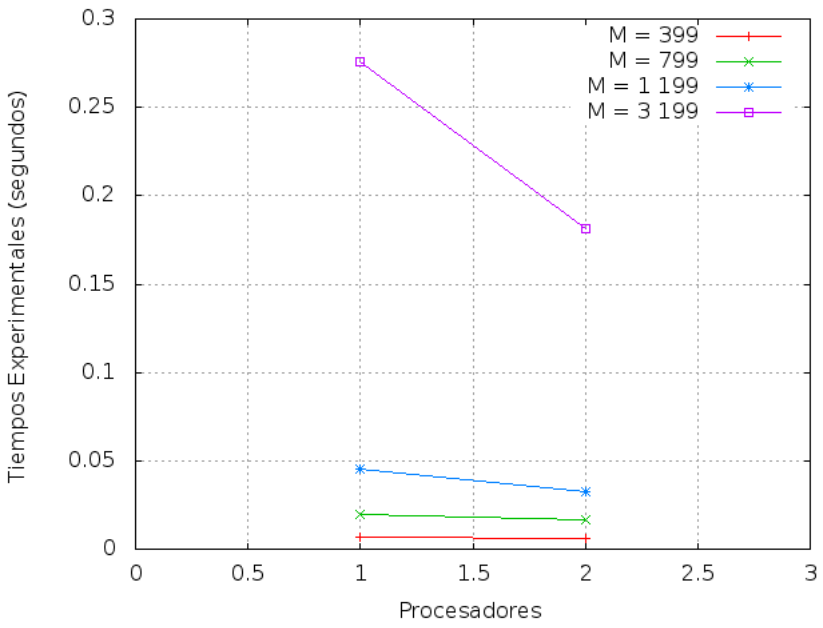

Figura 3.15: Tiempos experimentales para el Algoritmo 11.

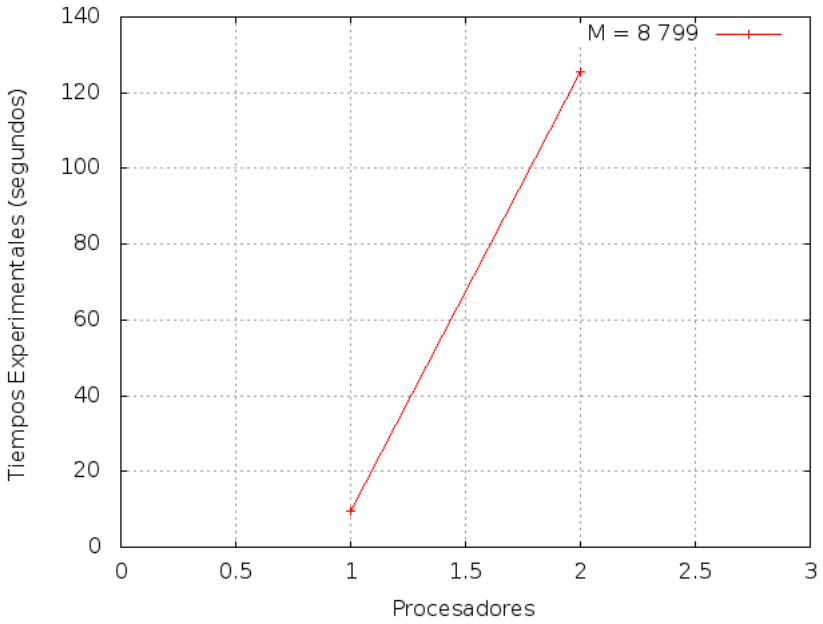

Figura 3.16: Tiempos experimentales para el Algoritmo 11 con M = 8 799.

P_{rc}	speed-up				
1	1	1	1	1	1
2	1.1841495	1.1763614	1.3820160	1.5206836	—
M	399	799	1 199	3 199	8 799

Tabla 3.12: Speed-up para el Algoritmo 11.

P_{rc}	eficiencia				
1	100 %	100 %	100 %	100 %	100 %
2	59 %	59 %	69 %	76 %	– %
M	399	799	1 199	3 199	8 799

Tabla 3.13: Eficiencia para el Algoritmo 11.

Capítulo 4

Conclusiones y Perspectivas

Existen dos grandes teorías para modelar el flujo vehicular, una tipo microscópico y otra de tipo macroscópico, cuya diferencia radica, principalmente, en que la primera considera un modelo discreto (vehículo por vehículo) y la otra considera la distribución de densidad de manera continua sobre la ruta. Esta tesis ha considerado un modelo de tipo macroscópico y donde se asume las condiciones básicas e ideales de una autopista. Este fenómeno es modelado coherentemente por una ley de conservación escalar. Para resolver esta ecuación se utilizó el método de volúmenes finitos de manera secuencial y además se introdujo una paralelización de estos. Ahora, en concordancia con los objetivos propuestos en la sección 1.4 se tiene lo siguiente

(a) Para cumplir con el objetivo (O_1) se construyeron los algoritmos 7 y 8. El Algoritmo 7 está basado en la descomposición de dominios y el algoritmo 8 en la descomposición del operador de diferencias finitas.

(b) Para el objetivo (O_2) se construyó el algoritmo 9. Este algoritmo es basado en el método de descomposición de dominios y el algoritmo 4.

(c) Para el objetivo (O_3) se diseño el algoritmo 10 el cuál está basado en la descomposición de dominios y el método de Godunov para la ecuación de Burger, presentado en su forma secuencial en el algoritmo 10.

(d) Para el objetivo (O_4) se constroyó el algoritmo 11 el cuál es basado en la descomposición de dominios y el método de Godunov para el modelo LWR, presentado en su forma secuencial en el algoritmo 6.

