



**CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITECNICO NACIONAL**

UNIDAD ZACATENCO

DEPARTAMENTO DE COMPUTACION

Núcleo de Monte Carlo y Camino Aleatorio en
Ambientes de Alto Desempeño

Tesis que presenta:
Fabiola Ortega Robles

Para obtener el grado de:
Maestra en Ciencias

En la especialidad de:
Ingeniería Eléctrica

Opción en computación

Director de Tesis: Dr. Sergio Victor Chapa Vergara

México, D.F.

Agosto 2008

Resumen

El cómputo de alto desempeño es utilizado dentro del área de la computación científica (principalmente), para dar solución a problemas que demandan una gran cantidad de manejo de instrucciones y datos. Dentro de este tipo de cómputo, se pueden encontrar diferentes paradigmas tales como: cómputo de alto desempeño local, distribuido y de red amplia.

En particular, el método de Monte Carlo (MC) ha sido utilizado en diferentes áreas de las ciencias como la Física, Mecánica Cuántica, Finanzas, Biología, etc. Su versatilidad se da por su naturaleza estadística, la cual engloba métodos básicos, lo que permite que MC pueda ser adaptado a una gama de diferentes áreas y problemas. Dicho método puede ser utilizado para realizar experimentación numérica o simulación de problemas difíciles de atacar en una forma analítica, así como problemas complejos tanto en el espacio de datos como en el número de instrucciones a ejecutar (lo que se traduce en mayores tiempos de procesamiento).

Este método se ha tornado durante varios años muy importante, debido a que a través de su uso es posible verificar modelos matemáticos, validar o corroborar experimentos, además que hace posible la obtención de predicciones. La complejidad al utilizar Monte Carlo depende del problema y del área en particular en donde es aplicado.

En nuestro trabajo de tesis se presenta un núcleo de Monte Carlo, el cual va estar compuesto de un conjunto de funciones de distribución para la generación de números aleatorios, tanto continuas como discretas. Estos números aleatorios son generados con un determinado comportamiento, según la distribución seleccionada. En nuestro caso son utilizados para simular procesos que son modelados mediante los métodos de integración Wiener, Ito y Feynman, las cuales requieren de la generación de trayectorias aleatorias.

Se realiza la creación de muestras, que se componen por un conjunto de trayectorias aleatorias. Para la creación de muestras con una gran cantidad de trayectorias, se requiere de un cómputo intensivo, una de las razones por las que este tema fue de nuestro interés.

El diseño e implementación de la biblioteca para la generación de muestras se realizó en forma paralela, lo que permite mejorar el desempeño en cuanto a cómputo se refiere. Para la paralelización se utilizó la biblioteca MPI (Message Passing Interface) haciendo uso del modelo SPMD (del inglés, Simple Program Multiple Data) en un cluster de computadoras.

Por otro lado, el núcleo de Monte Carlo y los métodos de Wiener e Ito fueron desarrollados como servicios Web, para su uso dentro de un *“framework”* llamado Taverna, el cual permite el desarrollo de *“workflows”* o flujos de trabajo. Además, el uso de Taverna

se da en la Grid para realizar cómputo de alto desempeño en forma distribuida a través de redes heterogéneas y de forma transparente al usuario. Todo esto con la finalidad de que pueda ser utilizado por un grupo de usuarios que no tengan conocimiento o no estén familiarizados con la programación, lo que permite que se concentren solo en su propio problema.

Nuestro diseño fue ejecutado en un cluster de computadoras utilizando diferentes parámetros para la generación de trayectorias, en donde cada procesador del cluster obtuvo paralelamente un conjunto de trayectorias las cuales conforman una muestra. En las pruebas realizadas se utilizaron diferentes números de procesadores con diferentes tamaños de población. Al aumentar el número de procesadores utilizando un mismo tamaño de población fue posible disminuir el tiempo de cómputo.

Abstract

High Performance Computing (HPC) is mainly used in scientific computing in order to find solutions to problems which requires multiple instructions and data processing. In this type of computing, one can find different paradigms, such as Local High Performance Computing, Distributed Computing and Wide Network Computing.

Particularly, the Monte Carlo method (MC) has been used in different areas of science, e.g., Physics, Quantum Mechanics, Finances, Biology, etc. Its versatility is given by its statistics nature that includes basic methods, this allows that MC to be adapted to a huge kind of problems and different areas. MC can be used to do numeric experimentation, simulation of problems which are difficult to solve in an analytic manner, complex problems in a sense of data space and in the number of instructions to execute (this means more processing time).

This method has become very important with the years because this method is useful to verify mathematical models, to validate and corroborate experiments, and furthermore it makes it possible to obtain predictions. The complexity in the use of Monte Carlo method depends on the problem and also depends on the particular area where it is applied.

In this thesis we present one Monte Carlo core. This core is composed of a set of distribution functions to generate continuous and discrete random numbers. These random numbers are generated with a certain behavior, according to the selected distribution. In our case, they are used to simulate processes that are modeled by Wiener integration methods, Ito and Feynman, which require generation of random trajectories.

We create samples that consist of a set of random trajectories. For the creation of samples with a huge amount of trajectories, it is required an intensive computing, that is one of the reasons for which this subject is of our interest.

The design and the implementation of the library required for the generation of samples was made in parallel, this allows improving the computing performance. MPI (Message Passing Interface) was used for the parallelization of the library, making use of the model SPMD (Single Program Multiple Data) in a cluster of computers.

On the other hand, Monte Carlo core, Wiener and Ito methods were developed as Web services, for their use within “framework” called Taverna, which allows the development of “workflows”. In addition, the use of Taverna occurs in the Grid to obtain calculation of high performance in a distributed form through heterogeneous networks and transparently to the user. All this is done in order to enable it to be used by users that do not have

programming knowledge or for those that are not familiarized with computer programming. It allows users to be concentrated only in their main problem.

Our design was executed in a cluster of computers using different parameters for the generation of trajectories. Each processor of cluster obtained, in a parallel form, a set of trajectories which conform a sample. In the tests different numbers of processors with different population size were used. When we increment the number of processors fixing the size of the population, it is possible to reduce the computing time.

DEDICATORIAS

He llegado al final de este camino para el comienzo de uno nuevo:

*Con todo mi corazón a una de las dos personas que más amo:
Ma. Guadalupe Robles H. mi madre y mi mejor amiga.*

*A mis hermanas: Mónica y Lizbet: ustedes, mi mamá y yo,
siempre juntas apoyándonos sin importar las distancias.*

*A mis hermanos: Eduardo, Saúl y Miguel, deseo que
algún día volvamos estar todos juntos.*

AGRADECIMIENTOS

Agradezco al Consejo Nacional de Ciencia y Tecnología por el apoyo económico recibido.

Agradezco al Centro de Investigación y de Estudios Avanzados (CINVESTAV) del IPN por el apoyo de la beca terminal.

Agradezco al Dr. Sergio V. Chapa Vergara por el apoyo brindado para poder concluir el presente trabajo de tesis.

Agradezco a la Dra. Julieta Medina García por su tiempo, apoyo, comentarios y sugerencias para el desarrollo del trabajo de tesis.

De igual forma agradezco a la Dra. Sonia Guadalupe Mendoza Chapa por su apoyo, consejos y sugerencias. Además de darle gracias por su disposición a escucharnos y apoyarnos incondicionalmente.

Agradezco al Dr. José Guadalupe Rodríguez García por sus sugerencias, comentarios y su amistad brindada.

Agradezco a Jesús Cazares por su disposición permanente e incondicional para aclarar mis dudas, así como por su amistad. Siempre tendrás en mí a una amiga.

Agradezco a todos los doctores investigadores del departamento de computación por sus enseñanzas y conocimientos transmitidos.

Agradezco a Jesús Muedano C. por su apoyo y comprensión durante el tiempo que estuve cursando mi maestría, por el tiempo y momentos que hemos pasado juntos.

Agradezco a mi madre, por todo su amor, cariño, comprensión, por haber sido mi guía durante muchos años; sino hubieses sido quien eres, yo no sería quien soy ahora. A mi padre por sus principios y valores, gracias.

Agradezco a mis amigos: Claudia Villalpando, Pascualita, Ernesto Meza, Chemita, Puch, Puga, Juan Ortiz, Solis (siempre serás mi amigo) y Araiza; por su amistad y apoyo durante todos estos años. Se que no siempre van a poder estar en todos mis proyectos de vida, pero sin embargo también se que siempre van a estar ahí para compartir con ustedes tanto mis éxitos como mis fracasos. Los llevo siempre en mis recuerdos y mi corazón.

Agradezco muy especialmente a José de J. Araiza Ibarra, por todo su apoyo, por darse el tiempo para escucharme, por sus recomendaciones, enseñanzas y consejos; por ser uno de mis ejemplos de que con trabajo y esfuerzo todo se puede lograr, por la fortaleza que inyecta en mí y por cree en mí. Siempre has sido y serás como mi hermano.

Agradezco a Felipa y Flor por su apoyo y disposición para darnos un consejo.

Agradezco a Sofia Reza por la amistad y consejos que nos brinda a todos, en especial a mí durante mi estancia en el CINVESTAV, además de darle gracias por su apoyo en todo momento.

