UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CÁLCULO DE LAS TRAYECTORIAS DE PARTÍCULAS EN LA ATMÓSFERA MEDIANTE CÁLCULOS PARALELIZADOS HACIENDO USO INTENSIVO DE GPU

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

NICOLÁS EUGENIO OZIMICA

PROFESOR GUÍA:
MARK FALVEY

MIEMBROS DE LA COMISIÓN:
NANCY VIOLA HITSCHFELD KAHLER
MARÍA CECILIA RIVARA ZÚÑIGA

SANTIAGO DE CHILE
OCTUBRE DE 2010

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN
POR: NICOLÁS EUGENIO OZIMICA
FECHA: 19 DE OCTUBRE DE 2010
PROF. GUÍA: SR. MARK FALVEY

CÁLCULO DE LAS TRAYECTORIAS DE PARTÍCULAS EN LA ATMÓSFERA MEDIANTE CÁLCULOS PARALELIZADOS HACIENDO USO INTENSIVO DE GPU

El cálculo de la trayectoria de partículas en la atmósfera es una actividad abordada desde hace mucho tiempo, tomando en cuenta un enfoque clásico del uso de los computadores, en el cual se utilizan sus recursos como Procesador Central (CPU) y Memoria RAM. Este enfoque, que ha servido para alcanzar los logros actuales, con predicciones bastante acertadas, aún adolece de problemas claves y en cierta medida irresolubles, que se ven acentuados cuando los procesos se hacen más complejos, ya sea incorporando muchas partículas, o tomando en cuenta modelos de desplazamiento más realistas.

El presente trabajo aborda un enfoque totalmente nuevo, el cual hace uso de una componente física de los computadores llamada "Tarjeta Gráfica", la cual cuenta con su propia unidad de procesamiento gráfico llamada GPU por sus siglas en inglés. Esta componente, gracias a su gran cantidad de núcleos, ofrece la posibilidad de realizar paralelamente entre sí todos, o una parte de los cálculos que le son asignados, de una manera mucho más potente que lo conseguido en la actualidad al hacer uso solamente de CPU.

Este problema es abordado mediante la implementación de un programa en dos versiones: una para funcionar exclusivamente en CPU y la otra para hacer uso de GPU en los cálculos. De esta manera se puede contar con un método directo para comparar el desempeño de estos dos enfoques, contrastarlos entre sí, y deducir los casos en que uno supera al otro. El principal insumo en la comparación de estos cálculos es la información del pronóstico del viento.

El programa fue aplicado a una situación real: la erupción del Volcán Chaitén, para un día cuyos datos se poseían de antemano. Los resultados obtenidos fueron graficados y comparados con una imagen satelital correspondiente al mismo día simulado, siendo posible comprobar la alta similitud entre ellas. El tiempo de cálculo empleado por la versión que funciona en GPU supera en algunos casos en más de doscientas veces lo que tarda su contraparte CPU, sin afectar en gran medida la exactitud de los cálculos. Esto permite comprobar efectivamente que las GPU superan ampliamente el desempeño de las CPU, cuando el problema abordado es altamente paralelizable.

Este es un tema que no está en absoluto cerrado, por cuanto son muchos los caminos donde seguir explorando las ventajas y desventajas del uso de GPU para estos cálculos. Por ejemplo se puede ver cómo funcionan las GPU incorporando procesos físicos más complejos para el cálculo de los desplazamientos de las partículas, o se puede considerar algunas propiedades físicas de estas partículas, como lo es la masa por ejemplo.

A mi Padre: Nicolás Eugenio Guillermo Ozimica Guerra

AGRADECIMIENTOS

En primer lugar agradezco profundamente a mi Padre por todo el amor y atención que me dedicó mientras estuvimos juntos. Siempre fue suyo el objetivo que yo alcanzase la educación universitaria que me permitiese lograr una realización personal tal como yo lo desease. El solo imaginar la tremenda alegría que le hubiese producido un hecho como mi titulación, es motivo de gran satisfacción para mí. Todo el trabajo realizado en esta memoria, como también mi título de Ingeniero Civil en Computación, están totalmente dedicados a él.

Agradezco también a mi hermana Verónica, siempre presente de una u otra forma a lo largo de toda mi vida. Su apoyo moral, como si fuera una madre, ha sido imprescindible. Y junto a ella, agradezco a mi hermana Maite, pues sin el apoyo de mis dos hermanas, tanto en los momentos difíciles como en los agradables, todo hubiese sido muchísimo más difícil en mi existencia. El ser parte de una familia tan grande, diversa y acogedora como son los Ozimica, es una de las mejores características de mi vida.

Nada de lo que en esta memoria se logró hubiera sido posible sin la total ayuda de los dos profesores con los cuales trabajo profesionalmente desde hace casi tres años: Mark Falvey y Rainer Schmitz. Gracias por la confianza depositada en mí al proponerme este tema de memoria, como por los recursos invertidos en la adquisición de un computador de última generación para llevar adecuadamente a cabo este trabajo, como también por haberme permitido asistir a la conferencia "CoV 6^{th}". Agradezco también a la profesora Nancy Hitschfeld, cuyos valiosos comentarios respecto de este informe final fueron de gran ayuda.

Durante los primeros años de mi carrera recibí la oportuna asistencia de la dirección de bienestar estudiantil, apoyado siempre por la excelente asistente social María Eugenia Robinson. Su ayuda y preocupación fueron muy importantes durante aquel tiempo.

Dedico un agradecimiento especial a Jocelyn, mi polola, con cuya compañía y apoyo incondicional desde que la conocí, mi vida ha sido mucho más agradable y feliz.

Agradezco también a todos mis amigos, con los cuales he pasado entretenidos y memorables momentos en horas de estudio, reuniones, viajes. Sé que sus amistades perdurarán en el tiempo.

ÍNDICE GENERAL

1. Introducción **1**

1.1. Contexto general 1

1.2. Contexto específico 2

1.2.1. Uso de CPU para los cálculos 2

1.2.2. Nuevas perspectivas de solución: GPU 2

1.2.3. GPU y cálculo de Trayectorias en la actualidad 4

1.3. Problema por resolver 4

1.4. Motivación 5

1.5. Objetivo General 6

1.6. Objetivos Específicos 7

1.6.1. Investigar los conocimientos necesarios 7

1.6.2. Implementar cálculos 8

1.6.3. Obtener información desde archivos de datos especiales 8

1.6.4. Manejar adecuadamente las proyecciones geográficas y espaciales . 8

1.6.5. Ejecutar el programa con datos reales 9

1.6.6. Comparar entre sí las implementaciones para CPU y GPU 9

1.6.7. Implementar un visualizador para las trayectorias calculadas 10

1.7. Contenido de la memoria 11

2. Antecedentes **12**

2.1. Sistemas de coordenadas verticales 12

2.1.1. Coordenadas sigma 12

2.1.2. Coordenadas curvilíneas usadas por ARPS 13

2.2. Características de los pronósticos del viento 14

2.3. Interpolación trilineal 15
2.4. Arquitectura física de las tarjetas de video NVIDIA GeForce GTX 200 . . . 15
2.4.1. Componentes y su organización al interior de la tarjeta 16
2.4.2. Doble propósito de la arquitectura GTX 200 17
2.4.3. Texturas 18
2.5. Especificación de la arquitectura CUDA de NVIDIA 18
2.5.1. Modelo de programación de CUDA 20
2.5.2. Modelo de memoria de CUDA 22
2.5.3. C for CUDA 22
2.5.4. Capacidad de Cómputo 23
2.5.5. Un patrón típico de programación en toda aplicación CUDA 23
2.6. Algoritmos actuales para el cálculo de trayectorias 24
2.7. Trabajos actuales que abordan temas similares 25

3. Descripción de la Solución **26**
3.1. Análisis de los datos del pronóstico del viento 26
3.2. Ecuación que describe las trayectorias 27
3.3. Pruebas en MATLAB 28
3.4. Diseño conceptual de la solución 28
3.5. Estructura de los datos y planificación de los cálculos 30
3.5.1. Información de la velocidad del viento 30
3.5.2. Información de los niveles de altura 32
3.5.3. Posiciones de las partículas a lo largo de los pasos de tiempo 32
3.5.4. Precisión numérica usada para los cálculos 32
3.6. Definición de los parámetros del problema a resolver 33
3.6.1. Cálculo de los pasos de tiempo totales de la simulación 34
3.6.2. Determinación de los momentos en que se guardan resultados . . . 35
3.7. Cálculo de la memoria ocupada por el programa 36
3.8. Determinación de la interfaz del programa 37
3.9. Módulos que componen la solución 38
3.9.1. Inicialización de cronómetros 38
3.9.2. Cálculo de la memoria ocupada y declaración de variables 40

3.9.3. Lectura y almacenamiento de archivos de datos 40
3.9.4. Definición de posiciones iniciales 41
3.9.5. Distribución de partículas en el tiempo 41
3.9.6. Ciclo de cálculos 42
3.9.7. Ciclo de cálculos en CPU 43
3.9.8. Ciclo de cálculos en GPU 43
3.9.9. Almacenamiento en disco de las trayectorias calculadas 53
3.10. Graficado de resultados 53

4. Resultados **55**
4.1. Obtención de los resultados 56
4.2. Validación de los resultados 57
4.3. Análisis del desempeño de la versión GPU 58

5. Conclusiones **63**

Bibliografía **65**

Fuentes en línea **67**

A. Características del entorno de desarrollo **69**
A.1. Hardware 69
A.1.1. Propiedades específicas de la tarjeta GeForce GTX 285 70
A.2. Software 71

B. Revisión bibliográfica **72**
B.1. Procesos físicos relacionados con la meteorología 72
B.2. Lenguajes de programación 72
B.3. Arquitectura física de las tarjetas de video 73
B.4. Especificación de la arquitectura CUDA de NVIDIA 74

C. Presentación del proyecto en la Conferencia "CoV6th" **75**

ÍNDICE DE FIGURAS

1.1. GFLOP máximos entre procesadores Intel y GPUs NVIDIA [15] 3
1.2. Un ejemplo del cálculo de trayectorias [18] 5
1.3. Un ejemplo de visualización gráfica de trayectorias. [17] 10

2.1. Ejemplo del modelo de coordenadas σ. [38] 13
2.2. Cubo dentro del cual se realiza una interpolación [22] 16
2.3. Detalle de un *Stream Multiprocessor*. [11] 17
2.4. Detalle de un *Thread processing cluster*. [11] 17
2.5. Arquitectura para cómputos en paralelo, en la tarjeta GTX 280. [11] 19
2.6. Arquitectura para procesamiento gráfico, en la tarjeta GTX 280. [11] 19
2.7. Grilla de Bloques, y un Bloque de Hilos. [15] 21

3.1. Módulos que componen el diseño preliminar del programa. 29
3.2. Esquema definitivo del diseño de los módulos del sistema 39
3.3. Esquema del ciclo de cálculos en GPU 45
3.4. Ejemplo de la graficación de trayectorias usando Python 54

4.1. Gráficos de los tiempos total y sólo de cálculo empleados por CPU y GPU 59
4.2. Representación de las trayectorias calculadas sobre un mapa de la zona . . 60
4.3. Imagen satelital del día para el cual se realizó la simulación. 61

A.1. Computador DELL XPS 70
A.2. Tarjeta GeForce GTX 285 70

C.1. Afiche oficial de la conferencia "Cities on Volcanoes 6^{th}" [24]. 75
C.2. Póster presentado en la conferencia internacional "Cities on Volcanoes 6^{th}". 77

ÍNDICE DE CUADROS

2.1. Descripción de los tipos de memoria disponibles en las tarjetas GTX 200 . . 22
2.2. Patrones de acceso de memoria en una aplicación CUDA 23
2.3. Características de los "*Compute Capability*" disponibles a la fecha 24

3.1. Parámetros generales del programa 34
3.2. Parámetros del programa referidos al archivo de datos 35

4.1. Tiempo de ejecución para ambas versiones según partículas 57

A.1. Componentes físicos del computador de pruebas 70
A.2. Propiedades específicas de la tarjeta GTX 285 71
A.3. Versiones de los sistemas usados en el entorno de desarrollo 71

CAPÍTULO 1

INTRODUCCIÓN

De entre de todas las cosas que han llamado la atención de los seres humanos a lo largo de su historia, el movimiento de los seres y los objetos es ciertamente una de ellas. Y sobre todo si ese movimiento se da en el aire –o en la atmósfera–, dado que en esa circunstancia es sabido que las trayectorias descritas por los objetos se tornan más inciertas e interesantes, pues son muchas las fuerzas que actúan sobre el objeto que se está moviendo, empujándolo en cambiantes direcciones, a veces oponiendo resistencia a sus esfuerzos, a veces apoyándolo en su trayectoria.

Y los seres humanos no se han conformado con observar estos procesos, también desean comprenderlo e incluso van más allá: pretenden predecirlo, y muchas veces lo logran con éxito.

Y dentro de las maneras que han usado para predecir estos movimientos, está el entregar ese problema a los computadores.

El explorar una nueva manera computacional de abordar este problema, para hacerlo más eficiente y rápido de lo que se hace en la actualidad, es el eje central del trabajo de memoria que este informe describe.

1.1. Contexto general

Calcular la trayectoria de partículas en la atmósfera ha sido abordado desde hace décadas, y su solución mediante computadores ha estado siempre restringida al uso

de los recursos típicos de estos equipos: su procesador central (CPU) y su memoria de sistema (Memoria RAM). En este escenario, la principal manera de acelerar los cálculos es usar computadores más rápidos y potentes, vale decir, tratando de mejorar la CPU usada, y/o la RAM de sistema.

1.2. Contexto específico

1.2.1. Uso de CPU para los cálculos

En la actualidad es posible advertir que las principales maneras de llevar a cabo el cálculo y la predicción de la trayectoria de partículas, es realizado usando computadores a la usanza tradicional, vale decir, ejecutando las instrucciones directamente en el procesador central de los computadores. Para obtener un mejor desempeño desde un punto de vista de *hardware*, se tiene que hacer más poderoso el computador usado, ya sea actualizando su procesador, aumentando su memoria RAM o agregándole más espacio de almacenamiento, pero todas estas mejoras tienen un límite asociado a la disponibilidad de estos componentes, y a los límites en las prestaciones que éstos son capaces de entregar, como también al hecho de que en determinados casos ni siquiera el aumento de las capacidades del equipo asegura un mejor funcionamiento de los cálculos, por cuanto ya han llegado a su máximo desempeño, y un equipo más potente no cambia las cosas.

1.2.2. Nuevas perspectivas de solución: GPU

Actualmente hay un componente de *hardware* computacional que está experimentando un notable desarrollo y potenciamiento: la Unidad de Procesamiento Gráfico (GPU, por sus siglas en inglés). Una de sus características principales es su capacidad de ejecutar múltiples tareas paralelamente (una necesidad inherente a muchas aplicaciones gráficas como por ejemplo los *juegos*). Estas tareas son en realidad simples cálculos de punto flotante, que sirven para realizar en un solo paso la suma de dos vectores, o la multiplicación de uno de ellos por un escalar, por mencionar solamente dos ejemplos. Cada uno de estos cálculos son ejecutados en los "Unified Shaders", que en el caso de la tarjeta NVIDIA GeForce GTX 285 llegan a un total de 240, y que son comparables a los "núcleos" con que cuentan los procesadores modernos.

En la actualidad las GPU superan, en cantidad de operaciones de punto flotante, a las CPU de última generación, como se puede observar en el gráfico 1.1. Esto se debe

tanto a que poseen una gran cantidad de procesadores, como se mencionó más arriba, pero también al hecho que estas GPU cuentan con su propia memoria RAM (la que será llamada GPU RAM a lo largo de este informe), separada de la memoria RAM del sistema, lo que les permite realizar de forma rápida tanto el cálculo de datos como su manipulación.

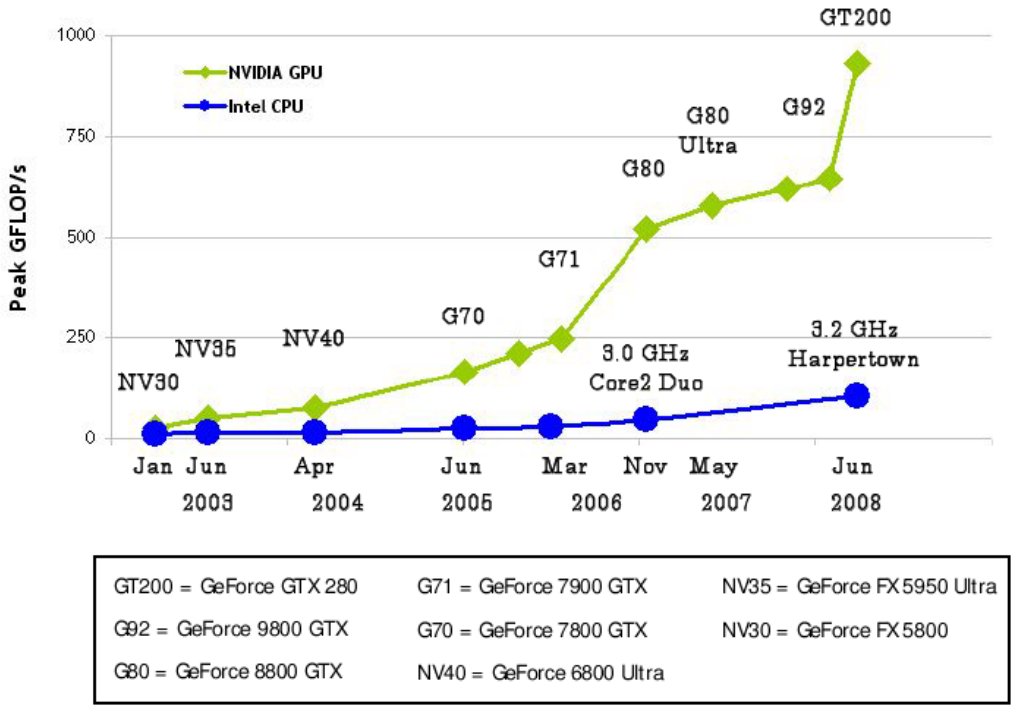


Figura 1.1: Gráfico comparativo de los GFLOP máximos (10^9 operaciones de punto flotante) por segundo logrados por diversos modelos de Procesadores Intel y GPUs NVIDIA, a través del paso de los últimos seis años [15].

Esta gran capacidad de cómputo en paralelo es la que ofrece grandes oportunidades para todas aquellas tareas que sean divisibles en múltiples subtareas, paralelas entre sí, o de aquéllas que involucren el procesamiento en paralelo de gran cantidad de información [3].

El gran desarrollo experimentado por las GPUs, y la aparición de cada vez mayores y más variadas aplicaciones cuyo funcionamiento está en gran parte apoyado por ellas, ha dado pie a la creación del concepto GPGPU (General-Purpose Computing on Graphics Processing Units)[27], el cual abarca las aplicaciones prácticas —científicas o no— de las GPU.

Y es precisamente en este contexto donde aparece el principal campo de acción del presente Tema de Título: aprovechar la alta capacidad de cómputo en paralelo de las

GPU, para implementar en ellas los algoritmos que calculan las trayectorias de partículas en un campo vectorial —como lo es el viento—. Estos cálculos son paralelizables, pues al dividir discretamente el tiempo en "pasos", separados entre sí por Δt, en cada uno de estos momentos se calcula la posición en que se ubicará cada partícula en el paso de tiempo siguiente, independientemente de las demás. El hecho de realizar los cálculos para cada partícula por separado es precisamente lo que representa su naturaleza *paralela*.

1.2.3. GPU y cálculo de Trayectorias en la actualidad

Todavía no existen implementaciones estables de algoritmos que hagan uso de GPU para el cálculo de trayectorias de partículas. No obstante, se ha investigado la existencia de algunos proyectos que serán mencionados más adelante en este informe (Sección 2.7). El que este campo aún no esté abordado masivamente indica que constituye una gran oportunidad para aportar a la predicción y cálculo de trayectorias de partículas en la atmósfera.

1.3. Problema por resolver

El tema que da vida a esta memoria es la necesidad de calcular el comportamiento y la trayectoria de partículas (consideradas como puntos infinitesimales) en suspensión en la atmósfera, y que están sujetas a un campo vectorial como el viento y la gravedad —entre otras fuerzas de la naturaleza— en una región geográfica en particular. Como ejemplo se puede observar en la figura 1.2 el resultado de un cálculo como éste. Entre las partículas cuya trayectoria se estudia se puede considerar al aire, cenizas volcánicas, polvo, gases contaminantes, etc. Es útil calcular las trayectorias de estas partículas, pues de esta manera se puede predecir su comportamiento global —en un período de tiempo arbitrario— lo que tiene diversas aplicaciones prácticas, como lo es la predicción del desplazamiento de una columna de cenizas proveniente de un volcán en erupción, o el estudio del impacto ambiental asociado a la emisión de contaminantes producidos por alguna faena industrial, por solamente nombrar algunos.

Dado que se está analizando la trayectoria de partículas que viajan a través de un campo vectorial, es necesario contar *a priori* con los valores de este campo en la región geográfica estudiada. Para efectos de este trabajo, se utilizó un pronóstico obtenido gracias a la aplicación del modelo meteorológico WRF [41] y otro producido por el modelo GFS [30].

Para calcular la trayectoria de estas partículas se considera la posición inicial de varios puntos (por lo general aleatorios), y se calcula sucesivamente sus posiciones para cada uno de los pasos de tiempo en que ha sido dividido el tiempo total de la simulación, considerando la influencia que ejerce sobre cada punto el campo vectorial. En algunos casos, si se le asigna propiedades físicas (masa o tamaño) a las partículas estudiadas, vale decir, se las modela como cenizas o granizo por mencionar sólo dos posibles casos, entonces se puede incorporar al cálculo su interacción con la gravedad, lo que se denomina sedimentación o precipitación.

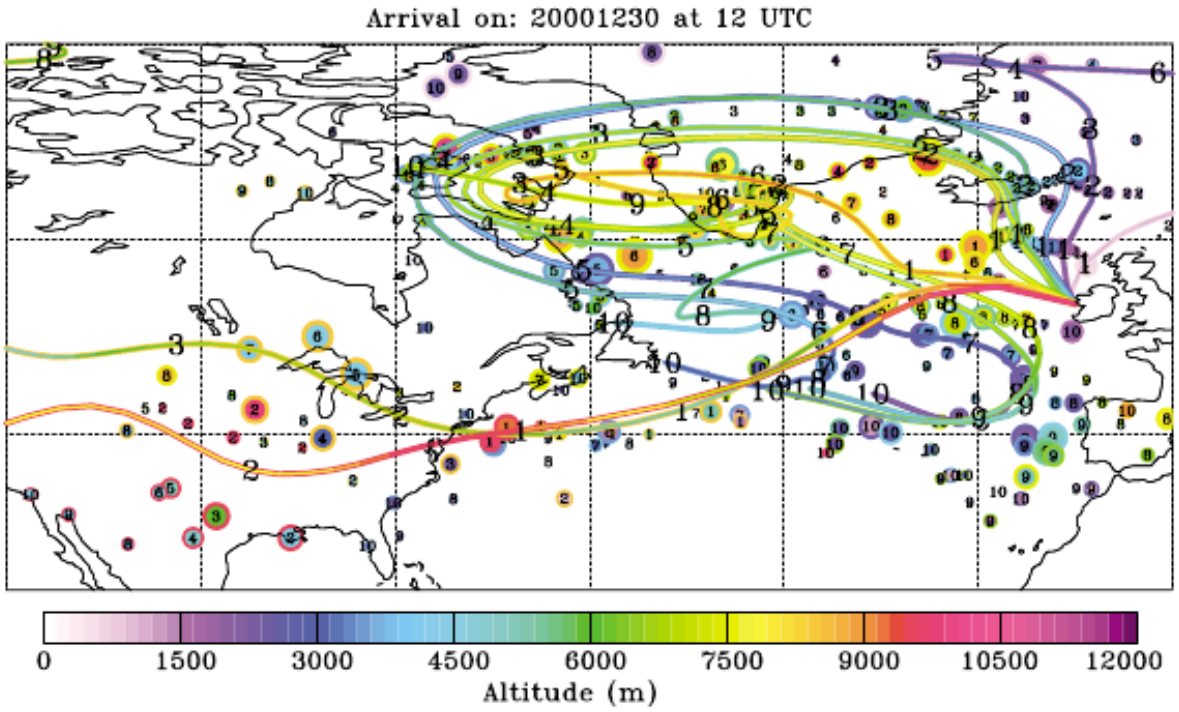


Figura 1.2: Ejemplo del cálculo de trayectorias. Las líneas de colores indican cada una de las diferentes trayectorias pronosticadas en función de la altura [18].