(d) Para el objetivo (O_4) se analizó en detalle el algoritmo 7 y se obtuvo que es de orden $O(n^2)$, donde $n = \text{máx}\{M, N\}$ con M el número de particiones de la variable espacial y N el número de particiones de la variable temporal. Un resultado análogo es válido para los otros algoritmos secuenciales, dado que están basados en volúmenes finitos explícitos.

A partir de las simulaciones numéricas en general se concluye que: (i) Los algoritmos paralelizados propuestos, en todos los casos reducen sustancialmente los tiempos de ejecución secuenciales. Aunque, muestran un comportamiento irregular para ciertos valores de M y esto es debido fundamentalmente al costo de paso de mensajes de datos; (ii) Los algoritmos

son eficientes; y (iii) Los algoritmos tienen spped-up aceptables para pocos procesadores. Además, se concluye que el esquema híbrido paralelizado responde directamente al problema que se planteó en la tesis para el caso lineal con velocidad constante. Sin embargo, este problema es abierto para el caso de la advección variable y para el caso no lineal.

Por otro lado, en cuanto a las perspectivas de esta tesis se señalan fundamentalmente las siguientes:

- Extender los algoritmos para aproximar sistemas de leyes de conservación unidimensionales, a fin de poder simular situaciones mucho más reales del tráfico vehicular. Incluyendo modelos de tráfico vehicular en redes tal como el presentado en [32].
- Diseñar algoritmos paralelos para aproximar sistemas de leyes de conservación en dos y tres dimensiones, a fin de simular sistemas análogos al del tráfico vehicular tales como la cromatografía o la sedimentación polidispersa [3, 42].
- Encontrar las condiciones que permitan paralelizar el esquema híbrido en el caso no lineal.

Además, algunas de las ideas de paralelización introducidas pueden ser aplicadas a otros modelos que dependen de una variable temporal.

Bibliografía

[1] *Foro de Mpi*. http://home.ifi.uio.no/kalie/fronttrack/ftrack/scalar.html (Accedido el 28 de mayo 2013).

[2] A. Aw and M. Rascle. Resurrection of "second order" models of traffic flow. *SIAM J. Appl. Math.*, 60(3):916–938 (electronic), 2000.

[3] S. Berres, R. Bürger, and E. M. Tory. Mathematical model and numerical simulation of the liquid fluidization of polydisperse solid particle mixtures. *Comput. Vis. Sci.*, 6(2-3):67–74, 2004.

[4] Raimund Bürger, Aníbal Coronel, and Mauricio Sepúlveda. A semi-implicit monotone difference scheme for an initial-boundary value problem of a strongly degenerate parabolic equation modeling sedimentation-consolidation processes. *Math. Comp.*, 75(253):91–112 (electronic), 2006.

[5] A.T. Chronopoulos and C.M. Johnston. A real-time traffic simulation system. *Vehicular Technology, IEEE Transactions on*, 47(1):321–331, 1998.

[6] A. Coronel and D. Crisosto. *Leyes de conservación en tráfico vehicular*. Universidad del Bio-Bio, Chile, 2009.

[7] Aníbal Coronel, Patricio Cumsille, and Rodrigo Quezada. A hybrid numerical method for shock capture in scalar conservation laws. *Rev. Integr. Temas Mat.*, 28(2):111–132, 2010.

[8] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.*, 100(1):32–74, 1928.

[9] Constantine M. Dafermos. *Hyperbolic conservation laws in continuum physics*, volume 325 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 2000.

[10] Idalia Flores de la Mota. *Apuntes de Programación entera*. Universidad Autónoma de México, Facultad de Ingeniería, México, 2002.

[11] J. Dongarra and T. Dunigan. *Message-passing performance of various computers*. Department of Computer Science, University of Tennessee, 1995.

[12] S. Doucouré. *Méthodes de décomposition de domaines pour les équations de Navier-Stokes en jonction fleuve/océan et les lois de conservation scalaires.* PhD thesis, Université de Neuchătel, 2012.

[13] Lawrence C. Evans. *Partial Differential Equations (Graduate Studies in Mathematics, V. 19) GSM/19.* American Mathematical Society, June 1998.

[14] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[15] Mauro Garavello, Roberto Natalini, Benedetto Piccoli, and Andrea Terracina. Conservation laws with discontinuous flux. *NHM*, 2(1):159–179, 2007.

[16] Mauro Garavello and Benedetto Piccoli. *Traffic flow on networks*, volume 1 of *AIMS Series on Applied Mathematics.* American Institute of Mathematical Sciences (AIMS), Springfield, MO, 2006. Conservation laws models.

[17] Edwige Godlewski and Pierre-Arnaud Raviart. *Hyperbolic systems of conservation laws*, volume 3/4 of *Mathématiques & Applications (Paris) [Mathematics and Applications].* Ellipses, Paris, 1991.

[18] Pedro Gonzáles and Casanova Henríquez. *Tortas Hiperbólicas: ¿Cómo se comen?* Notas de la Escuela de Verano en Ecuaciones Diferenciales, 6-10 junio 2011, Instituto de Matemáticas, Universidad Nacional Autónoma de México, 2011.

[19] V. Guinot. *Godunov-Type Schemes: An Introduction for Engineers.* Elsevier Science & Technology Books, 2003.

[20] Helge Holden and Nils Henrik Risebro. *Front tracking for hyperbolic conservation laws*, volume 152 of *Applied Mathematical Sciences.* Springer-Verlag, New York, 2002.

[21] Eberhard Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. *Comm. Pure Appl. Math.*, 3:201–230, 1950.

[22] C.M. Johnston and A.T. Chronopoulos. The parallelization of a highway traffic flow simulation. In *Frontiers of Massively Parallel Computation, 1999. Frontiers '99. The Seventh Symposium on the*, pages 192–199, 1999.