INDICE

1 Introducción.....	1
2 Ambientes computacionales de alto desempeño.....	7
2.1 Computación local	7
2.1.2 Procesadores multi-núcleo.....	8
2.1.3 Arquitectura de memoria para procesadores multi-núcleo.....	9
2.1.4 Características de los procesadores multi-núcleo	9
2.1.5 Impacto de los procesadores multi-núcleo en el desarrollo de SW.....	11
2.2 Computación distribuida.....	12
2.2.1 Antecedentes computación distribuida.....	12
2.2.2 Características.....	13
2.2.3 Diferentes enfoques computación distribuida.....	14
2.2.4 Visión laboratorio Mac.....	15
2.3 Computación paralela.....	18
2.3.1 Antecedentes de cómputo paralelo.....	19
2.3.2 Cluster.....	20
2.3.3 Paso de mensajes.....	22
2.3.4 Tipos de programación.....	22
2.3.5 Laboratorio cluster.....	24
2.3.6 Ambientes de desarrollo.....	24
2.3.7 El Hardware.....	25
2.4 Grid (Computación distribuida de área amplia).....	26
2.4.1 Uso de la Grid.....	27
2.4.2 La Grid y servicios web.....	27
2.4.3 Arquitectura Grid.....	29
2.4.4 Ventajas y desventajas del uso de la Grid	30
3 El método de Monte Carlo.....	31
3.1 Naturaleza del método de Monte Carlo.....	31
3.1.1 Generación de camino aleatorio.....	33
3.2 Distribuciones continuas.....	33
3.2.1 Uniforme continua.....	33
3.2.2 Exponencial.....	34
3.2.3 Gamma.....	34

3.2.4 Beta.....	35
3.2.5 Normal ó gaussiana.....	36
3.2.6 Logonormal.....	37
3.2.7 Cuachy.....	38
3.2.8 Weibull.....	39
3.2.9 Chi-cuadrada.....	40
3.2.10 T de Student.....	40
3.3 Distribuciones discretas.....	41
3.3.1 Binomial.....	41
3.3.2 Poisson.....	42
3.3.3 Geométrica.....	43
3.3.4 Hipergeométrica.....	43
3.3.5 Uniforme discreta.....	44
3.4 Problemas de simulación.....	45
3.4.1 Historia de un neutron.....	46
3.4.2 Cálculo de área sobre un plano.....	46
4 Integrales funcionales y de Feynman.....	49
4.1 Integral de Wiener.....	49
4.1.1 Algoritmo Wiener.....	51
4.2 Integral de Ito.....	52
4.2.1 Algoritmo de Ito.....	55
4.3 Integral de Feynman.....	55
4.3.1 Algorirmo de Feynman.....	60
4.4 Algoritmo paralelo para la generación de trayectorias.....	62
5 Arquitectura y diseño	65
5.1 Arquitectura para computación local.....	65
5.2 Diseño en OpenMP.....	66
5.3 Diseño y arquitectura en computación paralela distribuida.....	69
5.4 Diseño en MPI.....	71
5.5 Diseño paralelo para generación de trayectorias.....	74
5.6 Implementación de solución MPI.....	75
5.7 Arquitectura para computación Grid.....	79
5.8 Diseño en Taverna.....	82
5.9 Implementación en Taverna.....	84

6 Resultados	89
6.1 Resultados cómputo paralelo.....	89
6.2. Resultados de las pruebas realizadas a Wiener.....	90
6.2.1 Tiempos de ejecución Wiener 4 puntos.....	95
6.2.2 Tiempos de ejecución Wiener 8 puntos.....	97
6.2.3 Tiempos de ejecución Wiener 16 puntos.....	99
6.2.4 Tiempos de ejecución Wiener 32 puntos.....	101
6.3 Resultados de las pruebas realizadas a Ito y Feynman.....	103
6.3.1 Tiempos de ejecución para la integral de Ito con 10 puntos.....	109
6.3.2 Tiempos de ejecución para la integral de Ito con 20 puntos.....	112
6.3.3 Tiempos de ejecución para la integral de Ito con 30 puntos.....	114
6.3.4 Tiempos de ejecución para la integral de Feynman con 10 puntos.....	117
6.3.5 Tiempos de ejecución para la integral de Feynman con 20 puntos.....	119
6.3.6 Tiempos de ejecución para la integral de Feynman con 30 puntos.....	122
6.4 Resultados computación red amplia.....	124
6.5 Conclusiones.....	129
6.6 Trabajo futuro.....	131
Apéndice A	133
A.1 Uniforme.....	133
A.2 Normal.....	135
REFERENCIAS	139

LISTA DE FIGURAS

1.1 Mapa de ruta de la tesis.....	4
2.1 Arquitectura SMP (<i>Shared Memory Multiprocessors</i>).....	9
2.2 Arquitectura del procesador quad-core.....	10
2.3 Arquitectura multi-núcleo dual quad-core.....	10
2.4 Vista General del Laboratorio Mac a Futuro.....	16
2.5 Red Laboratorio Mac.....	16
2.6 Problema en forma secuencial.....	18
2.7 Problema en forma paralela.....	18
2.8 Ejemplo de Cluster.....	20
2.9 MPP (Massively Parallel Processors).....	21
2.10 Plataforma del cluster (software).....	24
2.11 Arquitectura laboratorio cluster.....	25
2.12 Esquema de una Grid.....	26
2.13 Esquema de comunicación entre servicios.....	28
3.1 Algoritmo ITM.....	33
3.2 Algoritmo E-1.....	34
3.3 Algoritmo G-1.....	35
3.4 Algoritmo Be-1.....	36
3.5 Algoritmo forma polar.....	37
3.6 Algoritmo LN-1.....	38
3.7 Algoritmo C-3.....	38
3.8 Algoritmo W-1.....	39
3.9 Algoritmo Chi-cuadrada.....	40
3.10 Algoritmo Student's t.....	41
3.11 Algoritmo IT-2.....	42
3.12 Algoritmo IT-2.....	42
3.13 Algoritmo GEO.....	43
3.14 Algoritmo IT-2.....	44
3.15 Algoritmo uniforme discreta.....	44
3.16 Relación entre teoría, experimento y simulación.....	45
3.17 Calculo de área.....	47
4.1 Algoritmo Wiener.....	51
4.2 Algoritmo de Ito.....	55
4.3 Ejemplo trayectorias.....	56
4.4 Trayectoria.....	57
4.5 Algoritmo de Feynman.....	60
4.6 Algoritmo apFuncion.....	61
4.7 Algoritmo valor IT_accion.....	61
4.8 a Algoritmo paralelo.....	62
4.8 b Algoritmo paralelo.....	63
4.8 c Algoritmo paralelo.....	64
5.1 Arquitectura computación local.....	65
5.2 Hilos con OpenMP.....	66
5.3 Diseño computación local.....	68

5.4 Distribución de datos en un cluster.....	69
5.5 Arquitectura general.....	70
5.6 Diseño de la solución.....	71
5.7 Componentes de módulo núcleo Monte Carlo.....	73
5.8 Diseño de módulo trayectorias.....	73
5.9 Diseño paralelo para generación de trayectorias.....	74
5.10 Conjunto de bibliotecas y su relación.....	75
5.11 Arquitectura para el diseño en Taverna.....	80
5.12 Diseño del conjunto de servicios a desarrollar.....	82
5.13 Ejemplo de componente en Taverna.....	83
5.14 Diagrama de los métodos Wiener e Ito.....	84
5.15 Diagrama núcleo Monte Carlo.....	85
5.16 a) Servicios del núcleo Monte Carlo.....	86
5.16 b) Servicios del núcleo Monte Carlo.....	87
5.17 Servicios web módulo integración.....	87

LISTA DE TABLAS

6.1 Ejecuciones realizadas para Wiener con 4 puntos.....	90
6.2 Ejecuciones realizadas para Wiener con 8 puntos.....	91
6.3 Ejecuciones realizadas para Wiener con 16 puntos.....	92
6.4 Ejecuciones realizadas para Wiener con 32 puntos.....	93
6.5 Ejecuciones realizadas para Ito y Feynman con 10 puntos.....	104
6.6 Ejecuciones realizadas para Ito y Feynman con 20 puntos.....	106
6.7 Ejecuciones realizadas para Ito y Feynman con 30 puntos.....	107
6.8 Servicios Taverna.....	124