1.4. Motivación

El uso de GPU para llevar a cabo los cálculos de trayectorias es muy interesante, pues abre un camino hasta ahora apenas explorado, y que promete solucionar este problema en un tiempo mucho menor a lo actualmente conseguido, dado el paralelismo inherente a las GPU de última generación existentes a la fecha en que se realizó este trabajo.

Desde un punto de vista personal, este tema es muy interesante pues abarca problemas de diversa índole:

- *Matemáticos y Físicos*: El cálculo de las trayectorias obedece a las leyes físicas del movimiento de los cuerpos [4, 5].
- *Computacionales*: Ocuparse de este problema implica la programación en CPU y GPU de última generación, como también adentrarse en sus características de *hardware*.
- *Meteorológicos*: La Meteorología es la ciencia que estudia —entre muchas otras cosas—, los procesos físicos que ocurren en la atmósfera, y el pronóstico del viento es un insumo fundamental para la realización de estos cálculos.
- *Sociales*: Una más exacta y rápida predicción del comportamiento de diversos fenómenos naturales, tales como erupciones volcánicas o emergencias nucleares, permitirá a las autoridades organizar de manera más oportuna las medidas necesarias para paliar los efectos de estos eventos en la población.
- *Industriales*: En particular, la industria de la aviación comercial necesita conocer de antemano cómo se comportarán fenómenos naturales como las erupciones volcánicas, para poder organizar óptimamente las rutas de vuelo de sus aeronaves.

Todo esto demuestra que este proyecto será una muy buena oportunidad para poner en práctica varios de los conocimientos adquiridos a lo largo del estudio de la carrera de Ingeniería Civil en Computación, como también para aprender nuevos temas en ámbitos que no están directamente relacionados con esta carrera, tales como aquellos que tienen que ver directamente con la Meteorología.

1.5. Objetivo General

Desarrollar un programa que calcule las trayectorias de partículas —representadas como puntos infinitesimales— que están inmersas en la atmósfera, y que están sometidas a la fuerza del viento, el cual es modelado como un campo vectorial cuyos valores se conocen solamente en puntos discretos del espacio y del tiempo.

Estos cálculos han de ser realizados utilizando tanto CPU de múltiples núcleos, como GPU de última generación, para lo cual hay que implementar una versión del programa para cada una de ellas, porque también se espera que el cálculo de trayectorias sea lo más *portable* posible (para que no se quede restringido a un equipo con un *hardware* específico), como también para poder apreciar directamente la diferencia en tiempo de cálculo que ofrece cada implementación.

1.6. Objetivos Específicos

Un análisis más detallado de los objetivos de este trabajo es expuesto a continuación:

1.6.1. Investigar los conocimientos necesarios

Dado que este trabajo abarca temas pertenecientes a diversos campos del conocimiento humano, los cuales deben ser conocidos y manejados adecuadamente para lograr las metas planteadas, es que el primer objetivo de este trabajo es la investigación de estos temas, los que fueron organizados de acuerdo al ámbito al cual pertenecen:

- *Matemática*: Repasar varios temas vistos en los cursos de plan común llamados "Cálculo Numérico" y "Matemáticas Aplicadas". Fundamentalmente todo lo relacionado con los diversos tipos de interpolación y cálculo de gradientes.
- *Hardware*: El implementar de manera correcta los algoritmos involucrados en este trabajo —extrayendo el máximo provecho de las tarjetas gráficas utilizadas— sólo es posible si se conoce de manera profunda tanto las características físicas de estos componentes de *hardware*, como la manera en que éstos se comunican con las otras componentes del computador.
- *Software*: Para hacer uso de las potencialidades ofrecidas por las GPU, y en el caso particular de las tarjetas producidas por NVIDIA, es necesario utilizar la arquitectura CUDA, desarrollada por esa misma empresa.
- *Algoritmos existentes*: Dado que la solución a estos problemas desde una perspectiva tradicional (haciendo uso solamente de CPU) no es un tema nuevo, ya existen algoritmos implementados en lenguajes como FORTRAN o MATLAB que son recomendables de estudiar, pues es necesario obtener de ellos ideas para una óptima implementación de los algoritmos propios de este trabajo.
- *Formatos de datos científicos*: En la actualidad, la información de un campo vectorial (viento por ejemplo) es almacenada en diversos formatos relativamente diferentes. En este trabajo se hace uso de datos (en particular la información de la velocidad del viento) almacenados en archivos con los formatos NetCDF [39] y HDF5 [34], cuyas bibliotecas de programación deben ser investigadas.

1.6.2. Implementar cálculos

Implementar los algoritmos que permiten calcular las trayectorias. Se hace uso fundamentalmente del lenguaje de programación C para estos efectos. Las características principales de este objetivo son expuestas a continuación:

- Implementación de los algoritmos para ser usados solamente en CPU —a la usanza tradicional—, usando el lenguaje de programación C.
- Implementación del mismo algoritmo, pero para ser ejecutado en GPU, haciendo uso del lenguaje de programación C con el apoyo de las herramientas de desarrollo CUDA (Compute Unified Device Architecture) [10, 15].
- Evaluar también el uso de MATLAB en conjunto con CUDA, mediante la interfaz MEX [36].

Las razones de implementar dos versiones del mismo programa —tanto para CPU como para GPU— son tanto la necesidad de comparar el desempeño del programa en ambas circunstancias (de las cuales es de esperar que la versión GPU sea la más rápida) como el hecho de tener una versión del programa en CPU permite una mejor *portabilidad* del mismo, dado que no se cuenta con GPU adecuadas en todos los computadores.

1.6.3. Obtener información desde archivos de datos especiales

Los datos sobre la velocidad del viento solamente son obtenibles a partir de archivos con formatos especiales que permiten obtener información de carácter científico. Por ello, otro objetivo es implementar para estos efectos las interfaces necesarias para hacer uso tanto del formato de datos NetCDF [39] como del formato de datos HDF5 [34].

1.6.4. Manejar adecuadamente las proyecciones geográficas y espaciales

El sistema utiliza pronósticos de la velocidad del viento que corresponden a un campo vectorial en función de cuatro variables: Longitud, Latitud, Altura y Tiempo. Para las tres primeras, por ser coordenadas geográficas, hay que considerar los efectos que las proyecciones geográficas producen sobre ellas. Para el caso de la Latitud y Longitud (coordenadas horizontales), los efectos de la proyección geográfica usada son más apreciables mientras más amplio sea el alcance geográfico donde se trabaja. Para escalas

menores, que abarcan solamente unos cuantos grados de extensión, estos efectos pueden ser despreciados sin perder mayor exactitud en los cálculos.

En el caso de la Altura (coordenada vertical), se tiene que trabajar considerando los "Sistemas de coordenadas verticales" usados en los datos del pronóstico del viento, lo que tiene como principal consecuencia que dos puntos diferentes que están ubicados en un mismo nivel de altura desde el punto de vista del pronóstico, no están necesariamente a la misma altura en metros con respecto al nivel del mar.

Es de suma importancia notar que no considerar adecuadamente estas características de los datos geográficos implica pésimas consecuencias para la exactitud de los cálculos realizados.

1.6.5. Ejecutar el programa con datos reales

Debido a su gran repercusión mediática a nivel nacional, se escogió la erupción del Volcán Chaitén ocurrida en mayo de 2008 como la circunstancia con la cual poner a prueba el software producido por este trabajo. Para ello se cuenta con la información de la velocidad del viento en el área geográfica alrededor de este volcán, para un tiempo posterior a su erupción, pero mientras todavía estaba emitiendo contaminantes a la atmósfera.

Con estos datos se ejecuta el programa tanto para apreciar cómo funciona, como también para comprobar que su predicción corresponda razonablemente a lo que realmente sucedió.

1.6.6. Comparar entre sí las implementaciones para CPU y GPU

Una vez elaboradas las implementaciones, y usados los datos reales, realizar varias pruebas con el fin de comparar los desempeños de estas dos implementaciones, para saber con certeza la conveniencia de utilizar las GPU en casos como el del presente trabajo, pero también con el fin de identificar en qué circunstancias conviene usar una u otra versión.

1.6.7. Implementar un visualizador para las trayectorias calculadas

Como un objetivo complementario está el aprovechar las ventajas de tener los datos calculados en la memoria RAM de la tarjeta de video [17], para simultáneamente con su cálculo, poder desplegarlos en pantalla de una manera entendible y clara.

Este es un objetivo que reviste gran complejidad, y que involucra desarrollar un visualizador en 3 dimensiones de las trayectorias dentro del campo vectorial, razón por la cual no se le asigna gran prioridad, para no descuidar los otros objetivos que son fundamentales según lo planteado en las secciones anteriores de este documento.

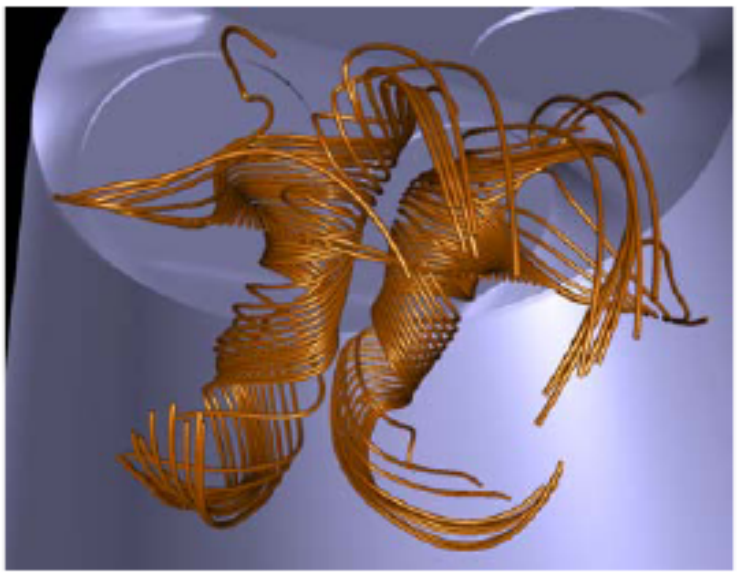


Figura 1.3: Un ejemplo de visualización gráfica de trayectorias. [17]

Una alternativa válida para este objetivo, y que no implica una complejidad tan grande, es hacer uso de la biblioteca Matplotlib de Python [29], la cual produce una amplia gama de gráficos de alta calidad. También se puede optar por usar programas con capacidades gráficas como MATLAB [35] o Gnuplot [6].

1.7. Contenido de la memoria

En el capítulo 2: **Antecedentes**, se expone los principales temas cuyo entendimiento es importante para una adecuada valoración del trabajo realizado en esta memoria de título.

En el capítulo 3: **Descripción de la Solución**, se describe en detalle la manera en que fueron abordados y llevados a cabo los objetivos de esta memoria. Se hizo especial hincapié tanto en explicar los algoritmos diseñados como en presentar todas las funcionalidades de CUDA utilizadas.

En el capítulo 4: **Resultados**, se presenta y discute los resultados obtenidos de la aplicación del programa implementado a un caso real, como fue la erupción del Volcán Chaitén, específicamente enfocado en el comportamiento de la columna de ceniza volcánica durante un día en particular.

En el capítulo 5: **Conclusiones**, se comentan las conclusiones a las que se llegó luego de implementar y analizar los resultados del programa fruto de este trabajo.

Se presenta también toda la bibliografía consultada para llevar a cabo este trabajo, así como las fuentes consultadas por internet, que dado su carácter más volátil, fueron listadas por separado.

En el apéndice A: **Características del entorno de desarrollo**, se presentan las principales características de *hardware* y *software* del equipo utilizado para realizar la implementación y las pruebas de este trabajo.

En el apéndice B: **Revisión bibliográfica**, se comentan los principales libros y manuales consultados.

En el apéndice C: **Presentación del proyecto en la Conferencia "CoV6th"**, se comenta brevemente la experiencia y resultados de haber asistido a presentar un póster acerca de los alcances y proyecciones que este trabajo de título representa para la investigación vulcanológica.

CAPÍTULO 2

ANTECEDENTES

A continuación se exponen los antecedentes técnicos y teóricos necesarios para comprender el aporte del presente trabajo. Su conocimiento es necesario para visualizar de mejor manera los alcances que abarca este proyecto. Se comienza hablando de los temas relacionados con la Meteorología, para pasar por problemas más Matemáticos, hasta llegar a temas directamente relacionados con las Ciencias de la Computación.

2.1. Sistemas de coordenadas verticales

Cuando se trabaja con fenómenos atmosféricos, se hace necesario poner especial atención a su estructura vertical, dado que existen muchas maneras de modelar esta dimensión espacial. Se introduce brevemente tanto el sistema de coordenadas verticales σ, como el sistema curvilíneo usado por ARPS [19, 23], los cuales son útiles para entender las transformaciones de coordenadas que debieron ser realizadas a la hora de trabajar con la componente vertical de la velocidad del viento entregada por los pronósticos.

2.1.1. Coordenadas sigma

En este sistema, para todo punto del espacio se define un valor llamado "σ", calculado mediante la ecuación 2.1, donde p es el valor de la presión atmosférica en el punto en cuestión, y p_{suelo} es el valor de la presión atmosférica a nivel del suelo, exactamente debajo el aquel punto. Esta ecuación define los "niveles σ", que son los puntos en el espacio que poseen un mismo valor σ y que tienen la propiedad de ser niveles que se

"ajustan" a la variación topográfica del terreno (ver figura 2.1). Esto último se afirma porque para un terreno a mayor altura, el valor p_{suelo} disminuye, por lo que el valor de σ aumenta consecuentemente. Por consiguiente, se dice que dos puntos están en el mismo "nivel σ" cuando la razón entre la presión atmosférica en ese punto y la presión a nivel del suelo es la misma. Es muy importante señalar que el valor de la presión atmosférica en un punto del espacio no es constante a lo largo del tiempo, razón por la cual la altura de un nivel σ varía también con respecto a esta dimensión. Es decir, es variable en las cuatro dimensiones: latitud, longitud, altura y tiempo.

$$\sigma = \frac{p}{p_{suelo}} \tag{2.1}$$

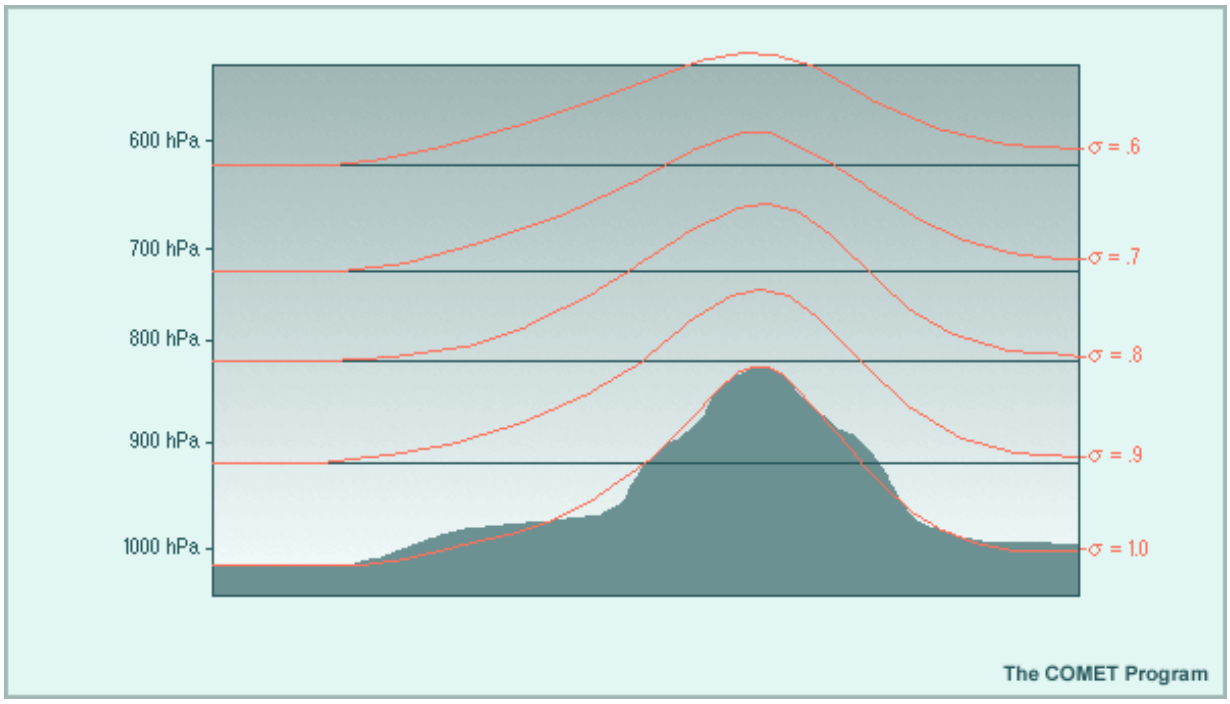


Figura 2.1: Ejemplo del modelo de coordenadas σ. [38]

2.1.2. Coordenadas curvilíneas usadas por ARPS

El sistema de Predicción Regional Avanzada (sigla en inglés ARPS) [23, 19] hace uso de un sistema de coordenadas curvilíneas (ξ, η, ζ), el que está definido en coordenadas

cartesianas (x, y, z) de la siguiente manera:

$$\xi = x \tag{2.2}$$
$$\eta = y \tag{2.3}$$
$$\zeta = \zeta(x, y, z) \tag{2.4}$$

donde $\zeta(x, y, z)$ corresponde a la transformación vertical de la posición, y que se adecúa a la topografía del terreno. Conforme a este sistema de coordenadas curvilíneas, la velocidad del viento en coordenadas cartesianas (u, v, w) está definido en función de las velocidades en coordenadas curvilíneas (U^c, V^c, W^C) de la siguiente manera:

$$u = U^c\frac{\partial x}{\partial \xi} + V^c\frac{\partial x}{\partial \eta} + W^c\frac{\partial x}{\partial \zeta} = U^c \tag{2.5}$$
$$v = U^c\frac{\partial y}{\partial \xi} + V^c\frac{\partial y}{\partial \eta} + W^c\frac{\partial y}{\partial \zeta} = V^c \tag{2.6}$$
$$w = U^c\frac{\partial z}{\partial \xi} + V^c\frac{\partial z}{\partial \eta} + W^c\frac{\partial z}{\partial \zeta} = U^c\frac{\partial z}{\partial x} + V^c\frac{\partial z}{\partial y} + W^c\frac{\partial z}{\partial \zeta} \tag{2.7}$$

de ello se puede apreciar que solamente la componente vertical de la velocidad del viento se ve alterada al transformar entre estos dos sistemas de coordenadas.

2.2. Características de los pronósticos del viento

Tal como se mencionó en la sección 1.6.4, es de vital importancia considerar las proyecciones geográficas y los sistemas de coordenadas verticales de los datos del pronóstico de la velocidad del viento utilizados en los cálculos de las trayectorias.

Hay que tener en cuenta que esta información corresponde a datos numéricos (la velocidad del viento en $[m/s]$) para cada celda en que está dividido el espacio tridimensional, y ello, a su vez, para cada paso de tiempo (lo que combinado, corresponde a las cuatro dimensiones de la "función velocidad del viento"). Para hacer referencia a cualquiera de estas celdas, hay que usar los índices (t, z, y, x), que corresponden respectivamente al tiempo, altura, latitud y longitud, y que aceptan valores que van desde 1 hasta $(t_{max}, z_{max}, y_{max}, x_{max})$. Por lo tanto, para obtener el valor de la velocidad del viento en un punto dado del espacio, para un momento dado, primero hay que convertir estas coordenadas a los índices correspondientes, y con estos índices se accede –dentro del

archivo de datos– al valor buscado. Producto de esto se tiene las siguientes funciones:

$$\begin{aligned} \text{Longitud, Latitud, Altura, Tiempo :} \quad & lon,\ lat,\ alt,\ tpo \\ \text{Índice para Longitud :} \quad & x(lon) \qquad (2.8) \\ \text{Índice para Latitud :} \quad & y(lat) \qquad (2.9) \\ \text{Índice para Altura :} \quad & z(alt, lat, lon) \qquad (2.10) \\ \text{Índice para Tiempo :} \quad & t(tpo) \qquad (2.11) \\ \text{Velocidad del viento :} \quad & \mathbf{V}(t, z, y, x) \qquad (2.12) \end{aligned}$$

donde $\mathbf{V}$ es el vector "Velocidad del viento", con tres componentes, una para cada coordenada espacial: $\mathbf{V} = (V_{alt}, V_{lat}, V_{lon})$. En el caso de la coordenada z —el índice correspondiente a la altura— su valor depende no solamente de la altura del punto, sino también de su ubicación horizontal (latitud y longitud). Ello se debe a que los niveles verticales —dentro del archivo de datos— no son superficies paralelas al nivel del mar, sino que siguen de forma aproximada la topografía del terreno (ver sección 2.1). Por ello, para saber el nivel vertical de un punto hay que conocer su posición completa en el espacio.

2.3. Interpolación trilineal

La interpolación trilineal es una herramienta de cálculo en extremo usada a lo largo de este trabajo. Con objeto de aplicarla al cálculo de los valores de la velocidad del viento que se necesitan, y que no coincidan exactamente con los valores discretos con los que se cuenta, pero que se encuentran entre medio de ellos, se ha recurrido a [22], artículo que da una expresión práctica para este cálculo, la que se expone en la ecuación 2.13, y que usa como valores de referencia a los puntos que se muestran en la figura 2.2.

2.4. Arquitectura física de las tarjetas de video NVIDIA GeForce GTX 200

La generación de tarjetas gráficas GeForce GTX 200, lanzada al mercado por NVIDIA el año 2008, tiene entre sus miembros a la tarjeta GTX 285, la cual fue usada para la implementación y la obtención de resultados de esta memoria. A continuación se describe las principales características de la arquitectura física de esta familia de tarjetas, en la medida que fueron relevantes para los efectos de este trabajo, y de esta manera permitir una mejor comprensión de aquellas propiedades especiales sin las cuales todo lo descrito

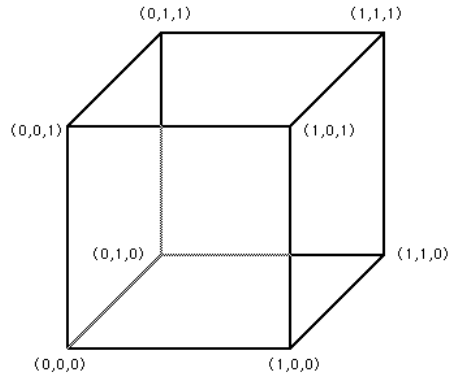


Figura 2.2: Cubo unitario en cuyos vértices están los valores conocidos del campo a interpolar [22].

$$\begin{aligned} V_{xyz} \;=\; & V_{000}\;(1-x)\;(1-y)\;(1-z) \\ & +\; V_{100}\;x\;(1-y)\;(1-z) \\ & +\; V_{010}\;(1-x)\;y\;(1-z) \\ & +\; V_{001}\;(1-x)\;(1-y)\;z \\ & +\; V_{101}\;x\;(1-y)\;z \\ & +\; V_{011}\;(1-x)\;y\;z \\ & +\; V_{110}\;x\;y\;(1-z) \\ & +\; V_{111}\;x\;y\;z \end{aligned} \tag{2.13}$$

más adelante no podría haber sido logrado. Cabe señalar que los componentes de *hardware* descritos a continuación, son referidos por sus nombres originales en inglés, por no tener una traducción de ellos al español lo suficientemente consolidada a la fecha.

2.4.1. Componentes y su organización al interior de la tarjeta

Toda tarjeta perteneciente a la generación GTX 200 está compuesta por diversos componentes de *hardware*, entre los cuales se tiene:

A nivel de multiprocesador (figura 2.3):

Stream Processor (SP): Son los núcleos, las unidades básicas de cálculo. Cada uno de ellos ejecuta una instrucción (*thread*) individual. En la figura 2.3 están representados por los cuadrados verde claro, con una de sus denominaciones en inglés: *Core*.

Instruction unit (IU): Esta componente crea, administra y ejecuta los *threads* en grupos de 32 *threads* paralelos llamados *Warp*. Está representado como el rectángulo verde obscuro en la figura 2.3.

Shared local memory: Un bloque de 16kB de memoria, compartida por todos los **SP** de un mismo **MP**. Corresponde al rectángulo morado que se ubica entre medio de los **SP** de la figura 2.3.

Stream Multiprocessor (MP): Es la agrupación de ocho **SP**. Corresponde a todo el conjunto de elementos representados en la figura 2.3.

A nivel de *cluster* (figura 2.4):

Texture address/filter (TF): Son las unidades encargadas de realizar las operaciones sobre el espacio de memoria GPU RAM asignado a ellas (sección 2.4.3).

L1 Cache: Un cache L1 de memoria de Textura, de 16kB de tamaño, usado por los **TF**.