[23] T. Kiesling and J. Luthi. Towards time-parallel road traffic simulation. In *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on*, pages 7–15, 2005.

[24] S. N. Kružkov. First order quasilinear equations in several independent variables. *Mathematics of the USSR-Sbornik*, 10(2):217–243, February 1970.

[25] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[26] Eil Kwon, Kyeongah Yoo, and Haesun Park. Parallel simulation of freeway traffic flows on a personal computer-based, distributed computing system. In *Vehicle Navigation and Information Systems Conference, 1996. VNIS '96*, volume 7, pages 36–42, 1996.

[27] Chang-Jen Lan and Shaw-Pin Miaou. Real-time prediction of traffic flows using dynamic generalized linear models. *Transportation Research Record: Journal of the Transportation Research Board*, 1678:168–178, 1999.

[28] Sangsoo Lee and Daniel B. Fambro. Application of subset autoregressive integrated moving average model for short-term freeway traffic volume forecasting. *Journal of the Transportation Research Board*, 1678:179–188, 1999.

[29] Randall J. LeVeque. *Finite volume methods for hyperbolic problems.* Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2002.

[30] M H Lighthill and G B Whitham. On kinematic waves II: a theory of traffic flow on long, crowded roads. In *Proceedings of the Royal Society of London series A*, volume 229, pages 317–345, 1955.

[31] Yadong Lu, S.C. Wong, Mengping Zhang, Chi-Wang Shu, and Wenqin Chen. Explicit construction of entropy solutions for the Lighthill-Whitham-Richards traffic flow model with a piecewise quadratic flow-density relationship. *Transportation Research Part B: Methodological*, 42(4):355–372, May 2008.

[32] Jennifer McCrea and Salissou Moutari. A hybrid macroscopic-based model for traffic flow in road networks. *European Journal of Operational Research*, 207(2):676–684, 2010.

[33] Haim Nessyahu and Eitan Tadmor. Nonoscillatory central differencing for hyperbolic conservation laws. *J. Comput. Phys.*, 87(2):408–463, 1990.

[34] Sebastian Noelle. The MoT-ICE: a new high-resolution wave-propagation algorithm for multidimensional systems of conservation laws based on Fey's method of transport. *J. Comput. Phys.*, 164(2):283–334, 2000.

[35] A. Louise Perkins and Garry Rodrigue. A domain decomposition method for solving a two-dimensional viscous Burgers' equation. *Appl. Numer. Math.*, 6(4):329–340, 1990.

[36] Benedetto Piccoli and Andrea Tosin. Vehicular traffic: A review of continuum mathematical models. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 9727–9749. Springer, 2009.

[37] P I Richards. Shock waves on a highway. *Operations Research*, 4:42–51, 1956.

[38] P. Rodriguez and D. Risso. *Apunte de Computación Paralela.* Universidad del Bío-Bío, Facultad de Ciencias Empresariales, Chile, 2004.

[39] B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 2004.

[40] Shiliang Sun, Changshui Zhang, and Guoqiang Yu. A bayesian network approach to traffic flow forecasting. *Trans. Intell. Transport. Sys.*, 7(1):124–132, March 2006.

[41] Andrea Toselli and Olof Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, 2004.

[42] G. B. Whitham. *Linear and nonlinear waves*. Wiley-Interscience [John Wiley & Sons], New York, 1974. Pure and Applied Mathematics.

[43] H. B. Yin, S. C. Wong, J. M. Xu, and C. K. Wong. Urban traffic flow prediction using a fuzzy-neural approach rid a-7258-2008. *Transportation Research Part C-emerging Technologies*, 10(2):85–98, Apr 2002.

[44] Guoqiang Yu, Jianming Hu, Changshui Zhang, Like Zhuang, and Jingyan Song. Short-term traffic flow forecasting based on markov chain model. In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, pages 208–212, 2003.