LISTA GRAFICAS

6.1 Trayectoria con 4 puntos de Wiener.....	91
6.2 Trayectoria con 8 puntos de Wiener.....	92
6.3 Trayectoria con 16 puntos de Wiener.....	93
6.4 Trayectoria con 32 puntos de Wiener.....	94
6.5 Tiempos de ejecución para 1000 trayectorias de Wiener (4 puntos).....	95
6.6 Tiempos de ejecución para 10000 trayectorias de Wiener (4 puntos).....	95
6.7 Tiempos de ejecución para 100000 trayectorias de Wiener (4 puntos).....	96
6.8 Tiempos de ejecución para 1000000 de trayectorias de Wiener (4 puntos).....	96
6.9 Tiempos de ejecución para 1000 trayectorias de Wiener (8 puntos).....	97
6.10 Tiempos de ejecución para 10000 trayectorias de Wiener (8 puntos).....	97
6.11 Tiempos de ejecución para 100000 trayectorias de Wiener (8 punto).....	98
6.12 Tiempos de ejecución para 1000000 de trayectorias de Wiener (8 punto).....	98
6.13 Tiempos de ejecución para 1000 trayectorias de Wiener (16 puntos).....	99
6.14 Tiempos de ejecución para 10000 trayectorias de Wiener (16 puntos).....	99
6.15 Tiempos de ejecución para 100000 trayectorias de Wiener (16 punto).....	100
6.16 Tiempos de ejecución para 1000000 de trayectorias de Wiener (16 punto).....	100
6.17 Tiempos de ejecución para 1000 trayectorias de Wiener (32 puntos).....	101
6.18 Tiempos de ejecución para 10000 trayectorias de Wiener (32 puntos).....	101
6.19 Tiempos de ejecución para 100000 trayectorias de Wiener (32 punto).....	102
6.20 Tiempos de ejecución para 1000000 de trayectorias de Wiener (32 punto).....	102
6.21 Trayectoria con 10 puntos de Ito.....	105
6.22 Trayectoria con 10 puntos de Feynman.....	105
6.23 Trayectoria con 20 puntos de Ito.....	106
6.24 Trayectoria con 20 puntos de Feynman.....	107
6.25 Trayectoria con 30 puntos de Ito.....	108
6.26 Trayectoria con 30 puntos de Feynman.....	108
6.27 Tiempos de ejecución para 1000 trayectorias de Ito (10 puntos).....	109
6.28 Tiempos de ejecución para 10000 trayectorias de Ito (10 puntos).....	110
6.29 Tiempos de ejecución para 100000 trayectorias de Ito (10 puntos).....	110
6.30 Tiempos de ejecución para 1000000 de trayectorias de Ito (10 puntos).....	111
6.31 Tiempos de ejecución para 1000 trayectorias de Ito (20 puntos).....	112
6.32 Tiempos de ejecución para 10000 trayectorias de Ito (20 puntos).....	112
6.33 Tiempos de ejecución para 100000 trayectorias de Ito (20 puntos).....	113
6.34 Tiempos de ejecución para 1000000 de trayectorias de Ito (20 puntos).....	113
6.35 Tiempos de ejecución para 1000 trayectorias de Ito (30 puntos).....	114
6.36 Tiempos de ejecución para 10000 trayectorias de Ito (30 puntos).....	115
6.37 Tiempos de ejecución para 100000 trayectorias de Ito (30 puntos).....	115
6.38 Tiempos de ejecución para 1000000 de trayectorias de Ito (30 puntos).....	116
6.39 Tiempos de ejecución para 1000 trayectorias de Feynman (10 puntos).....	117
6.40 Tiempos de ejecución para 10000 trayectorias de Feynman (10 puntos).....	117
6.41 Tiempos de ejecución para 100000 trayectorias de Feynman (10 puntos).....	118
6.42 Tiempos de ejecución para 1000000 de trayectorias de Feynman (10 puntos).....	118
6.43 Tiempos de ejecución para 1000 trayectorias de Feynman (20 puntos).....	119
6.44 Tiempos de ejecución para 10000 trayectorias de Feynman (20 puntos).....	120

6.45	Tiempos de ejecución para 100000 trayectorias de Feynman (20 puntos).....	120
6.46	Tiempos de ejecución para 1000000 de trayectorias de Feynman (20 puntos)...	121
6.47	Tiempos de ejecución para 1000 trayectorias de Feynman (30 puntos).....	122
6.48	Tiempos de ejecución para 10000 trayectorias de Feynman (30 puntos).....	122
6.49	Tiempos de ejecución para 100000 trayectorias de Feynman (30 puntos).....	123
6.50	Tiempos de ejecución para 1000000 de trayectorias de Feynman (30 puntos)...	123

Capítulo 1

Introducción

La computación de alto desempeño (HPC), surge para resolver problemas complejos (en distintas áreas), este tipo de computación a permitido otros desarrollos tecnológicos. Muchos de estos avances se dieron gracias a las necesidades y requerimientos que han aparecido a través de los años. La primera aplicación de la HPC fue dentro del área de la investigación, ya que conforme fue avanzando la ciencia, surgieron varios problemas como: la aplicación de Monte Carlo, análisis de cadenas de ADN, simulación de sistemas dinámicos, etc., los cuáles demandaban gran capacidad de cómputo para obtener resultados.

Inicialmente para dar solución a varios problemas con las características anteriores, aparecieron las supercomputadoras. Este tipo de equipo es muy costoso, por lo que no fue fácil tener acceso a él. Debido a esto se busco una solución alterna haciendo uso de tecnologías existentes en ese momento, como: concentradores, redes, computadoras personales, etc., con lo que fue posible compartir recursos entre un conjunto de computadoras interconectadas a través de una red. De ahí nacen los “*clusters*” con lo cual es posible tener acceso a un tipo de computación HPC a un relativamente bajo costo.

Actualmente haciendo una combinación de equipos, redes e Internet surge el paradigma computacional de **red amplia o Grid**, el cual es relativamente nuevo, hace uso de la red Internet para la interconexión de redes heterogéneas, con la finalidad de compartir sus recursos. Su inicio también se dió dentro del área de la investigación científica, con el propósito de fomentar la colaboración entre instituciones e individuos (en un inicio científicos), para compartir tanto recursos como datos e información, con la finalidad de mejorar la calidad de sus investigaciones.

El uso de la **Grid** ha permitido la creación de Centros de Investigación Virtuales, en donde es posible trabajar conjuntamente o individualmente, haciendo uso de recursos disponibles. Estos recursos pueden ir desde bases de datos hasta memoria o procesadores. Para hacer posible lo anterior, se ha trabajado en sistemas que permiten la creación de aplicaciones complejas, las cuales hacen uso de dichos recursos, todo eso de una forma transparente al usuario. Un ejemplo de esos sistemas es Taverna.

Taverna es un “*framework*” , que permite la creación de “*workflows*”, en base a un conjunto de servicios o componentes que están disponibles dentro del “*framework*”. Por medio de los “*workflows*”, es posible la creación de aplicaciones más complejas en base a módulos sencillos.

Un método muy utilizado desde los años 40's es el método de Monte Carlo, el cual por algún tiempo se dejó de lado, debido a que algunos problemas requerían de una gran cantidad de cálculos, los cuales no era posibles de realizar, ya que requerirían de mucho tiempo (incluso años). Dichos problemas en su momento fueron considerados intratables. Con la aparición de la computadora, el método de Monte Carlo resurge con un nuevo enfoque computacional. Haciendo uso de equipo de alto desempeño, problemas que no tuvieron solución anteriormente fue posible retomarlos y además obtener resultados aceptables de ellos.

La parte medular del método de Monte Carlo es la generación de números aleatorios que obedecen a cierta función de distribución, por medio de los cuales es posible la construcción de una población ficticia en base a muestras creadas por medio de números aleatorios. El método de Monte Carlo es un método estadístico, en donde la varianza y la reducción de la misma es un factor muy importante para la aproximación a buenos resultados. Aunque en sus inicios fue desarrollado dentro del área de la física, el método no está casado con ella. Sino que actualmente es utilizado en diferentes áreas y su complejidad depende del problema y el área en donde es aplicado.

Esté método es de nuestro interés, debido a la importancia del mismo y a que su aplicación para obtener resultados o dar solución a algún tipo de problema requiere de un HPC (computo de alto desempeño). En varias áreas es posible el uso y combinación de MC y la HPC como por ejemplo: Mecánica Cuántica, Finanzas, Biología, Mecánica estadística, etc. Su combinación da como resultado una herramienta muy útil.

Existen procesos que son modelados por medio de trayectorias aleatorias. Esta el caso del movimiento browniano, problemas de finanzas o en diferentes sistemas que implican un comportamiento errático o aleatorio; otro de los usos importantes es la simulación de sistemas, en donde es posible mediante las trayectorias generadas tener un historial en un segmento de tiempo dado, del comportamiento de un sistema diseñado previamente en forma matemática. Para producir dichas trayectoria es necesaria la generación de números aleatorios, aquí es en donde se hace uso del núcleo de MC.

Uno de nuestros intereses u objetivos del presente trabajo de tesis, es el interés de desarrollar una herramienta con la cual un usuario pueda hacer uso del método de Monte Carlo, así como la generación de trayectorias en diferentes áreas, con la finalidad de que el usuario final no necesite estar familiarizado con toda la "maquinaria" necesaria para implementar Monte Carlo, como por ejemplo el generador de números aleatorios. El objetivo es que el usuario final solo se preocupe por invocar un conjunto de programas y adaptarlos a sus necesidades, para poder obtener sus resultados, así como tener disponible una herramienta con la que sea posible resolver problemas que requieran de HPC.

Al generar una cantidad grande de trayectorias aleatorias o cuando algún problema a tratar es muy complejo, se demanda de gran capacidad de cómputo. Para ello, se consideró

el uso de un cluster y la Grid, los cuales permite aprovechar los recursos distribuidos, correr programas en paralelo y de esa forma obtener resultados de forma más rápida, lo que es posible bajando los tiempos de ejecución al hacer uso de varios procesadores al mismo tiempo.

Los objetivos específicos de este trabajo son:

El desarrollo de un núcleo basado en Monte Carlo y camino aleatorio para su uso en ambientes de alto desempeño como: computación de red amplia y computación paralela.

Extender el uso de Taverna a aplicaciones del método de Monte Carlo, como podrían ser: bioinformática, química, economía, etc., por mencionar algunas áreas.

Esta tesis presenta el diseño de un núcleo de Monte Carlo y camino aleatorio. El uso de este núcleo de Monte Carlo fue pensado originalmente para ambientes de alto desempeño en forma paralela. Sin embargo, puede ser utilizado tanto en un programa secuencial, como en un programa en paralelo, ya sea con hilos o alguna biblioteca de paso de mensajes.

El núcleo para camino aleatorio fue diseñado y desarrollado para su uso en un ambiente paralelo de alto desempeño con el uso de la librería MPI (Message Passing Interface), este núcleo se encarga del desarrollo de un conjunto de trayectorias en base a tres métodos diferentes que son: Wiener, Ito y Feynman. La finalidad de su desarrollo en paralelo es debido a que la generación de dichas trayectorias requiere de gran capacidad de cómputo. Con su implementación en paralelo se busco disminuir los tiempos para la generación de trayectorias, además de presentar una herramienta, la cual pueda ser utilizada por un conjunto de personas, las cuales no les sea necesario invertir mucho tiempo en programación. De esa forma se pretende facilitar su uso y esto permita concentrarse solamente en el problema principal en donde va ser utilizado el núcleo.

