UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aceleración de algoritmos para el cálculo de distribuciones de probabilidad alfa-estables mediante programación paralela

Juan Miguel Aranda Cobos
Tutor: Luis de Pedro Sánchez
Ponente: Jorge E. López de Vergara Méndez

Julio 2016

Aceleración de algoritmos para el cálculo de distribuciones de probabilidad alfa-estables mediante programación paralela

AUTOR: Juan Miguel Aranda Cobos
TUTOR: Luis de Pedro Sánchez
PONENTE: Jorge E. López de Vergara Méndez

High Performance Computing and Networking
Dpto. de Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Julio 2016

Resumen

Las distribuciones de probabilidad alfa-estables son una familia especial de distribuciones de probabilidad cuyo estudio puede resultar en sistemas predictivos avanzados de diversos tipos, como análisis de tráfico de red, procesamiento de imágenes médicas o cotizaciones bursátiles. Surgen a partir del Teorema del Límite Central Generalizado, que establece que aquella suma de variables aleatorias cuya varianza no es finita converge a distribuciones alfa-estables. Desafortunadamente, la mayoría de herramientas existentes en la actualidad no son capaces de realizar los cálculos necesarios para el procesamiento de las distribuciones con la suficiente velocidad para poder llegar a obtener herramientas verdaderamente útiles que trabajen a tiempo real con un sistema predictivo. Esto se debe principalmente a la gran complejidad computacional que supone realizar las operaciones de integración numérica necesarias para obtener una distribución alfa-estable con un nivel de precisión aceptable para su uso en un sistema predictivo. En el presente Trabajo Fin de Grado se estudia la posibilidad de conseguir acelerar un algoritmo para el cálculo de distribuciones alfa-estables mediante el uso de computación paralela, más concretamente en GPUs con el uso de la tecnología CUDA. Para ello se propone la utilización de un algoritmo de integración numérica adaptativo, diseñado para su uso en entornos paralelos CUDA como método primario de integración para el algoritmo encargado de calcular las distribuciones de probabilidad alfa-estables. De esta forma se consigue incrementar el rendimiento en la obtención de estas distribuciones, pudiendo lograr una aceleración relativa de hasta varias veces el rendimiento de la versión serie del programa, manteniendo la precisión necesaria para lograr que los resultados permanezcan igual de correctos.

Palabras clave

CUDA, GPU, integración numérica, distribución de probabilidad, alfa-estable, rendimiento, aceleración, computación paralela, paralelismo.

Abstract

Alpha-stable distributions are a special subset of probability distributions whose research can result in advanced predictive systems of diverse types, such as network traffic analysis, medic images processing or stock-market prices. They emerge because of the Central Limit Theorem, which establishes that the addition of random variables with infinite variance converges into an alpha-stable distribution. Unfortunately, most currently used tools are not capable of calculating fast enough the processing of these distributions with the necessary speed in order to obtain actually useful tools capable to work within real time in a predictive system. This is mainly due to the great computational complexity required to solve all of the numeric integration operations needed in order to calculate an alpha-stable distribution with an acceptable level of precision for its usage in a predictive system. The following End-of-Degree Project studies the possibility of accelerating an algorithm to calculate alpha-stable distributions by means of parallel computing, more specifically in GPUs equipped with CUDA technology. In order to do so, we propose the usage of a multi-dimensional adaptive numeric integration algorithm designed for CUDA environments as the primary integration method for the algorithm that calculates alpha-stables. By doing so, we manage to increase the performance in the calculation of these distributions, being capable of reaching an acceleration of multiple times the performance of the serial version while maintaining the necessary precision in order for the results to remain equally correct.

Key words

CUDA, GPU, numeric integration, probability distribution, alpha-stable, performance, acceleration, parallel computation, parallelism.

Agradecimientos

En primer lugar, a ti, por haber decidido tomarte un tiempo para leer este documento.

En segundo lugar, a Luis de Pedro y Jorge López de Vergara por su apoyo, ya que sin el cual habría resultado bastante más complicada la realización del presente trabajo fin de grado.

Por último, a todos los que me conocen, por soportarme.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Glosario

PDF	Probability Density Function (función de densidad de probabilidad).
CDF	Cumulative Distribution Function (función de distribución acumulada).
CPU	Central Processing Unit (unidad central de procesamiento).
GPU	Graphics Processing Unit (unidad de procesamiento gráfico).
API	Application Programming Interface (interfaz de programación de aplicaciones)
CUDA	Compute Unified Device Architecture (arquitectura unificada de dispositivos de cómputo).
OpenCL	Open Computing Language (lenguaje de computación abierto).
GSL	GNU Scientific Library (biblioteca científica de GNU): es la librería que permite realizar cálculos de integración numérica mediante diversos métodos.
Libstable	Algoritmo de cálculo de distribuciones alfa-estables para CPUs.
Cuhre	Algoritmo de integración numérica multidimensional para GPUs CUDA.

1 Introducción

1.1 Motivación

La computación paralela es un tema que desde hace unos años está experimentando un auge debido a las grandes posibilidades que ofrece a la hora de abordar problemas que resultan ser computacionalmente muy costosos.

Los sistemas predictivos basados en inferencia estadística tratan un concepto relativamente novedoso y muy interesante por el cual se puede generar un modelo predictivo a partir de una distribución de probabilidad matemática y usarlo para, como su nombre indica, predecir todo tipo de comportamientos en sistemas informáticos. Sin embargo, puesto que el principal requerimiento de estos sistemas suele ser la velocidad (en la mayoría de los casos lo ideal es disponer de procesamiento en tiempo real), no suele ser tarea fácil la implementación de uno de estos sistemas de forma que verdaderamente consiga producir predicciones correctas con el margen de tiempo necesario para poder reaccionar a dichas predicciones con antelación.

Asimismo, resolver los desafíos que conlleva paralelizar en GPUs un algoritmo matemático complejo como es el de un método de integración numérica resulta apasionante debido a la gran cantidad de conceptos que se entremezclan y son necesarios ordenar.

Uniendo estas dos ideas nace este proyecto: mediante la aceleración conseguida gracias a la computación paralela en GPUs, se puede conseguir un rendimiento lo suficientemente mayor como para que se puedan usar varios de estos sistemas predictivos que en caso contrario no podrían llegarse a usar debido a la falta de velocidad en los cálculos. Así es como, gracias a la paralelización en CUDA de un método de integración numérica, se prevé conseguir una aceleración visible en un algoritmo para el cálculo de distribuciones de probabilidad alfa-estables, que a su vez posibilitará el uso de sistemas predictivos en tiempo real.

1.2 Objetivos

La idea que promovió el desarrollo de este proyecto consistía en una aplicación para la detección de anomalías en el tráfico agregado de redes IP basada en inferencia estadística sobre un modelo alfa-estable de primer orden. Esta idea fue la que originó la creación de la librería *Libstable* para el cálculo eficiente de distribuciones de probabilidad alfa-estables en tiempo real. Sin embargo, no se consiguió alcanzar la suficiente velocidad en dichos cálculos como para poder usarse de manera totalmente satisfactoria en la aplicación de detección de anomalías en redes. Esto fue lo que, a su vez, suscitó la actual idea, que consiste en conseguir un mayor rendimiento en la librería *Libstable* mediante el uso de computación paralela en GPUs. Por tanto, el objetivo primario de este proyecto consiste en acelerar el algoritmo de cálculo de distribuciones de probabilidad alfa-estables usado en la librería *Libstable*. Este objetivo se puede subdividir en los siguientes:

- Conseguir un algoritmo de integración numérica cuyo procesamiento se realice de forma paralela en GPUs usando CUDA que sea matemáticamente equivalente al algoritmo usado por la versión serie de la librería *Libstable*.
- Integrar dicho algoritmo dentro del código de la librería *Libstable*, sustituyendo las llamadas a las funciones primarias de integración numérica por llamadas a la función del algoritmo paralelo en los casos en los que dicha sustitución tenga sentido.

- Probar el correcto funcionamiento equivalente entre ambas versiones de *Libstable*, comprobar que el nivel de precisión es el adecuado y estudiar el nivel de aceleración que se consigue mediante el uso de la versión paralela del algoritmo.