Apéndices

Apéndice A

Algoritmo Secuencial LWR

```
/*|=========================================================================|
  | Este programa contiene el codigo necesario para resolver el MODELO  LWR  |
  |                      u_t+f(u).u_x=0,  Implementacion  f(u) GREENSHIELDS  |
  | ------------------------------------------------------------------------|
  |Autor: Lenin Quiñones Huatangari                                          |
  |Version de Octubre 2012                                                   |
  |=========================================================================|*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
#include<math.h>
#define MINIMO(A,B) ((A) <= (B) ? (A) : (B))
#define MAXIMO 0.75             // valor maximo del flujo
#define DENSIDADCRITICA 0.1     // punto critico de la densidad


/*|-------------------------------------------------------------------------|
  |                            Estructuras Necesarias                       |
  |-------------------------------------------------------------------------|
  |   He considerado las siguientes estructuras "DatosFisicos", ademas he utiliza_|
  | do estructuras anidadas para poder crear una estructura mas general.     |
  |He tomado en cuenta las                                                   |
  | estructura de datos iniciales "malla" y a partir de esta he creado la    |
  | estructura "mallaCompleta". Se ha utilizado typedef adecuadamente.       |
  |-------------------------------------------------------------------------*/

// estructura DatosFisicos
typedef struct DATOS_FISICOS
{
        float ul;
        float ur;

}DatosFisicos;
```

```
// estructura malla
typedef struct MALLA
{
        float l0,l1,t0,t1,CFL;
        int   M;
}malla;


//estructura mallaCompleta
typedef struct MALLA_COMPLETA
{
        malla DatosRequeridos;
        float DeltaX,DeltaT;
        int N;
}mallaCompleta;


/*|-------------------------------------------------------------------------|
  |                  Funciones de la Ecuación de transporte LWR             |
  |-------------------------------------------------------------------------|
  |   Se necesitan de las Condiciones Iniciales y el flujo de la Ecuación.  |
  |                      u0(x)         :      Condición Inicial             |
  |           f(u)=15*u*(1-u/0.2) :     Flujo GREENSHIELDS para el Transporte |
  |-------------------------------------------------------------------------*/

// Funciones de las Condiciones Iniciales
float CondicionInicial1(float x)
{
       if(x<=0.0)
               return (0.0);
       else
               return(0.09);
}

float CondicionInicial2(float x)
{
       if(x<=0.0)
               return(0.09);
       else
               return (0.0);
}

float CondicionInicial3(float x)
{
       if((x >=-1.0)&& (x <= 1))
               return(0.0);
       else
               return (0.09);
}
```

```
float CondicionInicial4(float x)
{
       if((x >=-1.0)&& (x <= 1))
              return(0.09);
       else
              return (0.0);
}

// Función del flujo de GREENSHIELDS...
float Greenshields(float x)
{
            float resultado;
            if ((x > 0.0) && (x <0.2))
            {
               resultado= 15.0*x*(1.0-(x/0.2));
               return (resultado);
            }
            else
               return (0.0);
}


/*|-------------------------------------------------------------------------|
  |              Funciones Necesarias para       C F L                      |
  |-------------------------------------------------------------------------|
  |   He considerado las siguientes funciones "derivada","máximo" ademas de  |
  |   una función "CondicionCFl".                                            |
  |                                                                          |
  |                (DeltaT/Delta X) * Máximo |f‘(u)| <= 1          Cond. CFL  |
  |                                    u0                                    |
  |                                                                          |
  |-------------------------------------------------------------------------*/

// Derivada en una pto y una función dada
float derivada(float u, float(*fpuntero)(float))
{
       float resultado, h=1e-6;
       resultado = ((*fpuntero)(u+h)-(*fpuntero)(u))/h;
       return(resultado);
}

//Implementación del máximo de un vector
float maximo(float vector[], int num_elem)
{
      int i;
      float mayor;
      mayor= vector[0];

      for (i = 1; i < num_elem; i++)   // buscamos
```

```
      {
          if (mayor < vector[i])       // si hay otro mayor lo cambiamos
          {
                    mayor = vector[i];
          }
      }
      return(mayor);
}

//Implementacion de la CONDICION CFL
float CondicionCFL(float ValorMaximo,float TiempoFinal, float TiempoInicial,
                                          float Distancia, float Constante)
{
float ValorFinal;
ValorFinal=(int)ceil((double)(ValorMaximo*(TiempoFinal-TiempoInicial)/
           (Distancia*Constante)));
return(ValorFinal);
}

/*float FlujoDerivada(float x)
{
            float derivando;

            derivando =15.0-150.0*x;
            return (derivando);

}
*/

float FlujoInversaDerivada(float x)
{
            float inversa;
            if ((x > -15.0) && (x <15.0))
            {
                inversa = 0.1 - 0.007*x;
                return (inversa);
            }
            else
                return (0.0);
}


/*|-------------------------------------------------------------------------|
  |            Problemas de RIEMANN para modelo LWR de Transporte           |
  |-------------------------------------------------------------------------|