Thread processing *cluster* (TPC): Es la agrupación de tres **MP**, ocho **TF** y un **L1 Cache**. Corresponde a todo el conjunto de elementos representado en la figura 2.4.

Es muy importante destacar que a diferencia de las CPU tradicionales, que dedican aproximadamente un 20 % de sus transistores a los cálculos, las GPU elevan este valor a un 80 % aproximadamente [11].

Figura 2.3: Detalle de un *Stream Multiprocessor*. [11]

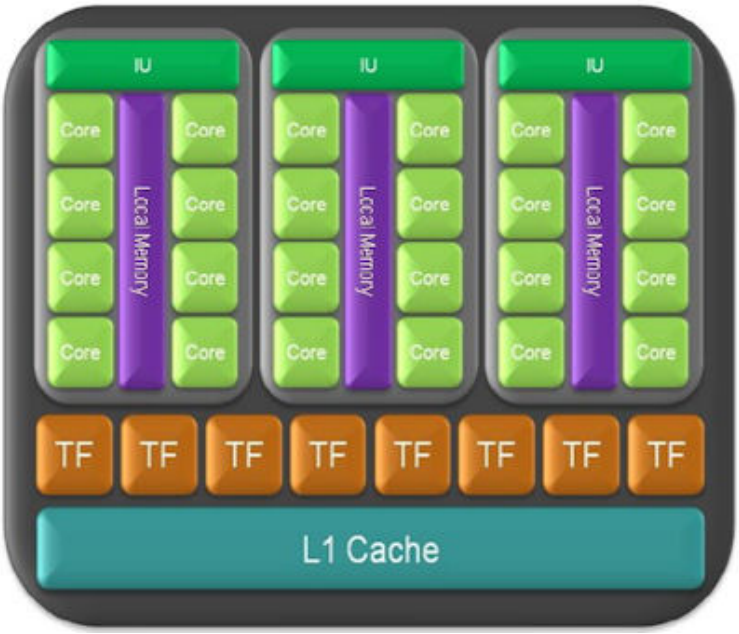


Figura 2.4: Detalle de un *Thread processing cluster*. [11]

2.4.2. Doble propósito de la arquitectura GTX 200

Una de las motivaciones de NVIDIA para desarrollar esta generación de tarjetas fue el abarcar dos mundos distintos: el de las aplicaciones que necesiten realizar un uso computacional intensivo —e incluso podría decirse que extremo—, y el de las aplicaciones gráficas. Como ejemplo de esta doble "personalidad", en la figura 2.5 se muestra la arquitectura global de la tarjeta GTX 280 al ser usada por aplicaciones que realizan cálculos intensos, y la figura 2.6 ilustra la arquitectura de la misma tarjeta, pero en el contexto de aplicaciones gráficas.

Si se observa la parte superior de ambos diagramas se observa la principal diferencia: en el primer caso existe una unidad llamada *Thread scheduler*, la cual está encargada de administrar todos los hilos (*threads*) de cálculos entre los distintos **MP** y **SP**; en el segundo caso lo que se activa son unidades encargadas de realizar tareas eminentemente gráficas.

2.4.3. Texturas

Un elemento central dentro de todos los componentes de hardware descritos más arriba, corresponde a la Memoria de Textura. Si bien se suele referir a las texturas como un tipo de memoria, no son tales desde un punto de vista físico (dentro de la tarjeta no existe un chip de memoria exclusivamente dedicado esta tarea), sino más bien es un filtro, a través del cual se accede a un sector de la memoria GPU RAM, y que permite ejecutar operaciones sobre estos datos.

Su nombre proviene del principal uso de estos elementos en el contexto de las aplicaciones gráficas: aplicar textura a objetos. Este es un proceso que requiere asignar un valor (un color por ejemplo) a cada *pixel* de un objeto representado gráficamente, contándose con estos valores solamente para algunos de estos *pixels*. En este caso, lo que se hace es inferir los valores faltantes en función de los que ya se conocen gracias a un cálculo matemático llamado interpolación.

Este es un proceso que está optimizado por hardware, de manera tal que si se tiene información lineal (vector), superficial (matriz) o volumétrica (cubo) —para la cual se necesita obtener algunos valores intermedios— las unidades de textura son altamente recomendables para calcularlos. Es muy importante señalar que la manera especial en que estas unidades de textura interpolan los valores tiene como consecuencia concreta que solamente se pueden interpolar de manera exacta 256 “valores interpolados” entre cada dos “valores conocidos consecutivos” [15].

Y es justamente el cálculo de interpolación lineal —en particular en tres dimensiones— una de las principales acciones que fueron ejecutadas en el trabajo descrito en los siguientes capítulos.

2.5. Especificación de la arquitectura CUDA de NVIDIA

La arquitectura CUDA (*Compute Unified Device Architecture*) corresponde a un conjunto de herramientas de desarrollo creadas por la empresa productora de tarjetas gráficas

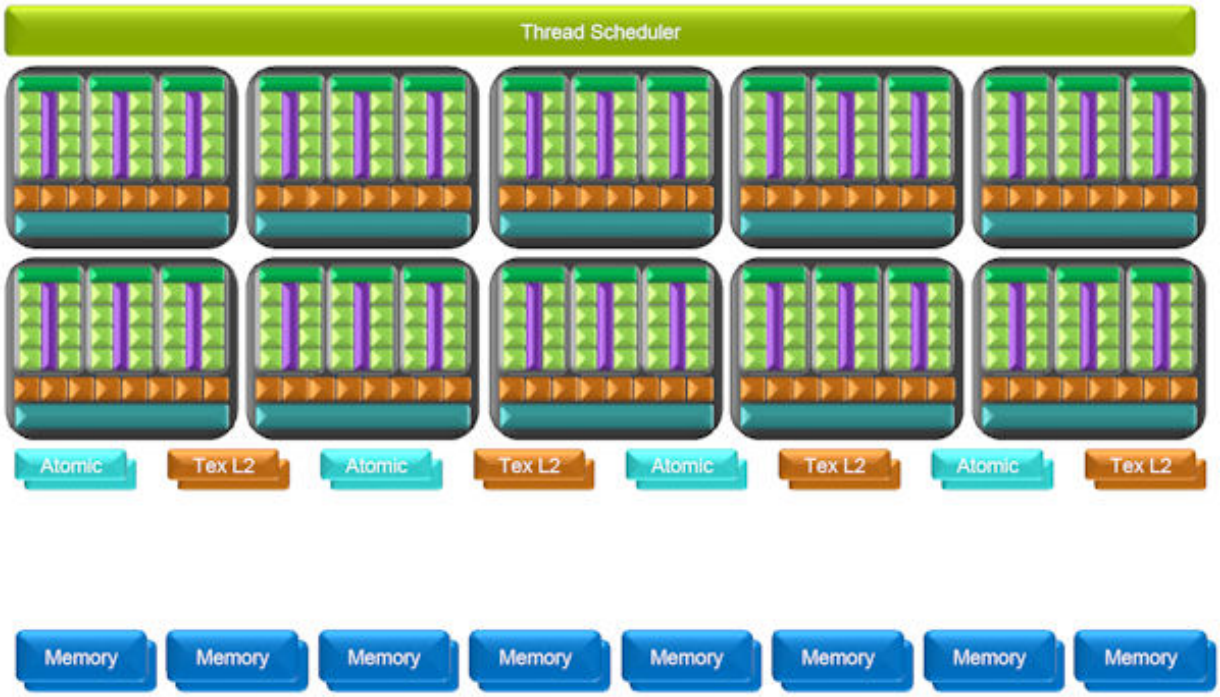


Figura 2.5: Arquitectura para cómputos en paralelo, en la tarjeta GTX 280. [11]

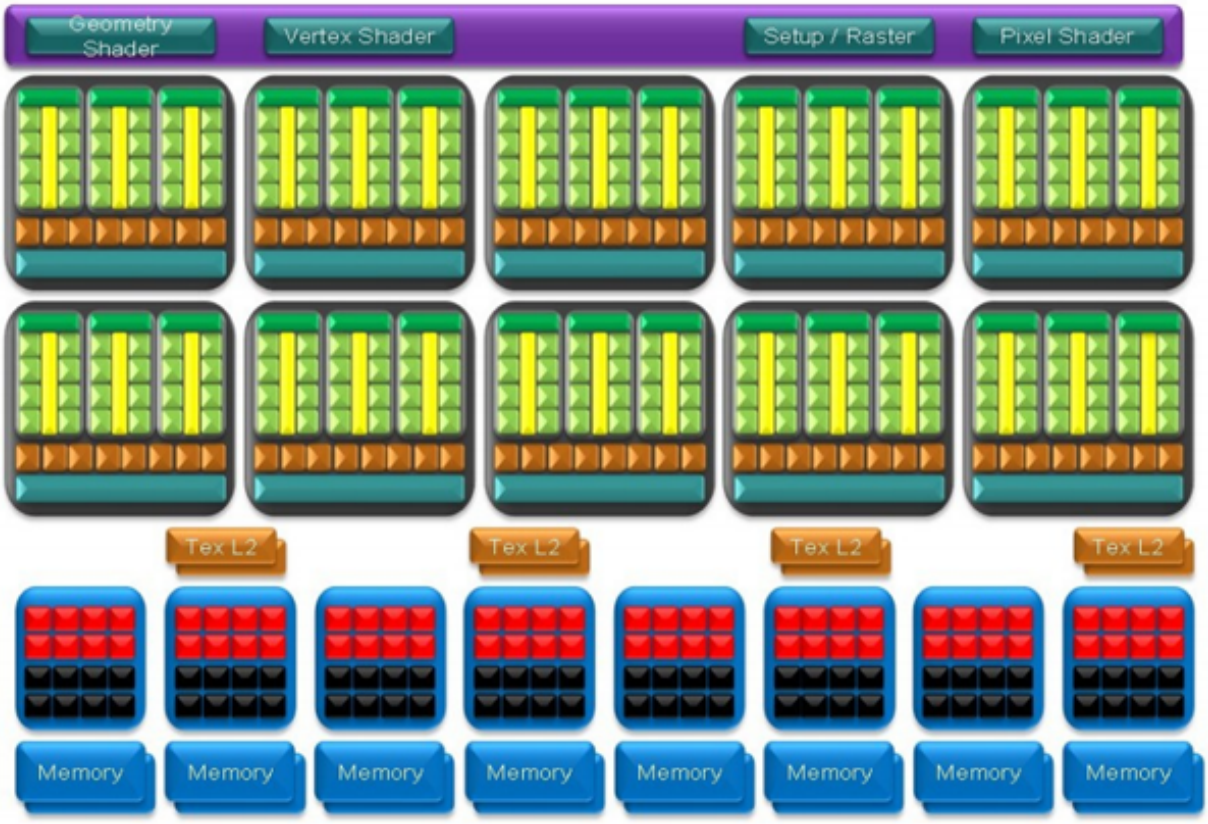


Figura 2.6: Arquitectura para procesamiento gráfico, en la tarjeta GTX 280. [11]

NVIDIA para desarrollar programas que hagan uso de las unidades de procesamiento gráfico (GPU por sus siglas en inglés) de estas tarjetas. CUDA define fundamentalmente un "Modelo de Programación" y un "Modelo de Memoria", que son expuestos a continuación.

2.5.1. Modelo de programación de CUDA

En general, el resolver de manera paralela un problema implica subdividirlo en partes más pequeñas, que sean independientes unas de otras. En CUDA, cada una de estas subrutinas paralelas es representada por un "*Thread*" (Hilo), ilustrado en la figura 2.7, y el cual dentro de la tarjeta es ejecutado en uno de los *Stream processors* (**SP**). A nivel de código fuente, todas las instrucciones que componen el plan de ejecución de un *Thread* son programadas en funciones llamadas "*Kernels*".

Los *Kernels* son funciones programadas para ser ejecutadas exclusivamente en la GPU, haciendo uso sólo de memoria GPU RAM. Dado que los *Threads* corresponden a tareas ejecutadas en paralelo, son asíncronas entre sí. También se puede señalar que no pueden ser recursivas y no pueden definir variables estáticas.

Los Hilos son agrupados en "*Blocks*" (Bloques), de manera tal que todos los Hilos de un mismo Bloque comparten un mismo espacio de memoria (sin perjuicio de la memoria propia que maneja cada Hilo: *Register* y *Local memory*). Físicamente, cada Bloque es ejecutado dentro de un *Multiprocessor* (**MP**). Es interesante notar que al momento de programar un Bloque, sus Hilos se pueden organizar como un vector o como una matriz de hasta tres dimensiones, lo cual es útil pues al manipular datos en matrices, cada elemento de esa matriz puede ser operado por un Hilo, el que es referenciable al interior del Bloque como un elemento de la matriz de Hilos, y se puede usar los mismos índices para ambos arreglos.

El conjunto de todos los Hilos y Bloques de una aplicación se denomina "*Grid*" (Grilla). Y se define una sola Grilla por cada tarjeta de video disponible en el sistema. Una Grilla también puede tener a sus Bloques organizados en la forma de un vector o de una matriz, pero en un máximo de dos dimensiones.

Como una característica de optimización, y dado que en CUDA una instrucción nunca puede tomar menos de cuatro ciclos, es que existe el concepto de "Warp", que corresponde a la cantidad total de Hilos que pueden ser ejecutados simultáneamente en un mismo *Multiprocessor* (**MP**). En el caso particular de la tarjeta GTX 285, se tiene que el tamaño

de su *Warp* es 32, lo que corresponde a cuatro veces los 8 *Stream processors* (**SP**) presentes en cada *MP*.

También cabe señalar que cada tarjeta tiene definido una cantidad máxima de Hilos a ejecutar por cada Bloque, lo que se describe en la sección 2.5.4, donde se explica el concepto de "*Compute Capability*" que corresponde a una forma de clasificar estas tarjetas en función de sus características principales.

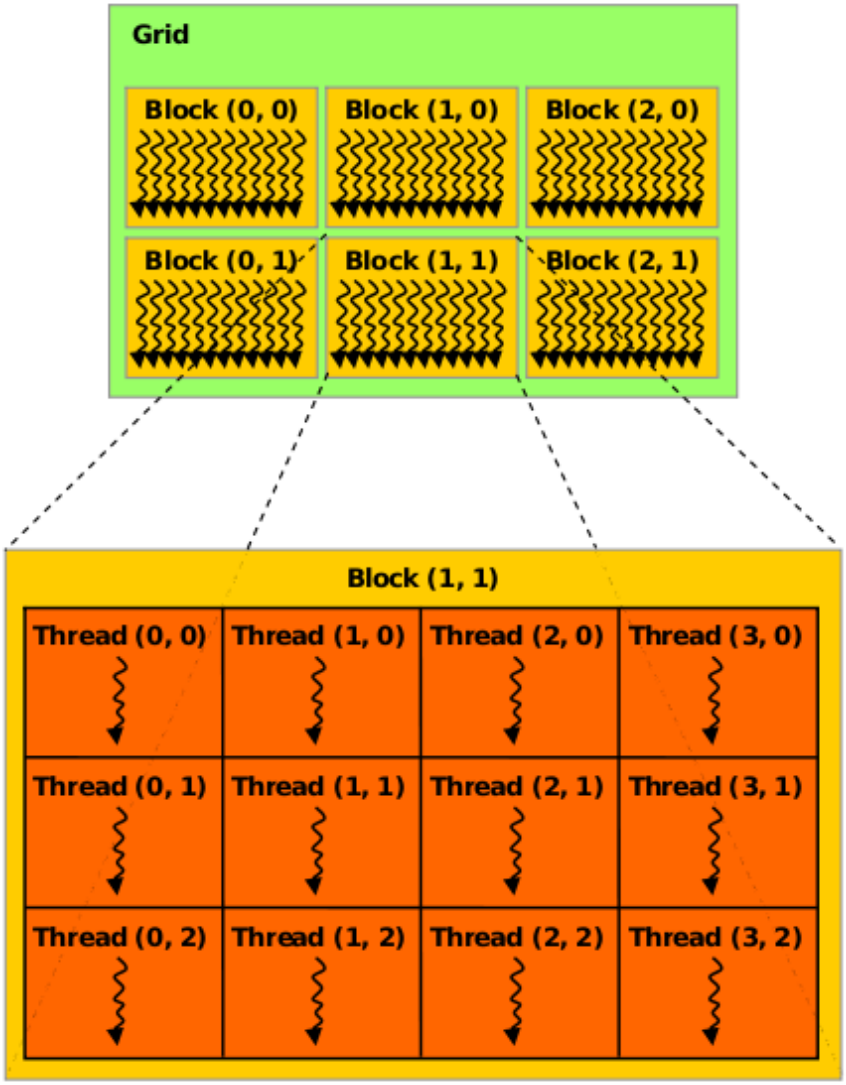


Figura 2.7: Grilla de Bloques, y un Bloque de Hilos. [15]

2.5.2. Modelo de memoria de CUDA

La generación de tarjetas gráficas GTX 200 no solamente cuenta con su propia Memoria RAM (GPU RAM), también ofrece distintos tipos de memorias, los cuales permiten distribuir los datos manipulados por el programa entre ellos, con el fin de optimizar su acceso en tiempo de ejecución (como es el caso de las memorias de textura). Estas características son descritas en el cuadro 2.1 (Algunos de los conceptos, tales como "Hilo", son explicados en la sección 2.5.

Nombre	Velocidad	Visibilidad	Vida total	Comentarios
Register	Un ciclo	Hilo	Hilo	
Shared	Un ciclo	Todo hilo de un mismo bloque	Bloque	Dividida en bancos de memoria.
Global	150x más lenta que "Register"	Global	Aplicación	
Local	150x más lenta que "Register"	Hilo	Hilo	Esta memoria solamente es una abstracción. Físicamente reside dentro de la memoria "Global"
Constant	Entre 1 a 100 ciclos.	Global	Aplicación	Hasta 64kB pueden ser ubicados en el cache de esta memoria, y 8kB por cada Multiprocesador.
Texture	Entre 1 a 100 ciclos.	Global	Aplicación	Puede entenderse más como un cache de memoria.

Cuadro 2.1: Descripción de los tipos de memoria disponibles en las tarjetas GTX 200

Dados los distintos tipos de memoria que este tipo de tarjetas ofrece, a la hora de usarlos es recomendable tener en cuenta sus características principales. En el cuadro 2.2 se contrastan sus características desde un punto de vista de "lectura/escritura".

2.5.3. C for CUDA

Para programar dentro de las tarjetas de video NVIDIA, CUDA ofrece una extensión al lenguaje de programación C, llamado "*C for CUDA*" el cual provee un conjunto de funciones especiales para acceder a los recursos propios de la GPU. Varias de estas funciones usadas a la hora de implementar el sistema son mencionadas en la sección 3.9.8.

Tipo de memoria	Patrón de acceso
Constant memory	Sólo lectura (sin estructura de vector o matriz)
Texture memory	Sólo lectura (con estructura de vector o matriz, y optimizada para ser referenciada de esta manera)
Shared memory	Lectura y escritura (común a todos los hilos de un mismo bloque)
Global memory	Lectura y escritura (donde se almacena los datos iniciales y los resultados)

Cuadro 2.2: Patrones de acceso de memoria en una aplicación CUDA

También provee de un compilador especial llamado **nvcc**, el cual recibe código fuente escrito en C o C++ que en su interior hace referencia a estas funciones adicionales.

Cabe destacar que si bien CUDA permite el uso de C tanto para ser ejecutado en CPU como GPU, en el caso de C++ solamente lo acepta para su uso en CPU.

2.5.4. Capacidad de Cómputo

El concepto de "*Compute Capability*" es usado por NVIDIA para clasificar a sus tarjetas gráficas (compatibles con CUDA) en función de sus características de *hardware*, capacidades a nivel de cálculo, y configuración de memoria. Es un indicador que permite saber de inmediato el nivel de exigencia máximo al que puede ser sometida una tarjeta gráfica. A la fecha de escritura de este informe existen oficialmente cuatro niveles "*Compute Capability*", cuyas principales características son expuestas en el cuadro 2.3, cuya información se basa en [16, Apéndice G].

2.5.5. Un patrón típico de programación en toda aplicación CUDA

De un modo muy general, se puede resumir la elaboración de una aplicación CUDA como la ejecución consecutiva de los siguientes pasos:

1. Dividir la tarea en subtareas.
2. Dividir los datos de entrada (que ya están cargados en la memoria del sistema) en trozos que quepan en los *Registers* y en *Shared memory*.
3. Cargar los trozos de información creados en el paso anterior en los *Registers* y en *Shared memory*.
4. Procesar cada uno de estos trozos de información con un *Thread block*.

	Compute Capability				
Especificación técnica	**1.0**	**1.1**	**1.2**	**1.3**	**2.0**
Max. *Threads* por cada *block*	512		512		1024
Tamaño de cada *warp*	32		32		32
Max. *blocks* por cada MP	8		8		8
Max. *warps* por cada MP	24		32		48
Max. *threads* por cada MP	768		1024		1536
Registros de 32 bits por cada MP	8192		16384		32768
Max *shared memory* por cada MP	16 kB		16 kB		48 kB
Local memory por cada *thread*	16 kB		16 kB		512 kB
Tamaño total de *constant memory*	64 kB		64 kB		64 kB
Max. tamaño de textura 2D asociada a memoria lineal o a "CUDA Array"	65536 x 32768				65536 x 65536
Max. tamaño de textura 3D asociada a memoria lineal o a "CUDA Array"	2048 x 2048 x 2048				
Max. cantidad de texturas asociadas a un *kernel*	128				
Max. cantidad de texturas 2D asociadas a un *kernel*	8				
Max. cantidad de instrucciones por cada *kernel*	2 millones				

Cuadro 2.3: Características de los "*Compute Capability*" disponibles a la fecha

5. Copiar los resultados desde la memoria de la GPU RAM a la memoria RAM de sistema (CPU).

2.6. Algoritmos actuales para el cálculo de trayectorias

Entre los algoritmos existentes actualmente y que son usados para resolver este problema se tiene:

HYSPLIT_4: "Hybrid Single-Particle Lagrangian Integrated Trajectory" [20], el cual es un sistema completo para el cálculo tanto de trayectorias simples como de simu-

laciones complejas de dispersión, usando ya sea un enfoque de remolinos o de partículas [1] [2].

FLEXPART y FLEXTRA [31]: Son modelos de trayectorias y dispersión de partículas en la atmósfera, escrito en FORTRAN 77 .

PUFF: "Puff-Volcanic Ash Tracking Model" [26], que usa formulaciones aleatorias Lagrangianas para calcular la trayectoria de un número arbitrario de partículas.

Entre los algoritmos usados hasta hace poco tiempo se puede mencionar, a modo de referencia, a **VAFTAD**: "Volcanic Ash Forecast Transport and Dispersion" [21], el cual es un modelo Euleriano que calcula la advección y la dispersión de ceniza volcánica usando información meteorológica y vulcanológica.

2.7. Trabajos actuales que abordan temas similares

Dado el gran interés que está despertando en la comunidad científica las notables capacidades de cálculo de las GPU, es que durante los últimos dos años se ha estado investigando en el uso de estas componentes de *hardware* para el cálculo de diversos problemas.

En este contexto cabe ser mencionado el *paper* "GPU Acceleration of Numerical Weather Prediction"[8], en el cual se presenta la implementación del Modelo WRF [41] haciendo uso de CUDA en una GPU. Cabe señalar que la presente memoria en lo que concierne a su aplicación práctica, hace uso de pronósticos de viento, los cuales son obtenibles también mediante este modelo meteorológico.

CAPÍTULO 3

DESCRIPCIÓN DE LA SOLUCIÓN

En el presente capítulo se describe los pasos llevados a cabo para desarrollar las dos versiones que componen el programa, y que son parte fundamental de los objetivos de este trabajo.

A lo largo de texto se describe los parámetros entregados al programa, como también se mencionan las variables más relevantes según sea el caso. Con el fin de destacar estas últimas, cuando sean mencionadas se utiliza letra cursiva negrita, como en este ejemplo: ***varName***.

3.1. Análisis de los datos del pronóstico del viento

Durante el desarrollo del programa, se contó con dos archivos de datos con pronósticos, el segundo de ellos (que correspondió al Volcán Chaitén y sus zonas aledañas) fue el más importante, por cuanto se utilizó para realizar las pruebas definitivas del sistema (ver sección 1.6.5).

Para estos dos archivos de datos utilizados, se analizó la influencia de las proyecciones geográficas y de las coordenadas verticales sobre los cálculos a implementar, llegándose a las siguientes conclusiones:

Coordenadas horizontales: En este caso la distancia abarcada por los datos tanto en la Latitud como en la Longitud es de unos cuantos miles de kilómetros —no muy cerca del polo sur— lo que no es suficientemente grande como para constituir un

caso en que la proyección geográfica produzca alteraciones mayores a los cálculos, por cuanto para representar tal peligro, la diferencia entre los datos geográficos obtenidos a partir de los datos y la geografía real que se está modelando, ha de ser del orden de los kilómetros, cosa que no se observó en este problema, donde el error solamente fue de algunos metros, y fundamentalmente en los sectores más extremos de la región geográfica cubierta por los datos.