Además se presenta el diseño de un núcleo de Monte Carlo, así como de camino aleatorio para los métodos de Wiener e Ito, para su uso en un ambiente de alto desempeño, pero en esta ocasión de red amplia como lo es la Grid. El diseño e implementación para la Grid fue hecho en servicios Web, los cuales permiten la transferencia de información a través de aplicaciones sin importar la plataforma o tipo red en donde se encuentre el servicio. El núcleo consta de un conjunto de servicios Web, lo cuales permiten la generación de números aleatorios con diferentes comportamientos, dependiendo de la distribución deseada. El núcleo de camino aleatorio hace uso de estos servicios.

Estos núcleos fueron incorporados al “*framework*” Taverna. En Taverna los servicios anteriormente mencionados, son manejados como componentes, en donde podrán ser utilizados en la construcción de diferentes “*workflows*”, los cuales requieran el uso del método de Monte Carlo o camino aleatorio.

A continuación es descrita y presentada la organización de la tesis:

La figura 1.1 muestra el mapa de ruta de la tesis, con el cual se pretende presentar en una forma esquemática la organización de la tesis:

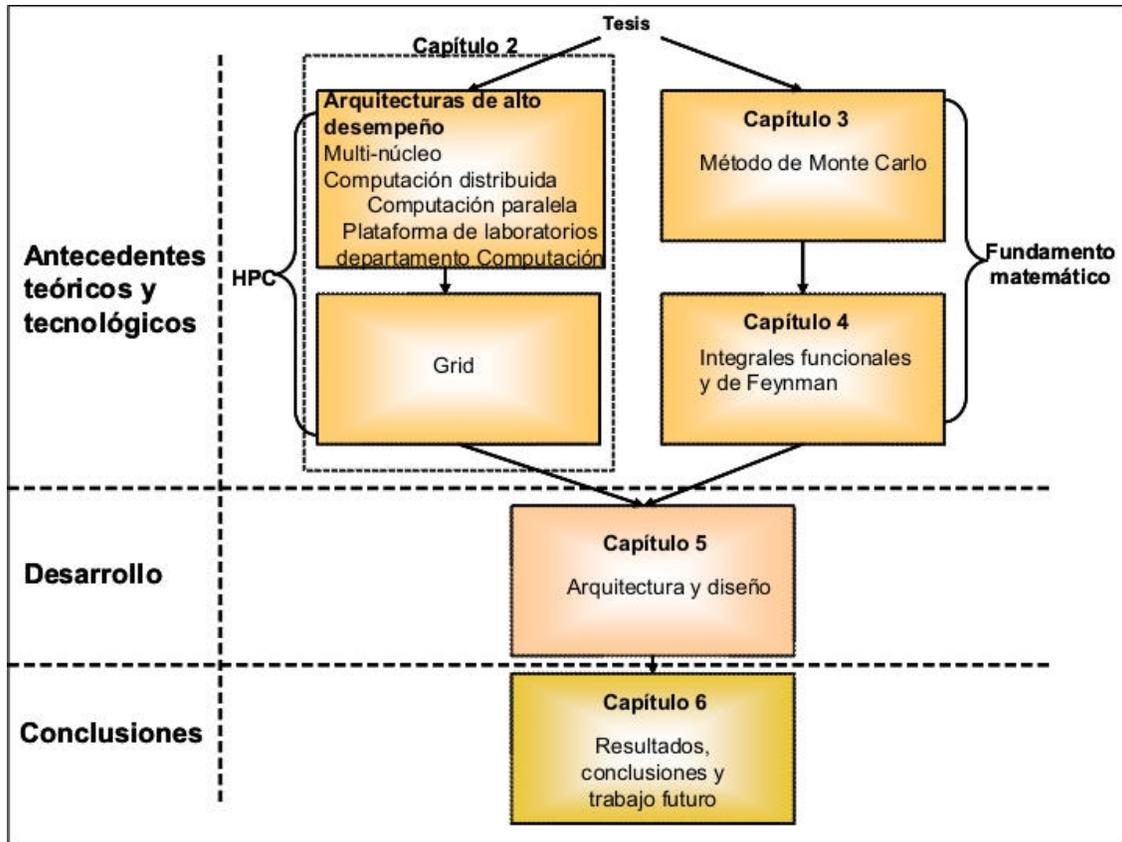


Fig: 1.1 Mapa de ruta de la tesis.

La tesis se organiza en 3 secciones que agrupan 6 capítulos. La sección de antecedentes teóricos y tecnológicos (capítulos 2, 3 y 4), la sección de desarrollo (capítulo 5), y la sección de conclusiones (capítulo 6).

El capítulo 2: “*Ambientes computacionales de alto desempeño*” presenta los diferentes paradigmas computacionales: computación local, computación distribuida y computación de red amplia, haciendo mención de sus arquitecturas como: procesadores multi-núcleo, clusters y Grid. Este capítulo tiene la finalidad de introducir al lector en los diferentes ambientes de alto desempeño, dando a conocer su beneficio y la necesidad actual de hacer uso de este tipo de ambientes.

El capítulo 3: “*El método de Monte Carlo*” presenta la importancia del uso del método de Monte Carlo, su naturaleza, cada uno de los algoritmos que componen el núcleo, los cuales se dividen en dos grupos que son: la generación de números aleatorios continuos y la

generación de número aleatorios discretos, lo cuales obedecen a un comportamiento según la función de probabilidad seleccionada. Por último se mencionan dos ejemplos de la aplicación del método.

El capítulo 4: “*Integrales funcionales y de Feynman*” presenta cada una de la integrales funcionales Wiener, Ito y Feynman, así como los algoritmos utilizados para la construcción de trayectorias, según sus propiedades. Por último se presenta el algoritmo utilizado para la generación de trayectorias en forma paralela, en base a los algoritmos de cada integral.

El capítulo 5: “*Arquitectura y diseño*” presenta las arquitecturas utilizadas para el desarrollo del proyecto. Se presenta una propuesta para el uso del núcleo de Monte Carlo y de trayectorias para procesadores multi-núcleo. Además de presentar el diseño del proyecto tanto para su versión en paralelo como su versión en servicios Web, para su uso en Taverna.

El capítulo 6: “*Resultados, conclusiones y trabajo futuro*” presenta los resultados obtenidos de la ejecución de los programas en paralelo, las conclusiones sobre el trabajo realizado y algunas recomendaciones como trabajo futuro.

Capítulo 2

Ambientes computacionales de alto desempeño

A través del tiempo, la capacidad de cómputo se ha incrementado considerablemente, esta mejora se debe a los avances en el desarrollo de la tecnología en el campo de la electrónica, en base a dichos avances se ha dado (por el momento) solución a varios problemas como: el manejo y la ejecución de grandes volúmenes de información, la creación de programas más sofisticados con el fin de hacer más fácil su manejo al usuario final, programas para la resolución de problemas que demanda cada día mayor capacidad de cómputo. Lo anterior, ha llevado al desarrollo de nuevas tecnologías y paradigmas de computación que puedan cumplir con los requerimientos actuales.

En el presente capítulo se dará una breve descripción del concepto: *Computación de Alto Desempeño* o HPC (High Performance Computer), dentro de los diferentes paradigmas computacionales.

2.1 Computación local

Como computación local puede considerarse a un ambiente de red LAN (Local Area Network). Sin embargo, en este trabajo de tesis la computación local de la que se habla, se refiere a la capacidad de cómputo, ambiente y arquitectura de una máquina computadora local.

Con el paso del tiempo han aparecido diferentes equipos. A partir del año 1945 aparecen los primeros proyectos de computadoras con alta velocidad electrónica y gran escala, como por ejemplo la ENIAC y EDVAC. Con el advenimiento de los transistores surgió una nueva computadora denominada “*Mainframe*”, la cual es una máquina de propósito general, pero con una tendencia hacia aplicaciones científicas que requieren alta velocidad [1], equipo que no es accesible en cuanto al costo se refiere.

La *ley de Moore* ha sido muy importante en el campo de la computación y la electrónica, nace de la observación de *Gordon Moore* en el año de 1965, en el desarrollo de tecnología digital. Dicha observación es la siguiente: *La densidad de circuitos integrados por chip, se dobla aproximadamente cada dos años [2]*. Con lo anterior, se hace posible el aumento de velocidad en procesadores, lo que lleva primeramente al desarrollo de microprocesadores con mayor capacidad de procesamiento y después a abaratar los costos por transistor; lo cual permite que dicha tecnología este al alcance de un mayor número de personas.

Entre los años de 1983 al 2002 el poder de ejecución se incrementó en una frecuencia de 5 Mhz a 3 Ghz, lo que da un porcentaje o una razón de 600 veces la velocidad, sin embargo, el poder de crecimiento en la densidad de transistores y el calor producido por los procesadores, fueron algunos de los limitantes para seguir aumentando la capacidad de los mismos [3]. Para dar solución a los limitantes antes mencionados, fue necesario pensar en una nueva tecnología. El advenimiento de los transistores y su miniaturización, ha hecho posibles nuevas arquitecturas como los procesadores multi-núcleo actualmente, los cuales pueden ser considerados como un ambiente de computación local de alto desempeño, éste tipo de arquitectura ha permitido continuar con la predicción de la ley de Moore.

2.1.2 Procesadores multi-núcleo

La tecnología multi-núcleo incorpora dos o más núcleos en un procesador, lo que incrementa la capacidad de procesamiento. Según [3] “Con la incorporación de varios núcleos, cada uno es capaz de correr a baja frecuencia, dividiendo entre cada núcleo el poder dado normalmente a uno solo”. La finalidad del diseño es permitir la ejecución de varias tareas simultáneamente y de esa forma alcanzar mayor capacidad de ejecución.