  |   Esta es una funcion principal en la Ecuación del modelo LWR, de donde |
  |  se pueden obtener "Ondas de Choque" u "Ondas de Rarefacción".          |
  |-------------------------------------------------------------------------*/
// Funcion que hace las aproximaciones
float FuncionRiemann(float x, float y)
```

```
{
      //Rarefaccion
      if (x > y)
      {
              //sigma[k] = 0.5*(u[k] + uaux[k]);
              if(x <= DENSIDADCRITICA)
              {
                          return (Greenshields(x));
              }
              else if(x > DENSIDADCRITICA)
              {
                          return (Greenshields(y));
              }
              else
              {
                          return (MAXIMO);
              }

      }
      //Choque
      else
      {
              if(y <= DENSIDADCRITICA)
              {
                         return (Greenshields(x));
              }
              else if(x >= DENSIDADCRITICA)
              {
                         return (Greenshields(y));
              }
              else
              {
                         return (MINIMO(Greenshields(x),Greenshields(y)));
              }
      }

}

/*|-------------------------------------------------------------------------|
  |                           Ecuación LWR                                  |
  |-------------------------------------------------------------------------
  |   Esta es la funcion principal del programa, para esto se necesitan las dos   |
  | estructuras definidas anteriormente "mallaCompleta" y "DatosFisicos"          |
  |-------------------------------------------------------------------------*/
void EcuacionLWR(mallaCompleta CalculosNecesarios,DatosFisicos CondicionesIniciales,
                                    float *u0,float *uf,float *x,float *SolExacta)
{
       int h,n,j,i;
       float lambda,AuxiliarCFL;
       float *u,*v,*VectorDerivada;
```

```
    float (*fptr)(float); // declara puntero a una funcion
    fptr= Greenshields;


    //Localizacion de MEMORIA DINAMICA
    u=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
    v=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
    VectorDerivada= (float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)
                                                          *sizeof(float));


if(u==NULL)
    {
              puts("NO se puede asignar memoria");
              exit(1);
    }


    //Desplazamiento en el Eje X
    CalculosNecesarios.DeltaX = (CalculosNecesarios.DatosRequeridos.l1
                               -CalculosNecesarios.DatosRequeridos.l0)/
                               (CalculosNecesarios.DatosRequeridos.M);

    //CONDICION INICIAL
    for(i=0;i<=CalculosNecesarios.DatosRequeridos.M;i++)
    {
                         x[i]   =     CalculosNecesarios.DatosRequeridos.l0 +
                                                 i*CalculosNecesarios.DeltaX;
              u0[i]=u[i]       =     CondicionInicial2(x[i]);
              VectorDerivada[i]=   fabs(derivada(CondicionInicial2(x[i]),fptr));
    }

    AuxiliarCFL = maximo(VectorDerivada,CalculosNecesarios.DatosRequeridos.M);


    //Numero de Iteraciones en el eje del TIEMPO.
    CalculosNecesarios.N      = CondicionCFL(AuxiliarCFL,
                                CalculosNecesarios.DatosRequeridos.t1,
                                CalculosNecesarios.DatosRequeridos.t0,
                                CalculosNecesarios.DeltaX,
                                CalculosNecesarios.DatosRequeridos.CFL);

    CalculosNecesarios.DeltaT = (CalculosNecesarios.DatosRequeridos.t1
                                -CalculosNecesarios.DatosRequeridos.t0)/
                                (float)CalculosNecesarios.N;
    lambda                    = CalculosNecesarios.DeltaT /
                                CalculosNecesarios.DeltaX;


    // Solución exacta de un Problema de Riemann para la Ecuación de LWR
```

```
if (CondicionesIniciales.ul > CondicionesIniciales.ur)
{
//(CondicionesIniciales.ul > CondicionesIniciales.ur)
           for(j=0;j <=CalculosNecesarios.DatosRequeridos.M;j++)
           {
                       if(x[j] <= (derivada(CondicionesIniciales.ul,fptr)*
                                   CalculosNecesarios.DatosRequeridos.t1))
                            v[j]=CondicionesIniciales.ul;
                       else
                            if(x[j] >= (derivada(CondicionesIniciales.ur,
                                       fptr)*CalculosNecesarios.DatosRequeridos.t1))
                                  v[j]=CondicionesIniciales.ur;
                            else
                                  v[j]=FlujoInversaDerivada(x[j]
                                        /CalculosNecesarios.DatosRequeridos.t1);
           }
}
else
{
           for(i=0;i <= CalculosNecesarios.DatosRequeridos.M;i++)
           {
                   if(x[i] <((Greenshields(CondicionesIniciales.ul)-
                             Greenshields(CondicionesIniciales.ur))/
                             (CondicionesIniciales.ul-
                             CondicionesIniciales.ur))*
                             CalculosNecesarios.DatosRequeridos.t1)
                             v[i]=CondicionesIniciales.ul;
                    else
                             v[i]=CondicionesIniciales.ur;
           }
}

//Solución aproximada de la Ecuación de LWR usando METODO de GODUNOV
for(n=0; n<CalculosNecesarios.N;n++)
{
            //EVOLUCION
            for(h=1;h<CalculosNecesarios.DatosRequeridos.M;h++)
            {
                        u[h] = u[h]+(CalculosNecesarios.DeltaT