1.3 Fases de realización

Este proyecto se ha realizado siguiendo una serie de fases estructuradas para facilitar la elaboración del mismo y minimizar el trabajo erróneo, las cuales se detallan a continuación:

- **Investigación**: lo primero e indispensable que fue necesario hacer consistió en llevar a cabo un estudio del estado del arte de todas las tecnologías involucradas en la realización del proyecto, haciendo especial hincapié en los métodos de integración numérica ya existentes para intentar encontrar alguno ya desarrollado para sistemas paralelos en GPUs, así como en toda la teoría matemática que hay detrás, tanto del cálculo de distribuciones de probabilidad alfa-estables, como de algoritmos para integración numérica (y las distintas estrategias existentes para abordarlos), para poder llegar a entender el funcionamiento de los algoritmos matemáticos pertinentes.
- **Análisis**: una vez investigado todo lo necesario, el siguiente paso trató de analizar con precisión los requerimientos del proyecto para poder tener una idea más concreta de todo lo que debía realizar para su completitud y organizar los pasos necesarios.
- **Implementación**: tras ello vino la programación de todo el código necesario para la integración de los algoritmos obtenidos y lograr una correcta interacción entre ellos para así conseguir un funcionamiento equivalente a la versión sin paralelizar.
- **Pruebas**: lo primero que fue necesario probar fue la correcta equivalencia en cuanto a resultados obtenidos entre las versiones en serie y en paralelo para cerciorar que se implementó correctamente el algoritmo paralelo. Posteriormente se llevaron a cabo unos tests de precisión para asegurar que la integración numérica en paralelo era igual de capaz que la versión serie, y por último, unos tests de rendimiento para comprobar el nivel de aceleración que se obtuvo tras paralelizar el algoritmo.

1.4 Organización de la memoria

En el presente capítulo se ha realizado una introducción para divulgar la idea general de este proyecto antes de entrar en más detalle en los siguientes capítulos.

En el capítulo que viene a continuación se detallará el estudio del estado del arte llevado a cabo, explicando por separado todo lo necesario acerca de los conceptos teóricos.

En el capítulo 3 se enumerarán los requisitos del proyecto necesarios para conseguir la completitud del mismo, así como un análisis de las aplicaciones con las que ya contamos.

En el capítulo 4 se explicará en detalle todo aquello que se ha realizado para desarrollar el proyecto y completar los requisitos del capítulo anterior.

En el capítulo 5 se hablará de las pruebas realizadas para la validación del desarrollo realizado, y se estudiarán los resultados obtenidos a partir de las mismas.

Para terminar, en el capítulo 6 quedarán definidas las conclusiones a las que se ha llegado tras la validación de los resultados y mejoras futuras que se puedan realizar al proyecto.

2 Estado del arte

2.1 Introducción

En este capítulo se explicarán los conceptos teóricos y las técnicas ya conocidas para manejar y entender los siguientes 3 puntos clave: distribuciones alfa-estables, integración numérica y paralelización en CUDA. También se detallarán la búsqueda y obtención de algoritmos ya realizados para procesar los cálculos que interesen en el ámbito de este proyecto y de esa forma conseguir evitar el tener que reinventar la rueda.

2.2 Distribuciones de probabilidad alfa-estables

Las distribuciones de probabilidad estables son aquellas distribuciones de probabilidad que cumplen la siguiente propiedad: "cualquier combinación lineal de 2 copias independientes de una muestra aleatoria tiene la misma distribución, salvo posición y escala". [1]

Estas distribuciones constan de cuatro parámetros que las definen:

1. Alfa (α): denota la estabilidad de la distribución. Abarca el intervalo (0, 2].
2. Beta (β): denota la asimetría de la distribución. Abarca el intervalo [-1, 1].
3. Sigma (σ): denota la escala de la distribución. Abarca el intervalo (0, +∞).
4. Mu (μ): denota la posición relativa de la distribución (la cual define la moda, la media y la mediana). Abarca el intervalo (-∞, +∞).

El parámetro más relevante para el estudio de estas distribuciones es alfa, lo que hace que se suela utilizar el término "alfa-estables" para referirse a ellas.

Las distribuciones α-estables constan de una peculiaridad matemática: no existe expresión analítica cerrada capaz de definir el caso general de las mismas. Es por eso por lo que estas distribuciones se definen típicamente mediante su función característica: [2]

$$\Psi(t) = \begin{cases} -|\sigma t|^{\alpha}\left[1 - i\beta\tan\left(\frac{\pi\alpha}{2}\right)\operatorname{sign}(t)\right] + i\mu t, & \alpha \neq 1, \\ -|\sigma t|\left[1 + i\beta\frac{2}{\pi}\operatorname{sign}(t)\ln(|t|)\right] + i\mu t, & \alpha = 1, \end{cases}$$

$$\text{donde} \quad \operatorname{sign}(t) = \begin{cases} 1, & t > 0, \\ 0, & t = 0, \\ -1, & t < 0. \end{cases}$$

Para lograr obtener una expresión con la que poder trabajar, se hace uso de la transformada de Fourier (según el teorema de inversión de Fourier), cuya expresión es la siguiente:

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty}\phi(t)\mathrm{e}^{-itx}\,\mathrm{d}t = \frac{1}{2\pi}\int_{-\infty}^{+\infty}\mathrm{e}^{\psi(t)-itx}\,\mathrm{d}t.$$

La falta de expresiones analíticas cerradas para la función de densidad de probabilidad (PDF) y para la función de distribución acumulada (CDF) es consecuencia de que al sustituir la expresión de la función característica en la integral de Fourier, se obtiene un resultado para el que no se conoce expresión general como combinación de funciones analíticas.

Por el contrario, existen una serie de casos especiales de distribuciones alfa-estables que particularmente sí que disponen de expresiones cerradas para el cálculo analítico de las funciones de densidad y distribución. Son aquellas que cumplen con las parametrizaciones detalladas a continuación: [2]

1. Distribución de Gauss: $\alpha = 2$ (β se vuelve irrelevante).
2. Distribución de Cauchy: $\alpha = 1$ y $\beta = 0$.
3. Distribución de Lévy: $\alpha = 0'5$ y $\beta = \pm 1$.
4. Caso estándar: $\sigma = 1$ y $\mu = 0$.

En el resto de casos, se dispondrá de una distribución alfa-estable del caso general, lo que significa que no existirá ninguna expresión cerrada que la pueda definir analíticamente. Es precisamente la necesidad de procesar las distribuciones del caso general la que ha causado todo el estudio dedicado a lograr calcularlas numéricamente y de manera eficiente.

Todo esto conlleva que los únicos métodos viables para realizar el cálculo de las funciones de densidad y distribución para las distribuciones del caso general sean métodos numéricos, los cuales quedarán detallados en el capítulo 2.3.

Las distribuciones alfa-estables se pueden usar como fuente principal de información para sistemas predictivos de todo tipo, incluyendo temas tan variados como hidrología, finanzas, comunicaciones, análisis de tráfico de redes o segmentación de imágenes médicas. [2] De ahí que resulte tan útil el estudio de herramientas dedicadas al cálculo de distribuciones estables.

Debido a la imposibilidad de resolver las integrales de manera analítica, se hace necesario calcularlas de manera numérica. Al ser un proceso computacionalmente muy costoso, se plantea la necesidad de utilizar arquitecturas paralelas. Es esta parte en concreto la que motiva la realización de este proyecto, que consiste precisamente en lograr un aumento en el rendimiento del algoritmo del cálculo de las alfa-estables mediante la aceleración conseguida gracias a la paralelización del algoritmo de integración numérica.

2.3 Integración numérica

La integración numérica es la familia de métodos matemáticos diseñados para calcular el valor numérico de una integral definida mediante aproximaciones computacionales, es decir, no se trata de una resolución analítica a partir de la expresión de la integral en cuestión. Se suele utilizar el término "cuadratura" (que significa "calcular área" desde la época de los matemáticos de la Grecia antigua) como sinónimo de la integración numérica, sobre todo para aquellos casos en los que la integral posee sólo una dimensión:

$$\int_a^b f(x)\,dx$$

Existen varias razones que justifican llevar a cabo integración numérica, siendo la principal la imposibilidad de realizar la integración de forma analítica, bien porque no existe método para resolverla con la teoría matemática actual, o bien porque no se dispone del conocimiento y manejo de matemática avanzada necesarios para resolverla. También existen funciones integrables cuya primitiva no puede ser calculada, siendo la integración numérica de vital importancia en este caso.

La solución analítica de una integral proporciona una solución exacta de la misma en forma de expresión que se puede calcular con la precisión que en principio se desee, mientras que la solución numérica produce una solución aproximada. El error de la aproximación, que depende del método que se utilice y de lo fino que sea, puede llegar a ser tan pequeño que es posible obtener un resultado idéntico a la solución analítica hasta un cierto número de cifras decimales dependiendo de dicho error.

Los métodos de integración numérica pueden ser descritos generalmente como combinación de una serie de evaluaciones del integrando para obtener una aproximación de la integral. Una parte importante del análisis de cualquier método de integración numérica es estudiar el comportamiento del error de aproximación como una función del número de evaluaciones del integrando. Un método que produzca un pequeño error para un pequeño número de evaluaciones se considera superior. Por una parte, un número elevado de evaluaciones del integrando suele ser necesario para alcanzar una precisión elevada. [3] Pero por otra parte, reduciendo el número de evaluaciones del integrando se reduce el número de operaciones aritméticas involucradas y, por tanto, se reduce el error de redondeo total, además del tiempo de ejecución total (ya que cada evaluación lleva su tiempo y además el integrando puede ser arbitrariamente complicado).