Coordenada vertical: En este caso, por el contrario, los datos no estaban expresados en coordenadas cartesianas, sino que curvilíneas (ver sección 2.1), las cuales por seguir de forma aproximada la topografía del terreno y no ser constantes con respecto a la altura por sobre el nivel del mar, obligan a ejecutar una transformación del valor de la velocidad vertical para poder manejarlo como cartesiano.

Tal como se comenta en la sección 2.2, los datos de la velocidad del viento corresponden al valor de esta variable en cada una de las celdas en que se ha subdividido el espacio, y por lo tanto, corresponde a un valor discreto. Ello significa un problema, por cuanto para describir de forma más realista el desplazamiento de las partículas, es conveniente considerar desplazamientos continuos. Por lo tanto la solución pasa por interpolar los valores del viento, y así inferir cuánto valdrían en posiciones intermedias al interior de la zona estudiada.

Este hecho constituyó un aspecto fundamental de todo el problema que este trabajo resolvió, por cuanto el cálculo de las trayectorias —haciendo uso del pronóstico del viento como su principal insumo—, implica necesariamente la interpolación de estos datos para obtener resultados más válidos. Más adelante en este informe se explica en detalle todos los casos en que la aplicación de interpolación pasó a ser fundamental.

3.2. Ecuación que describe las trayectorias

Se consideró fundamental definir, primero que todo, la forma final que tendría la ecuación con la cual se calculasen las posiciones de las partículas a través del espacio (tridimensional y continuo) y del tiempo (discreto, dividido en pasos regulares de duración Δt). Existen muchas maneras de modelar el movimiento de las partículas en la atmósfera, dependiendo de qué fenómenos físicos se consideran y cuáles se desprecian, pero para el caso del presente trabajo se usó la ecuación de movimiento clásica:

$$x_{i+1} = x_i + v\Delta t + \tilde{x}_i \tag{3.1}$$

donde x_i y x_{i+1} corresponden a la posición de la partícula en dos pasos de tiempo sucesivos, y $\tilde{x}_i$ corresponde a un término genérico para expresar el desplazamiento adicional

ocasionado por la turbulencia, el cual, para efectos de este trabajo, fue siempre eliminado de los cálculos.

Esta ecuación debe ser aplicada a cada dimensión espacial, y debe considerarse también que la velocidad del viento es una función de las tres dimensiones espaciales más el tiempo. Como resultado se obtienen las siguientes ecuaciones:

$$\text{Longitud:} \quad x_{i+1} = x_i + v_x(t_i, z_i, y_i, x_i)\Delta t + \tilde{x}_i \tag{3.2}$$

$$\text{Latitud:} \quad y_{i+1} = y_i + v_y(t_i, z_i, y_i, x_i)\Delta t + \tilde{y}_i \tag{3.3}$$

$$\text{Altura:} \quad z_{i+1} = z_i + v_z(t_i, z_i, y_i, x_i)\Delta t + \tilde{z}_i \tag{3.4}$$

3.3. Pruebas en MATLAB

La primera actividad concreta que se realizó fue usar MATLAB para abrir los archivos de datos, ver cómo estaban definidas las variables, y cómo éstas se podían administrar para calcular eficientemente la ecuación descrita en la sección 3.2. Cabe señalar que estas pruebas solamente consideraron el uso de CPU, y que este paso sirvió finalmente para descartar el uso de MATLAB como una alternativa válida para la versión CPU del programa debido al tiempo tomado para ejecutar las interpolaciones de los campos de viento. Esta decisión se fundamentó al analizar los tiempos tomados por MATLAB para calcular las interpolaciones comparándolo con el mismo MATLAB pero haciendo uso de bibliotecas externas (código C compilado en archivos MEX, aplicando la ecuación descrita en la sección 2.3) para calcular las mismas interpolaciones. Ello dio la pista final que se necesitaba para optar por el uso total de C como lenguaje para la implementación del programa.

3.4. Diseño conceptual de la solución

El siguiente paso fue planificar y definir los módulos en que fueron divididas ambas versiones del programa. Esto fue importante, porque así se pudo tener una visión más completa de cómo iba a ser el proceso de desarrollo y se pudo saber desde un principio cuáles serían las etapas que iban a presentar mayor dificultad.

De esta manera, la implementación se dividió en módulos, algunos aplicables a ambas versiones del programa, y otros exclusivos para una de ellas. La visión general de este diseño puede ser apreciada en la figura 3.1.

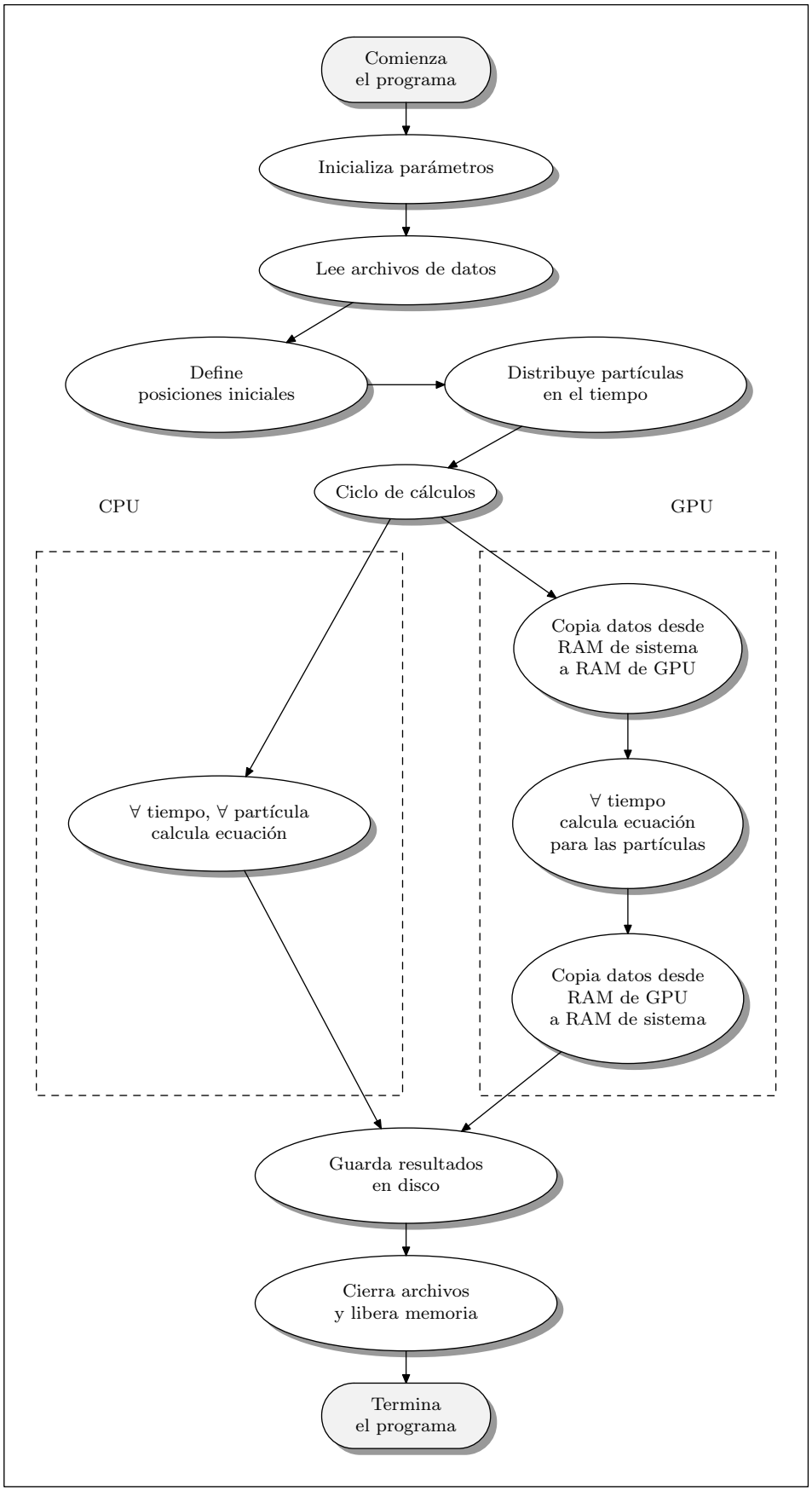


Figura 3.1: Módulos que componen el diseño preliminar del programa.

Este diseño preliminar sirvió para comenzar a desarrollar la implementación de las versiones del programa, razón por la cual a continuación se explica cómo fueron abordados cada uno de estos módulos, dejando en claro en qué manera fueron evolucionando a lo largo del proceso de desarrollo.

3.5. Estructura de los datos y planificación de los cálculos

Al momento de planificar la estructura de la información almacenada en memoria, se privilegió el asegurar que su acceso por parte de los módulos del programa fuese hecho de manera rápida, aunque ello implicase tanto un proceso más lento a la hora de leerla desde los archivos de datos, como de organizarla en memoria.

Se revisó la manera en que este proceso fue planificado para los dos más importantes conjuntos de datos procesados por el programa, a saber, la información del pronóstico del viento y las sucesivas posiciones de las partículas calculadas durante el programa, y que constituyen su trayectoria.

3.5.1. Información de la velocidad del viento

Para el caso de la velocidad del viento, su información es modelada como un hipercubo, por cuanto está distribuida en cuatro dimensiones: tiempo, altura, latitud y longitud. Por lo que en un principio se necesitaría usar un arreglo de cuatro dimensiones para almacenar sus datos. Sin embargo, temprano en la implementación del programa apareció el problema de que no era recomendable aplicar interpolación lineal en la dimensión correspondiente a la altura, por cuanto esta dimensión no es ortogonal con respecto a la longitud y la latitud (sección 2.1, página 12). En este caso, la primera solución ideada fue almacenar en memoria, para cada valor de la coordenada vertical, sendos cubos (llamados *cubos TYX* de ahora en adelante) con la información asociada a las dimensiones de tiempo, latitud y longitud. Vale decir, se divide el hipercubo de datos en *cubos TYX*, uno por cada altura.

De esta manera, para obtener un valor cualquiera al interior del hipercubo de datos, primero se realiza una interpolación trilineal en los dos cubos más cercanos a la altura buscada, y posteriormente estos dos valores se interpolan entre ellos de una manera especial, con el objeto de representar la irregularidad de la dimensión altura.

Por ejemplo, si se necesita la velocidad del viento en el punto P (ec. 3.5), se define un punto P' (ec. 3.6) asociable a los *cubos TYX*, y los dos valores $z_1, z_2 \in \mathbb{N}$ (ec. 3.7 y 3.8) que corresponden a los dos índices de altura más cercanos al valor buscado.

$$P = (t, z, y, x) \qquad t, z, y, x \in \mathbb{R} \tag{3.5}$$
$$P' = (t, y, x) \tag{3.6}$$
$$z_1 = \lfloor z \rfloor \tag{3.7}$$
$$z_2 = \lceil z \rceil \tag{3.8}$$

Entonces, se calcula las velocidades asociadas a z_1 y z_2 (mediante interpolación trilineal) en el punto P' (ec. 3.9 y 3.10) y se usan estos dos valores para calcular, mediante una interpolación especial (simbolizada por f en ec. 3.11), el valor final de la velocidad buscada. Es importante destacar que como $\mathbf{V}$ es un vector con tres componentes: (u, v, w), esta operación se realiza tres veces, una para cada componente.

$$\mathbf{V}_1 = \mathbf{V}_{z_1}(P') \tag{3.9}$$
$$\mathbf{V}_2 = \mathbf{V}_{z_2}(P') \tag{3.10}$$
$$\mathbf{V}(P) = f(\mathbf{V}_1, \mathbf{V}_2) \tag{3.11}$$

Todo esto, no obstante, no se pudo implementar exactamente como ha sido descrito, porque si bien el lenguaje de programación C permite usar arreglos de más de una dimensión, a la hora de traspasar la información a la memoria GPU RAM haciendo uso de "C for CUDA", esta extensión del lenguaje C recomienda que los datos en RAM de sistema estén almacenados en arreglos lineales [15], razón por la cual debía ser usado este tipo de arreglos para representar la información.

Entonces, la solución implementada finalmente fue usar arreglos lineales para guardar estos datos, pero ordenados en memoria de tal manera que lo que antes eran los *cubos TYX* —uno por cada nivel de altura—, ahora fuesen arreglos lineales, y todos ellos estuviesen "uno después del otro", —desde el primer al último nivel de altura—, en un mismo gran arreglo lineal con toda la información junta. De esta manera, para acceder a un valor en el vector del campo de viento se tiene que usar un índice producto de una transformación de los índices originales, es decir, que para obtener el valor de la velocidad del viento en una componente r (con $r \in \{u, v, w\}$), en la altura z, para un paso de tiempo t, latitud y y longitud x, hay que aplicar el *mapeo* [1] descrito en la ecuación 3.12, que permite pasar desde una notación $vel_r^{4d}[z][t][y][x]$ (para datos en cuatro dimensiones), a la notación $vel_r^{1d}[index]$ (para datos en una sola dimensión). Nótese que en esta última ecuación, los valores con prefijo FIELD_COUNT_ corresponden a los parámetros

[1] Con *mapear* se hace referencia al proceso de traducir la posición de un valor en un arreglo tridimensional (cubo) o bidimensional (matriz) a una posición en un arreglo lineal (vector)

del sistema descritos en el cuadro 3.2.

$$\begin{aligned} index(z,t,y,x) &= z * (\text{FIELD\_COUNT\_T} * \text{FIELD\_COUNT\_Y} * \text{FIELD\_COUNT\_X}) \\ &+ t * (\text{FIELD\_COUNT\_Y} * \text{FIELD\_COUNT\_X}) \\ &+ y * (\text{FIELD\_COUNT\_X}) + x \end{aligned} \tag{3.12}$$

3.5.2. Información de los niveles de altura

Análogo al caso de los campos de viento, la información de la altura (en metros sobre el nivel del mar) a la que está un punto dentro de la simulación, se obtiene mediante un campo escalar de alturas, el cual también depende de las cuatro dimensiones manejadas para el caso del pronóstico de la velocidad del viento (sección 2.1, página 12).

3.5.3. Posiciones de las partículas a lo largo de los pasos de tiempo

Para el caso de la estructura de datos que almacena las distintas posiciones en que se encuentran las partículas a lo largo del tiempo, también se eligió usar arreglos lineales, por las mismas razones mencionadas en la sección anterior. Como estos datos son referenciables mediante dos dimensiones: la que representa a las partículas, y la que representa a los pasos de tiempo, es necesario aplicar otra transformación, gracias a la cual para acceder a la posición de la partícula p, en la dimensión r (con $r \in \{x, y, z\}$), en el paso de tiempo t, el valor de esta posición representado por el arreglo $pos_r^{2d}[p][t]$ (de dos dimensiones), es *mapeado* a un valor $pos_r^{1d}[index]$ (en una sola dimensión), mediante la ecuación 3.13.

$$index(t,p) = (t * \text{NUM\_PARTICLES}) + p \tag{3.13}$$

Nuevamente, el valor NUM_PARTICLES corresponde a un parámetro del sistema, el cual representa la cantidad total de partículas cuyas trayectorias se calculan.

3.5.4. Precisión numérica usada para los cálculos

Todos los datos comentados más arriba son datos numéricos que para ser almacenados a nivel computacional han de estar expresados en una "precisión numérica" establecida, vale decir, con cuántos *bits* son representados al interior de la memoria RAM.

Lo interesante de este tema es que mientras todas las tarjetas gráficas son capaces de trabajar con precisión simple (32 *bits*), solamente algunas de ellas pueden trabajar con precisión doble (64 *bits*). Por esta razón se hizo importante definir —antes de realizar todas las implementaciones— en qué manera este sistema iba a discriminar qué tipo de precisión usar.

Para la implementación y las pruebas de este sistema, la tarjeta gráfica utilizada fue la GTX 285, que por pertenecer a la categoría "*Compute Capability 1.3*" soporta cálculos de precisión doble (64 *bits*) [15, página 103]. No obstante ello, son muchas las tarjetas gráficas que pertenecen a "*Compute Capability*" menores, como son aquéllos con códigos *1.2* ó *1.1*. Para permitir que este sistema sea ejecutable también en dispositivos con esas características, pero que también sea capaz de usar precisión doble cuando sea posible, es que en todo el sistema se usó lo que en el lenguaje de programación C se denomina **`typedef`**, que permite usar un *alias* en vez de un tipo de datos específico.

Por consiguiente se hizo uso de tres *alias*: **`pstn_t`** para tipos de datos que representan posiciones (que corresponden a los arreglos que almacenan las trayectorias calculadas para todas las partículas); **`wind_t`** para tipos de datos que representan velocidades del viento; y **`height_t`** para la información de los niveles de altura.

3.6. Definición de los parámetros del problema a resolver

Con el objeto de construir desde un principio un programa flexible, es que se definió leer dinámicamente los parámetros de configuración del problema a resolver, de tal manera que con solo cambiar estos parámetros, el programa sea capaz de adaptarse a distintas circunstancias en que sea usado, como puede ser tener que leer otro archivo de datos, utilizar otras posiciones iniciales, manejar un distinto número de partículas, etc.

Definir de qué manera el programa accede a esta información también es un tema a considerar, por cuanto son varias las alternativas, cada una con su correspondientes ventajas. Finalmente se escogió una de las soluciones disponibles, consistente en dejar todas estas variables de inicialización en un archivo de cabecera C (los llamados archivos .h), de tal manera que cada uno de estos parámetros es en realidad una constante para el programa. Las ventajas que ello conlleva son que cualquier error de formato en el archivo no repercute en tiempo de ejecución (por ser un archivo usado solamente en tiempo de compilación); también es bueno porque se ahorra todo el costo adicional (*overhead*) que implica tener que abrir un archivo de configuración especial, hacer un calce (*matching*), y detectar en tiempo de ejecución cualquier error en el formato de este archivo. La

única gran desventaja es que hay que compilar nuevamente el programa cada vez que se cambie alguno de estos parámetros.

Los parámetros principales del programa se muestran en los cuadros 3.1 y 3.2, donde estos valores son ordenados en función de si su alcance es general o es específico a los archivos de datos con la información de la velocidad del viento y de los niveles de altura.

Nombre de parámetro	Descripción
MEM_HOST	Bytes de memoria RAM que posee el sistema (CPU).
MEM_DEVICE	Bytes de memoria GPU RAM que posee la tarjeta de video (GPU).
SEEDEDFILE	Archivo con las posiciones iniciales de las partículas.
DELTA_T_SIM	Cuántas veces es dividido cada paso de tiempo de los datos del viento.
NUM_PARTICLES	Cantidad de partículas a desplazar simultáneamente.
PLOTPARTICLES	Cantidad de partículas cuyas trayectorias se almacenan como resultados (sección 3.9.9).
DELTA_T_TRAJ	Cada cuántos pasos de tiempo de la simulación se almacena en disco un resultado del cálculo de trayectorias.
BLOCKSIZE_X	Tamaño de la primera dimensión de los Bloques (sección 2.5).
BLOCKSIZE_Y	Tamaño de la segunda dimensión de los Bloques.
BLOCKSIZE_Z	Tamaño de la tercera dimensión de los Bloques.

Cuadro 3.1: Parámetros generales del programa

3.6.1. Cálculo de los pasos de tiempo totales de la simulación

Es muy importante señalar que el parámetro **DELTA_T_SIM** da cuenta de que los pasos en que está dividido el tiempo en este cálculo de trayectorias, no son necesariamente los mismos pasos de tiempo correspondientes a los datos de la velocidad del viento. Ello es así dado que aquel parámetro representa la cantidad de subdivisiones realizadas dentro de los cálculos de las trayectorias de partículas, sobre cada paso de tiempo asociado a los datos sobre velocidad del viento y niveles de altura. Vale decir que las trayectorias se calculan dividiendo el tiempo total de simulación en ***totalTimeSteps*** pasos de tiempo, valor calculado como:

$$\textbf{\textit{totalTimeSteps}} = \text{FIELD\_COUNT\_T} * \text{DELTA\_T\_SIM} \tag{3.14}$$

La razón de ser de esta subdivisión es que los datos de la velocidad del viento y de los niveles de altura están expresados en pasos de tiempo que en muchos casos abarcan

Nombre de parámetro	Descripción
FIELD_START_T	Índice inicial de la dimensión tiempo en el arreglo de datos.
FIELD_START_Z	Índice inicial de la dimensión altura en el arreglo de datos.
FIELD_START_Y	Índice inicial de la dimensión latitud en el arreglo de datos.
FIELD_START_X	Índice inicial de la dimensión longitud en el arreglo de datos.
FIELD_COUNT_T	El largo total de la dimensión tiempo que tiene el arreglo de datos.
FIELD_COUNT_Z	El largo total de la dimensión altura que tiene el arreglo de datos.
FIELD_COUNT_Y	El largo total de la dimensión latitud que tiene el arreglo de datos.
FIELD_COUNT_X	El largo total de la dimensión longitud que tiene el arreglo de datos.
DATA_FILE_NAME	Nombre del archivo con los datos.
SEG_T_SIM	Segundos reales entre cada paso de tiempo en el archivo de datos.
METERS_Z_SIM	Cantidad de metros reales entre cada subdivisión del espacio en la vertical.
METERS_Y_SIM	Cantidad de metros reales entre cada subdivisión del espacio en la latitud.
METERS_X_SIM	Cantidad de metros reales entre cada subdivisión del espacio en la longitud.
VARNAME_VEL_Z	Nombre de la variable que contiene la componente vertical velocidad.
VARNAME_VEL_Y	Nombre de la variable que contiene la componente latitud de la velocidad.
VARNAME_VEL_X	Nombre de la variable que contiene la componente longitud de la velocidad.
VARNAME_HGT	Variable que entrega la altura real en metros para un nivel dado (ver sección 2.1).

Cuadro 3.2: Parámetros del programa referidos al archivo de datos de velocidades y alturas

horas completas, pero para calcular el desplazamiento de las partículas es recomendable utilizar pasos de tiempo mucho más breves, del orden de los minutos.

3.6.2. Determinación de los momentos en que se guardan resultados

También cabe señalar que si bien los cálculos de las trayectorias de las partículas son realizados con la subdivisión del tiempo mencionada en el párrafo anterior, al momento de guardar los resultados en memoria —y posteriormente en disco— se hace necesario descartar algunos de estos valores con el fin de ahorrar espacio (ahorro que se realiza tanto en memoria RAM cuando se almacenan los datos, como en disco cuando éstos ya han sido escritos a un archivo de resultados). Es en este contexto que se hace uso del parámetro **DELTA_T_TRAJ**, el cual indica cada cuántos pasos de tiempo de la simulación es guardado en disco una posición para cada partícula modelada. Vale decir, que en total se almacena ***totalTrajSteps*** posiciones de la partícula a lo largo de toda la simulación,

valor que se calcula como:

$$\textbf{\textit{totalTrajSteps}} = \left\lceil \frac{\textbf{\textit{totalTimeSteps}}}{\text{DELTA\_T\_TRAJ}} \right\rceil = \left\lceil \frac{\text{FIELD\_COUNT\_T} * \text{DELTA\_T\_SIM}}{\text{DELTA\_T\_TRAJ}} \right\rceil \quad (3.15)$$

3.7. Cálculo de la memoria ocupada por el programa

Un tema muy sensible para este programa es la cantidad total de memoria que se consume: tanto para almacenar la información de la velocidad del viento en sus tres componentes; como para los niveles de altura; como también para los cálculos de las trayectorias de cada partícula en cada paso de tiempo.

Dado que para un hipercubo de n dimensiones, la cantidad total de datos que almacena corresponde a la multiplicación de los tamaños de cada una de sus dimensiones, se tiene que para el hipercubo de cuatro dimensiones que representa tanto las componentes de la velocidad del viento como la información de los niveles de altura, su cantidad total de valores almacenados —denominado para efectos de este trabajo como ***totalFieldElems***— se calcula como:

$$\textbf{\textit{totalFieldElems}} = \left(\prod_{i \in (z,t,y,x)} \text{FIELD\_COUNT\_}i \right) \quad (3.16)$$

Cabe señalar que los cálculos de consumo de memoria están expresados como el producto entre la cantidad total de datos que cada estructura almacena y el tamaño en *bytes* del tipo de datos usado para representar esta información en memoria (ya sea RAM de sistema o GPU RAM). Este último valor se expresa en las ecuaciones siguientes mediante la función "sizeof(**`data_type`**)", con **`data_type`** tomando el nombre del tipo de datos correspondiente al valor cuya memoria total ocupada se está calculando.