                                              /CalculosNecesarios.DeltaX)
                            *( FuncionRiemann(u[h-1],u[h])
                                         -FuncionRiemann(u[h],u[h+1]));
            }
            u[0]=u[1];
            u[CalculosNecesarios.DatosRequeridos.M-1]=
                   u[CalculosNecesarios.DatosRequeridos.M];
}


//SALIDA
```

```
        for(i=0;i<=CalculosNecesarios.DatosRequeridos.M;i++)
        {
                    uf[i]=u[i];
                    SolExacta[i]=v[i];
        }

        free(v);
        free(u);
        free(VectorDerivada);
}


/*|-------------------------------------------------------------------------------|
  |                              FUNCION MAIN                                     |
  |-------------------------------------------------------------------------------
  |   En esta funcion se define las variables, que vienen a ser las estructuras   |
  | "mallaCompleta", "DatosFisicos" además de los tipos de datos float,char       |
  | y un archivo para la escritura de los datos.                                  |
  | En este programa recibe por teclado los elementos de las estructuras       |
  | "malla" y "DatosFisicos". Ademas se llama a la funcion "AdveccionPositiva".   |
  |  Finalmente utilizaremos los ficheros para escribir los resultados obtenidos. |
  |-------------------------------------------------------------------------------*/
int main(int argc, char **argv)
{
        /* _________________VARIABLES_____________________ */
        mallaCompleta CalculosNecesarios;
        DatosFisicos CondicionesIniciales;
        int n,j,i;
        float lambda;
        float *u0,*uf,*x,*SolExacta;
        //el nombre del archivo
        char nombre[30]="LWRGodunovSecuencial.dat";
        FILE* fichero;

        /* _________________DATOS INICIALES_________________ */
        printf("\nINGRESE LOS DATOS DE LA MALLA\n");
        printf("\n**********************************\n");
        printf("\nINGRESAR el menor valor en el EJE X\n");
        scanf("%f",&CalculosNecesarios.DatosRequeridos.l0);
        printf("\nINGRESAR el mayor valor en el EJE X\n");
        scanf("%f",&CalculosNecesarios.DatosRequeridos.l1);
        printf("\nINGRESAR el menor valor con respecto a  T\n");
        scanf("%f",&CalculosNecesarios.DatosRequeridos.t0);
        printf("\nINGRESAR el mayor valor con respecto a  T\n");
        scanf("%f",&CalculosNecesarios.DatosRequeridos.t1);
        printf("\nINGRESAR el  valor del M (numero de partes
                                    a fraccionar eje X)\n");
        scanf("%d",&CalculosNecesarios.DatosRequeridos.M);
        printf("\nINGRESAR la constante CFL (para garantizar
                                estabilidad del METODO)\n");
```

```
scanf("%f",&CalculosNecesarios.DatosRequeridos.CFL);
printf("\n**********************************\n");

printf("\nINGRESE EL VALOR INICIAL\n");
printf("\n**********************************\n");
printf("\nINGRESAR El valor izquierdo del PROBLEMA
                                    DE RIEMANN\n");
scanf("%f",&CondicionesIniciales.ul);
printf("\nINGRESAR El valor derecho del PROBLEMA
                                    DE RIEMANN\n");
scanf("%f",&CondicionesIniciales.ur);
printf("\n**********************************\n");

printf ( "\n" );
printf ( "  Calcula una solucion aproximada de la\n" );
printf ( "  la Ecuacion con Flujo Lineal Constante:\n" );
printf ( "\n" );
printf ( "    dU/dt + u.dU/dx = 0\n" );
printf ( "\n" );
printf ( "  para \n %f = < x < = %f\n",CalculosNecesarios.DatosRequeridos.l0,
                                    CalculosNecesarios.DatosRequeridos.l1 );
printf ( "\n" );
printf ( "  y \n %f =  < t <= %f\n", CalculosNecesarios.DatosRequeridos.t0,
                                     CalculosNecesarios.DatosRequeridos.t1 );
printf ( "\n" );

struct timeval t_ini, t_fin;
double  tiempo;


/*_________________ Iniciando a medir el tiempo_____________________*/
gettimeofday(&t_ini,NULL);
// pedir memoria para u0,uf,x
u0=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
uf=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
x=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
SolExacta=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)
                                                *sizeof(float));

EcuacionLWR(CalculosNecesarios,CondicionesIniciales,u0,uf,x,SolExacta);

gettimeofday(&t_fin,NULL);
/*_________________Terminando de medir el tiempo_____________________*/



/*_____________________FICHEROS_________________________*/
// Abrir y Manejo de Errores
fichero = fopen( nombre, "w" );
printf( "Fichero: %s (para escritura) -> ", nombre );
```

```
        if( fichero != NULL)
            printf( "creado (ABIERTO)\n" );
        else
        {
            printf( "Error (NO ABIERTO)\n" );
                    return 1;
        }

        for(i=0;i<=CalculosNecesarios.DatosRequeridos.M;i++)
            fprintf(fichero, "\t%f\t%f\t%f\t%f\n",x[i],u0[i],uf[i]
                                                    ,SolExacta[i]);

        //garantizando el  ESTADO del fichero.
        fprintf( stdout,"Datos guardados en el fichero: %s\n", nombre );
        if( !fclose(fichero) )
                  printf( "Fichero cerrado\n" );
        else
        {
                  printf( "Error: fichero NO CERRADO\n" );
                  return 1;
        }
        free(x);
        free(uf);
        free(u0);
        free(SolExacta);


        tiempo=(double)(t_fin.tv_sec+(double)t_fin.tv_usec/1000000)-
              (double)(t_ini.tv_sec+(double)t_ini.tv_usec/1000000);


        printf("\nTiempo Empleado %.6g\n",tiempo);

        return 0;
}
```