Existen varios tipos de integración numérica, dependiendo de la estrategia utilizada, aunque no todos son apropiados para el cálculo de distribuciones alfa-estables. Para estos métodos se suele utilizar una estrategia con el fin de mejorar la precisión de los cálculos que consiste en lo siguiente: se divide el intervalo [a, b] en una serie de subintervalos, se aplica la regla de cuadratura a cada uno de ellos por separado, y se suman los resultados parciales. Todos ellos se basan, además, en el uso de una función de interpolación (distinta para cada regla) para lograr la aproximación de la función que se desea integrar.

- Reglas que usan un polinomio fijo de un grado en concreto:

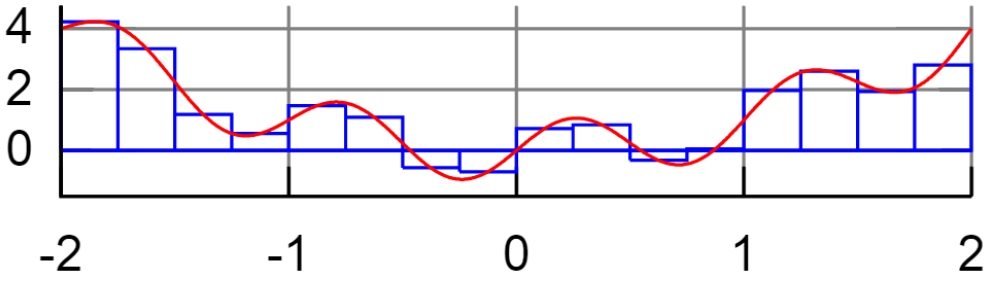
  - **Regla del punto medio**

    - Su función de interpolación pasa por el punto $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$

    - Su función de interpolación es un polinomio de orden 0 (función constante):

$$\int_a^b f(x)dx \sim (b-a)\, f\left(\frac{a+b}{2}\right)$$

    - Ejemplo de uso:



    - También se le conoce como **regla del rectángulo** si en lugar de usar el punto medio para la función constante se usa (a, f(a)).

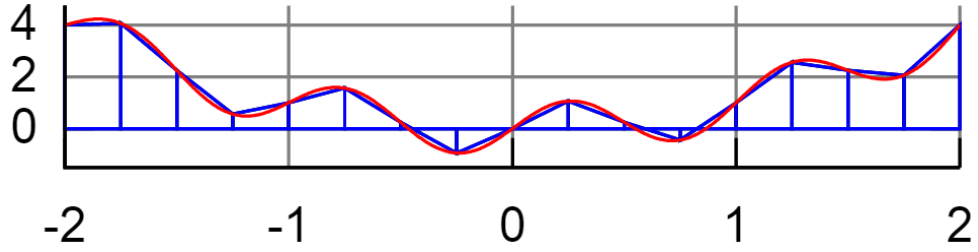
❖ **Regla del trapecio**

- Su función de interpolación pasa por los puntos $(a, f(a)), (b, f(b))$

- Su función de interpolación es un polinomio de orden 1 (función lineal):

$$\int_a^b f(x)\,dx \approx (b-a)\,\frac{f(a)+f(b)}{2}$$

- Ejemplo de uso:



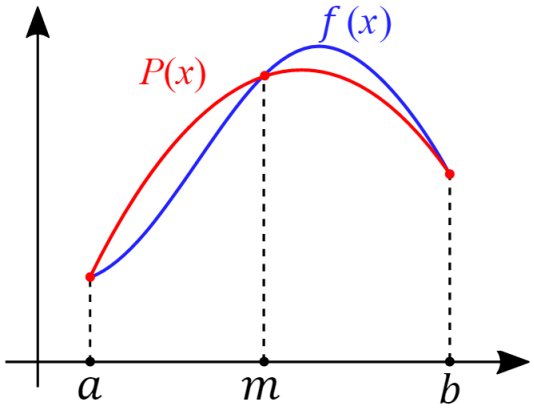
❖ **Regla de Simpson**

- Pasa por los puntos: $(a, f(a)), \left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$ y $(b, f(b))$

- Su función de interpolación es un polinomio de orden 2 (función parabólica):

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

- Ejemplo de uso:



- Reglas que usan un número de polinomios variable con diversos grados:

❖ **Regla de Gauss**

La regla de Gauss de orden *n* es una cuadratura que selecciona los puntos a evaluar de manera óptima en vez de hacerlo de forma igualmente espaciada y así conseguir resultados exactos para polinomios de grado *2n – 1* o inferior, mediante una elección apropiada de los *n* puntos (x_i) y sus pesos (w_i). Por tanto, la regla queda definida mediante la siguiente función, para un dominio típico que abarca desde -1 hasta 1:

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

La regla de Gauss solamente producirá buenos resultados si la función está bien aproximada por un polinomio en el rango [-1, 1]. Esta cuadratura, por tanto, no es adecuada para funciones que presenten singularidades en ese intervalo. [4] Pero en caso de estar definida, se consigue mucha más precisión que con las reglas polinómicas anteriores, y en ello radica su importancia.

Una conocida desventaja de la cuadratura de Gauss consiste en que cuando se divide el intervalo [a, b], los puntos de evaluación de los nuevos subintervalos nunca coinciden con los puntos de las evaluaciones previas (salvo en cero y en números impares), [4] requiriendo nuevas evaluaciones del integrando en cada uno de los nuevos puntos. Este problema queda resuelto por la siguiente regla.

- **Regla de Gauss-Kronrod**

  La regla de Gauss-Kronrod es una extensión de la regla de Gauss que consiste en añadir $n + 1$ puntos a una regla gaussiana de orden n de tal manera que la regla resultante sea de orden $2n + 1$. Los puntos de la extensión Kronrod son los ceros de los polinomios de Stieltjes. [5]

  Es una regla **anidada**, ya que dado un conjunto de puntos de evaluación para la función, posee dos reglas para realizar la cuadratura: una de orden mayor y otra de orden menor (también conocida como regla empotrada).

  La principal ventaja del uso de esta regla es que se pueden reutilizar los valores de la función de orden superior para estimar los de orden inferior, logrando un rendimiento mucho mayor en comparación a la regla gaussiana normal cuando se necesita mayor precisión en intervalos concretos (y por tanto es necesario dividirlos en subintervalos).

La característica de la integración numérica por la cual se realizan subdivisiones del intervalo solicitado para mejorar la precisión se puede explotar aún más para obtener versiones que se denominan **adaptativas** de las reglas anteriormente detalladas, que funcionan mediante una subdivisión dinámica en los intervalos de integración realizada de la siguiente manera: para cada subintervalo "normal", se compara una estimación del error relativo calculado con el de la precisión que se desea obtener, y en caso de existir una diferencia superior a un umbral, se divide reiteradamente dicho subintervalo en otros dos, repitiendo el proceso en cada uno de ellos. De esta forma se consigue mayor precisión en la parte problemática de la integral.

La regla de Gauss-Kronrod adaptativa es la que se utiliza en la librería *Libstable* debido a su elevada precisión y relativamente buena rapidez de cálculo. La implementación usada para dicha regla es la que viene dada por la librería GSL. Para ciertos casos en los que no hace falta tanta precisión también se utiliza una versión no adaptativa de dicha regla.

2.4 Técnicas de paralelización en GPUs

La computación de propósito general en GPUs es una estrategia relativamente reciente que consiste en aprovechar la gran cantidad de núcleos existentes en una tarjeta gráfica para realizar un gran número operaciones en coma flotante de forma paralela en cada núcleo. Los frameworks y APIs existentes otorgan un nivel de abstracción que permite dividir el trabajo a realizar por el programa en diversos hilos de ejecución, los cuales se organizarán posteriormente en bloques de *n* hilos y se asignarán a la tarjeta gráfica para ser procesados en paralelo, distribuyéndose de manera automática y transparente para el programador.

- **OpenCL**: es la tecnología abierta para computación paralela compatible con todas las GPUs (NVIDIA, AMD, Intel). Consta de una API (interfaz de programación de aplicaciones) y de un lenguaje de programación basado en C99, los cuales permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse en CPUs, en GPUs, en DSPs (procesadores digitales de señales) o FPGAs (Field-Programmable Gate Arrays). [6]

  Al tratarse de una tecnología pensada para su uso general en dispositivos variados, puede no llegar a ofrecer tanto rendimiento como CUDA en determinados casos para algunas GPUs concretas.