Velocidad del viento: En este caso la información corresponde a tres vectores iguales (uno para cada componente: u, v, w) que contienen datos para las tres dimensiones espaciales y la dimensión temporal, lo que es expresado como:

$$\textbf{\textit{totalWindMem}} = 3 * \textbf{\textit{totalFieldElems}} * \text{sizeof}(\textbf{\texttt{wind\_t}}) \quad (3.17)$$

donde se aprecia que cada uno de estos tres vectores almacena la información de un hipercubo que tiene las mismas dimensiones que la información original proveniente de los archivos de datos científicos.

Niveles de altura: En este caso la información corresponde a un solo vector con datos en función de cada dimensión espacial y temporal, por lo que el tamaño total es expresado como:

$$\textit{totalHgtMem} = \textit{totalFieldElems} * \text{sizeof}(\texttt{height\_t}) \tag{3.18}$$

Cálculos de trayectorias: Las trayectorias se calculan para cada una de las tres dimensiones espaciales, y son tantos valores como partículas se está modelando en total (parámetro **NUM_PARTICLES**), multiplicado por la cantidad total de pasos de tiempo en que se almacenan posiciones de las trayectorias de las partículas (valor ***totalTrajSteps***, definido en la sección 3.6.2).

$$\textit{totalTrajMem} = 3 * \text{NUM\_PARTICLES} * \textit{totalTrajSteps} * \text{sizeof}(\texttt{pstn\_t}) \tag{3.19}$$

El tipo de dato usado —al interior de la implementación— para representar la información fue siempre el mismo para todos los casos, razón por la cual —sin pérdida de generalidad— se puede redefinir este valor como ***typeSize***:

$$\textit{typeSize} = \text{sizeof}(\texttt{wind\_t}) = \text{sizeof}(\texttt{height\_t}) = \text{sizeof}(\texttt{pstn\_t}) \tag{3.20}$$

Por último, hay que considerar la memoria ocupada por todas las demás variables usadas en el programa —cuyo tamaño por cierto no es comparable al de las estructuras ya mencionadas— y que es denotado como ***otherVars***. Por lo tanto, sumando los valores de las ecuaciones 3.17, 3.18 y 3.19, y considerando el valor de ***otherVars***, se tiene que la memoria total consumida por el programa, denominada como ***totalMem***, es:

$$\begin{aligned}\textit{totalMem} &= \textit{totalWindMem} + \textit{totalHgtMem} + \textit{totalTrajMem} + \textit{otherVars} = \\ &(4 * \textit{totalFieldElems} + 3 * \text{NUM\_PARTICLES} * \textit{totalTrajSteps}) * \textit{typeSize} + \textit{otherVars}\end{aligned} \tag{3.21}$$

donde se puede apreciar que este valor es lineal con respecto al número total de partículas modelado, un aspecto muy importante a la hora de realizar el análisis de los resultados de este programa.

3.8. Determinación de la interfaz del programa

Para todo programa, una de sus características principales corresponde a la manera en que éste se comunica con el "mundo exterior", ya sea para recibir instrucciones, cargar datos iniciales, o entregar resultados, entre muchas otras cosas. Para el caso particular de

este proyecto se definió que la única forma de interacción del programa con el exterior tendría las siguientes propiedades:

- Su ejecución se realiza mediante una invocación a través de una "línea de comandos" de un terminal conectado al computador donde este sistema esté instalado.
- Los datos de entrada son obtenidos de diferente manera dependiendo de la naturaleza de esta información:
  - La versión de los cálculos por ejecutar (si es CPU o GPU) se define mediante un parámetro en la "línea de comandos".
  - Los parámetros iniciales se definen en un "archivo de cabecera .h" (ver sección 3.6).
  - Las posiciones iniciales de las partículas a modelar están definidas en un archivo ad-hoc, accesible por este programa.
- Los mensajes emitidos por el programa durante su ejecución son direccionados a la "salida estándar" del sistema que lo invocó.
- Los resultados finales que corresponden a las trayectorias calculadas son escritos en un archivo de texto, para su posterior análisis y graficado.

3.9. Módulos que componen la solución

Una vez analizada la información a ser usada por el programa, y considerado el diseño preliminar ilustrado en la figura 3.1, se implementó las dos versiones que componen el programa cuyo principal objetivo es calcular las trayectorias de partículas en la atmósfera sometidas a la acción de la fuerza del viento. El diseño final de esta solución, tal como fue implementada, se puede apreciar en la figura 3.2.

A continuación se describe en detalle cada uno de los pasos que componen la solución definitiva a este problema.

3.9.1. Inicialización de cronómetros

Como uno de los objetivos de este trabajo es evaluar el desempeño de las versiones CPU y GPU del programa, es que se cuenta el tiempo total tomado por este programa en su ejecución, como también el tiempo tomado solamente para realizar los cálculos por cada una de aquellas versiones.

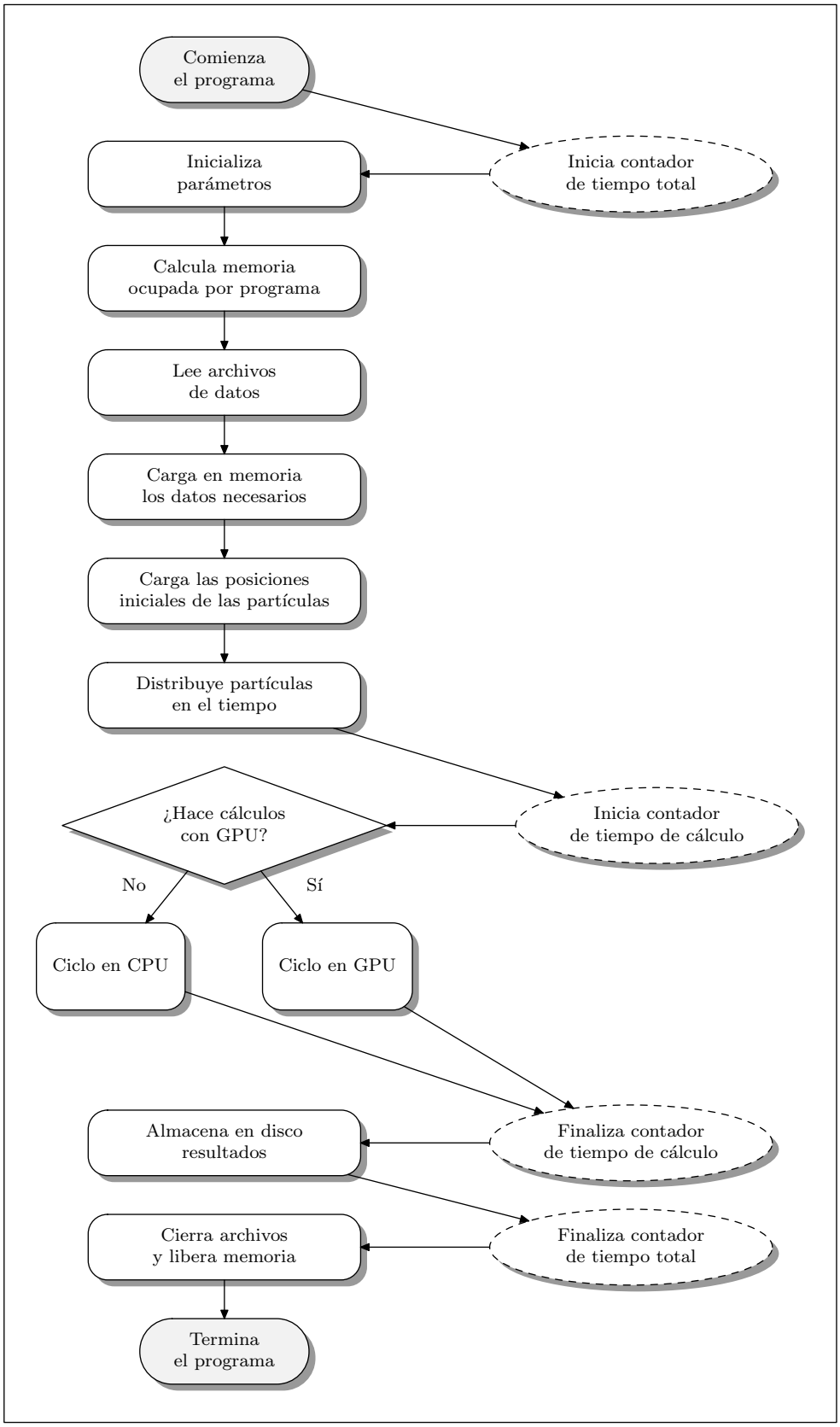


Figura 3.2: Esquema definitivo del diseño de los módulos del sistema

3.9.2. Cálculo de la memoria ocupada y declaración de variables

En este módulo el programa verifica si hay disponible suficiente memoria RAM de sistema y GPU RAM para almacenar tanto la información a extraer de los archivos de datos como la información de las posiciones de las partículas a medida que avanzan a lo largo del tiempo. Ello quiere decir que toda esta memoria no puede exceder ni la memoria RAM total del sistema (parámetro **MEM_HOST**) ni la memoria GPU RAM (parámetro **MEM_DEVICE**). En caso de calcularse que cualquiera de estas dos memorias pueda ser excedida, el programa termina con un mensaje de error.

En caso de sí contarse con la memoria necesaria, el sistema realiza la declaración de las variables y la reserva de la memoria RAM que ellas necesitan, mediante la invocación de la función que en el lenguaje de programación C se llama **`malloc`**.

3.9.3. Lectura de los archivos de datos y su almacenamiento en memoria RAM de sistema

Para acceder a la información contenida en los archivos de datos de velocidad del viento y niveles de altura, a través de un programa escrito en C, hay que usar la biblioteca de acceso que corresponda al formato del archivo en cuestión, y que sea compatible con el sistema operativo donde se realiza la compilación. Para el caso del formato NetCDF se utilizó la biblioteca NetCDF versión 4.0 para Linux, y para el caso del formato HDF5 se utilizó la biblioteca HDF5 versión 1.8.5 para Linux. En ambos casos las bibliotecas fueron compiladas desde sus propios códigos fuente.

Esta biblioteca provee diversas funciones para abrir los archivos, consultar cuáles variables tienen y extraer subconjuntos de datos para estas variables. Para acceder adecuadamente a la información de una variable, es necesario saber de antemano el nombre de ella tal cual está especificado en el archivo; la posición inicial desde la cual se extraen los datos; y la cantidad total de éstos que se quiere obtener.

Es importante considerar que al final de este módulo, toda la información de las velocidades del viento para la zona geográfica estudiada ya está cargada en la memoria RAM del sistema. También lo está la información de los niveles de altura.

3.9.4. Definición de posiciones iniciales

Otra de las características de este programa es que calcula las trayectorias de un número arbitrario de partículas (parámetro **NUM_PARTICLES**, ver cuadro 3.1). Por lo tanto es necesario saber en qué posiciones se ubican éstas en el momento $t = 0$ de la simulación.

Se decidió implementar un programa (llamado "Sembrador") independiente del calculador de trayectorias, que entregue para una zona predeterminada del espacio (un cubo con posiciones y dimensiones arbitrarias), las posiciones iniciales de las partículas cuyas trayectorias son calculadas. Este "Sembrador" escribe en un archivo de texto plano estas coordenadas, de manera tal que el programa principal conoce su formato y sabe dónde encontrarlo (parámetro **SEEDEDFILE**). Así, si en el futuro se necesita calcular las trayectorias de partículas que comiencen su viaje en posiciones arbitrarias, solamente hay que dejar sus posiciones iniciales en ese archivo y el programa principal funciona sin problemas.

Para el caso de las pruebas finales a las que fue sometido el programa principal, las partículas fueron "sembradas" en un cubo ubicado justo por encima de la chimenea del Volcán Chaitén, que es la "posición inicial" de las partículas una vez que abandonan el interior del Volcán.

3.9.5. Distribución de partículas en el tiempo

Este programa está diseñado para que modele el viaje de muchas partículas (parámetro **NUM_PARTICLES**), las cuales no inician su movimiento simultáneamente, sino que son distribuidas de manera tal que en cada paso de tiempo de la simulación, un grupo de ellas es liberado desde su posición inicial. Vale decir, que si la simulación consta de ***totalTimeSteps*** pasos de tiempo, entonces en cada uno de ellos la cantidad de partículas liberadas corresponde al valor ***particlesByTime***, definido como:

$$\boldsymbol{particlesByTime} = \left\lceil \frac{\text{NUM\_PARTICLES}}{\boldsymbol{totalTimeSteps}} \right\rceil \tag{3.22}$$

cuyo cálculo se realiza en el marco del algoritmo 1, línea 1.

Cabe señalar que este diseño implica una tasa constante de emisión de partículas a lo largo del tiempo, lo que ciertamente no es el modelo más realista, no obstante para efectos del principal objetivo de este trabajo, cual es, hacer uso de GPU para calcular la

trayectoria de las partículas y comprobar que este método es más eficiente que el uso clásico de CPU, no se consideró como prioritario implementar un modelo estocástico de emisión de contaminantes con respecto al tiempo. También hay que considerar que la manera en que son emitidas las partículas es un problema que depende íntimamente de la naturaleza del proceso modelado, por lo que para un acercamiento genérico como es este programa, es algo que reviste complicaciones que van más allá de sus límites naturales.

Algoritmo 1. Distribución de las partículas modeladas a lo largo del tiempo

```
   required : NUM_PARTICLES, la cantidad total de partículas
   required : totalTimeSteps, la cantidad total de pasos de tiempo
1  particlesByTime ← ⌈NUM_PARTICLES / totalTimeSteps⌉
2  releasedNow ← 0
3  for t ← 1 to totalTimeSteps do
4      releasedNow ← releasedNow + particlesByTime
5      for p ← 1 to NUM_PARTICLES do
6          if p < releasedNow then
7              Calcula nueva posición de la partícula.
8          else
9              Copia la posición anterior de la partícula como la posición actual.
```

3.9.6. Ciclo de cálculos

Una vez completados los pasos anteriores, el programa está listo para empezar a realizar los cálculos propiamente tales. Es en este momento cuando el programa se comporta de forma diferente dependiendo de cuál versión es la usada: CPU o GPU. Ello se controla mediante un parámetro entregado en "línea de comandos".

No obstante las diferencias entre estos dos procesos, existe un punto en común que es conveniente explicitar de inmediato: dado que ambas versiones buscan obtener el mismo tipo de resultado —las trayectorias finales de las partículas—, estos valores son almacenados en la memoria RAM (del sistema o de la tarjeta de video, dependiendo del caso), y es de fundamental importancia asegurarse de que el almacenamiento de estos valores no sobrepase la memoria total disponible. En el caso GPU se aprecia más adelante que esto es especialmente crítico.

A continuación se explica por separado los pasos específicos llevados a cabo para cada una de las versiones del programa.

3.9.7. Ciclo de cálculos en CPU

Para el caso de trabajar en CPU, hay que realizar los cálculos al interior de dos ciclos anidados: uno para cada paso de tiempo y otro para cada partícula, proceso que se muestra en el algoritmo 2. El procedimiento es el siguiente: en cada paso de tiempo se obtiene —según lo explicado en la sección 3.5— las velocidades del viento para cada una de las partículas, y se calcula entonces, haciendo uso de la ecuación de movimiento (sección 3.2, ecuación 3.1), los desplazamientos durante el paso de tiempo en cuestión.

3.9.8. Ciclo de cálculos en GPU

Para describir la implementación de este módulo —que corresponde a la parte central de todo este trabajo—, se menciona a continuación cada uno de los pasos llevados a cabo, junto con mencionar las funciones propias del API de CUDA [15] que fueron usadas en cada caso. Estos pasos están esquematizados en el diagrama de la figura 3.3.

Selecciona GPU

El computador usado para este trabajo (sus características están descritas en el apéndice A), cuenta con dos tarjetas de video, ambas compatibles con CUDA, razón por la cual es posible en tiempo de ejecución del programa, decidir cuál de las dos usar. Para estos efectos, CUDA provee de dos funciones: **`cudaGetDeviceCount()`**, la cual entrega la cantidad de tarjetas disponibles, y **`cudaSetDevice()`**, con la cual se define, mediante un índice que parte desde 0, cuál dispositivo utilizar. En el caso de este trabajo, dado que la primera tarjeta estaba siendo ocupada por el sistema operativo para todas las tareas gráficas usuales (tales como la gestión del escritorio), se ocupó siempre la segunda tarjeta, por cuanto ésta tenía toda su GPU RAM disponible. También hay que señalar que esta es la primera acción que se debe realizar al tratar de usar una GPU, dado que al primer momento en que se declara o inicializa una variable, CUDA hace uso de la tarjeta gráfica por omisión, tras lo cual ya no se puede usar otra durante la ejecución del programa.

Declara variables

Una vez seleccionada la tarjeta, las variables ya pueden ser declaradas. Para las información de las trayectorias —las posiciones de todas las partículas a lo largo de la simulación— se usó un arreglo lineal (**`float *`**), tal como se menciona en la sección 3.5.3, página 32. Sin embargo, para el caso del pronóstico del viento se hizo uso, dentro de la memoria GPU RAM, de una estructura especial provista por "C for CUDA": los "Arreglos 3D CUDA", los cuales son arreglos en tres dimensiones —existen también en dos— especialmente diseñados para ser utilizados en conjunto con unidades de textura.

Algoritmo 2. Ciclo de cálculos realizados en CPU

required : NUM_PARTICLES, la cantidad total de partículas
required : SEG_T_SIM, cantidad de segundos entre cada paso de tiempo
required : METERS_Z_SIM, cantidad de metros entre cada división del espacio en el eje Z
required : METERS_Y_SIM, cantidad de metros entre cada división del espacio en el eje Y
required : METERS_X_SIM, cantidad de metros entre cada división del espacio en el eje X

required : ***totalTimeSteps***, la cantidad total de pasos de tiempo
required : ***trajMat_uVector***, vector de posiciones en el eje X
required : ***trajMat_vVector***, vector de posiciones en el eje Y
required : ***trajMat_wVector***, vector de posiciones en el eje Z

required : ***velMat_uVector***, campo de velocidades en el eje X
required : ***velMat_vVector***, campo de velocidades en el eje Y
required : ***velMat_wVector***, campo de velocidades en el eje Z
required : ***hgtVector***, campo escalar niveles de altura
required : ***particlesByTime***, partículas que inician su movimiento en cada paso de tiempo

1 $particlesByTime \leftarrow \lceil$ NUM_PARTICLES / ***totalTimeSteps*** $\rceil$
2 $releasedNow \leftarrow 0$
3 $metersToSeconds_x \leftarrow$ SEG_T_SIM / (DELTA_T_SIM $*$ METERS_X_SIM)
4 $metersToSeconds_y \leftarrow$ SEG_T_SIM / (DELTA_T_SIM $*$ METERS_Y_SIM)
5 $metersToSeconds_z \leftarrow$ SEG_T_SIM / (DELTA_T_SIM $*$ METERS_Z_SIM)
6 **for** $t \leftarrow 1$ **to** ***totalTimeSteps*** **do**
7 $releasedNow \leftarrow releasedNow + particlesByTime$
8 **for** $p \leftarrow 1$ **to** $NUM\_PARTICLES$ **do**
9 $t' \leftarrow t - 1$
10 **if** $p < releasedNow$ **then**
11 $pos_x \leftarrow$ ***trajMat_uVector***$[t'][p]$
12 $pos_y \leftarrow$ ***trajMat_vVector***$[t'][p]$
13 $pos_z \leftarrow$ ***trajMat_wVector***$[t'][p]$
14 $vel_x \leftarrow$ `interpolateVelocity(`***velMat_uVector***$,t', pos_x, pos_y, pos_z$`)`
15 $vel_y \leftarrow$ `interpolateVelocity(`***velMat_vVector***$,t', pos_x, pos_y, pos_z$`)`
16 $vel_z \leftarrow$ `interpolateVelocity(`***velMat_wVector***$,t', pos_x, pos_y, pos_z$`)`
 `/* Transforma velocidad vertical */`
17 $gradHgt_x \leftarrow$ `getGradientX(`***hgtVector***$,t', pos_x, pos_y, pos_z$`)`
18 $gradHgt_y \leftarrow$ `getGradientY(`***hgtVector***$,t', pos_x, pos_y, pos_z$`)`
19 $vel_z \leftarrow vel_z - (vel_x * gradHgt_x + vel_y * gradHgt_y)$
 `/* Calcula nuevas posiciones */`
20 ***trajMat_uVector***$[t][p] = pos_x + vel_x * metersToSeconds_x$
21 ***trajMat_vVector***$[t][p] = pos_y + vel_y * metersToSeconds_y$
22 ***trajMat_wVector***$[t][p] = pos_z + vel_z * metersToSeconds_z$
23 **else**
24 ***trajMat_uVector***$[t][p] =$ ***trajMat_uVector***$[t'][p]$
25 ***trajMat_vVector***$[t][p] =$ ***trajMat_vVector***$[t'][p]$
26 ***trajMat_wVector***$[t][p] =$ ***trajMat_wVector***$[t'][p]$

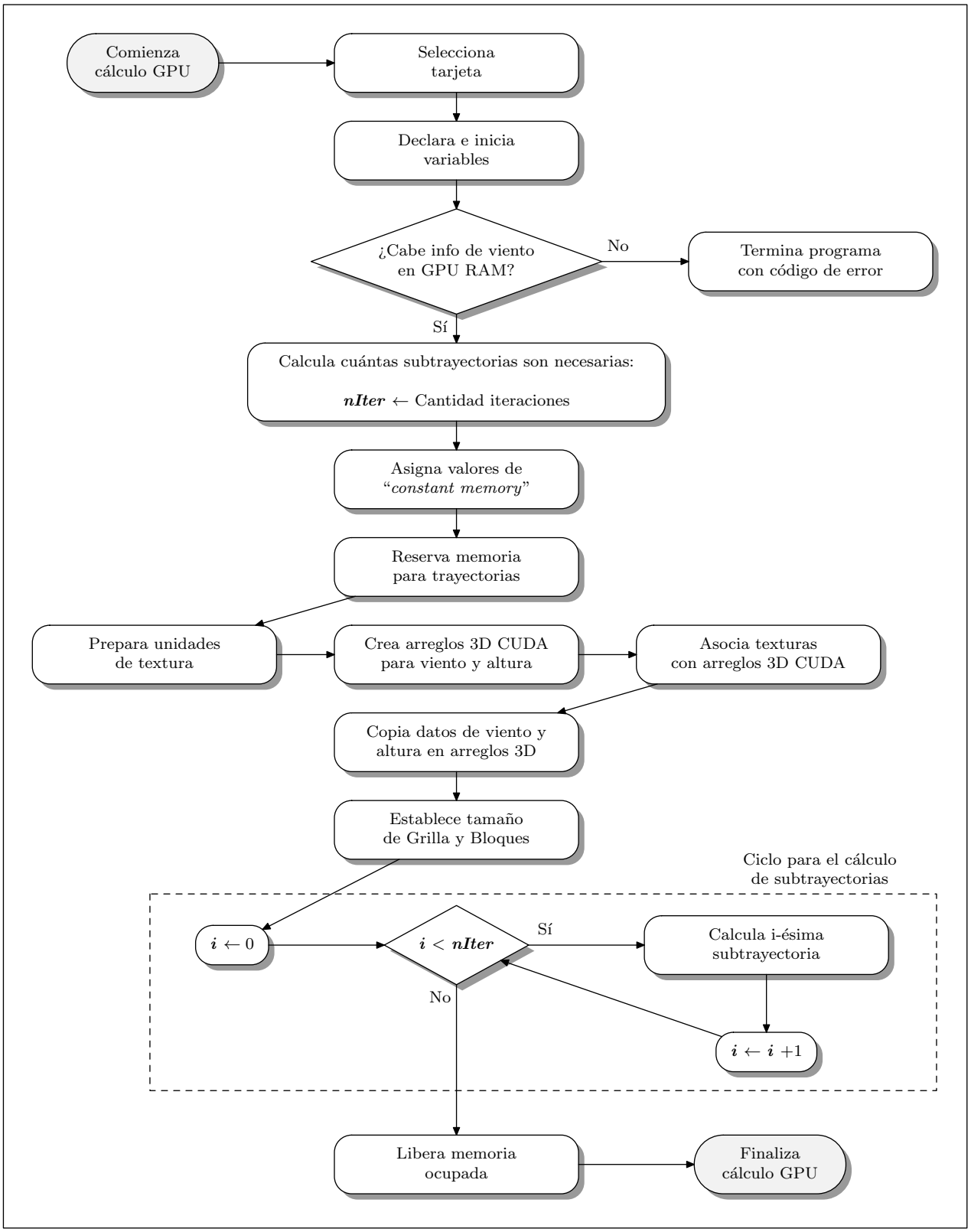


Figura 3.3: Esquema de los pasos llevados a cabo en el ciclo de cálculos en GPU

Verifica si información del viento cabe en GPU RAM

Medida se seguridad, para evitar reservar más memoria de la que se tiene a disposición. En caso de no caber toda la información del viento en memoria, no es posible seguir realizando cálculos, por lo que el programa termina con un mensaje de error.