Apéndice B

Algoritmo Paralelo LWR

```
/*|=========================================================================|
  | Este programa contiene el codigo necesario para resolver el MODELO  LWR  |
  |                      u_t+f(u).u_x=0,  Implementacion  f(u) GREENSHIELDS  |
  | ------------------------------------------------------------------------
  |Autor: Lenin Quiñones H                                                   |
  |Version de Octubre 2012                                                   |
  |=========================================================================|*/
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
#include<math.h>
#define MINIMO(A,B) ((A) <= (B) ? (A) : (B))
#define MAXIMO 0.75                                  // valor maximo del flujo
#define DENSIDADCRITICA 0.1                          // punto critico de la densidad


/*|-------------------------------------------------------------------------|
  |                            Estructuras Necesarias                       |
  |-------------------------------------------------------------------------
  |   He considerado las siguientes estructuras "DatosFisicos", ademas he utiliza_|
  | do estructuras anidadas para poder crear una estructura mas general.     |
  |He tomado en cuenta las                                                   |
  | estructura de datos iniciales "malla" y a partir de esta he creado la    |
  | estructura "mallaCompleta". Se ha utilizado typedef adecuadamente.       |
  |-------------------------------------------------------------------------*/

// estructura DatosFisicos
typedef struct DATOS_FISICOS
{
        float ul;
        float ur;
}DatosFisicos;

// estructura malla
```

```
typedef struct MALLA
{
        float l0,l1,t0,t1,CFL;
        int   M;
}malla;

//estructura mallaCompleta
typedef struct MALLA_COMPLETA
{
        malla DatosRequeridos;
        float DeltaX,DeltaT;
        int N;
}mallaCompleta;


/*|-------------------------------------------------------------------------|
  |                   Funciones de la Ecuación de Flujo LWR                 |
  |-------------------------------------------------------------------------|
  |   Se necesitan de las Condiciones Iniciales y el flujo de la Ecuación.  |
  |    |
  |    u0(x)        :       Condición Inicial                  |
  |     f(u)=0.5*u^2 :       Flujo para la Advección          |
  |   |
  |-------------------------------------------------------------------------*/


// Funciónes de las Condiciónes Iniciales
float CondicionInicial1(float x)
{
if(x<=0.0)
               return (0.0);
else
               return(0.09);
}


float CondicionInicial2(float x)
{
if(x<=0.0)
               return(0.09);
else
               return (0.0);
}


float CondicionInicial3(float x)
{
if((x >=-1.0)&& (x <= 1))
               return(0.0);
else
```

```
                return (0.09);
}

float CondicionInicial4(float x)
{
if((x >=-1.0)&& (x <= 1))
                return(0.09);
else
                return (0.0);
}


// Función del flujo de GREENSHIELDS...
float LWR(float x)
{
        float resultado;

        if ((x > 0.0) && (x <0.2))
        {
                resultado= 15.0*x*(1.0-(x/0.2));
                return (resultado);
        }
        else
                return (0.0);
}


/*|-------------------------------------------------------------------------------|
  |              Funciones Necesarias para       C F L                            |
  |-------------------------------------------------------------------------------
  |   He considerado las siguientes funciones "derivada","máximo" ademas de       |
  |   una función "CondicionCFl".                                                 |
  |                                                                               |
  |                (DeltaT/Delta X) * Máximo |f‘(u)| <= 1          Cond. CFL      |
  |                                    u0                                         |
  |                                                                               |
  |-------------------------------------------------------------------------------*/

// Derivada en una pto y una función dada
float derivada(float u, float(*fpuntero)(float))
{
       float resultado, h=1e-6;
       resultado = ((*fpuntero)(u+h)-(*fpuntero)(u))/h;
       return(resultado);
}


//Implementación del máximo de un vector
float maximo(float vector[], int num_elem)
{
```

```
      int i;
      float mayor;
      mayor= vector[0];

      for (i = 1; i < num_elem; i++)   // buscamos
      {
          if (mayor < vector[i])       // si hay otro mayor lo cambiamos
          {
                    mayor = vector[i];
          }
      }
      return(mayor);
}


//Implementacion de la CONDICION CFL
float CondicionCFL(float ValorMaximo,float TiempoFinal, float TiempoInicial,
                                          float Distancia, float Constante)
{
float ValorFinal;
ValorFinal=(int)ceil((double)(ValorMaximo*(TiempoFinal-TiempoInicial)/
           (Distancia*Constante)));
return(ValorFinal);
}




//float FlujoDerivada(float x)
//{
  //      float derivando;

    //    derivando =15.0-150.0*x;
     //   return (derivando);
//}


float FlujoInversaDerivada(float x)
{
        float inversa;
        if ((x > -15.0) && (x <15.0))
        {
                inversa = 0.1 - 0.007*x;
                return (inversa);
        }
        else
                return (0.0);
}
```

```
/*|-------------------------------------------------------------------------|
  |                 Problemas de RIEMANN para Ec. LWR                       |
  |-------------------------------------------------------------------------
  |   Esta es una funcion principal en la Ecuación de Burguer, de donde     |
  |  se pueden obtener "Ondas de Choque" u "Ondas de Rarefacción".          |
  |-------------------------------------------------------------------------*/
float Funcion(float x, float y)
{
//Rarefaccion
if (x > y)
{
//sigma[k] = 0.5*(u[k] + uaux[k]);
if(x <= DENSIDADCRITICA)
{
                        return (LWR(x));
}
else if(x > DENSIDADCRITICA)
{
                        return (LWR(y));
}
else
{
                        return (MAXIMO);
}

}
//Choque
else
{
if(y <= DENSIDADCRITICA)
{
                        return (LWR(x));
}
else if(x >= DENSIDADCRITICA)
{
                        return (LWR(y));
}
else
{
                        return (MINIMO(LWR(x),LWR(y)));
}
}
}


/*|-------------------------------------------------------------------------|
  |                     Ecuación LWR  Paralelo                              |
  |-------------------------------------------------------------------------
  |   Esta es la funcion principal del programa, para esto se necesitan las dos   |
  | estructuras definidas anteriormente "mallaCompleta" y "DatosFisicos" y además |
```

```
  | los pasos de mensajes adecuados en las esquinas.                              |
  |-------------------------------------------------------------------------------*/

void EcuacionLWRParalelo(int id,int p)
{
mallaCompleta CalculosNecesarios;
DatosFisicos CondicionesIniciales;
int h,n,i,j,r,proc,tag,subarreglo,aux,master;
float lambda,temporal,temporal1,AuxiliarCFL;
float *u,*u0,*v,*buf,*uf,*x,*y,*VectorDerivada;
FILE* fichero;
float (*fptr)(float);               // declara puntero a una funcion

fptr= LWR;

MPI_Status status;
char nombre[30]="LWRGodunovParalelo.dat";

j=1;
master =0;


//Localizacion de MEMORIA DINAMICA para la condicion CFL
y=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+2)*sizeof(float));