- **CUDA**: es la tecnología propietaria de NVIDIA para realizar computación paralela en GPUs de su fabricación. [7] El término hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten usar una variación del lenguaje de programación C para codificar algoritmos en sus GPUs. CUDA intenta explotar las ventajas de las GPUs frente a las CPUs de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un número muy alto de hilos simultáneos. Por ello, si una aplicación se diseña para utilizar todos estos hilos de forma que realicen tareas independientes, una GPU podrá ofrecer un gran rendimiento en comparación a lo que podría llegar a ofrecer una CPU con una versión no paralela del mismo algoritmo.

  CUDA presenta ciertas ventajas sobre otros tipos de computación para GPUs:

  - **Lecturas dispersas**: se puede leer cualquier posición de memoria de la GPU de manera arbitraria en cualquier momento.
  - **Memoria compartida y unificada**: CUDA pone a disposición del programador un área de memoria que se comparte entre hilos. Dado su tamaño y rapidez, puede ser utilizada como una caché manejada por el usuario.
  - **Lecturas y escrituras más rápidas** desde y hacia la GPU.
  - Soporte completo para **operaciones con enteros y a nivel de bit**.

  Por el contrario, también posee algunos inconvenientes que impiden que CUDA sea una tecnología perfecta:

  - Esta tecnología es propietaria y solamente se encuentra disponible en las GPUs fabricadas por NVIDIA.
  - El compilador de CUDA no soporta completamente el estándar C, ya que es un wrapper de un compilador de C++, lo que implica que no se puede utilizar código C válido que sea inválido en C++.

- No se puede utilizar recursividad, punteros a funciones, variables estáticas dentro de funciones o funciones con número de parámetros variable.
- Puede existir un cuello de botella entre la CPU y la GPU por los anchos de banda de los buses y sus latencias a la hora de transferir datos.
- Los hilos de ejecución, por razones de eficiencia, deben lanzarse en grupos de al menos 32 hilos cada uno.
- Cada proceso debe lanzarse a través de espacios de memoria disjuntos.
- No se soporta el manejo de excepciones, ya que penalizarían el rendimiento.

Como ya se ha visto, CUDA intenta aprovechar el gran paralelismo y el alto ancho de banda de la memoria existente en GPUs para ejecutar aplicaciones con un gran coste aritmético (pero que no requieran de numerosos accesos a memoria principal, lo que podría actuar de cuello de botella, sino que se basten con la caché de la GPU).

El modelo de programación de CUDA está diseñado para desarrollar aplicaciones que de manera transparente puedan escalar su paralelismo para así incrementar el número de núcleos computacionales. Este diseño se basa en la jerarquía de grupos de hilos, las memorias compartidas y las barreras de sincronización. La estructura que se utiliza en este modelo está definida por un grid (cuadrícula), dentro del cual hay bloques de hilos formados por un número de hilos configurable.

Cada hilo está identificado por un identificador único, al que se puede acceder con una variable específica, la cual es muy útil para repartir el trabajo entre distintos hilos. Además, permite realizar cálculos multi-dimensionales (hasta dimensión 3), ya que dispone de tres componentes (x, y, z) con las que se puede trabajar de forma análoga. Al igual que los hilos, los bloques también se acceden mediante una variable que los identifica, en este caso con dos componentes (x, y). Por último, existe otra variable para modificar el tamaño de cada bloque, esto es, el número de hilos que existirán en cada uno. Todas estas variables son las que dan juego para realizar las divisiones de los conjuntos de datos de entrada y repartirlos adecuadamente a lo largo de todos los recursos de la GPU. [7]

Este proyecto se ha centrado en la tecnología CUDA, puesto que ya existía un estudio previo realizado con OpenCL para acelerar el rendimiento de la librería *Libstable*. [8] En el capítulo 5 se comparará dicha versión con la de este proyecto para estudiar las diferencias entre ellas.

2.5 Conclusiones

En este capítulo se ha detallado el estado del arte en cuanto a distribuciones de probabilidad alfa-estables, métodos de integración numérica y mecanismos para paralelizar en GPUs.

Se han explicado los conceptos teóricos que existen detrás de cada apartado, incluyendo la base matemática que será necesario entender adecuadamente para lograr resultados correctos en el desarrollo del proyecto.

En el siguiente capítulo se realizará un análisis de las tareas solicitadas para este proyecto, así como un análisis de la librería *Libstable* para encontrar el mejor punto de optimización a paralelizar en la aplicación serie.

3 Análisis

3.1 Introducción

En el presente capítulo se estudia qué tipo de aplicación es necesario realizar para este proyecto mediante un análisis de sus requisitos. Puesto que ya se ha explicado el estado del arte de las tecnologías relacionadas, se puede proceder con la enumeración de los requisitos y valorar rápidamente lo que implica cumplir cada uno en términos de trabajo necesario. La solución propuesta consiste en sustituir la integración numérica de la librería *Libstable* por una versión en paralelo, como se verá a continuación.

3.2 Análisis de Requisitos

Las tareas (requisitos funcionales) que se han solicitado para la elaboración de este proyecto son las siguientes:

- **Identificar los cuellos de botella presentes en el algoritmo de la librería *Libstable* de cálculo de distribuciones alfa-estables.**
  Para realizar esta tarea se realizará un perfilado de la aplicación *Libstable* mediante el uso de la herramienta callgrind para detectar aquellos puntos en los que el tiempo de ejecución se vea penalizado por exceso de cálculos.

- **Mejorar el rendimiento de las funciones previamente detectadas.**
  Esta tarea se realizará mediante la paralelización en CUDA de aquellas partes de la librería que se hayan identificado en la tarea anterior como responsables del "bajo" rendimiento (es decir, no lo suficientemente alto como para ser útil en todos los sistemas predictivos de inferencia estadística) de la librería *Libstable*.

- **Acelerar el tiempo de ejecución de la librería *Libstable*.**
  Se requerirá realizar la integración de la versión paralela de las funciones "malas" a la librería *Libstable*, sustituyendo la versión serie de dichas funciones por la nueva y reprogramando toda la funcionalidad necesaria para conseguir un funcionamiento equivalente entre la versión original de *Libstable* y aquella con código CUDA.

- **Comprobar que no se haya perdido precisión en el cálculo de distribuciones.**
  Para llevar a cabo esta tarea se ejecutarán una serie de tests de precisión comparando las dos versiones de *Libstable* para observar los resultados de ambas.

- **Estudiar la aceleración conseguida tras la obtención de la nueva librería.**
  Será necesario realizar una batería de pruebas de rendimiento ejecutando mediciones de tiempo comparativas entre ambas versiones de *Libstable*, todo esto con respecto a ejecuciones variables en cuanto a complejidad computacional para poder también estudiar la escalabilidad de la aplicación.

3.3 Estudio de la aplicación serie

Para poder localizar los cuellos de botella del algoritmo serie de cálculo de distribuciones alfa-estables, es necesario realizar un perfilado y observar qué llamadas son las que más ciclos de reloj consumen y, por tanto, las que más tiempo tardan en ejecutarse.

Tal y como se puede comprobar en la ilustración 3, que es un resultado de la ejecución de callgrind (usado debido a su sencillez y buen funcionamiento) a un binario de ejemplo de *Libstable*, se ve que, efectivamente, las llamadas a las funciones de integración numérica de GSL son lo que más tiempo de ejecución consume, llegando hasta el 80% del tiempo total para un caso con un tamaño del conjunto de datos problema relativamente grande.

Por tanto, se puede aplicar la Ley de Amdahl para calcular la relación de aceleración entre la parte que se va a paralelizar (p) y la parte en serie (s): [9]

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \qquad S_{\text{latency}}(s) = \frac{1}{\mathbf{0'2} + \frac{\mathbf{0'8}}{s}}$$

Al sustituir p por 0’8, se ha podido simplificar la expresión para tener la relación exacta entre la aceleración total de la aplicación y la aceleración del algoritmo de integración numérica.

De aquí se puede deducir que la aceleración máxima teórica que se podrá llegar a conseguir mediante la aceleración del 80% del tiempo de cómputo del algoritmo será de un x5 (esto se deduce mediante el cálculo del límite de la expresión para $s \to \infty$).

Todo esto sirve para concluir que la estrategia para conseguir acelerar la librería *Libstable* consistirá en paralelizar el algoritmo de integración numérica mediante CUDA, para así obtener una aceleración parcial del orden de x100 y acercarse lo máximo posible al x5 total calculado anteriormente, lo cual se consigue gracias a la presencia de cientos de núcleos en la GPU disponibles para su uso de forma paralela. El factor x5 es en la práctica inalcanzable debido a las limitaciones de la arquitectura paralela de las GPUs utilizadas.

Por último, conviene recordar que, según la Ley de Gustafson, el límite dado por la fórmula anterior es directamente proporcional al tamaño del conjunto de datos de entrada, [10] por lo que un aumento lo suficientemente grande en dicho tamaño conllevaría un aumento en el factor de aceleración máxima obtenible por la versión paralela del algoritmo.