Gracias a los avances en la tecnología digital, hoy en día se cuenta con **procesadores multi-núcleo** en computadoras personales convencionales, con lo cual actualmente se puede tener un ambiente local de alto desempeño. Además, con el uso de este tipo de tecnología en clusters, se proporciona una mayor capacidad de cómputo.

Actualmente se vuelve cada vez más indispensable el uso de la tecnología **multi-núcleo**, ya que día a día se requiere la ejecución de:

- Varias aplicaciones (más complejas) al mismo tiempo.
- Programas que demandan cada vez mayor capacidad de cómputo;
- Aplicaciones visuales, en donde es imprescindible la rapidez en la ejecución de la información, entre otras.

La computación de alto desempeño, como la disponible a través de la tecnología **multi-núcleo**, esta ayudando a cambiar la visión de la computación intensiva, ya no solo es usada en investigaciones, sino que cada día es más común en computadoras personales, en la industria o la educación.

La tecnología multi-núcleo se puede ver como una arquitectura SMP (Shared Memory Multi Processors). Consiste en un número de procesadores idénticos que comparten una memoria global principal, en donde espacios de memoria son compartidos localmente a través de diferentes hilos, sin necesidad de paso de mensajes.

2.1.3 Arquitectura de memoria para procesadores multi-núcleo

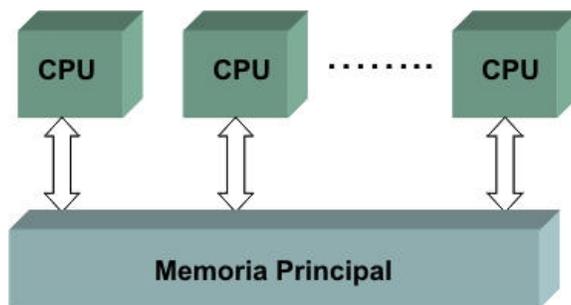


Fig 2.1: Arquitectura SMP (*Shared Memory Multiprocessors*) para procesadores multi-núcleo

La figura 2.1 muestra la arquitectura general del equipo SMP con procesadores multi-núcleo.

La diferencia con respecto a la arquitectura general de los “*mainframes*” y las supercomputadoras [4], es que éstos comparten también el bus de datos, lo que no sucede con los procesadores multi-núcleo.

2.1.4 Características de los procesadores multi-núcleo

Una de las características importantes de los procesadores multi-núcleo es la forma en que se lleva a cabo el manejo y ejecución de la información, que se realizan mediante el uso de “*threads*” o hilos. Los hilos son secuencias de instrucciones relacionadas.

Ya se cuenta con mejoras en sistemas operativos que ofrecen la ejecución de “*multitasking*” o multitarea en un procesador con un solo núcleo, lo cual es posible con la ejecución del programa aparentemente en paralelo, por medio del intercambio rápido de entrada y salida de dicho programa. Sin embargo, el procesador corre todo el tiempo un solo hilo [5]. Posteriormente llegó tecnología “*Hyper-Threading*” (HT), la cual hizo posible la ejecución de dos hilos simultáneamente en el mismo procesador, pero con una limitante: no se puede ejecutar dos programas al mismo tiempo, en la tecnología HT para reconocer un procesador como si fuese dos, el procesador simula dos procesadores lógicos para el sistema operativo.

A diferencia de la tecnología HT, en un equipo multi-núcleo cada uno de los hilos creados en una ejecución, corre en un procesador y comparten memoria entre ellos. Todos los hilos dentro de un proceso comparten las mismas estructuras y datos. Cuando varios hilos hacen uso de los mismos datos, el acceso es controlado por variables de

sincronización, haciendo uso de exclusión mutua, otra forma de sincronización es el uso interno de señales en los hilos [6].

La distribución de trabajo entre los diferentes núcleos se realiza mediante la programación en “paralelo”, para poder aprovechar al máximo la capacidad de cómputo ofrecida por el procesador. Actualmente, uno de los lenguajes utilizados para la creación de programas en “paralelo” con memoria compartida es el OpenMP.

Características de procesadores quad-core y dual quad-core

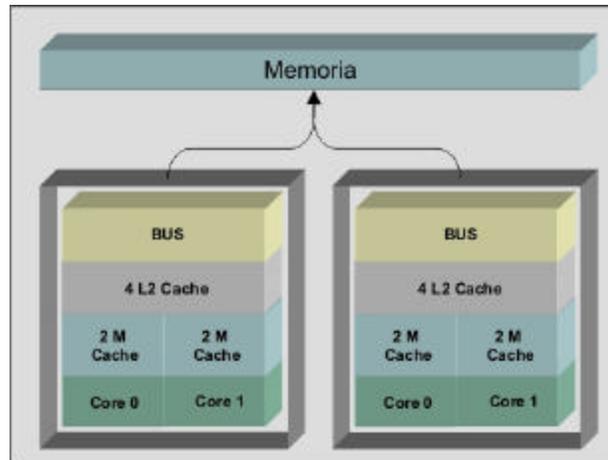


Fig 2.2: Arquitectura del procesador quad-core

Quad-Core Es un procesador multi-núcleo de alto desempeño, basado en tecnología de 45 nm (El manómetro es la millonésima parte de un milímetro) y procesador Intel Xeon. Está compuesto de dos CPU's dual-core. Cada CPU cuenta con dos núcleos, memoria cache de 2M para cada uno de los núcleos, obteniendo así un total de cache L2 de 8M, ésta tiene una función de chache secundaria. En el quad-core además, su bus es 1600 Mhz para las series 5400, mientras que para las serie 5300 su Bus es de 1333 Mhz. Los cuatro CPU's comparten el bus de datos y la memoria RAM [7]. La figura 2.2 presenta su arquitectura.

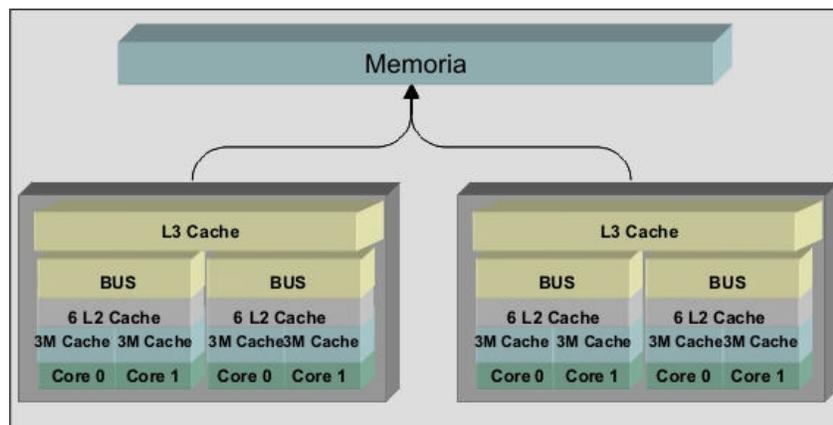


Fig 2.3: Arquitectura de multi-núcleo dual quad-core

Dual Quad-Core Es un procesador multi-núcleo de alto desempeño, también basado en tecnología de 45nm y procesador Intel Xeon. En este caso se compone de dos CPU's quad-core, con un total de 8 núcleos, cada CPU cuenta con un cache L2 de 12M, con lo cual se tiene una cantidad de 3M por núcleo, los bus de datos son de 1600 Mhz. Para la comunicación entre núcleos de un quad-core, se cuenta con una memoria cache L3. De igual forma que la descripción anterior, la memoria RAM entre los Procesadores es compartida. La figura 2.3, muestra su arquitectura.

Actualmente el laboratorio Mac de Matemáticas Computacionales y Bases de datos del departamento de Computación, cuenta en su infraestructura con tecnología multi-núcleo en equipo Macintosh. Se planea en una segunda etapa incorporar equipo con procesadores egith-core y quad-core, con el fin de aumentar la capacidad de cómputo en la infraestructura de HPC actual; prestar nuevos servicios de éste tipo como: ambientes resolvedores de problemas, cálculos que demandan grandes cantidades de cómputo y servicios Web.

Los servicios Web desarrollados para el núcleo de Monte Carlo, así como los servicios para la creación de trayectorias aleatorias por medio de los métodos de Wiener e Ito desarrollados en el presente trabajo de tesis, para la segunda etapa del laboratorio, serán instalados he incorporados en un servidor *quad-core* asignado para servicios de HPC. El núcleo y los métodos de Wiener e Ito serán vistos más adelante en los capitulos 3 y 4.

2.1.5 Impacto de los procesadores multi-núcleo en el desarrollo de software

La llegada de la tecnología **multi-núcleo**, significa múltiples cambios para los desarrolladores [8].

Uno de los cambios marcados se da en el desarrollo de software. Por décadas, la mayor parte de los programadores han estado acostumbrados a pensar y programar en forma secuencial. La habilidad de diseñar y programar en paralelo, es otro tipo de percepción que de igual forma lleva a la solución de problemas reales. La programación en paralelo, hasta el día de hoy, es más conocida en áreas de investigación en donde se requiere de computación intensiva, como por ejemplo el área de Astrofísica, Física Estadística, Economía, etc. Actualmente debe ser desarrollada dicha habilidad por la comunidad del área de la computación, para hacer uso y aprovechar al máximo la tecnología **multi-núcleo**. Este tipo de programación es importante en ambientes de alto desempeño como es el caso.

Otros de los cambios que pueden ser mencionados son los siguientes: la necesidad de reprogramar o crear parches a software ya existente, para que así sea posible su ejecución en “paralelo” y de esa forma aprovechar la capacidad de computo disponible. El desarrollo

de nuevos lenguajes como: Parallel C para la Agencia de Seguridad Nacional (en E.U.) o el X10 en el cual trabaja actualmente IBM [8].