v=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+2)*sizeof(float));
VectorDerivada=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+2)
               *sizeof(float));
u0=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+2)*sizeof(float));

//tamaño del trabajo para cada Procesador
subarreglo               = (CalculosNecesarios.DatosRequeridos.M+1)/p;
aux                      = p*subarreglo;


//Localizacion de MEMORIA DINAMICA para cada sub-arreglo
u=(float*)malloc((subarreglo+3)*sizeof(float));
//buf=(float*)malloc(aux*sizeof(float));
buf=(float*)malloc((CalculosNecesarios.DatosRequeridos.M+1)*sizeof(float));
x=(float*)malloc((subarreglo+1)*sizeof(float));
uf=(float*)malloc((subarreglo)*sizeof(float));


if(u==NULL)
{
               puts("NO se puede asignar memoria");
               exit(1);
}

CalculosNecesarios.DatosRequeridos.l0 = -2000;
CalculosNecesarios.DatosRequeridos.l1 = 2000;
```

```
CalculosNecesarios.DatosRequeridos.t0 = 0;
CalculosNecesarios.DatosRequeridos.t1 = 100;
CalculosNecesarios.DatosRequeridos.M  = 399;
CalculosNecesarios.DatosRequeridos.CFL= 0.75;
CondicionesIniciales.ul               = 0;
CondicionesIniciales.ur               = 0.09;


//Desplazamiento en el Eje X
CalculosNecesarios.DeltaX = (CalculosNecesarios.DatosRequeridos.l1
                                -CalculosNecesarios.DatosRequeridos.l0)/
                                (CalculosNecesarios.DatosRequeridos.M);

        //Operaciones auxiliares para la Condición CFL
for(i=0;i<=CalculosNecesarios.DatosRequeridos.M;i++)
{
               y[i]              = CalculosNecesarios.DatosRequeridos.l0+
                                   i*CalculosNecesarios.DeltaX;
               u0[i]             = CondicionInicial1(y[i]);
               VectorDerivada[i] = derivada(CondicionInicial1(y[i]),fptr);
}

AuxiliarCFL = maximo(VectorDerivada,CalculosNecesarios.DatosRequeridos.M);

//CONDICION INICIAL
for(i=1;i<=subarreglo;i++)
{
               x[i] = ( ( float ) (         id * subarreglo + i - 1 )
                      * CalculosNecesarios.DatosRequeridos.l1
                      + ( float ) ( p * subarreglo - id * subarreglo - i)
                      * CalculosNecesarios.DatosRequeridos.l0 )
                      / ( float ) ( p * subarreglo              - 1 );

                  //x[i]=CalculosNecesarios.DatosRequeridos.l0+(id*(subarreglo)
                         *CalculosNecesarios.DeltaX + (i-1)*CalculosNecesarios.DeltaX);
u[i] = CondicionInicial1(x[i]);
}


//Numero de Iteraciones en el eje del TIEMPO.
CalculosNecesarios.N      = CondicionCFL(AuxiliarCFL,
                            CalculosNecesarios.DatosRequeridos.t1,
                                           CalculosNecesarios.DatosRequeridos.t0,
                                           CalculosNecesarios.DeltaX,
                                           CalculosNecesarios.DatosRequeridos.CFL);

CalculosNecesarios.DeltaT =(CalculosNecesarios.DatosRequeridos.t1
                                          -CalculosNecesarios.DatosRequeridos.t0)/
                                           (float)CalculosNecesarios.N;
lambda                    = CalculosNecesarios.DeltaT /
```

```
                        CalculosNecesarios.DeltaX;


//Solución aproximada de la Ecuación de LWR usando METODO de GODUNOV
for(n=1;n<=CalculosNecesarios.N;n++)
{
                /*
                  Send u[1] to ID-1.
                */

                if ( 0 < id )
                {
                        tag = 1;
                        MPI_Send ( &u[1], 1, MPI_FLOAT, id-1, tag,
                                                              MPI_COMM_WORLD );
                }

                /*
                  Receive u[N+1] from ID+1.
                */
                if ( id < p-1 )
                {
                        tag = 1;
                        MPI_Recv ( &u[subarreglo+1], 1,  MPI_FLOAT, id+1, tag,
                                                   MPI_COMM_WORLD, &status );
                }


                /*
                   Send u[subarreglo-1] to ID+1.
                */
                if(id < p-1)
                {
                       tag = 2;
                       MPI_Send (&u[subarreglo], 1, MPI_FLOAT, id+1,
                                                 tag, MPI_COMM_WORLD);
                }


                /*
                   Receive u[0] from ID-1.
                */

                if(0 < id )
                {
                        tag = 2;
                        MPI_Recv (&u[0], 1, MPI_FLOAT, id-1, tag,
                                                MPI_COMM_WORLD, &status);
                }
```

```
if(id == 0)
                {
                        u[0]=u[1];
                }

if(id == p-1)
{
u[subarreglo]=u[subarreglo+1];
}


                //EVOLUCION
                for(h=1;h<=subarreglo;h++)
                {
                        u[h] = u[h]+lambda*( Funcion(u[h-1],u[h])-
                                                        Funcion(u[h],u[h+1]));
                }

}

//Soluciones aproximadas para cada sub-arreglo
for(i=1;i<=subarreglo;i++)
{
                uf[i]=u[i];
}


//Todos los procesadores tienen los datos
        MPI_Barrier(MPI_COMM_WORLD);


// ..................RECOGIENDO LA INFORMACION..................
MPI_Gather(uf,subarreglo,MPI_FLOAT,buf,subarreglo,MPI_FLOAT,master
                                                        ,MPI_COMM_WORLD);
        //MPI_SAFE(MPI_Gather(uf,subarreglo,MPI_FLOAT,buf,subarreglo,
                                                MPI_FLOAT,0,MPI_COMM_WORLD));

if( id == 0 )
{
      fichero = fopen ( nombre,"w" );
      for( i=0;i<=CalculosNecesarios.DatosRequeridos.M;i++)
      {
                        fprintf(fichero,"\t%f\t%f\t%f\t%f\n",y[i],u0[i],
                                                                buf[i],v[i]);
      }

}

        free(uf);
```

```
free(x);
free(y);
free(buf);
free(u);
free(v);
free(VectorDerivada);
free(u0);
}


/*|-----------------------------------------------------------------------------|
  |                              FUNCION MAIN                                   |
  |-----------------------------------------------------------------------------
  |   En esta funcion se define las variables generales de   MPI.               |
  |   Ademas se tomará la medida del tiempo utilizado en la rutina de encontrar |
  |   la solución final paralelizada entre n-procesadores.                      |
  |-----------------------------------------------------------------------------*/

main(int argc, char *argv[])
{
        int id;
        int p;
        double start,finish,time;


        MPI_Init ( &argc, &argv );
        MPI_Comm_rank ( MPI_COMM_WORLD, &id );
        MPI_Comm_size ( MPI_COMM_WORLD, &p );


        //para controlar el tiempo
        MPI_Barrier(MPI_COMM_WORLD);
        start=MPI_Wtime();

        EcuacionLWRParalelo(id,p);


        MPI_Barrier(MPI_COMM_WORLD);
        finish=MPI_Wtime();
        time=finish- start;


        if(id==0)
        {
                printf ( "\n" );
                printf ( "LWR_MPI:\n" );
                printf ( "  C/MPI version\n" );
                printf ( "\n" );
                printf ( "  Resuelve el Modelo de Transporte GREENSHIELDS.\n" );
                printf ( "    dU/dt + f(u).dU/dx = 0\n" );
```

```
                printf ( "\n" );
                printf("\nEL TIEMPO TOTAL EMPLEADO es: %f\n",time);
        }

        MPI_Finalize( );
        return 0;
}
```