3.4 Conclusiones

En este capítulo se han explicado las tareas requeridas para la elaboración del proyecto, así como el estudio de la aplicación serie mediante un perfilado para detectar los cuellos de botella causantes del “bajo” rendimiento de la librería *Libstable*, que han resultado ser (tal y como se había predicho) las llamadas a las funciones de integración numérica de GSL, lo cual permite deducir que es precisamente la integración numérica lo que se debe paralelizar para lograr una aceleración positiva.

En el siguiente capítulo se podrán ver con exactitud los pasos seguidos hasta conseguir la implementación del método paralelo de integración numérica acoplado a *Libstable* y cómo va tomando forma el proyecto gracias a ello.

4 Desarrollo

4.1 Introducción

En este capítulo se explicarán el diseño y el desarrollo derivados del análisis del capítulo anterior, es decir, el procedimiento que se ha llevado a cabo para poder generar la aplicación que se había pedido y cumpliendo con todos los requisitos del apartado anterior. Para ello se hablará primero de cómo se ha obtenido el algoritmo de integración numérica en CUDA, seguido de la explicación acerca de cómo se ha añadido dicho algoritmo a *Libstable* para realizar el cálculo de las distribuciones alfa-estables.

4.2 Integración numérica en paralelo

Lo primero que se estudió fue la posibilidad de modificar el algoritmo de una versión estándar de la integración numérica de Gauss-Kronrod para hacerla paralela mediante CUDA. Este enfoque resultó ser excesivamente complicado debido a la complejidad del algoritmo de la cuadratura de Gauss-Kronrod, que poseía interdependencias paralelas muy perjudiciales en la ejecución de los bucles principales del algoritmo. Es por esto por lo que se optó por seguir una estrategia distinta: investigar acerca de algoritmos de integración numérica para CUDA que ya se encontrasen disponibles para poder usarse.

Cuhre es una librería realizada por matemáticos e informáticos de la universidad *Old Dominion* de Virginia que implementa un algoritmo paralelo, eficiente y determinista de integración numérica adaptativa multidimensional para CUDA. [11] Su alta portabilidad y la gran aceleración que consigue (hasta un x100) la convirtieron en la principal candidata a tener en cuenta para este proyecto. Su diagrama de flujo se encuentra en la ilustración 4.

Al tratarse de un algoritmo **adaptativo** de integración, consta de dos fases diferenciadas. En la primera, se realiza una división en partes iguales del intervalo que se pretende integrar y se aplican unas reglas para calcular la integral de cada subintervalo. Para cada una de esas integrales, se comprueba si superan el umbral de error relativo que se haya solicitado. En caso negativo, se guarda su valor para la suma total. De lo contrario, se marca dicho subintervalo como "malo". Esta fase continúa hasta que se obtiene un número suficiente de subintervalos "malos", en cuyo caso se procede con la segunda fase. La segunda fase del algoritmo se dedica a aplicar una serie de subdivisiones sucesivas a los intervalos "malos" y a ejecutar una rutina secuencial de integración sobre los mismos que finaliza cuando se obtiene un error relativo inferior a la tolerancia definida. Cada una de estas rutinas se ejecuta en paralelo en un núcleo CUDA de la GPU, de forma que se reparten todos los subintervalos "malos" entre todos los núcleos de la GPU. Para finalizar, se actualizan los valores globales de la integral y la estimada del error, sumando los valores calculados en esta fase a los valores previos de la fase anterior de los intervalos "buenos".

Por otra parte, dado que es un algoritmo para integración **multidimensional**, fue necesario realizar varias modificaciones para asegurar su funcionamiento correcto en integrales de una sola dimensión. Además, los parámetros de configuración de hardware tuvieron que ser ajustadose para la GPU presente en el entorno de pruebas (en este caso, el *número de bloques*). Tras una serie de sencillos tests para comprobar que la precisión de las integrales *Cuhre* era equivalente a la de las integrales Gauss-Kronrod, se decidió apostar por el uso de esta librería como principal método paralelo de integración numérica para *Libstable*.

4.3 Cálculo de distribuciones alfa-estables

La librería *Libstable* realiza las operaciones de integración numérica mediante una serie de llamadas a las funciones de GSL encargadas de integrar mediante la regla Gauss-Kronrod.

Tal y como está diseñada la librería, sustituir estas llamadas no es algo trivial, ya que existe un nivel de abstracción diseñado para poder cambiar entre métodos con facilidad a la hora de ejecutar los programas principales de *Libstable*, el cual se ha decidido respetar.

Se tuvo que añadir un wrapper para la función *Cuhre* de integración:

```
extern "C" void Cuhre(const size_t ndim, const size_t ncomp, const double epsrel,
const double epsabs, const double a, const double b, const int flags, size_t
*nregions, size_t *neval, size_t *fail, double integral[], double error[]);

void singlegpucuhre(double (*func)(double, void *), void *args,
                    double a, double b, const double epsabs, const double epsrel,
                    unsigned short limit, double *result, double *abserr)
{
  size_t nregions, neval, fail;

  Cuhre(1, 1, epsrel, epsabs, a, b, 0, &nregions, &neval, &fail, result, abserr);
}
```

La llamada a esta función se situó junto al resto de métodos de integración. De esta forma se consigue implementar el uso de *Cuhre* de manera totalmente transparente para la librería *Libstable*, lo que conlleva una mayor facilidad para cambiar entre un método u otro para cada ejecución de los programas de prueba.

Algunos problemas relacionados con la portabilidad que se tuvieron que resolver fueron:

- No poder enlazar librerías en la compilación debido al uso de flags incompatibles en los ficheros Makefile de cada una. Hubo que reestructurar la compilación.

- No poder compilar algunas partes de *Libstable* debido al cambio de compilador, al tratarse de código C válido pero inválido en C++. Hubo que reescribir ese código.

- No disponer del mismo nivel de precisión de coma flotante en partes del código de las librerías (partes de *Cuhre* que usaban precisión simple se adaptaron para usar precisión doble en su lugar y así ser equivalentes a la integración de GSL).

Una vez finalizada la integración de *Cuhre* en *Libstable*, tras solucionar los problemas anteriores, se dio por finalizado el desarrollo y se comenzó a realizar la validación.

4.4 Conclusiones

En este capítulo se ha mostrado cómo se han adaptado las 2 librerías (*Cuhre* y *Libstable*) para hacerlas compatibles la una con la otra, así como la implementación del acoplamiento de *Cuhre* dentro de *Libstable*. Una vez finalizado el desarrollo, pues, el siguiente paso es obvio en cualquier ciclo de desarrollo: se deben realizar pruebas para cerciorarse del correcto funcionamiento de la aplicación y validar el cumplimiento de los requisitos.

5 Validación

5.1 Introducción

Este capítulo abordará la realización de todas las pruebas y tests necesarios para comprobar el funcionamiento correcto de la versión paralela de *Libstable*, tanto en precisión como en rendimiento, así como el cálculo empírico de la aceleración obtenida gracias al uso de dicha versión paralela, en comparación a la versión serie.

Para ello se ha llevado a cabo una serie de mediciones con distinto número de puntos para, de esa forma, poder observar la evolución de la relación de aceleración conforme aumenta el número de datos a calcular. Lo esperado es obtener unos resultados de rendimiento que sean similares a aquellos obtenidos en la versión OpenCL de *Libstable* [8], teniendo en cuenta las particularidades y diferencias a nivel de hardware que se explicarán a continuación.

5.2 Pruebas

- **Precisión**

  Se ha diseñado un test para comprobar que los resultados dados por *Libstable-cuda* en el cálculo de PDF(x) y CDF(x) sean lo suficientemente equivalentes a aquellos calculados por *Libstable-serie* en función del nivel de error relativo especificado.

  Probando para un nivel de error absoluto de 10^{-10} y una distribución alfa-estable del caso general ($\alpha = 1'5$, $\beta = 0'75$, $\sigma = 5$, $\mu = 15$), se ha comprobado que los resultados para puntos arbitrarios de la distribución coinciden exactamente igual hasta llegar a la cifra definida por el error (el noveno dígito decimal), a partir de la cual pueden no llegar a coincidir, lo que es previsible dado el nivel de error elegido.