Verifica si hay que dividir las trayectorias por falta de GPU RAM

En caso que la información requerida para representar todo el desplazamiento de las partículas a lo largo de la simulación sea demasiado grande para caber en GPU RAM (la cual ya está parcialmente ocupada por la información de la velocidad del viento), el procedimiento que se realiza es procesar tantos pasos de tiempo como sea posible, hasta ocupar todo el espacio disponible en GPU RAM, después de lo cual se copia los resultados obtenidos a RAM de sistema, y se sigue procesando los siguientes pasos de tiempo, hasta completar toda la simulación. Esta división de las trayectorias se denomina "subtrayectorias" a lo largo de este informe. Este proceso está especificado en detalle en el algoritmo 3, donde se puede observar que se está realizando un total de *nIter* iteraciones (tal como se calcula en las líneas 4 y 7 de este algoritmo).

Cabe señalar que para la última de estas *nIter* iteraciones, eventualmente puede quedar por calcular una cantidad de datos menor a *mAvail* (caso que se produce cuando la división de la línea 4 no es exacta) lo que el programa detecta (línea 15) procesando entonces sólo la cantidad restante de pasos de tiempo (evitando de esta manera excederse en el límite final de pasos de tiempo para esta última iteración).

Carga valores en la sección de Memoria de Constantes de la GPU

Las GPU cuentan con una memoria especial, destinada a alojar valores constantes (ver sección 2.5.2, página 22), la cual es inicializada de manera especial mediante la función `cudaMemcpyToSymbol()`. Dentro de los kernels, estas variables son accesadas directamente y mantienen su valor durante toda la vida de la aplicación.

Reserva memoria para la información de las trayectorias

En este caso, luego de saber la cantidad de memoria GPU RAM que puede ser destinada para almacenar la información de las trayectorias, esta memoria es reservada mediante el llamado a la función `cudaMalloc()`.

Prepara las unidades de textura

Aquí se lleva a cabo otra de las actividades más importantes de todo el cálculo con GPU: la declaración de las unidades de textura a través de las cuales se obtienen los valores interpolados de los campos de viento (campo vectorial) y de la altura (campo escalar). En esta sección se define la manera en que son operadas las texturas, cuántas dimensiones tienen los datos por ellas manipulados y la forma en que se manejan los índices de cada una de esas dimensiones. Para más detalle consultar sección 2.4.3, página 18.

Algoritmo 3. Determinación y cálculo de subtrayectorias

required : NUM_PARTICLES, cantidad total de partículas modeladas
required : MEM_DEVICE, cantidad de memoria RAM GPU disponible
required : DELTA_T_TRAJ, Cada cuántos pasos de tiempo se guarda una posición calculada de las trayectorias

required : ***totalTimeSteps***, pasos de tiempo totales de la simulación
required : ***totalTrajSteps***, pasos de tiempo necesarios para guardar info de trayectorias
required : ***totalWindMem***, memoria total ocupada por la velocidad del viento
required : ***totalHgtMem***, memoria total ocupada por los niveles de altura
required : ***typeSize***, cantidad de *bytes* necesarios para almacenar cada valor en memoria

required : ***trajMat_uVector***, vector de posiciones en el eje X
required : ***trajMat_vVector***, vector de posiciones en el eje Y
required : ***trajMat_wVector***, vector de posiciones en el eje Z

$mAvail \leftarrow$ MEM_DEVICE $-$ ***totalWindMem*** $-$ ***totalHgtMem***
$maxTimeSteps \leftarrow \lfloor mAvail/(3 * \text{NUM\_PARTICLES} * \textbf{typeSize}) \rfloor - 1$
if ***totalTrajSteps*** $>$ $maxTimeSteps$ **then**
    $nIter \leftarrow \lceil \textbf{totalTrajSteps}/maxTimeSteps \rceil$
    $timeStepsPerIteration \leftarrow maxTimeSteps - 1$
else
    $nIter \leftarrow 1$
    $timeStepsPerIteration \leftarrow$ ***totalTrajSteps***
`/* Divide el cálculo de trayectorias en` $nIter$ `subtrayectorias */`
for $i \leftarrow 1$ **to** $nIter$ **do**
    `cleanVectors(`$positions_x, positions_y, positions_z$`)` `/* Inicia en 0 estos arreglos */`
    $trajTimeStepStart \leftarrow timeStepsPerIteration * (i-1)$
    $trajTimeStepEnd \leftarrow timeStepsPerIteration * i$
    $timeStepStart \leftarrow timeStepsPerIteration * (i-1) *$ DELTA_T_TRAJ
    $timeStepEnd \leftarrow timeStepsPerIteration * i *$ DELTA_T_TRAJ $- 1$
    if $timeStepEnd$ $>$ ***totalTimeSteps*** **then**
        $trajTimeStepEnd \leftarrow$ ***totalTrajSteps***
        $timeStepEnd \leftarrow$ ***totalTimeSteps***
    `/* Copia las posiciones iniciales de las trayectorias */`
    $positions_x \leftarrow$ Posiciones de partículas en ***trajMat_uVector*** para $t = timeStepStart$
    $positions_y \leftarrow$ Posiciones de partículas en ***trajMat_vVector*** para $t = timeStepStart$
    $positions_z \leftarrow$ Posiciones de partículas en ***trajMat_wVector*** para $t = timeStepStart$
    {Se invoca al Kernel para calcular i-ésima subtrayectoria}
    `kernel(`$positions_x, positions_y, positions_z, timeStepStart, timeStepEnd$`)`
    `/* Una vez terminado el Kernel se sincronizan los Hilos */`
    `cudaThreadSynchronize()`
    `/* Se copia los resultados desde GPU RAM a RAM de sistema */`
    trajMat_uVector$[trajTimeStepStart \,..\, trajTimeStepEnd] \leftarrow positions_x$
    trajMat_vVector$[trajTimeStepStart \,..\, trajTimeStepEnd] \leftarrow positions_y$
    trajMat_wVector$[trajTimeStepStart \,..\, trajTimeStepEnd] \leftarrow positions_z$

Crea los arreglos 3D de CUDA para el viento y la altura

Una de las estructuras de datos especiales que provee CUDA corresponde a los "Arreglos 3D", que son disposiciones de memoria optimizadas para su uso con información matricial que va a ser interpolada dentro de la GPU (ver sección 2.4.3, página 18). Con el objetivo de reservar memoria GPU RAM para un arreglo 3D, se ejecuta la función `cudaMalloc3DArray()`, indicándole el tamaño de cada dimensión.

Asocia los arreglos 3D de CUDA con las texturas

En este paso se asocia los arreglos 3D de CUDA con las unidades de textura ya preparadas, mediante la función `cudaBindTextureToArray()`.

Copia los datos de viento y altura en los arreglos 3D de CUDA

Una vez listos los arreglos 3D y las unidades de textura, se copia los datos de viento y altura que están en RAM a GPU RAM. Para el caso de copiar datos a arreglos 3D, se hace uso de la función `cudaMemcpy3D()`, previa preparación de los datos originales mediante la función `make_cudaPitchedPtr()`, la cual optimiza la disposición de los datos en GPU RAM para obtener el máximo beneficio del uso de texturas, y así poder realizar interpolaciones sobre esos datos.

Establece el tamaño de la Grilla y de los Bloques

El siguiente paso es muy importante: se establece las dimensiones tanto de la Grilla como de los Bloques en que están distribuidos los Hilos del proceso. Para este caso, se definió un Bloque de Hilos unidimensional, con un valor dado por el parámetro de sistema **BLOCKSIZE_X** (cuadro 3.1), que para efectos de las últimas pruebas tuvo un valor de 256. Para el caso de la Grilla, ésta también fue definida unidimensionalmente, con un tamaño dado por la ecuación 3.23, la cual da cuenta de que todas las partículas son procesadas en sendos Hilos, y por lo tanto hay tantos Bloques (cabe recordar que la dimensión total de una Grilla corresponde al número total de Bloques ejecutados) como sea necesario para acoger el total de partículas modeladas por el sistema, y cada uno de estos Bloques tiene un total de **BLOCKSIZE_X** Hilos.

$$gridSizeX = \left\lceil \frac{\text{NUM\_PARTICLES}}{\text{BLOCKSIZE\_X}} \right\rceil \tag{3.23}$$

Para toda iteración de subtrayectorias

En caso de tener que dividir las trayectorias por falta de GPU RAM, los siguientes pasos —explícitamente indicados como tales en sus propios títulos— se repiten tantas veces como sea necesario, para llevar a cabo los cálculos requeridos hasta completar todos los pasos de tiempo en que está dividido el modelo. Cabe señalar que estos pasos fueron diseñados de manera tal que si la memoria GPU RAM sí fuese suficiente para almacenar

todos los cálculos de las trayectorias de las partículas, entonces se realiza un solo ciclo de "subtrayectorias", que comienza en el primer paso de tiempo de la simulación, y finaliza en el último de estos pasos de tiempo, tal como se aprecia en las líneas 6 a 8 del algoritmo 3.

En iteración "i": Limpia los arreglos de trayectorias

Al momento de empezar a realizar los cálculos —y por lo tanto comenzar a almacenar sus resultados en memoria—, se inicializan en 0 los valores del arreglo lineal donde son almacenadas las trayectorias calculadas de las partículas, haciendo uso de la función `cudaMemset()`. Esto es muy importante, porque si llegase a ser necesario dividir las trayectorias por falta de memoria, en este paso el programa asegura que el espacio en memoria está *fresco* para recibir nuevos datos. También es importante porque para el caso de la última subtrayectoria —que por lo general abarca un número menor de pasos de tiempo que las anteriores—, es recomendable que el arreglo con resultados no esté contaminado con datos de iteraciones pasadas que pueden ser mal utilizados por esta última iteración. Esto está representado en la línea 10 del algoritmo 3.

En iteración "i": Copia las posiciones iniciales de las trayectorias

En el principio de toda iteración, es necesario saber la posición inicial de las partículas, lo cual se hace traspasando estos valores desde la memoria RAM de sistema a GPU RAM mediante el llamado a la función `cudaMemcpy()`, dándole el parámetro especial `cudaMemcpyHostToDevice`, que indica que el copiado se realiza en el sentido que corresponde a este caso. Esto está representado en entre las líneas 18 a 20 del algoritmo 3.

En iteración "i": Se invoca al Kernel

He aquí el punto cúlmine de todo el proceso de cálculo en GPU: la invocación del Kernel, el cual es precisamente el conjunto de rutinas a ser ejecutadas por cada uno de los Hilos paralelos en que está dividido el problema, y que para el caso particular de este trabajo, corresponde al cálculo paralelo de cada una de las partículas cuyas trayectorias se está determinando. En el marco del cálculo de subtrayectorias este paso está representado en la línea 21 del algoritmo 3 y todos sus pasos están descritos en más detalle en el algoritmo 4.

Dado que la cantidad total de Hilos que componen la Grilla puede ser mayor que la cantidad de partículas modeladas por el sistema (sección 2.5), lo primero que hace el Kernel es verificar que su *identificación* (línea 1 del algoritmo 4) sea asociable a una partícula, y en caso de no ser así, simplemente termina. En caso de ser un Hilo válido ejecuta —para cada paso de tiempo en que esté dividida la subtrayectoria actual— todos los pasos descritos en detalle en los párrafos siguientes, los cuales están indicados como tales de forma explícita en su propio título, y que cuando corresponda hacen referencia a determinadas líneas del algoritmo 4.

Algoritmo 4. Kernel

required : NUM_PARTICLES, la cantidad total de partículas

required : ***particlesAlreadyReleased***, partículas ya liberadas al momento de invocar al kernel

required : ***particlesByTime***, partículas que inician su movimiento en cada paso de tiempo

required : ***timeStepStart***, paso de tiempo inicial para la subtrayectoria actual

required : ***timeStepEnd***, paso de tiempo final para la subtrayectoria actual

required : ***trajMat_uVector***, vector de posiciones en el eje X

required : ***trajMat_vVector***, vector de posiciones en el eje Y

required : ***trajMat_wVector***, vector de posiciones en el eje Z

required : ***texUvel***, referencia a la textura que maneja la velocidad en el eje X

required : ***texVvel***, referencia a la textura que maneja la velocidad en el eje Y

required : ***texWvel***, referencia a la textura que maneja la velocidad en el eje Z

required : ***texHgt***, referencia a la textura que maneja los niveles de altura

`/* Factores para obtener el desplazamiento en metros producido por una velocidad dada, para los pasos de tiempo de la simulación */`

required : ***metersToSeconds_x***

required : ***metersToSeconds_y***

required : ***metersToSeconds_z***

`/* Índice del thread actual (en base a variables internas de CUDA) */`

1 $threadIdx \leftarrow (blockIdx.y * gridDim.x + blockIdx.x) * blockDimx + threadIdx.x$

`/* Verifica que es un thread asociable a una partícula */`

2 **if** $threadIdx < NUM\_PARTICLES$ **then**

3 **for** $t \leftarrow$ ***timeStepStart*** $+ 1$ **to** ***timeStepEnd*** **do**

4 $t' \leftarrow t - 1$

`/* Verifica si la partícula ya puede moverse */`

5 **if** $threadIdx <$ (***particlesAlreadyReleased*** $+$ ***particlesByTime***) **then**

6 $pos_x \leftarrow$ ***trajMat_uVector***$[t'][p]$

7 $pos_y \leftarrow$ ***trajMat_vVector***$[t'][p]$

8 $pos_z \leftarrow$ ***trajMat_wVector***$[t'][p]$

9 $vel_x \leftarrow$ `interpolate4D(`***texUvel***$,t', pos_x, pos_y, pos_z$`)`

10 $vel_y \leftarrow$ `interpolate4D(`***texVvel***$,t', pos_x, pos_y, pos_z$`)`

11 $vel_z \leftarrow$ `interpolate4D(`***texWvel***$,t', pos_x, pos_y, pos_z$`)`

`/* Transforma velocidad vertical */`

12 $gradHgt_x \leftarrow$ `getGradientX(`***texHgt***$,t', pos_x, pos_y, pos_z$`)`

13 $gradHgt_y \leftarrow$ `getGradientY(`***texHgt***$,t', pos_x, pos_y, pos_z$`)`

14 $vel_z \leftarrow vel_z - (vel_x * gradHgt_x + vel_y * gradHgt_y)$

`/* Calcula nuevas posiciones */`

15 ***trajMat_uVector***$[t][idx] = pos_x + vel_x *$ ***metersToSeconds_x***

16 ***trajMat_vVector***$[t][idx] = pos_y + vel_y *$ ***metersToSeconds_y***

17 ***trajMat_wVector***$[t][idx] = pos_z + vel_z *$ ***metersToSeconds_z***

18 **else**

19 ***trajMat_uVector***$[t][idx] =$ ***trajMat_uVector***$[t'][idx]$

20 ***trajMat_vVector***$[t][idx] =$ ***trajMat_vVector***$[t'][idx]$

21 ***trajMat_wVector***$[t][idx] =$ ***trajMat_wVector***$[t'][idx]$

Algoritmo 5. Interpolación en cuatro dimensiones versión GPU

input : ***texRef***, referencia a la textura que maneja la variable (velocidad o altura) por interpolar
input : $\boldsymbol{t}$, paso de tiempo en el cual se interpola
input : $\boldsymbol{pos_x}$, longitud en la cual se interpola
input : $\boldsymbol{pos_y}$, latitud en la cual se interpola
input : $\boldsymbol{pos_z}$, altura en la cual se interpola
output : $varInterp$, el valor interpolado

1 $posFloor_z \leftarrow \lfloor \boldsymbol{pos_z} \rfloor$
2 $posRemainder_z \leftarrow \boldsymbol{pos_z} - posFloor_z$
3 $varInterp \leftarrow$ `tex3D(`***texRef***, $\boldsymbol{t}$, $\boldsymbol{pos_x}$, $\boldsymbol{pos_y}$, $posFloor_z$`)`
4 **if** $posRemainder_z \neq 0$ **then**
5 $\quad varInterp' \leftarrow$ `tex3D(`***texRef***, $\boldsymbol{t}$, $\boldsymbol{pos_x}$, $\boldsymbol{pos_y}$, $(posFloor_z + 1)$`)`
6 $\quad varInterp \leftarrow varInterp * (1 - posRemainder_z) + varInterp' * posRemainder_z$

En iteración "i", dentro del Kernel: Declara las variables para expresar posiciones

Dado que la posición de una partícula está definida por sus tres coordenadas espaciales más el tiempo, corresponde a un vector de cuatro componentes. Por esta razón se hizo uso del tipo de datos especial de "C for CUDA" para estos efectos: **`float4`**, mediante la función que los crea: **`make_float4()`**. Estas estructuras están optimizadas a nivel de bits dentro de la GPU RAM para efectuar de la mejor manera los cálculos [15].

En iteración "i", dentro del Kernel: Verifica si la partícula ya puede moverse

Dado que el momento en que las partículas comienzan su recorrido se distribuye a lo largo de todo el período de modelación (sección 3.9.5), en el caso de que el Kernel esté asociado a una partícula que todavía no deba desplazarse, simplemente considera la posición actual de ella como resultado de todo el cálculo (líneas 19 a 21 del algoritmo 4), y procesa el paso de tiempo siguiente.

En iteración "i", dentro del Kernel: Calcula desplazamientos en ejes x e y

En este paso se obtienen las componentes horizontales de la velocidad del viento —componentes asociadas a la longitud y latitud— (líneas 9 y 10 del algoritmo 4). Esto se logra mediante el uso de las unidades de textura asociadas a estas componentes de la velocidad del viento, calculando con ellas la interpolación lineal en cuatro dimensiones que permite obtener este valor (ver algoritmo 5). Es importante señalar que para obtener un valor a través del uso de una unidad de textura, se hace uso de la función **`tex3D()`** (líneas 3 y 5 del algoritmo 5).

Luego de ello, se aplica estos valores en la ecuación de trayectoria (ecuación 3.1, página 27) y se calcula entonces los desplazamientos en los ejes x e y (líneas 15 y 16 del algoritmo 4).

En iteración "i", dentro del Kernel: Interpola la componente vertical de la velocidad

Luego procede a usar la unidad de textura para la componente vertical (línea 11 del algoritmo 4). Este valor, como ya se sabe, es obtenido mediante la aplicación de una interpolación lineal, lo cual por tratarse de información relativa a la altura no es suficiente, siendo necesario calcular su corrección gracias al paso siguiente.

En iteración "i", dentro del Kernel: Calcula el gradiente de la altura

En este momento se hace necesario calcular los gradientes de los datos de altura con respecto a los ejes x e y (líneas 12 y 13 del algoritmo 4) para aplicar la ecuación 2.7 (página 14), y corregir la velocidad vertical calculada en el paso anterior.

El cálculo de este gradiente se ejecuta también al interior de la GPU, usando las unidades de textura asociadas a la información de la altura (que también corresponde a un campo variable en las cuatro dimensiones: sección 2.1, página 12), aplicando los pasos del algoritmo 5, y con los cuales calcula de forma aproximada su gradiente.

En iteración "i", dentro del Kernel: Calcula desplazamiento en eje z

Una vez corregida la velocidad vertical, se calcula el desplazamiento correspondiente en el eje z (línea 17 del algoritmo 4) de forma análoga a lo ya hecho con los ejes x e y.

En iteración "i": Se finaliza el Kernel y se sincronizan los Hilos

Una vez invocado el Kernel, y debido a su carácter asíncrono, es necesario ejecutar la función `cudaThreadSynchronize()`, la cual se asegura de continuar con los siguientes pasos solamente cuando todos los Hilos ejecutando las rutinas del Kernel hayan terminado sus cálculos y guardado sus datos en memoria. Esto se representa en la línea 23 del algoritmo 3.

En iteración "i": Se copia a RAM de sistema los resultados y finaliza iteración

Una vez realizados los cálculos, y teniendo completos todos los valores en el arreglo de trayectorias para la actual iteración, se procede a copiar estos datos desde GPU RAM a RAM de sistema, utilizando la función `cudaMemcpy()` —la misma usada al copiar las posiciones iniciales—, solamente que en este caso se le entrega el parámetro especial `cudaMemcpyDeviceToHost`, para indicarle el sentido correcto de copiado que corresponde.

Esto se logra más cómodamente haciendo uso de aritmética de punteros, por cuanto para indicar la posición inicial hacia donde se escribe los resultados, solamente basta con entregar a la función `cudaMemcpy()` la referencia al arreglo de trayectorias al cual se le suma la cantidad de elementos que son saltados. Este paso se menciona simbólicamente entre las líneas 24 a 26 del algoritmo 3.

Se libera la memoria ocupada

Una vez realizados los cálculos, se libera los distintos tipos de memoria usados. Para el caso de las trayectorias, por ser arreglos lineales normales, su memoria se libera mediante el llamado a la función **`cudaFree()`**. Para el caso de las unidades de textura, se utiliza la función **`cudaUnbindTexture()`** y para el caso de los campos de viento y de altura, por ser arreglos 3D de CUDA, son liberados llamando a la función **`cudaFreeArray()`**.

3.9.9. Almacenamiento en disco de las trayectorias calculadas

Una vez terminados los cálculos, que ya están todos en memoria RAM de sistema, se guardan en un archivo en disco para poder ser utilizados posteriormente. Se escogió un formato de texto simple, dado que de esta manera es más fácil para todo tipo de programa externo ver de qué manera está organizada la información y poder utilizarla. Tal es el caso, por ejemplo, de graficar estos datos.

3.10. Graficado de resultados

Como un módulo independiente del programa principal, fue implementado un programa auxiliar que toma los resultados de aquél para mostrar en pantalla la manera en que las partículas se desplazan en el tiempo. Para ello se desarrolló primero un *script* para ser usado con **Gnuplot** [6, 40] que realizaba una animación del movimiento de unas cuantas partículas. Más tarde, se utilizó la librería Matplotlib de Python [29], con la cual se generó el gráfico presentado en la figura 3.4.

Para el momento de elaborar el póster para la conferencia "CoV6" (Apéndice C), se combinó los resultados del programa principal con una imagen satelital, haciendo uso de MATLAB (imagen 4.2, página 60). Todo esto fue realizado con la directa ayuda del Profesor Mark Falvey.

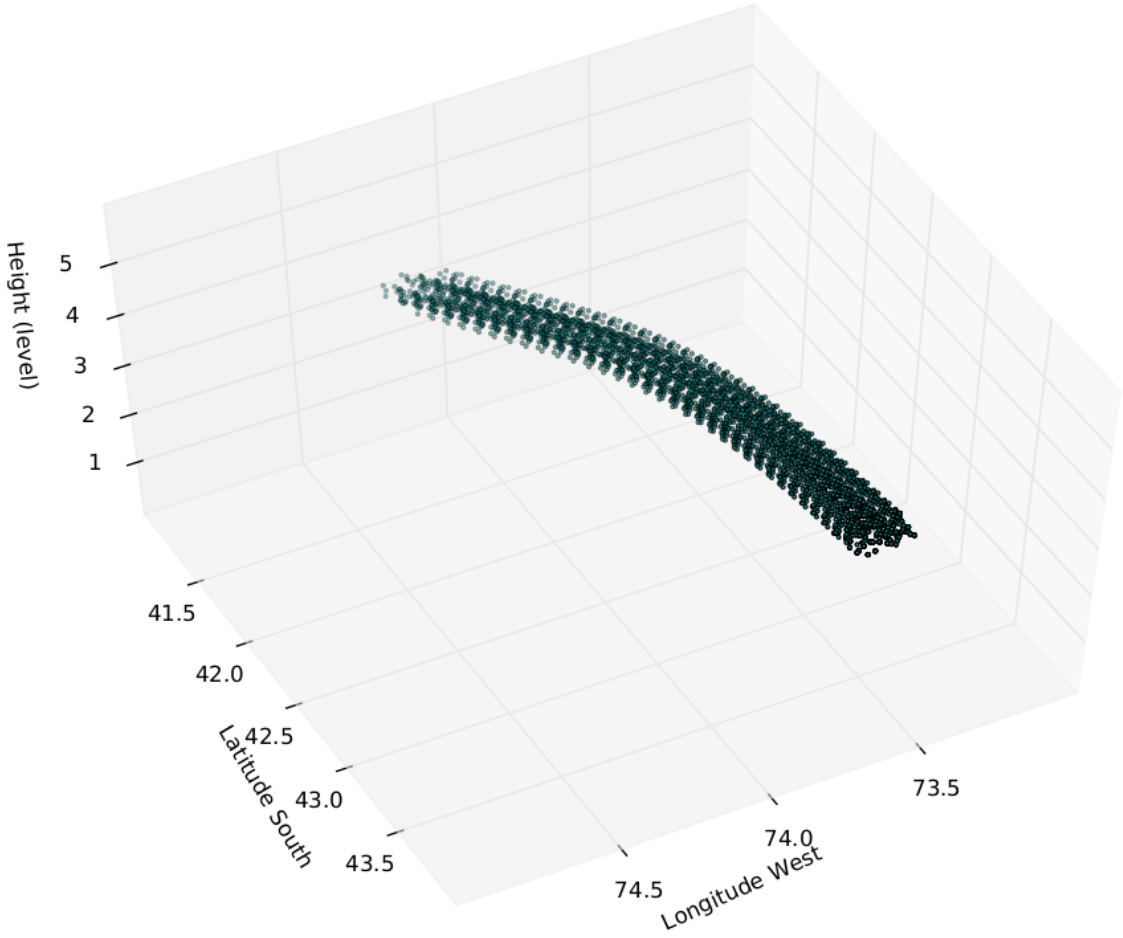


Figura 3.4: Ejemplo de la graficación de trayectorias usando Python

CAPÍTULO 4

RESULTADOS

Una vez realizadas todas las implementaciones descritas en el capítulo anterior, es muy importante ver en acción al programa, tanto en su versión CPU como GPU, para poder observar su desempeño y comparar estas versiones entre sí.