Los programas desarrollados para procesadores multi-núcleo trabajan con hilos, en este tipo de procesadores el paralelismo es simulado. Una de las ventajas que se puede mencionar en el uso de varios núcleos es que se hace posible el diseño y ejecución de un problema en diferentes hilos.

Son algunos de los impactos en el uso de circuitos integrados multi-núcleo.

En la presente tesis, se presenta más adelante en la sección 5.1 se presenta una propuesta del proyecto de tesis, diseñada para el uso de la tecnología multi-núcleo a través de hilos.

2.2 Computación distribuida

Una definición encontrada que describe de la mejor manera posible la *Computación Distribuida* es la siguiente:

La *computación distribuida* se considera “el acto de ejecución de un cálculo en un determinado número de diferentes procesadores, los cuales pueden residir en la misma computadora (como en máquinas con procesadores multi-núcleo), en diferentes computadoras en la misma red (como en el caso de clusters), o en diferentes computadoras ubicadas en diferentes redes, las cuales están físicamente localizadas lejos una de la otra (como en la grid) [9].

De igual forma este paradigma es considerado una infraestructura que permite el uso de recursos como un sistema único.

2.2.1 Antecedentes computación distribuida

Anteriormente los programas estaban enfocados a ser monousuario. Posteriormente, con el desarrollo de equipos cada día más potentes, fue posible tener equipo conectado a otros más grandes, para hacer uso de los recursos centralizados. Los crecientes requerimientos y el progreso en tecnologías tanto de comunicación, como de información tales como: sistemas distribuidos, redes locales LAN (*Local Area Network*), redes WAN (*Wide Area Network*) y el surgimiento de la Internet, son factores que han influido en la descentralización de recursos y la aparición de la *Computación Distribuida*. Con el término “recursos” nos referimos a hardware como:

- Impresoras,
- Pc's, *scanners*

- *Software* como: sistemas, bases de datos, archivos, etc.

Dentro de los ambientes computacionales de alto desempeño, podemos encontrar este paradigma de **Computación Distribuida**. La motivación principal para el surgimiento de éste es el compartir recursos [10]. Además del aprovechamiento de recursos computacionales de computadoras personales, estaciones de trabajo, clusters por medio de la red Internet, otro beneficio de este paradigma es que permite tener acceso a una mayor capacidad de cómputo a un relativamente bajo costo, con respecto a equipos grandes como por ejemplo supercomputadoras.

2.2.2 Características

La Computación Distribuida cuenta con 7 características importantes [10]:

Heterogeneidad. La ejecución y el acceso a servicios se realiza sobre un conjunto heterogéneo de redes y computadoras. La heterogeneidad es aplicada a los siguientes elementos:

- Redes.
- Hardware de computadoras.
- Lenguajes de programación.
- Implementaciones de diferentes desarrolladores

Extensibilidad. La extensibilidad de un programa de cómputo es la característica que determina si el sistema puede ser extendido y re-implementado en diversos aspectos. No es posible obtener extensibilidad a menos que la especificación y la documentación de un sistema estén disponibles para los desarrolladores de *software*. Es decir, que las interfaces clave estén *publicadas*.

Seguridad. La seguridad de los recursos de información tiene tres componentes: confidencialidad (protección contra el descubrimiento por individuos no autorizados); integridad (protección contra la alteración o corrupción); y disponibilidad (protección contra interferencia con los procedimientos de acceso a los recursos).

Escalabilidad. Se dice que un sistema es escalable si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y de usuarios. El diseño de los sistemas distribuidos escalables presenta los siguientes retos: control del coste de los recursos físicos, control de las pérdidas de prestaciones, prevención de desbordamiento de recursos software.

Tratamiento de fallos. Los fallos en un sistema distribuido son parciales; es decir, algunos componentes fallan mientras otros siguen funcionando. Consecuentemente, el tratamiento

de fallos es particularmente difícil; algunos fallos son detectables; el reto está en arreglárselas en presencia de fallos que no pueden detectarse pero que si pueden esperarse.

Concurrencia. Tanto los servicios como las aplicaciones proporcionan recursos que pueden compartirse entre los clientes en un sistema distribuido. Existe por lo tanto una posibilidad de que varios clientes intenten acceder a un recurso compartido a la vez.

Los servicios y aplicaciones permiten, usualmente, procesar concurrentemente múltiples peticiones de los clientes. Suponga que cada recurso se encapsula en un objeto y que las invocaciones se ejecutan en hilos de ejecución concurrentes (*threads*). En este caso es posible que varios "*threads*" estuvieran ejecutando concurrentemente el contenido de un objeto, en cuyo caso las operaciones en el objeto pueden entrar en conflicto entre sí y producir resultados inconsistentes.

Transparencia. Se define como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes en un sistema distribuido, de forma que se perciba el sistema como un todo más que como una colección de componentes independientes.

2.2.3 Diferentes enfoques computación distribuida

Hoy en día, el paradigma de **Computación Distribuida** con respecto a la computación de alto desempeño del inglés "High Performance Computing" (HPC), se presenta mediante alguno de los 3 siguientes aspectos: computación en red NOW, computación en cluster y una computación distribuida de área amplia, como lo es la Grid. En todos los casos, las escalas del sistema están de acuerdo al número de máquinas disponibles [1].

Redes NOW

El paradigma NOW comprende de una red local convencional de computadoras, típicamente estaciones de trabajo, con frecuencia con diferentes arquitecturas, trabajando juntas para el soporte a computación paralela, mejor conocida como computación distribuida. La computación es distribuida a través de la red, de tal forma que las tareas se distribuyen en los nodos según su disponibilidad, con el uso de un sistema de carga balanceada, como por ejemplo: el sistema de software Condor y el LSF (Load Sharing Facility) [1] [5].

Cada estación de trabajo cuenta con su propia memoria y usualmente un sistema de archivos local para compartir archivos comunes tales como directorios de trabajo. Para aprovechar al máximo el poder computacional prestado por este tipo de redes, el modelo de programación MIMD (por sus siglas en inglés Multiple Instruction Multiple Data) es el apropiado, aunque requiere más esfuerzo en programación.

En término de los nodos las redes pueden ser homogéneas o heterogéneas:

Homogéneas. Consiste de estaciones de trabajo con la misma plataforma. En el sistema los procesos usan la misma representación de datos, haciendo más simple su intercambio.

Heterogéneas. Consiste de estaciones de trabajo de la misma plataforma, provee de flexibilidad para combinar recursos, y hace posible el uso de una mayor cantidad de ellos para la resolución de una tarea.

Computación en cluster. Es el uso de múltiples equipos tales como: computadoras (computadoras personales o estaciones de trabajo), equipos de almacenamiento bajo sistemas UNIX o Windows, interconectados a través de una red, el cual simula un único sistema de alto rendimiento. Los clusters son considerados una aproximación a los “*mainframes*”. Debido a la importancia de este tipo de computación en el tema de tesis, será visto con más detalle en la sección 2.3.2.

Computación de área amplia Grid. Ha emergido como una nueva área de la computación distribuida. Es una red de redes y equipo distribuido geográficamente, por medio de la cual es posible compartir recursos entre un conjunto de personas y organizaciones, poniendo a su disposición capacidad de cómputo y almacenamiento. Puede ser usada para cálculos a gran escala. Dentro de la ciencia es usada para dar solución a diferentes problemas en forma colaborativa. Este tema se retoma en la sección 2.4 para una explicación más detallada, ya que forma parte integral del presente trabajo.

2.2.4 Visión laboratorio Mac

En el laboratorio de Matemáticas Computacionales y Bases de datos del departamento de Computación, se planea contar en un corto plazo con la siguiente arquitectura que, prestará un servicio de cómputo distribuido y HPC, sobre la cual en un servido Mac serán instalados los servicios Web desarrollados con respecto al núcleo de Monte Carlo e integrales de trayectoria.

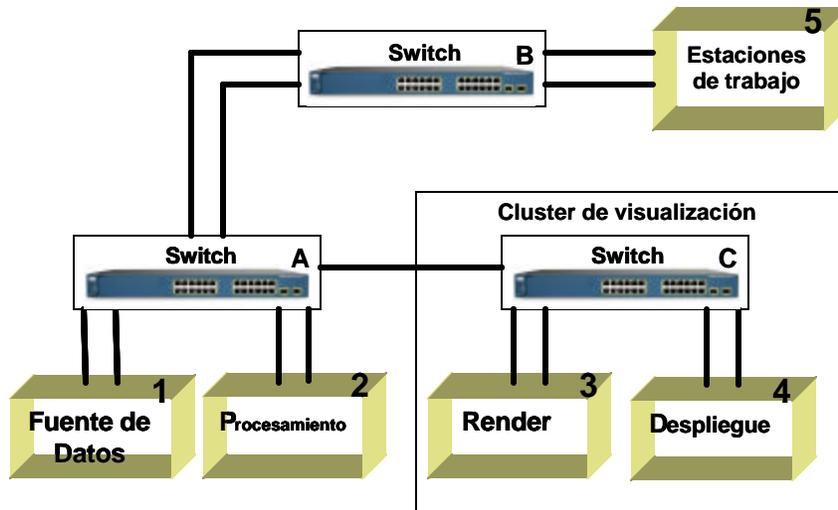


Fig 2.4: Vista General del Laboratorio Mac a Futuro

En la figura 2.4 se muestra una vista general de la estructura que se espera tener en el laboratorio Mac. Contará con una Fuente de Datos (almacenamiento de la información), un Área de Procesamiento de datos, un equipo “Render” (procesamiento de imágenes), un equipo de Despliegue, y un conjunto de Estaciones de Trabajo, todo interconectado a través de concentradores de red.