  El código C del programa de test de precisión es el siguiente:

```
#include <stdio.h>
#include "stable_api.h"

int main (void)
{
  double alfa = 1.5, beta = 0.75, sigma = 5.0, mu = 15.0;
  double x = 0, q = 0.1, pdf, cdf;
  int param = 0;

  StableDist *dist = stable_create(alfa, beta, sigma, mu, param);

  pdf = stable_pdf_point(dist, x, NULL);
  cdf = stable_cdf_point(dist, x, NULL);

  printf("PDF(%g) = %.16lf\n", x, pdf);
  printf("CDF(%g) = %.16lf\n", x, cdf);

  stable_free(dist);
  return 0;
}
```

- **Rendimiento**

  Para realizar las pruebas de rendimiento, se ha utilizado un programa de test que ya viene incluido junto con la librería *Libstable*, denominado **stable_test**. Este binario recibe como argumentos todos los parámetros que puedan variar en el cálculo de una distribución alfa-estable, que se detallan a continuación:

  a) Parámetros de la distribución (α, β, σ, μ).
  b) Flag de parametrización (puede ser 1 ó 0).
  c) Extremos del intervalo a integrar (a, b).
  d) Distancia de paso entre puntos. Se usa para definir qué (y cuántos) puntos evaluar para la división en subintervalos del intervalo [a, b].
  e) Número de muestras aleatorias a calcular.
  f) Número de hilos de CPU a utilizar para paralelizar mediante OpenMP.
  g) Tolerancia del error absoluto y tolerancia del error relativo.
  h) Métodos de integración que se desean usar (método A y método B), según la enumeración definida en el código. *Cuhre* se correspondería con el índice 8.

  Un ejemplo de ejecución del programa sería el siguiente:

```
./stable_test 1.5 0.75 5 15 0 -3 3 0.01 1000 1 1e-10 1e-10 8 8
```

  En cuanto al funcionamiento del programa, consiste en ejecutar las funciones de *Libstable* encargadas de calcular PDF y CDF para los puntos que se hayan definido en función del intervalo solicitado y la distancia de paso (es decir, la distancia entre cada punto que se va a evaluar en [a, b]), y de realizar una medición de tiempo de cómputo necesitado para completar las operaciones. De esta forma se obtiene como resultado el número de puntos por segundo que se podrían procesar dados los parámetros solicitados al programa. Así, comparar resultados entre ambos métodos de integración (Gauss-Kronrod de GSL y *Cuhre*) es tan sencillo como modificar los parámetros de entrada en cuanto al método de integración a utilizar (los dos últimos).

5.3 Resultados

- **Precisión**

  El test desarrollado permite calcular valores de PDF(x) y CDF(x) para valores arbitrarios de x. Estos son los resultados del test para la versión serie de *Libstable*:

  - PDF(-10) = **0’000316674**0403176
  - CDF(-10) = **0’004629690**1418765

  - PDF(0) = **0’002431008**0029025
  - CDF(0) = **0’013367535**4865993

  - PDF(10) = **0’042316628**2042123
  - CDF(10) = **0’181683592**9584865

Por otro lado, aquí están los resultados para la versión paralela:

- PDF(-10) = **0’000316674**1228703
- CDF(-10) = **0’004629690**2481572

- PDF(0) = **0’002431008**0947842
- CDF(0) = **0’013367535**3472363

- PDF(10) = **0’042316628**1752953
- CDF(10) = **0’181683593**0848259

Dado un nivel de error absoluto de 10^{-10}, esto demuestra que ambas versiones son equivalentes, al disponer de números cuyas cifras son exactamente iguales hasta llegar a la que define el error.

De esta forma se puede comprobar que, efectivamente, la precisión alcanzada por la versión CUDA de *Libstable* es lo suficientemente buena como para poder usarse de la manera esperada, es decir, igual que la versión serie.

- **Rendimiento**

Antes de detallar los resultados obtenidos en la ejecución del test explicado en el apartado anterior, es necesario realizar un preámbulo y hablar acerca de la versión de *Libstable* cuya integración numérica se ha paralelizado con OpenCL [8].

En dicha versión se ha logrado una aceleración máxima de x10 en el cálculo de PDF y CDF en comparación a la versión serie de *Libstable*. La ejecución se realizó en una tarjeta gráfica Tesla M2090, que posee 512 núcleos CUDA a 1’3 GHz y 177 GB/s de velocidad de ancho de banda para la memoria.

Por el contrario, la tarjeta gráfica del presente entorno de pruebas es una GTX 480 con 480 núcleos a 700 MHz y 177 GB/s de ancho de banda de memoria. Esto significa que para realizar una comparación entre ambas versiones, se deben tener en cuenta las diferencias en hardware para estimar la aceleración máxima proporcional que se podrá obtener en nuestro caso (para el mismo conjunto de datos variable y exigiendo la misma precisión que en el caso OpenCL).

La tarjeta GTX 480 tiene casi tantos núcleos como la Tesla M y el mismo ancho de banda para la memoria (177 GB/s), pero la frecuencia de cada núcleo es de casi la mitad, por lo que teóricamente cabría esperar una aceleración máxima aproximada 2 veces menor que la obtenida con la tarjeta Tesla M. Es decir, habría una aceleración máxima (al alza) de x5 en la tarjeta GTX 480 en caso de recrear todas las condiciones que se definieron para realizar las pruebas en OpenCL.

Una vez explicado esto, se pasa a observar los resultados obtenidos en la ejecución del test de rendimiento en la tarjeta GTX 480 con unos parámetros similares a los utilizados en los tests OpenCL de la tarjeta Tesla M (intervalo entre -100 y 100, error de 10^{-10} y distribución alfa-estable de caso general) en función del número de puntos a evaluar (para tener en cuenta la Ley de Gustafson):