En este contexto, fueron dos los principales objetivos a comprobar: que los cálculos fuesen razonablemente correctos y que el tiempo ocupado en llevarlos a cabo fuese mucho menor con la versión GPU que con la CPU.

Todo esto se realizó en el marco de la preparación de un póster para ser expuesto en la Conferencia Internacional "Cities on Volcanoes 6^{th}" [24], desarrollada desde el 30 de mayo al 4 de junio de 2010 en la ciudad de Puerto de la Cruz, Tenerife, Islas Canarias, España. Los dos principales resultados expuestos en este póster fueron: la posibilidad concreta de usar GPU para calcular la trayectoria de una nube de ceniza volcánica que viaja libremente a través de la atmósfera; y que este programa realiza los cálculos en órdenes de magnitud más rápido que si solamente hiciera uso de CPU a la manera tradicional (más detalles de esta experiencia se presentan en el Apéndice C).

Se exponen a continuación los pasos llevados a cabo para obtener, validar y analizar los resultados, y como complemento de ello también se exponen gráficos que permiten analizar esta información de manera visual.

4.1. Obtención de los resultados

Para obtener resultados a partir de la ejecución del programa ya implementado, se eligió ponerlo a prueba analizando el comportamiento de la nube de ceniza volcánica emitida por el Volcán Chaitén durante el día 28 de mayo de 2008, que corresponde a uno de los días durante los cuales estuvo en erupción aquel año (ver sección 1.6.5, página 9). Para realizar esto de manera adecuada, se utilizó un pronóstico de la velocidad del viento para el día en cuestión (y los dos siguientes), lo que permite someter al programa al uso para el cual está diseñado.

Se definió en esta etapa que la obtención de estos resultados debían ser útiles tanto para poder validarlos como para poder comparar entre sí las versiones CPU y GPU de este sistema. Por ello el programa fue ejecutado varias veces, cada una con un distinto número de partículas iniciales (cifra representada por el parámetro **NUM_PARTICLES**, ver cuadro 3.1, página 34), manteniendo fija la cantidad de pasos de tiempo en que es dividido el tiempo total de simulación (parámetro **DELTA_T_TRAJ** del mismo cuadro).

Los pasos llevados a cabo fueron:

- Identificar la zona geográfica donde se encuentra la chimenea del Volcán Chaitén, y definir una zona sobre ella desde donde inicien su viaje las partículas a estudiar.
- Luego de ello, para cada partícula por simular, "sembrar" en esta zona sus posiciones iniciales definidas aleatoriamente.
- Ejecutar el programa para distintos escenarios, diferenciados entre sí por el número de partículas simuladas, las que iban desde 100 hasat 10.000.000. Estas ejecuciones se realizan tanto por la versión CPU como GPU.

Una vez realizados estos pasos, se cuenta tanto con las trayectorias calculadas como con los tiempos totales y de cálculo para cada uno de los escenarios ejecutados. Los valores de estos tiempos se pueden apreciar en el cuadro 4.1.

En la figura 4.1a se muestra un gráfico —con escala logarítmica en sus dos ejes— que muestra la relación entre los valores de tiempo total y de cálculo para las versiones CPU y GPU de este programa. También se elaboró un gráfico con escala normal en ambos ejes, y que solamente considera los tiempos totales y de cálculo para la versión GPU, el cual se presenta en la figura 4.1b.

	Tiempo de cálculo [s]		Tiempo total [s]	
Partículas	**CPU**	**GPU**	**CPU**	**GPU**
100	0,04	0,16	0,06	0,18
1.000	0,21	0,16	0,23	0,18
5.000	0,96	0,15	0,99	0,17
10.000	1,86	0,15	1,90	0,18
50.000	9,22	0,19	9,31	0,25
100.000	18,86	0,23	19,01	0,36
200.000	36,60	0,34	36,80	0,53
400.000	75,48	0,52	75,86	0,86
600.000	113,25	0,70	113,78	1,22
800.000	147,38	0,88	148,05	1,56
1.000.000	187,24	1,13	188,08	2,07
2.000.000	373,71	2,10	375,39	3,74
3.000.000	566,40	2,93	568,93	5,42
4.000.000	742,62	3,73	745,92	6,98
5.000.000	929,13	4,81	933,18	8,92
6.000.000	1129,72	5,61	1134,78	10,49
7.000.000	1293,45	6,35	1299,46	11,97
8.000.000	1487,40	6,66	1493,84	13,53
9.000.000	1668,01	7,26	1675,39	14,92
10.000.000	1847,48	8,73	1855,58	16,83

Cuadro 4.1: Distintos tiempos de ejecución del programa, en sus dos versiones, para diferentes cantidades de partículas modeladas

4.2. Validación de los resultados

Con estos datos, se pudo elaborar con la ayuda del profesor Mark Falvey la imagen presentada en la figura 4.2, la cual al ser comparada con la imagen satelital presentada en la figura 4.3 (captada por el satélite Aqua) permite apreciar que las trayectorias calculadas por este programa se comportan de una manera muy similar a lo que sucedió en la realidad aquel día 28 de mayo de 2008. En la imagen que presenta los resultados de este trabajo se puede apreciar que la nube de ceniza volcánica viaja fundamentalmente en dirección norte, con algunas dispersiones en dirección este y oeste, correspondiente a algunas partículas. En la imagen satelital se observa, sin embargo, que prácticamen-

te toda la nube de ceniza tiene una dirección predominantemente norte. Inspecciones más directas de los resultados demostraron que la gran mayoría de las partículas cuya trayectoria se desviaba tanto hacia el oeste como hacia el este correspondían a las que se ubicaban en alturas ubicadas en los extremos superior e inferior de esta columna de cenizas, lo que da la idea de que en la situación real probablemente en aquellas alturas no viajaban partículas provenientes del volcán.

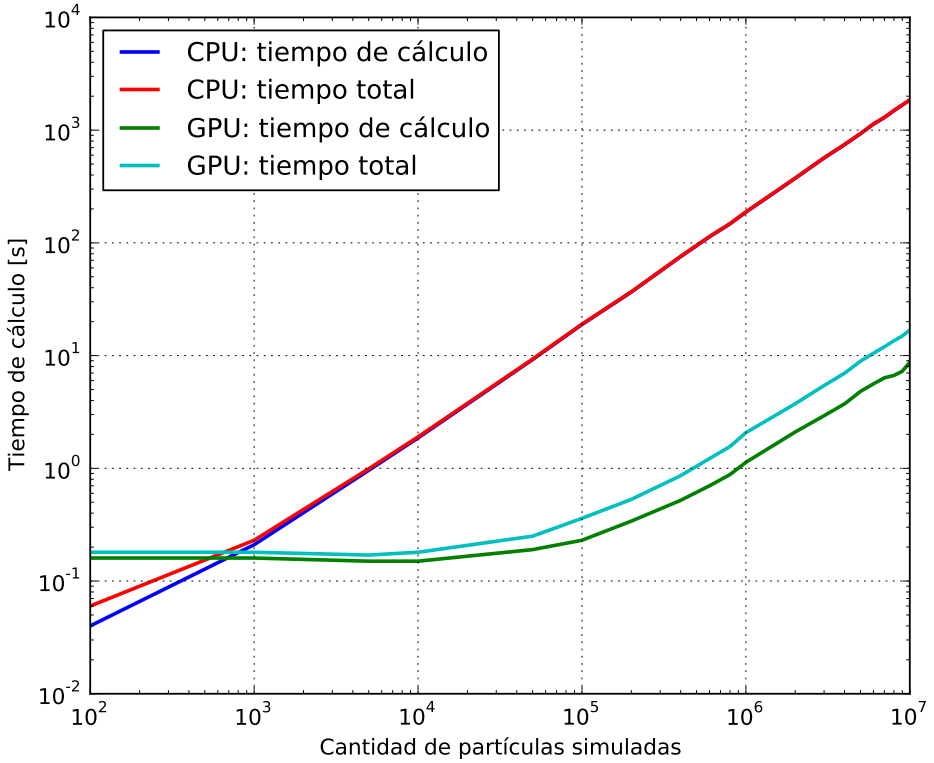
También se pudo comparar los resultados producidos por las dos versiones de este programa, para un mismo escenario (igual número de partículas). Esto es muy importante porque una diferencia entre estos datos puede indicar que un mal cálculo se está realizando —al menos— en una de estas versiones. Como era de esperarse, sí existe una leve discrepancia entre ambos resultados, debido a la diferente precisión numérica existente entre los cálculos de interpolación de la velocidad del viento producidos por la versión CPU —que realiza estos cálculos aritméticamente—, y la versión GPU —que los realiza gracias a las unidades de hardware denominadas "unidades de textura"—. Estas últimas componentes de hardware tienen una precisión limitada (consultar sección 2.4.3, página 18), lo que implica que los valores interpolados que entrega solamente son exactos para 256 valores intermedios entre cada dos valores consecutivos conocidos.

4.3. Análisis del desempeño de la versión GPU

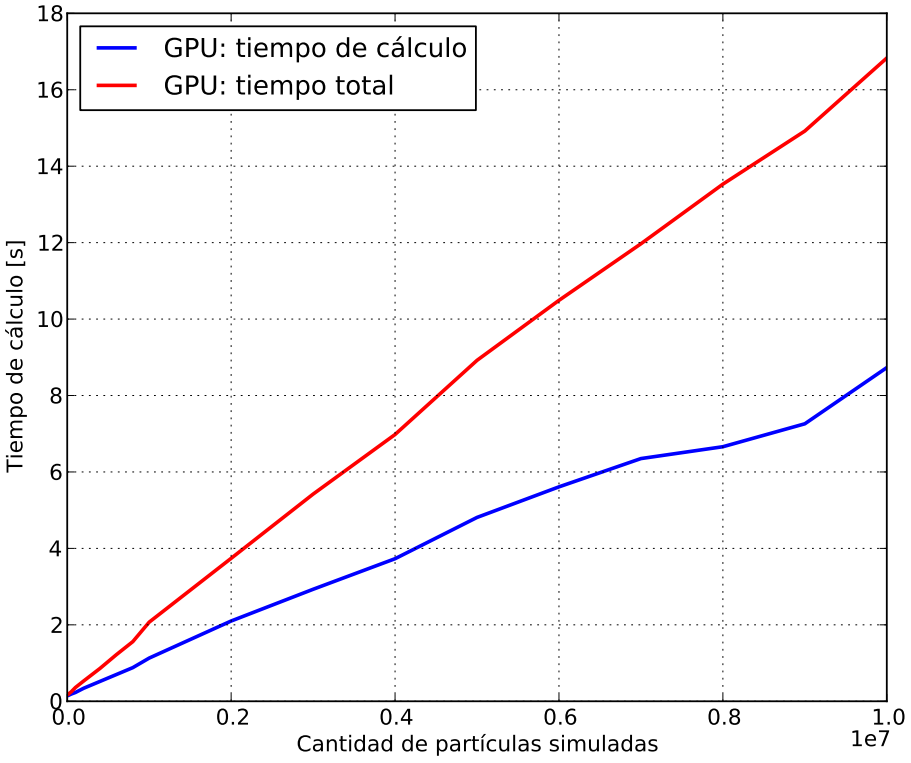
En el cuadro 4.1, se muestra tanto para CPU como para GPU el tiempo —total y sólo de cálculo— empleado por estas versiones del programa para calcular las trayectorias de las partículas modeladas, manteniendo fijos todos los parámetros salvo precisamente la cantidad total de estas partículas. El tiempo total en este cuadro incluye no solamente el cálculo, sino también la inicialización de variables, la lectura de los archivos de datos y su proceso para acomodarlos en las estructuras de datos correspondientes, por sólo mencionar algunos de todos los procesos llevados a cabo en cada ejecución.

El gráfico de la figura 4.1a, muestra tanto los tiempos totales que tomaron ambas versiones del programa, como también el tiempo que demoraron solamente en realizar los cálculos correspondientes. Dado que la diferencia entre los valores arrojados por ambas versiones eran en extremo diferentes, al punto que mientras los valores asociados a la CPU se veía como una recta inclinada hacia arriba y los de GPU se veían prácticamente siguiendo el eje horizontal, es que se los ordenó en un gráfico "log-log", el cual permite apreciar mejor estos datos para compararlos entre sí.

Analizando el gráfico de la figura 4.1a se puede llegar a las siguientes conclusiones:



(a) Gráfico "log-log" del tiempo de cálculo y tiempo total ocupado en procesar distinta cantidad de partículas, para las dos versiones del programa.



(b) Gráfico del tiempo de cálculo y tiempo total ocupado en procesar distinta cantidad de partículas solo para la versión GPU.

Figura 4.1: Gráficos con la comparación de los tiempos de cálculo y tiempo total ocupados por ambas versiones, para distinta cantidad de partículas simuladas.

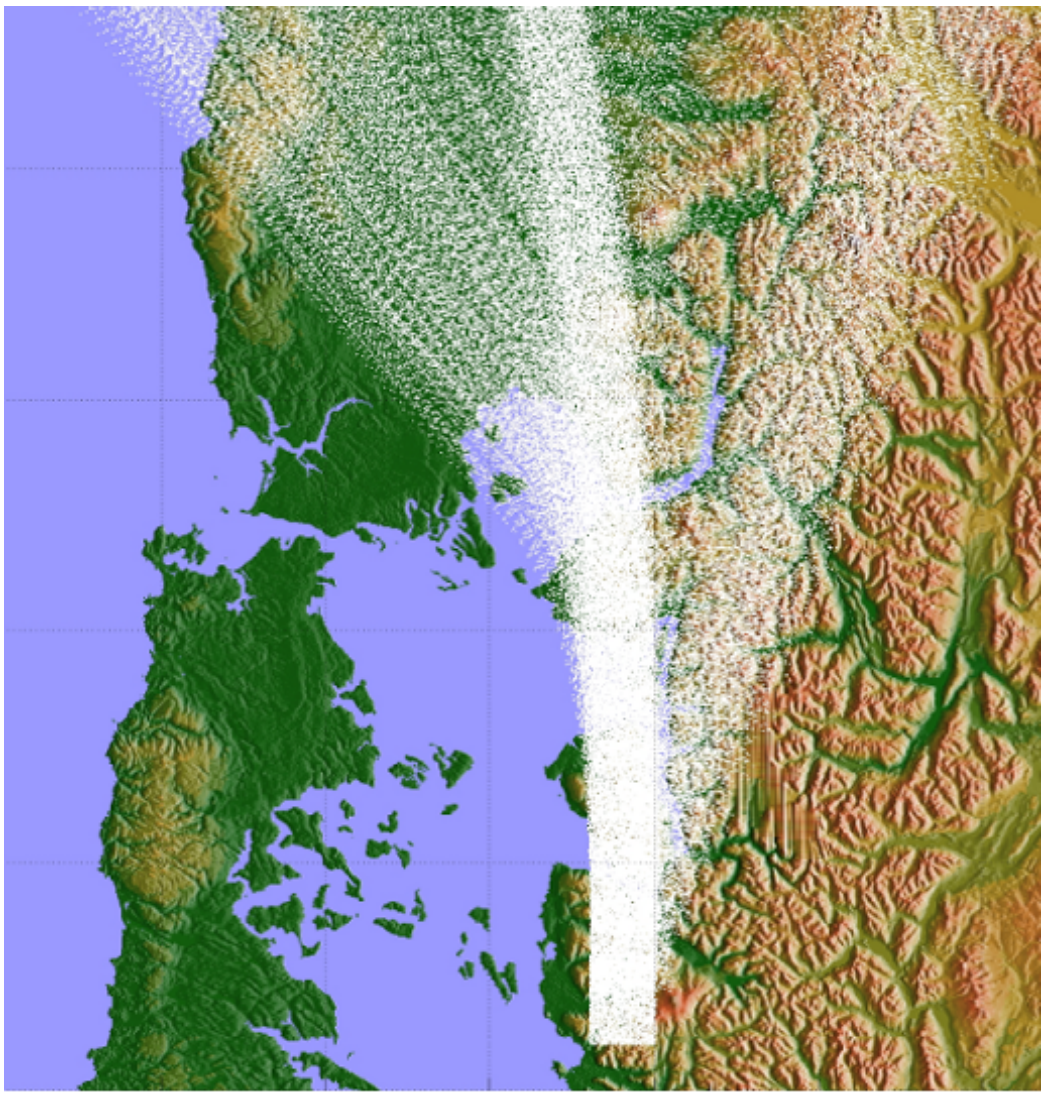


Figura 4.2: Representación de las trayectorias calculadas sobre un mapa de la zona

- El tiempo —tanto total como sólo de cálculos— tomado por la versión CPU es lineal con respecto a la cantidad de partículas.
- La diferencia es mínima entre los dos tiempos tomados (total y sólo de cálculo) para la versión CPU, lo que demuestra que en esta versión lo más intensivo es ciertamente los cálculos realizados para obtener las trayectorias de las partículas.
- El tiempo —tanto total como sólo de cálculos— tomado por la versión GPU también es lineal con respecto a la cantidad de partículas.
- En el caso de la versión GPU, a diferencia de su contraparte CPU, sí se advierte una diferencia entre los tiempos de cálculo y totales, lo que evidencia que en este caso las tareas complementarias a los cálculos, como son la lectura de archivos de datos, o su copiado en memoria, por ejemplo, tardan un tiempo que sí es comparable al que es ocupado en realizar solamente los cálculos.

Figura 4.3: Imagen satelital (Aqua MODIS) obtenida el mismo día de la simulación (28 de mayo de 2008)

- Se advierte que hasta poco menos de 1.000 partículas la versión CPU es la más rápida. Sin embargo, para todo el resto de escenarios calculados la versión GPU pasa a ser claramente la que toma menos tiempo para toda su ejecución.
- Es interesante que hasta los escenarios que comprenden 50.000 partículas la versión GPU toma aproximadamente el mismo tiempo total de ejecución y de cálculos. Esto se debe a que para esos casos el tiempo ocupado en procesar las partículas es insignificante comparado con el costo fijo de tener que alistar todos los procesos necesarios para hacer los cálculos en GPU.

De este análisis, y revisando la ecuación 3.21 (página 37), que expresa la cantidad total de memoria consumida por este programa, y analizando los algoritmos del capítulo 3, se ve que existe una relación lineal entre la cantidad total de partículas modeladas y el tiempo empleado en calcular sus trayectorias. Cabe destacar que otras pruebas realizadas con este programa, en las cuales se divide el tiempo de simulación en un paso de tiempo menor al utilizado en las pruebas recién comentadas, arrojó los mismos resultados.

El hecho que la CPU sea más rápida que la GPU para un número muy pequeño de partículas, no hace otra cosa que dar cuenta del hecho que esta última —independientemente de la cantidad de Hilos que deba procesar— tiene que llevar a cabo tareas fijas, entre las cuales la más exigente es sin duda la copia hacia y desde GPU RAM. Dada la gran rapidez de transferencia entre estas memorias, aproximadamente 2 GB/s [25], en el caso de tener que manejar muchos Hilos, este proceso de transferencia deja de ser un problema mayor, comparativamente hablando, dado que en ese caso la gran cantidad de procesos involucrados toman especial relevancia. En el caso de la CPU esta transferencia no existe, y sin embargo no tarda en quedar atrás de la GPU en el tiempo total de cálculo.

CAPÍTULO 5

CONCLUSIONES

Ha quedado en evidencia que el uso de GPU para realizar cálculos matemáticos en paralelo es realmente una alternativa válida cuando se busca reducir los tiempos de ejecución de estos cálculos sin comprometer su exactitud, y efectivamente —como era de esperarse debido a las altas expectativas creadas desde un inicio—, esto se cumplió con creces.

Es un gran logro el hecho que este trabajo haya conseguido demostrar que un mismo cálculo —como lo es el determinar la trayectoria de partículas en la atmósfera— es realizable perfectamente tanto en CPU como en GPU, y que esta última, para el caso particular de este trabajo, lo lleva a cabo en un tiempo considerablemente menor que la primera. Fue muy interesante constatar que el *kernel* programado para ser ejecutado en GPU constaba de muchas operaciones, muy complejas, pero que la GPU de todas maneras —cual procesador CPU— pudo calcular en cada hilo la trayectoria de una sola partícula, para todos los pasos de tiempo de la simulación, sin tener que hacer pausas intermedias como se temía en un principio. Asimismo es destacable que todos estos hilos se encargasen de las partículas paralelamente entre sí.

También se considera como un logro el hecho de haber abarcado un tema como la programación con tarjetas gráficas, lo que constituye un ámbito que todavía es relativamente novedoso, y en el que los ejemplos y experiencias externas no abundan. De hecho, la documentación oficial de NVIDIA (en particular [15]) es más bien una guía de las funciones de "C for CUDA", que una verdadera guía de programación con la cual se pueda, en forma completa, dominar este lenguaje. En efecto, fueron muchas las dudas solucionadas tras una larga búsqueda en el foro oficial de NVIDIA [32], o incluso en páginas individuales.

En lo que se refiere a las proyecciones a futuro para este proyecto, es claro que se abren muchos caminos por recorrer, en pos de seguir profundizando lo logrado en él. Sería muy interesante ver cómo se comporta la GPU al programar en ella ecuaciones de movimiento más complejas, que involucren por ejemplo el cálculo de valores aleatorios para modelar procesos como la turbulencia, o que se explore en procesos atmosféricos que tengan como directa consecuencia una pérdida en la absoluta independencia de las partículas unas con otras, como sucede por ejemplo con interacciones de orden químico y reactivo.

También será bueno abordar temas como una manera más exacta de calcular la transformación de las coordenadas verticales, o el manejo de otros formatos de datos científicos distintos de NetCDF o HDF5.

Mirando hacia atrás lo realizado, queda la agradable sensación que dos ámbitos del conocimiento humano como son las Ciencias de la Computación y las Ciencias de la Tierra, en un principio tan diferentes, pueden ser extremadamente complementarios: el primero ofreciendo herramientas para solucionar problemas geofísicos; el segundo proponiendo estos problemas como una manera de expandir el conocimiento generado por la Computación, al punto de desplazar los límites de lo que se creía posible alcanzar hasta el momento.

Y por último, no se puede dejar afuera la experiencia de haber participado en una conferencia internacional como "Cities on Volcanoes 6^{th}", que fue una excelente experiencia en lo académico, por el acercamiento a otros trabajos de memorias y tesis y por haber asistido a un ambiente donde la computación es, y seguirá siendo, una herramienta de gran utilidad.

BIBLIOGRAFÍA

[1] Roland R. Draxler and G. D. Hess. Description of the hysplit_4 modelling system for trajectories, dispersion and deposition. *Australian Meteorological Magazine, 47*, pages 295–308, 1998.

[2] Roland R. Draxler and G. D. Hess. *Description of the HYSPLIT_4 modelling system.* Air Resources Laboratory Silver Sprint, Maryland., 2004.

[3] Kayvon Fatahalian and Mike Houston. A closer look at gpus. *Communications of the ACM, Vol. 51, Number 10*, pages 50–57, October 2008.

[4] Robert G. Fleagle and Joost A. Businger. *An Introduction to Atmospheric Physics, 2nd Edition.* Academic Press, 2nd edition, 1980.

[5] George J. Haltiner and Roger T. Williams. *Numerical prediction and dynamic meteorology*. John Wiley & sons, 2nd edition, 1980.