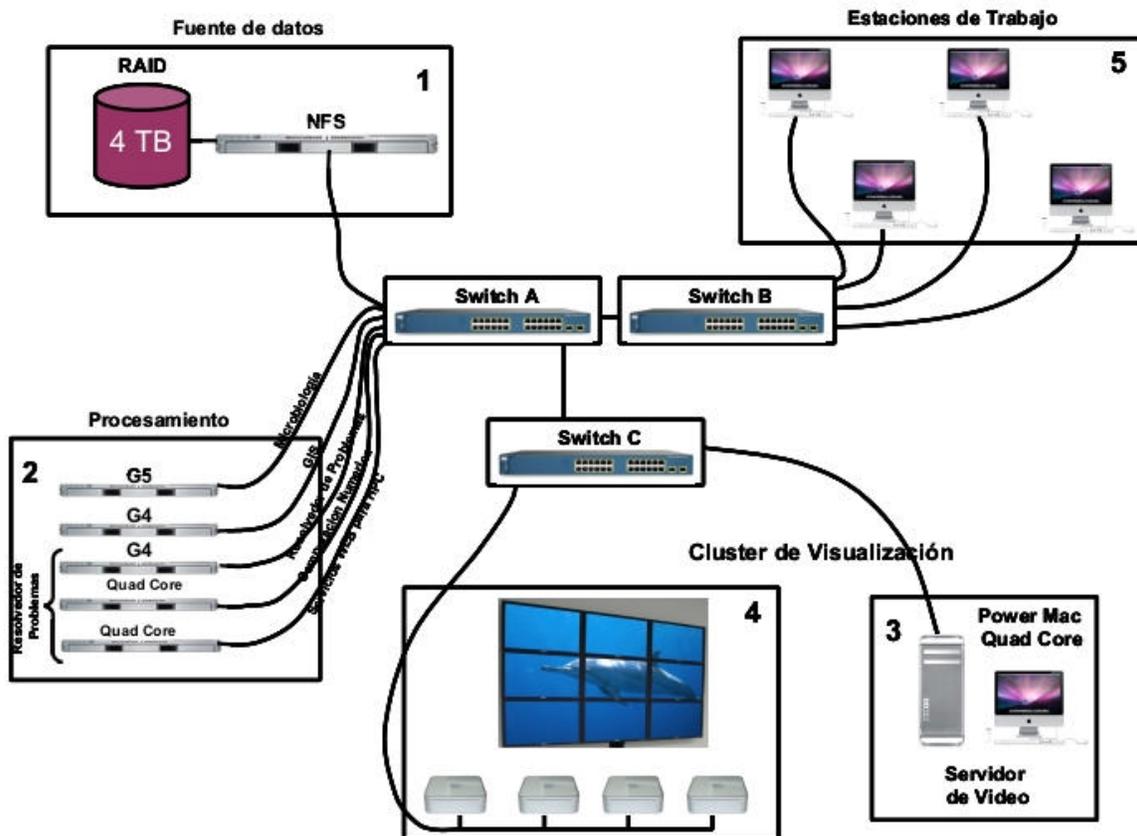


Fig 2.5: Red Laboratorio Mac

La figura 2.5 muestra el esquema de la red planeada a futuro del laboratorio Mac. Va a estar compuesta de tres concentradores de red, los cuales se van a encargar de conectar las 5 diferentes áreas de las cuales se planea va estar compuesto el laboratorio.

Las áreas son descritas a continuación:

1.- **Fuente de datos.** Consta de un RAID con una capacidad de 4 TB, conectado a un servidor, los cuales van a permitir el almacenamiento de la información manejada por los diferentes sistemas desarrollados en el laboratorio Mac.

2.- **Procesamiento.** Consta de un conjunto de servidores Mac los cuales se van a encargar del la ejecución y el procesamiento de los diferentes sistemas y aplicaciones utilizados. Los servidores van estar distribuidos de la siguiente forma:

XServer G5 El cual va albergar la información y el sistema de microbiología.

XServer G4 El cual va albergar los desarrollos relacionados a sistemas GIS (Geographic Information System).

XServer G4 Este servidor se va encargar de albergar los sistemas revolvedores de problemas.

Quad Core Servidor asignado para encargarse de la computación numérica.

Quad Core Servidor asignado para albergar y prestar el servicio de los servicios Web desarrollados para computación de alto desempeño (HPC).

3.- **Render.** Va a constar de un servidor Power Mac Quad Core, el cual va ser el encargado de realizar el procesamiento de las imágenes, para su posterior despliegue.

4.- **Despliegue.** Consta de un conjunto de pantallas, en las cuales se va realizar el despliegue de las imágenes anteriormente procesadas por el **Render**.

5.- **Estaciones de trabajo.** Aquí se engloban las estaciones de trabajo con las que cuenta el laboratorio para uso de los alumnos.

Además se contará con tres concentradores de red. El concentrador de red **C** va ser un concentrador inalámbrico, va ser el encargado de conectar el conjunto de estaciones de trabajo a la red.

2.3 Computación paralela

Otro ambiente computacional de alto desempeño, utilizado desde los años 60`s, es el que comprende a la computación paralela.

Algunas de las definiciones encontradas sobre *Computación Paralela* son las siguientes:

- a) Aplicación de dos o más procesos unidos para resolver un problema. Así un programa puede contener unidades de trabajo paralelas [11].
- b) Es el uso de múltiples recursos computacionales para resolver un problema [13]:
 - El problema es dividido en partes discretas que pueden ser resueltas en forma concurrente.
 - Cada parte es dividida en una serie de instrucciones.
 - Las instrucciones son ejecutadas simultáneamente en múltiples CPU's.

Este tipo de computación permite: aprovechar los recursos computacionales disponibles, lo que conlleva a la disponibilidad de gran capacidad de cómputo a un bajo costo; dividir problemas en módulos más pequeños y disminuir el tiempo de ejecución al solucionar problemas en forma paralela, debido a que sus partes son ejecutadas simultáneamente.

Esquema de un problema en forma secuencial y en forma paralela:

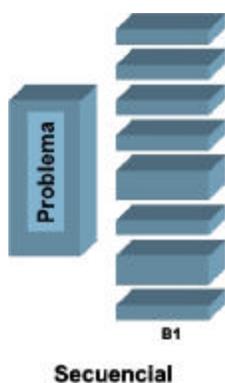


Fig. 2.6: Problema en forma secuencial

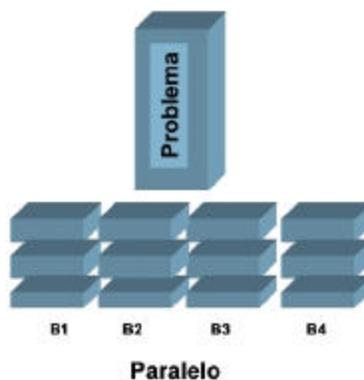


Fig. 2.7: Problema en forma paralela

La figura 2.6 muestra un esquema, el cual representa un problema descompuesto en una serie de bloques de instrucciones. Cuando el problema es programado, el conjunto de bloques de B1 es ejecutado en un solo CPU en forma secuencial.

En cambio, la figura 2.7 muestra el esquema del mismo problema pero en esta ocasión descompuesto en conjuntos de bloques de instrucciones (B1, B2, B3 y B4), los cuales deben ser programados y ejecutados en forma paralela, cada uno en un CPU. El paralelismo radica en la ejecución de dichos bloques en forma simultánea.

2.3.1 Antecedentes de cómputo paralelo

El surgimiento de computadoras, de propósito general con tendencia a aplicaciones científicas, ha sido un parteaguas en la forma de dar solución a grandes problemas en diferentes áreas. Los avances en la tecnología digital, hacen eco en la creación de equipo de cómputo cada vez más potente, lo que ha permitido proponer diferentes arquitecturas como son: los “*mainframes*” en los 60’s y 70’s; las supercomputadoras en los 80’s, con sus diferentes arquitecturas (procesadores vectoriales, arreglo de procesadores, procesadores vectoriales paralelos...) y posteriormente los clusters [1].

Las supercomputadoras y los “*mainframes*” son muy costosos, pocos usuarios o instituciones tienen la oportunidad de contar con algún equipo de éste tipo. Las supercomputadoras cuentan a menudo con miles de procesadores, suelen dedicarse a la ciencia y la milicia, son usadas en complicados cálculos que tienen lugar en la memoria, mientras que los “*mainframes*” cuentan con un conjunto pequeño de procesadores. El limitado paralelismo está escondido al programador, son utilizados para cálculos simples que implican grandes cantidades de datos externos, son dedicados a las empresas y aplicaciones administrativas del gobierno [12].

Algunos de los factores que llevaron al surgimiento del paradigma *Computación Paralela* fueron [11]:

-**Limitaciones físicas.** En algunos momentos en la historia de la computación se han enfrentado problemas que sobrepasan las capacidades del equipo disponible (hablando en términos de: procesamiento, memoria, almacenamiento, velocidad en bus...).

- **Factores económicos.**
- **Escalabilidad,** (Empatar la computadora al tamaño del problema).
- **Mejoramiento de arquitecturas.** (Reflejando el creciente rol de memoria comparado con el poder de procesamiento).

En el paradigma de *Computación Paralela* son referenciados dos enfoques diferentes:

- a) Computación paralela local.
- b) Computación paralela distribuida.

Como computación paralela local pueden ser mencionados los mainframes, las supercomputadoras y hoy en día las computadoras personales multi-núcleo. En la Computación paralela distribuida es posible mencionar los clusters de computadoras. Cada enfoque cuenta con sus propias arquitecturas. Para aprovechar al máximo los recursos computacionales tanto locales como distribuidos, se hace uso de la **Programación en Paralelo**.