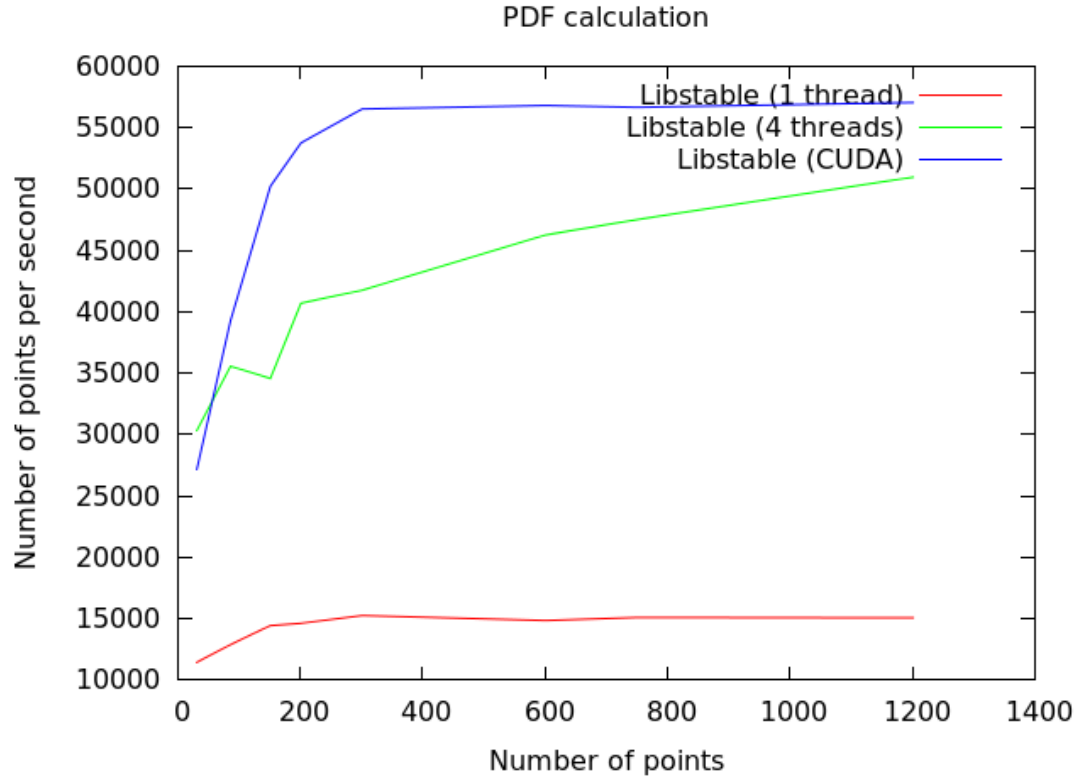


Ilustración 1: cálculo de PDF mediante Libstable-CUDA

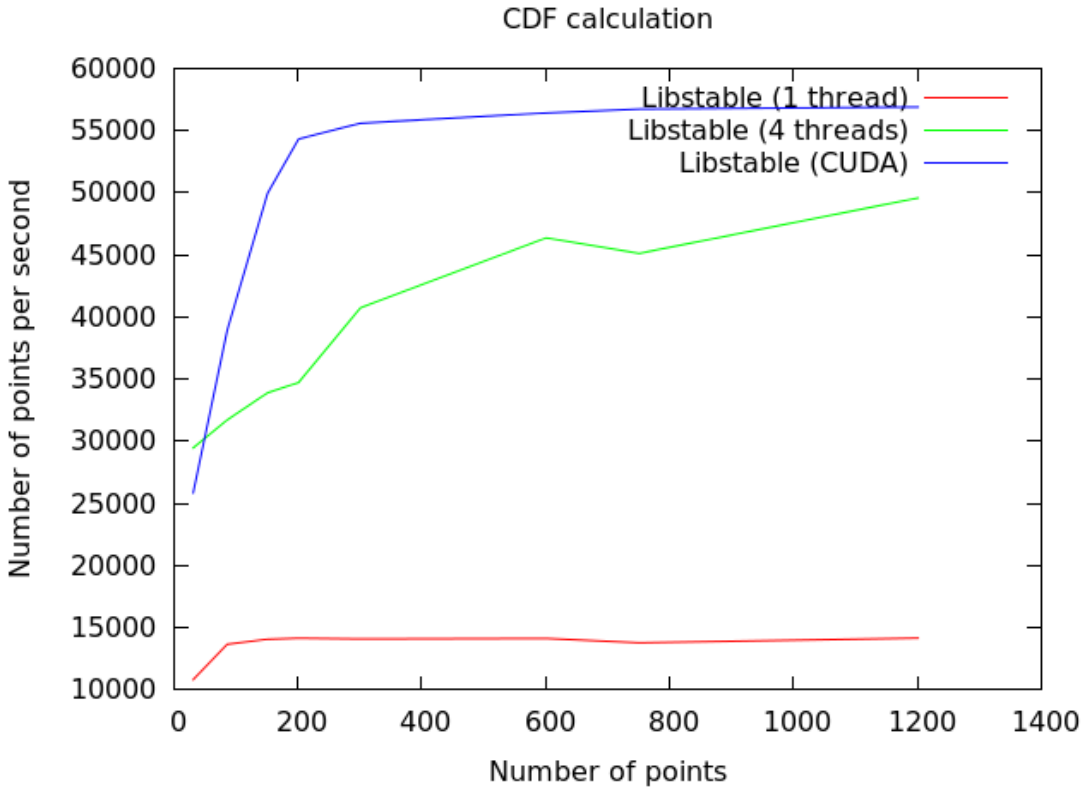


Ilustración 2: cálculo de CDF mediante Libstable-CUDA

Como se puede observar en las gráficas anteriores de las ilustraciones 1 y 2, se ha logrado una aceleración de aproximadamente x3'8 con respecto a la versión serie de *Libstable* cuando el número de puntos a evaluar es mayor a 200. Esto es un 76% de la aceleración máxima estimada en la comparación anterior, pudiéndose justificar el no haber llegado al 100% por los tres siguientes motivos:

- La estimación estaba realizada al alza, puesto que realmente siempre existirán más parámetros aparte del número de núcleos y la frecuencia que conllevarán una mayor diferencia en la potencia de procesamiento entre dos tarjetas gráficas distintas, y al tratarse la GTX de una tarjeta más antigua que la Tesla M, se encuentra por ello en desventaja. Aparte de la diferencia entre el lanzamiento de las tarjetas, la gama Tesla está específicamente pensada para su uso en servidores de propósito computacional, mientras que la GTX es una tarjeta general para usuario medio. Esto conlleva una desventaja aún más elevada para la GTX, que puede disponer de características menos potentes que la Tesla M debido al propósito en mente con el que se fabricó.

- El método de integración numérica implementado en *Libstable-OpenCL* se realizó de manera específica para ser usado en la librería de cálculo de distribuciones alfa-estables, lo que conlleva un grado de optimización muy alto. Por el contrario, *Cuhre* es una librería de integración numérica de propósito general, lo que implica que haya realizado sacrificios en cuanto a rendimiento para poder otorgar toda la portabilidad que ofrece. Esto significa que *Libstable-CUDA* también está en desventaja por usar la librería Cuhre en lugar de un algoritmo específicamente diseñado para *Libstable*.

- Las CPUs usadas en ambas máquinas son distintas: la usada en los tests de CUDA junto con la tarjeta GTX posee menor rendimiento que su homóloga, lo que puede ocasionar penalizaciones de rendimiento debido a las operaciones que son necesarias realizar en CPU para preparar la ejecución de la integración en CUDA (por ejemplo, las lecturas y escrituras de memoria desde el kernel GPU al host CPU, y viceversa). El resto de operaciones que se realizan en CPU no afectan, en principio, a nuestra comparativa, ya que precisamente nos comparamos con respecto a la versión serie.

Pese a todo esto, Libstable-CUDA ofrece incluso mejor rendimiento que una ejecución de la versión sin GPU con 4 hilos de ejecución OpenMP en CPU, logrando aceleraciones entre x1'5 y x2 (dependiendo del número de puntos) con respecto a ella, lo cual es positivo.

5.4 Conclusiones

En este capítulo se han detallado las pruebas realizadas para demostrar el funcionamiento del programa de este proyecto y los resultados de dichas pruebas, separándolos en 2 tipos: precisión y rendimiento.

Tras comprobar los resultados, se puede llegar a la conclusión de que la aplicación ha conseguido su objetivo, teniendo en cuenta las limitaciones de hardware presentes en las máquinas donde se han realizado las pruebas. Se ha conseguido una aceleración muy cercana a la que se había predicho con anterioridad y sin perder precisión en el cálculo de las alfa-estables, lo cual es todo un éxito.

En el siguiente capítulo se tratarán las conclusiones finales de todo el proyecto, mencionando aquellas cosas que se hayan aprendido durante su realización y las mejoras para el futuro.

6 Conclusiones

6.1 Introducción

En este capítulo, para concluir, se analizarán tanto el trabajo desarrollado durante el proyecto como el resultado del mismo. Se extraerán conclusiones finales y, además, se mencionarán aquellas cosas que podrían ser interesantes de cara al futuro como posibles mejoras para ampliar aún más la funcionalidad de la aplicación o para mejorar la ya existente.

6.2 Conclusiones

El objetivo de este proyecto consistía en la aceleración de un algoritmo para el cálculo de distribuciones de probabilidad alfa-estables. Tal y como ha quedado demostrado por los resultados del capítulo anterior, es justo afirmar que el objetivo se ha cumplido.

Se ha conseguido implementar un algoritmo de integración numérica paralelizado en CUDA para el algoritmo de cálculo de distribuciones alfa-estables, lo que ha ocasionado un aumento en el rendimiento de la aplicación, consiguiendo una aceleración de hasta 3,8 veces en comparación a la versión original del algoritmo.

También se ha podido comprobar de primera mano el efecto de la Ley de Gustafson, por el cual el tamaño de los datos de entrada de un problema está directamente relacionado con la aceleración posible de conseguir mediante la optimización parcial de solamente una parte del mismo. De igual forma, se ha validado la Ley de Amdahl para datos de tamaño fijo.

Se ha visto cómo la paralelización de un algoritmo en CUDA es capaz de funcionar con un rendimiento similar a una versión equivalente de dicho algoritmo realizado en OpenCL.

6.3 Trabajo futuro

Para finalizar, se enumerarán aquellas tareas posibles de realizar para conseguir una mejora en la aplicación en cuanto a funcionalidad y facilidad de uso como actualización futura.

- Realizar de manera dinámica los ajustes relativos a hardware (nº de bloques GPU) para no necesitar configurarlos manualmente antes de la compilación del programa.

- Optimizar aún más la integración numérica para aumentar la aceleración mediante la simplificación de aquél código que no sea necesario para el caso unidimensional de la integración numérica.

- Construir una batería de pruebas que abarque todos los casos de uso en cuanto a número de puntos a evaluar, número de hilos usados y método de integración usado.

- Realizar pruebas de rendimiento en una máquina con una mejor tarjeta gráfica.

Un desarrollo futuro de gran utilidad podría consistir en la implementación de un sistema predictivo capaz de caracterizar en tiempo real el tráfico de una red para detectar anomalías como intrusiones o ataques de denegación de servicio mediante inferencia estadística basada en un modelo de distribuciones de probablidad alfa-estables. [12]

Referencias

[1] John P. Nolan (1997), "Numerical calculation of stable densities and distribution functions". *Communications in Statistics. Stochastic Models*.

[2] Javier Royuela del Val, Federico J. Simmross Wattenberg and Carlos Alberola López, "Libstable: Fast, parallel and high-precision computation of α-stable distributions in C/C++ and MATLAB", *Journal of Statistical Software*, **in press**.

[3] Philip J. Davis and Philip Rabinowitz (2007), *Methods of Numerical Integration*.

[4] Gene H. Golub, John H. Welsch (1969), "Calculation of Gauss Quadrature Rules", *Mathematics of Computation*.

[5] Sven Ehrich (2001), "Stieltjes polynomials", *Encyclopedia of Mathematics*, Springer.

[6] OpenCL official website, https://www.khronos.org/opencl/ (retrieved 2016-06-20).

[7] CUDA Zone, https://developer.nvidia.com/cuda-zone (retrieved 2016-06-20).