[6] Philipp K. Janert. *Gnuplot in Action: Understanding Data with Graphs*. Manning Publications, 2009.

[7] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Prentice Hall, 2nd edition, 1988.

[8] John Michalakes and Manish Vachharajani. Gpu acceleration of numerical weather prediction. *Parallel Processing Letters Vol. 18 No. 4. World Scientific*, pages 531–548, December 2008.

[9] NVIDIA Corporation. *Technical Brief: NVIDIA GeForce® 8800 GPU Architectural Overview.* NVIDIA, November 2006.

[10] NVIDIA Corporation. *Accelerating MATLAB with CUDA™ using MEX files.* NVIDIA, September 2007.

[11] NVIDIA Corporation. *Technical Brief: NVIDIA GeForce® GTX 200 GPU Architectural Overview.* NVIDIA, May 2008.

[12] NVIDIA Corporation. *Getting Started. NVIDIA CUDA™ Installation and Verification on Linux*. NVIDIA, July 2009.

[13] NVIDIA Corporation. *NVIDIA CUDA™ Architecture. Introduction & Overview*. NVIDIA, April 2009.

[14] NVIDIA Corporation. *NVIDIA CUDA™ C Programming Best Practices Guide*. NVIDIA, July 2009.

[15] NVIDIA Corporation. *NVIDIA CUDA™ . Programming Guide, Version 2.3.1.* NVIDIA, August 2009.

[16] NVIDIA Corporation. *NVIDIA CUDA™ . C Programming Guide, Version 3.1.1*. NVIDIA, July 2010.

[17] Marc Schirski, Torsten Kuhlen, and Christian Bischof. Particles with a history: Visualizing flow fields with gpu-based streamlines. *ACM SIGGRAPH 2005 Posters*, page 127, 2005.

[18] Andreas Stohl, Sabine Eckhardt, Caroline Forster, Paul James, Nicole Spichtinger, and Petra Seibert. A replacement for simple back trajectory calculations in the interpretation of atmospheric trace substance measurements. *Atmospheric Environment 36*, pages 4635–4648, 2002.

[19] M. Xue, K.K. Droegemeier, V. Wong, A. Shapiro, and K. Brewster. *ARPS Version 4.0 User's Guide*. CAPS: Center for Analysis and Prediction of Storms, University of Oklahoma, September 1995.

FUENTES EN LÍNEA

[20] ARL: Air Resources Laboratory. Hysplit: Hybrid single particle lagrangian integrated trajectory model. http://ready.arl.noaa.gov/HYSPLIT.php; última visita: 20 de julio de 2010.

[21] ARL: Air Resources Laboratory. Volcanic ash forecast transport and dispersion. http://www.geo.mtu.edu/volcanoes/vc_web/tools/t_VAFTAD.html; última visita: 20 de Julio de 2010.

[22] Paul Bourke. Interpolation methods - trilinear interpolation. http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/interpolation/; última visita: 20 de julio de 2010.

[23] CAPS: Center for Analysis and Prediction of Storms, University of Oklahoma. Arps: Advanced regional prediction system. http://wwwcaps.ou.edu/index.html; última visita: 20 de julio de 2010.

[24] Comission of Cities and Volcanoes (CaV). Cities on volcanoes 6th, tenerife 2010. http://www.citiesonvolcanoes6.com; última visita: 20 de julio de 2010.

[25] Rob Farber. Supercomputing for the masses, part 1 to 15. http://www.drdobbs.com/architecture-and-design/207200659; última visita: 13 de julio de 2010.

[26] Geophysical Institute, University of Alaska Fairbanks. Puff-volcanic ash tracking model. http://puff.images.alaska.edu/index.shtml; última visita: 20 de julio de 2010.

[27] GPGPU. General-purpose computation on graphics hardware. http://gpgpu.org; última visita: 23 de julio de 2010.

[28] Joel Hruska. Nvidia's gtx 280: G80, evolved. http://arstechnica.com/hardware/reviews/2008/06/nvidia-geforce-gx2-review.ars; última visita: 20 de julio de 2010.

[29] John Hunter, Darren Dale, and Michael Droettboom. Matplotlib v1.0.0 documentation. http://matplotlib.sourceforge.net; última visita: 24 de mayo de 2010.

[30] NOAA: National Oceanic and Atmospheric Administration. Global forecast system. http://wwwt.emc.ncep.noaa.gov/?branch=GFS; última visita: 20 de agosto de 2010.

[31] Norwegian Institute for Air Research. Flexpart. http://transport.nilu.no/flexpart; última visita: 20 de julio de 2010.

[32] NVIDIA Corporation. Nvidia forums. http://forums.nvidia.com; última visita: 24 de julio de 2010.

[33] Alexey Stepin. Nvidia geforce gtx 200 graphics architecture review: Born to win? http://www.xbitlabs.com/articles/video/display/geforce-gtx200-theory.html; última visita: 20 de julio de 2010.

[34] The HDF Group. Hdf5 tools documentation. http://www.hdfgroup.org/HDF5/doc/index.html; última visita: 22 de agosto de 2010.

[35] The MathWorks, Inc. Matlab - the language of technical computing. http://www.mathworks.com/products/matlab/; última visita: 20 de julio de 2010.

[36] The MathWorks, Inc. Mex-files guide. http://www.mathworks.com/support/tech-notes/1600/1605.html; última visita: 20 de julio de 2010.

[37] Gabriel Torres. Geforce gtx 200 series architecture. http://www.hardwaresecrets.com/article/569; última visita: 20 de julio de 2010.

[38] UCAR: University Corporation for Atmospheric Research. Comet program. http://www.comet.ucar.edu/index.htm; última visita: 20 de julio de 2010.

[39] UCAR: University Corporation for Atmospheric Research. Network common data format. http://www.unidata.ucar.edu/software/netcdf/; última visita: 20 de julio de 2010.

[40] Thomas Williams and Colin Kelley. Gnuplot. http://www.gnuplot.info; última visita: 20 de julio de 2010.

[41] WRF. Weather research & forecasting model. http://www.wrf-model.org; última visita: 20 de julio de 2010.

APÉNDICE A

CARACTERÍSTICAS DEL ENTORNO DE DESARROLLO

Se lista a continuación las características del computador de pruebas usado para implementar, probar y ejecutar el programa —en dos versiones: CPU y GPU— fruto del trabajo de esta memoria.

A.1. Hardware

El computador usado corresponde a un DELL modelo XPS Dimension 730x, fabricado en el mes de septiembre de 2009. La figura A.1 muestra cómo se ve externamente este equipo. Dentro de esta máquina están las dos tarjetas GeForce GTX 285 usadas en este trabajo de memoria, cuya presentación externa se puede apreciar en la figura A.2.

Las componentes de *hardware* de este computador están listadas en el cuadro A.1.

Características de *hardware*	
Marca:	Dell
Modelo:	XPS
Placa base:	DELL Inc. 0P270J, versión A00
CPU:	Intel® Core™ i7, 2.93 GHz; 8 MB Cache
RAM:	6 GB (3 x 2GB)
GPU 1:	Tarjeta NVIDIA GeForce GTX 285; 1 GB RAM; 300 MHz
GPU 2:	Tarjeta NVIDIA GeForce GTX 285; 1 GB RAM; 300 MHz
Disco duro:	Samsung HD103UJ, 1 TB

Cuadro A.1: Componentes físicos del computador de pruebas

Figura A.1: Computador DELL XPS

Figura A.2: Tarjeta GeForce GTX 285

A.1.1. Propiedades específicas de la tarjeta GeForce GTX 285

Para efectos de las pruebas realizadas en este trabajo de memoria, se utilizó dos tarjetas NVIDIA GeForce GTX 285, unidas entre sí mediante la tecnología SLI[1]. Esta tarjeta fue lanzada al mercado el 15 de enero de 2009 y corresponde a la penúltima integrante de la serie GTX 200.

[1]Tecnología propia de NVIDIA™ , que permite usar en un mismo computador dos o tres tarjetas de video, conectadas tanto entre sí como directamente al computador a través del puerto PCI Express.

Componente	Característica
Thread processing cluster	10 TPC en total
Stream multiprocessor	30 (3 por cada TPC)
Stream processor	240 (8 por cada MP)
Global memory	1GB
Constant memory	64kB por cada MP
Shared memory	16kB por cada MP
Registers	16384 registros de 32 *bits* por cada MP
Warp size	32 *warps* (*threads* paralelos) por cada MP

Cuadro A.2: Propiedades específicas de la tarjeta GTX 285

En el cuadro A.2 se presenta un resumen de sus principales características.

A.2. Software

El sistema operativo usado por este computador, junto con los controladores (*drivers*) de la tarjeta gráfica, la versión de CUDA, el compilador de C, versión de MATLAB y versiones de las biblioteca para manipular archivos NetCDF y HDF5, están listados en el cuadro A.3.

Características de *software*	
Sistema operativo:	Debian GNU/Linux 5.0.3 "Lenny"
Driver NVIDIA:	195.36.24
CUDA Toolkit:	3.0
Compilador C:	GCC 4.3.2
MATLAB:	R2009a
NetCDF library:	4.0
HDF5 library:	1.8.5

Cuadro A.3: Versiones de los sistemas usados en el entorno de desarrollo

APÉNDICE B

REVISIÓN BIBLIOGRÁFICA

Se presenta a continuación una breve revisión bibliográfica destacando las principales fuentes de información usadas en este trabajo, las cuales pueden ser consultadas para una más profunda investigación en este tema.

Los contenidos están separados por el ámbito al cual pertenecen, para una mejor claridad y más fácil búsqueda.

B.1. Procesos físicos relacionados con la meteorología

De las muchas aplicaciones prácticas que tiene el hacer uso de las GPU para cálculos en paralelo, la aplicación práctica a abordar en este trabajo fue el cálculo de las trayectorias de partículas en la atmósfera. El libro "An Introduction to Atmospheric Physics, 2nd Edition", de Robert Fleagle y Joost Businger [4], fue la referencia central para este tema.

B.2. Lenguajes de programación

Para sacar provecho a las tarjetas de video NVIDIA mediante el uso de la arquitectura CUDA, los lenguajes de programación C y C++ fueron las opciones más directas. Para el caso del lenguaje C, se hizo uso —como es de esperarse— del clásico libro "The C Programming Language" de Brian Kernighan y Dennis Ritchie [7].

También se consultó la referencia de "C for CUDA" [15], la cual es la principal fuente oficial de información para aprender este lenguaje. Fue de valiosa ayuda también el foro oficial de NVIDIA [32].

Para varias tareas se utilizó el software matemático MATLAB [35] que provee su propio lenguaje de programación llamado "lenguaje M", sirviendo de referencia para él la extensa documentación disponible en su web. Y para hacer uso dentro de MATLAB de librerías escritas en el lenguaje C, es que se revisó la documentación de los "Archivos MEX" [36] que sirven de interfaz entre estos dos lenguajes.

B.3. Arquitectura física de las tarjetas de video

Un punto fundamental para poder extraer el máximo provecho de la programación para tarjetas gráficas, fue el conocer en detalle la arquitectura con que han sido diseñadas y construidas. Esto se hace aún más importante con las tarjetas de última generación con las que se trabajó en este proyecto, dado que sus variadas características solamente son explotables al máximo cuando son utilizadas adecuadamente. El caso particular de los distintos tipos de memoria y de *caches* de memoria ofrecidos por estas tarjetas, cada uno para una necesidad específica, es un buen ejemplo.

La información en este tópico, dada su todavía reciente publicación, solamente está cubierta por algunos libros y artículos. Los que fueron revisados hasta el momento son:

- **"Technical Brief: NVIDIA GeForce® 8800 GPU Architectural Overview"** [9]: Este manual explica la arquitectura de la familia de tarjetas 8800, que son las anteriores a aquellas que serán usadas en este proyecto.
- **"Technical Brief: NVIDIA GeForce® GTX 200 GPU Architectural Overview"** [11]: Este manual cubre la actual familia de tarjetas de video: GTX 200. Con con estas tarjetas fueron llevadas a cabo las pruebas realizadas en esta memoria. En este texto se describen los objetivos y características clave de estas tarjetas, y también se explica la implementación técnica del hardware que las compone.

También se revisó material de páginas web, fundamentalmente sitios especializados en hardware que elaboran una detallada descripción de las tarjetas de video apenas salen éstas al mercado. Entre los principales artículos consultados están "GeForce GTX 200 Series Architecture" [37], "Nvidia GeForce GTX 200 Graphics Architecture Review: Born to Win?" [33], y "NVIDIA's GTX 280: G80, Evolved" [28].

B.4. Especificación de la arquitectura CUDA de NVIDIA

Para este tema se consultó fundamentalmente información disponible en internet, como también la información que provee NVIDIA en su propia página web. En el caso de la documentación provista por NVIDIA, se revisó detalladamente los siguientes documentos:

- **"Getting Started. NVIDIA CUDA™ Installation and Verification on Linux"** [12]: Manual que explica los pasos para instalar adecuadamente CUDA en una máquina con Linux. Expone tanto la instalación del controlador para la tarjeta de video, como la instalación de las bibliotecas y ejemplos, y su uso para verificar el correcto funcionamiento de CUDA en el sistema.
- **"NVIDIA CUDA™ Architecture. Introduction & Overview"** [13]: Una breve introducción a lo que es CUDA, presentando su arquitectura lógica, su entorno de desarrollo, y los lenguajes de programación que se pueden usar.
- **"NVIDIA CUDA™ Programming Guide, Version 2.3.1."** [15]: Ésta es la guía de programación, que explica los conceptos necesarios para comenzar a programar en CUDA. Para ello describe las extensiones al lenguaje C (llamado "C for CUDA") y sus usos tanto para la definición de las funciones a ser ejecutadas en paralelo (llamadas "kernels") como para la administración y asignación de la memoria exclusiva con que cuenta la GPU.
- **"NVIDIA CUDA™ C Programming Best Practices Guide"** [14]: Un libro enfocado para quienes necesiten obtener el mejor desempeño posible de sus aplicaciones CUDA, mediante la exposición de técnicas de optimización y atajos de programación.

APÉNDICE C

PRESENTACIÓN DEL PROYECTO EN LA CONFERENCIA "CoV6th"

Las conferencias internacionales "Cities on Volcanoes" son una serie de eventos en los cuales científicos, estudiantes y público en general pueden compartir experiencias y conocimientos en el ámbito de la vulcanología, sobre todo cuando ésta se aplica a la coexistencia de poblaciones humanas y volcanes. La sexta edición de esta conferencia se desarrolló desde el 30 de mayo al 4 de junio de 2010 en la ciudad española de Puerto de la Cruz, Tenerife, Islas Canarias.

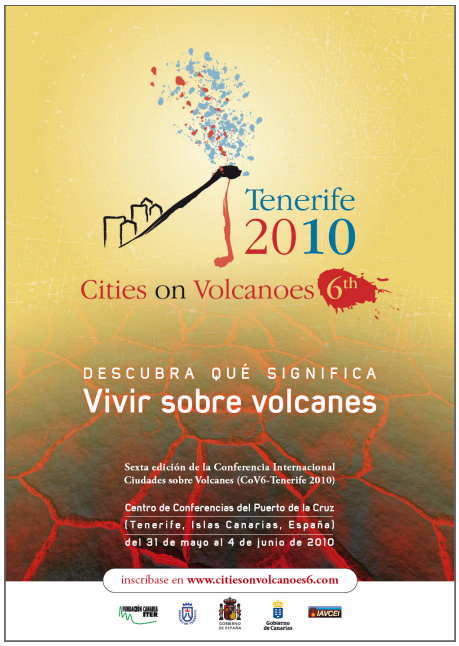


Figura C.1: Afiche oficial de la conferencia "Cities on Volcanoes 6th" [24].

Debido a las ya conocidas aplicaciones vulcanológicas del presente trabajo, se decidió presentar —a principios del mes de marzo de 2010— un resumen de éste al Comité de la Conferencia, el cual confirmó la propuesta, y definió que el formato de la presentación sería el de un póster.

Dada esta oportunidad, se planteó como objetivos fundamentales los siguientes puntos:

- Ver cuán masificado está el uso de GPU en aplicaciones vulcanológicas en la actualidad.

- Ver el interés que una aplicación como la del presente trabajo despierta en una concurrencia que fundamentalmente se desenvuelve en el contexto de la geología y la geofísica.
- Establecer eventuales contactos con profesores o alumnos que trabajen en temas similares.

Durante la conferencia, fue posible conocer otros proyectos relativos al cálculo de trayectorias de partículas, como también se pudo asistir a una decena de charlas —previamente seleccionadas de entre todas las ofrecidas— que presentaban temas relacionados a la colaboración entre las ciencias de la computación y la vulcanología. También se conversó con varios de los asistentes, tanto de sus proyectos como de éste.

Si bien hay que considerar que no es posible sacar conclusiones categóricas, tomando como única muestra a los asistentes a esta conferencia, se pudo observar que en ninguno de los proyectos presentados —tanto en póster como en charlas— se consideraba el uso de GPU para el cálculo de trayectorias. Tampoco fueron presentados trabajos que implementasen algoritmos para hacer cálculos en paralelo a nivel de CPU. Y la existencia del uso de GPU para acelerar los cálculos de la dispersión en la atmósfera de contaminantes de origen volcánico no era valorada en su justa medida, por cuanto todavía era visto como un tema lejano e incluso totalmente desconocido.

Esta experiencia se puede tomar como una oportunidad de apreciar que la aplicación de GPU para este tipo de cálculos todavía es una actividad en ciernes. También se pudo ver que los métodos usados todavía corresponden a la aplicación "clásica" del poder computacional en este contexto.

En la figura C.2 se muestra el póster que fue finalmente presentado en esta conferencia.

Trajectory Calculation of Volcanic Ash Dispersion using Graphics Processing Units (GPU)

Nicolás Ozimica
nozimica@dcc.uchile.cl

Rainer Schmitz
schmitz@dgf.uchile.cl

Mark Falvey
falvey@dgf.uchile.cl

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

Introduction

The contaminants (ash and gases) released into the atmosphere by a volcanic eruption pose a major hazard to many human activities, including those related to aviation. In order to mitigate these risks it is of great importance to be able to forecast ash dispersion as reliably and as quickly as possible.

Despite the fact there are curently many methods and models available (mainly using the Lagrangian approach) calculations are almost exclusively done on conventional computer architecture, that is, on the Central Processing Unit (CPU). In this work we present a relatively new approach that makes use of parallel power of the Graphical Processing Units (GPU) of the last generation Video Cards. GPU's are typically used to perform advanced 3D graphics rendering operations. However, modern graphics cards are capable of performing more general calculations and, depending on the type of numerical problem GPUs, may enable very fast parallel computing at an affordable price.

Trajectory calculations are able to take advantage of the aforementioned parallel GPU capacity as the trajectories of individual particles can be modelled independently. For example, if a certain number of particles are emitted to the atmosphere at a given time, each of their trajectories can be solved in parallel calculations.

In this work we present preliminary results which demonstrate a significant increase in calculation speed when computing large numbers of trajectories in GPU as compared to similar calculations in conventional (CPU) architecture.

Trajectories and interpolation

We have implemented a basic trajectory calculation algorithm which works both in GPU and conventional CPU architectures. Trajectories are computed from 3D wind fields provided by numerical weather prediction models (NWP).

The trajectory of a particle can be calculated, for each spatial coordinate, with the formula below:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i \mathbf{\Delta t} + \bar{\mathbf{x}}$$

where $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are the trajectory coordinates at times $\mathbf{i}$ and $\mathbf{i+1}$, $\mathbf{v}_i$ is the wind velocity component and $\mathbf{\Delta t}$ is the time step ($\mathbf{t}_{i-1} - \mathbf{t}_i$). $\bar{\mathbf{x}}$ describes the (optional) effect of stochastic turbulence.

The trajectory calculator is able to handle terrain following vertical coordinate systems (such as σ coordinates). In this case, the vertical wind velocity is transformed to orthogonal system prior to trajectory integration using the formula below:

$$\mathbf{w}_f = \mathbf{w} - \left(\mathbf{u}\frac{\partial \mathbf{h}}{\partial \mathbf{x}} + \mathbf{v}\frac{\partial \mathbf{h}}{\partial \mathbf{y}}\right)$$

where $\mathbf{h}$ is the altitude of the untransformed grid cooordinates on a given vertical level.

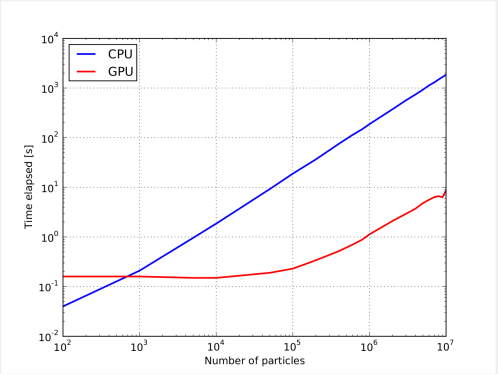
Both of the above calculations involve the spatial interpolation of the wind and height data (which are only known for discrete points). This is ideal for the GPU, since it incorporates a specialized hardware unit designed to perform interpolations on arrays of image data (called *textures* in GPU terminology). This is very important for many basic graphical operations such as zooming on an image. However, our GPU implentation makes optimal use of the texture processor to perform interpolations within the 3D wind and height fields.

Results

Here we compare the calculation times for the GPU and CPU implementations. The results are based on trajectory calculations for the Chaitén Volcano, using NWP data from the Global Forecast system model (see panel to the right for more information). Because algorithm effeciency depends mainly on the number of trajectories that are calculated we present results for groups of trajectories consisting of 100 to 10 million particles. Each trajectory is computed over 160 integration steps.

The times elapsed by the CPU and GPU versions, are shown in the plot below. The results show that for groups of more than 1000 particles, the GPU gives an improved performance compared to a conventional CPU. Indeed, for the experiment involving 10 million particles the GPU version is about **211 times faster** than its CPU counterpart. In this case, the CPU takes almost 20 minutes to complete the calculation, whereas using the GPU, the same result is available in less than 10 seconds.

Note that for small groups of particles the GPU actually runs slower than the CPU. This is due to overhead associated with the transfer of data between conventional RAM and the internal GPU memory.

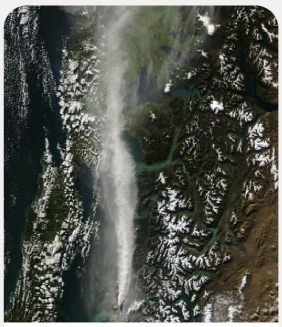


What is a GPU?

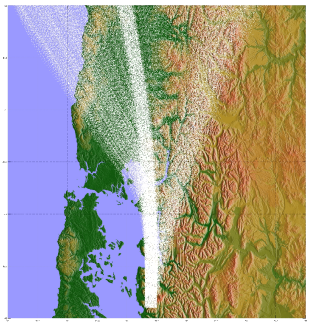
A Graphics Processing Unit (GPU) is a specialized processor that forms part of a computers graphics card, and is able to perform floating point operations. Over recent years, the processing power of high-end GPU's increased extrodinarily, mainly as a result of the demands of the video gaming and digital animation industries (among others). For example, the GTX285 NVIDIA GPU consists of 240 processing cores and has at least more than 1 GB of RAM memory. In the past, GPU applications have been restricted to very specific graphics operations. However, with the advent of NVIDIA CUDA programming interface, it has become posible to exploit the capabilities of the GPU for scientific computing aplications.

Example for Chaitén Volcano

The Chaitén Volcano, located in the south of Chile, is a good example for the application of this work. The two images below show the dispersion of the volcanic plume: the first one is a MODIS satellite image, the second is the forecast based on trajectories calculated using the GPU. The trajectories are seeded between heights of 3000 and 6000 meters directly over the Chaitén Volcano, and are advanced in time (12 hours) using the winds from the GFS forecast model.



Aqua 2008/149 05/28/08 18:40 UTC
Image courtesy of MODIS Rapid Response Project at NASA/GSFC



Similar to the satelite data, the ash particles are predicted to travel north, from the top of the volcano and beyond the extent of the satellite photo, where they move slightly westwards.

Conclusions and future work

The results of this study, while preliminary, are very promising and indicate that the use of high-end GPU's may offer enormous performance advantages for trajectory calculations. Apart from allowing the faster calculation of larger groups of trajectories, the use of GPU's is also more cost effetive because a single GPU is much cheaper than the hundreds of CPU that are needed to achieve the same performance.

The next steps in our work are to further optimize the code and algorithms, and also to incorporate other physical phenomena (such as deposition and turbulence, among others).

Finally, a web page with information about this project, along with the complete source code of this implementation is scheduled for the next few months.

Figura C.2: Póster presentado en la conferencia internacional "Cities on Volcanoes 6th".