[8] Guillermo Julián Moreno, Jorge E. López de Vergara, Iván González, Luis de Pedro, Javier Royuela del Val, and Federico J. Simmross Wattenberg, "Fast parallel α-stable distribution function evaluation and parameter estimation using OpenCL in GPGPUs", *Statistics and Computing*, Springer, **submitted.**

[9] Gene M. Amdahl (1967), "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" *AFIPS Conference Proceedings*.

[10] John L. Gustafson, "Reevaluating Amdahl's Law", *Communications of the ACM*.

[11] Kamesh Arumugam, Alexander Godunov, Desh Ranjan, Balsa Terzic and Mohammad Zubair, "An Efficient Deterministic Parallel Algorithm for Adaptive Multidimensional Numerical Integration on GPUs". **PDF available**: http://goo.gl/bf63rD

[12] Federico J. Simmross Wattenberg, Juan Ignacio Asensio Pérez and Marcos Martín Fernández, "Detección de anomalías en el tráfico agregado de redes IP basada en inferencia estadística sobre un modelo α-estable de primer orden".

Anexos

Anexo A:

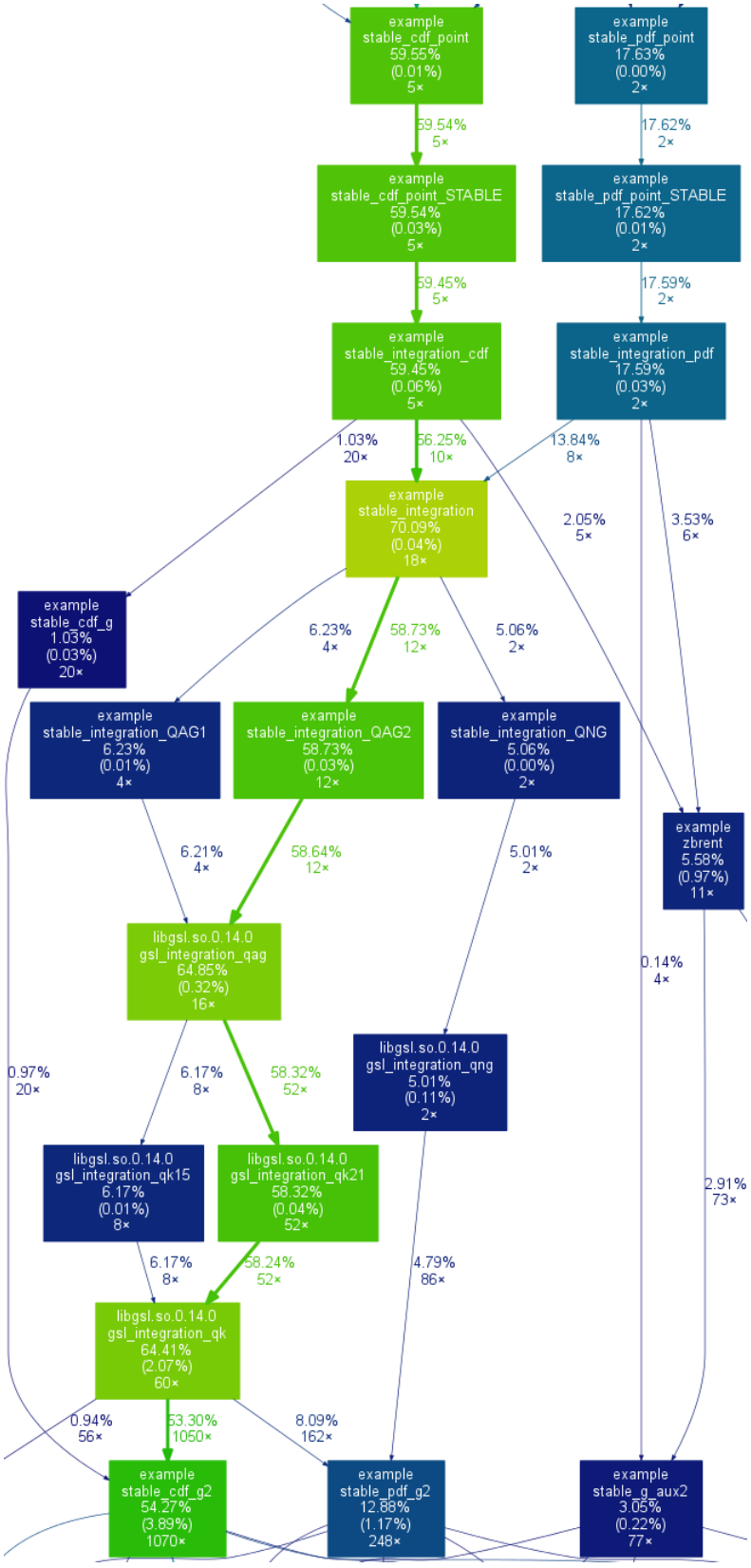


Ilustración 3: perfilado de Libstable-serie para un conjunto de datos pequeño

Anexo B:

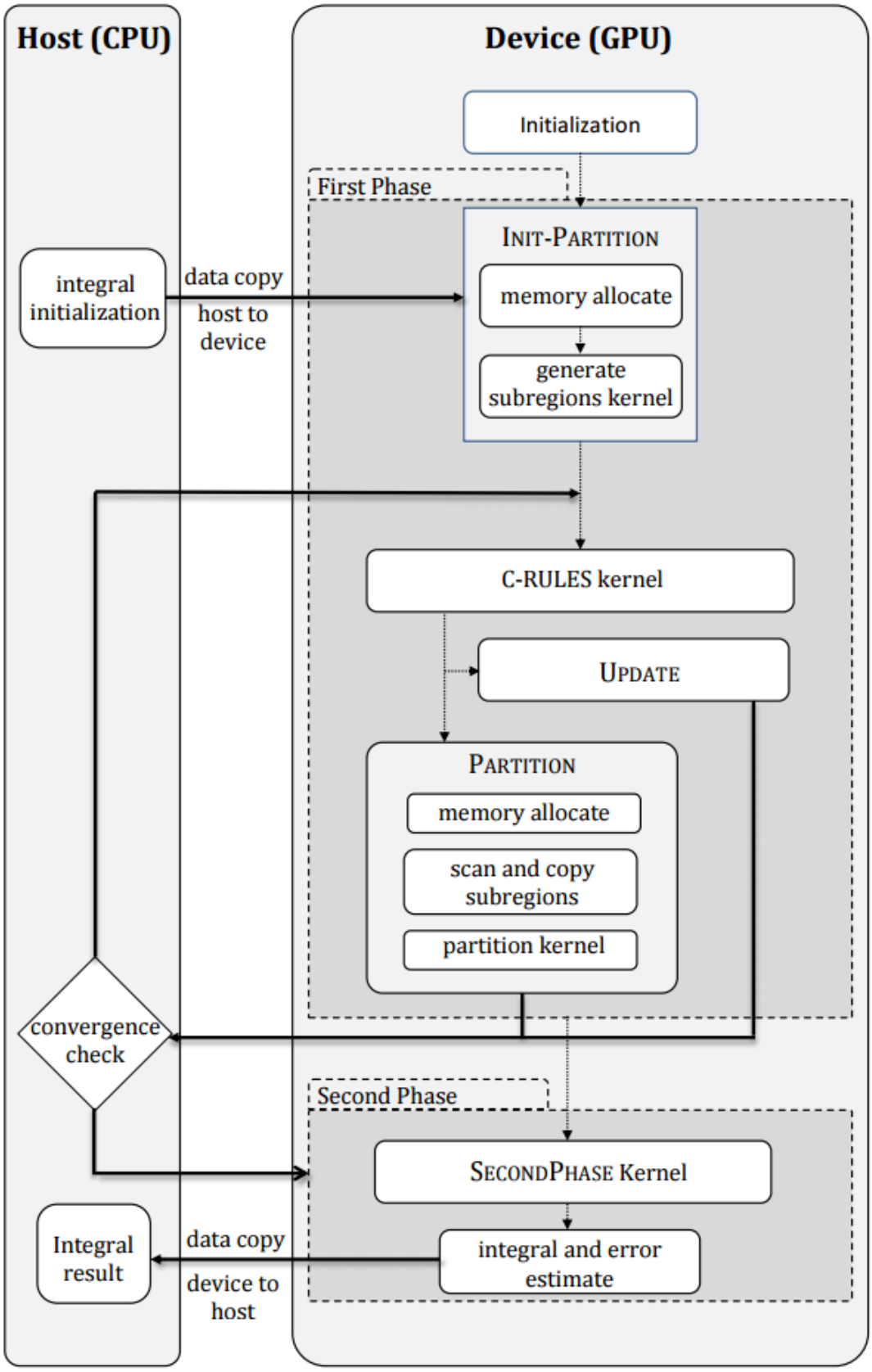


Ilustración 4: diagrama de flujo del algoritmo de integración Cuhre