El hecho de haber enfocado el **cómputo paralelo** a una red, involucra nuevos factores para un buen desempeño, como por ejemplo: la **latencia** que es el tiempo que tarda el envío de paquetes entre nodos; el **ancho de banda**, mientras mayor sea el ancho de banda se va a obtener una comunicación más rápida, debido a la disminución de la latencia, ya que puede ser enviada mayor cantidad de paquetes sin ningún problema.

Un aspecto importante para nuestro trabajo es el uso del cluster, debido a que parte del proyecto fue desarrollado para su uso y ejecución en un de ellos, para esto a continuación se da una descripción de cluster:

2.3.2 Cluster

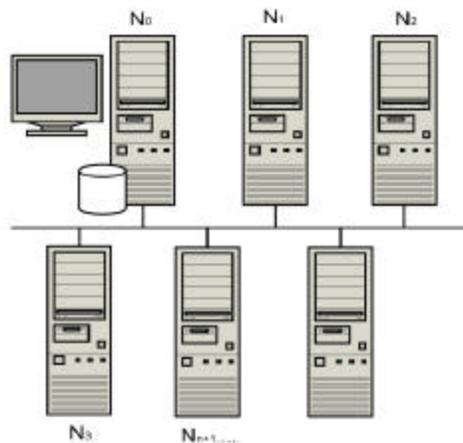


Fig. 2.8: Ejemplo de Cluster

Los clusters son una tecnología que apareció antes que las “grids”, las cuales están enfocadas a resolver problemas dentro de la ciencia, la ingeniería, y el comercio; son utilizadas en problemas que requieren una gran capacidad de cómputo.

Esta tecnología, aparece como resultado de varias tendencias, incluyendo la disponibilidad de microprocesadores, el desarrollo de herramientas de software estándar

para grandes ejecuciones; tanto para la ciencia de la computación, el cómputo científico y aplicaciones comerciales [13].

Muchos de los clusters disponibles hoy en día, son heterogéneos en cuanto a arquitectura se trata. El cluster puede estar compuesto de componentes de hardware y software. Los componentes de hardware pueden incluir computadoras y redes. Los nodos de los que se compone pueden ser: computadoras personales, estaciones de trabajo y SMP (Symmetric Multiprocessors). Las redes son usadas para la interconexión de los nodos, pueden ser redes de área local. Varios sistemas operativos como: Linux, Solaris, y Windows pueden ser usados para el manejo de recursos. La comunicación esta basada en el protocolo TCP/IP [13]. El esquema de un cluster es presentado en la figura 2.8.

Los “clusters” son usados en muchas disciplinas incluyendo biología (mapeo de genomas, manejo de proteínas), ingeniería (diseño de automóviles), física (simulaciones, armas nucleares), astrofísica (simulación de galaxias), y meteorología (simulación de climas), entre otras.

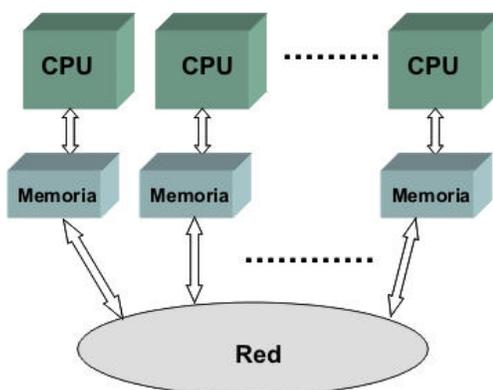


Fig. 2.9: MPP (*Massively Parallel Processors*)

La figura 2.9 muestra la arquitectura general del cluster utilizado para el desarrollo del trabajo de tesis en cuanto a memoria se refiere. La arquitectura utilizada es una arquitectura de multi-procesadores con memoria distribuida, siendo necesaria su comunicación a través de paso de mensajes.

En este tipo de arquitectura, no se tiene acceso directo a memoria global por parte de los procesadores. Los procesadores se encuentran interconectados por medio de una red. Cada uno cuenta con su memoria local y no hay memoria compartida disponible. Es utilizado un mayor paralelismo, además, se puede tener a disposición cientos y en ocasiones miles de procesadores, debido a que es escalable. Estas arquitecturas pueden ser vistas como una gráfica, donde cada vértice representa un nodo, y cada línea de comunicación es

representada por una arista, lo cual en teoría de grafos puede ser visto de la siguiente forma: $G=(V,L)$ cada nodo $\in V$ y cada par $e = (j,i) \in E$, siendo E el conjunto de aristas. Este tipo de arquitectura es llamada MPP (*Massively Paralle Processors*) [6] [14]. La idea de la arquitectura MPP es normalmente el resultado del compromiso entre el costo y la calidad de la red de comunicación.

En esta arquitectura los procesos son paralelos, cada uno corre en un procesador diferente, no se comparte memoria principal con otros procesos y se usa el paso de mensajes para la comunicación, con el fin de coordinar el trabajo [6].

2.3.3 Paso de mensajes

Los mensajes pueden ser usados en los siguientes casos: cuando un procesador utilice algún dato que haya sido calculado en otro procesador, cuando sea necesario sincronizar el trabajo entre los procesadores. El *paso de mensajes* es similar al envío de correo electrónico ya que contiene información sobre quien lo envía y hacia quien va dirigido. Hay dos formas de paso de mensajes, una es *asíncrona* y la otra es *síncrona*. Con la *asíncrona* los canales de comunicación se comportan parecidos a semáforos, en la *síncrona* es utilizado un mecanismo de exclusión mutua. El uso de paso de mensajes, es debido, a que en un modelo de memoria distribuida no es posible el intercambio de información, a través de variables compartidas.

El paradigma de paso de mensajes ha sido implementado en diferentes bibliotecas como son PARMACS, Chameleon, CHIMP, PCL, MPI (Message Passing Interface), PVM (*Parallel Virtual Machin*)... Siendo las dos últimas las más populares, difieren cada una en detalles en la implementación, pero el modelo básico es el mismo [6]. Estas bibliotecas van a ser vistas con un poco más de detalle en la siguiente sección.

2.3.4 Tipos de programación

OpenMP

Corresponde a un conjunto de directivas que permite el uso de hilos en la programación, con el fin de distribuir el trabajo en diferentes procesadores y reducir con ello el tiempo de ejecución de un programa. El conjunto de directivas es una extensión de C, C++ y Fortran, pueden ser usadas bajo plataformas como UNIX y Windows [12]. Es utilizado en aplicaciones de HPC y en arquitecturas de multiprocesadores de memoria compartida. Actualmente en los procesadores multi-núcleo puede ser explotado el paralelismo a nivel de hilos eficientemente con el uso de *OpenMP*.

Modo de ejecución. Un programa en OpenMP comienza su ejecución como un simple proceso, llamado hilo maestro. Una directiva define una región del programa en paralelo. Cuando el hilo principal o maestro entra a la región paralela del programa, genera un conjunto de n hilos y la ejecución del programa continua en forma paralela a través de los múltiples hilos. Los hilos se sincronizan y al final de la región el único que continua con la ejecución es el hilo maestro. Se cuenta con otro conjunto de directivas que son las de compartir trabajo (*Work Sharing Directives*), permiten dividir el trabajo entre los hilos, por ejemplo la directiva “for” especifica que las iteraciones relacionadas deben ser divididas, posteriormente cada iteración es ejecutada por un hilo[15].

MPI

No es completamente un ambiente de programación. Como sus siglas lo dicen: *Message Passing Interface (MPI)*, es una interfaz de paso de mensaje, la cual se compone por un conjunto de directivas. MPI permite la comunicación entre redes homogéneas. Su portabilidad radica en que: los programas pueden ser compilados y posteriormente ejecutados en diferentes arquitecturas. Provee dos tipos de comunicación punto-a-punto y operaciones colectivas. La punto-a-punto envuelve dos procesos: uno que se encarga de enviar un mensaje y el otro de recibirlo. Las operaciones colectivas tal como la sincronización con un *barrier*, como el envío de información por medio de un *broadcast*, envuelven un grupo de procesos [6]. No cuenta con el concepto de Máquina Virtual.

PVM

Fue desarrollada bajo el concepto de máquina virtual. El concepto de máquina virtual para la perspectiva de PVM provee las bases para la heterogeneidad, portabilidad y encapsulamiento de funciones. Algunas de las características se describen a continuación [16]:

- Hace posible que, un conjunto de computadoras heterogéneas, aparezcan lógicamente para el usuario como una sola, donde es posible la ejecución de un programa paralelo y el intercambio de información.
- Posee una buena interoperabilidad y un conjunto de manejo de recurso dinámico y funciones de control de procesos.
- Intercambio de información entre tareas paralelas a través del paso de mensajes.
- Transparencia de utilizar memoria compartida en redes de alta velocidad para mover datos entre clusters de memoria compartida de multiprocesadores.

- Portabilidad ya que los programas pueden ser copiados a diferentes arquitecturas y ejecutados sin ninguna modificación. Un programa en C puede enviar un mensaje que es recibido por un programa en Fortran y viceversa.

2.3.5 Laboratorio cluster

Como se puede observar, con el paso del tiempo y el desarrollo de la ciencia de la computación, van surgiendo nuevos requerimientos y con ello surgen nuevos paradigmas computacionales.

La computación paralela, en especial, su uso en un cluster, es de importancia en el trabajo de tesis, debido a que parte del trabajo fue desarrollado en uno, el cual se encuentra en el departamento de Computación. Para la programación en paralelo fue utilizada la biblioteca MPI junto con el lenguaje C. A continuación, se presenta la arquitectura del laboratorio de cluster.

Fue considerada la plataforma como la combinación del hardware, y del sistema operativo. Esto se muestra en la figura 2.10.

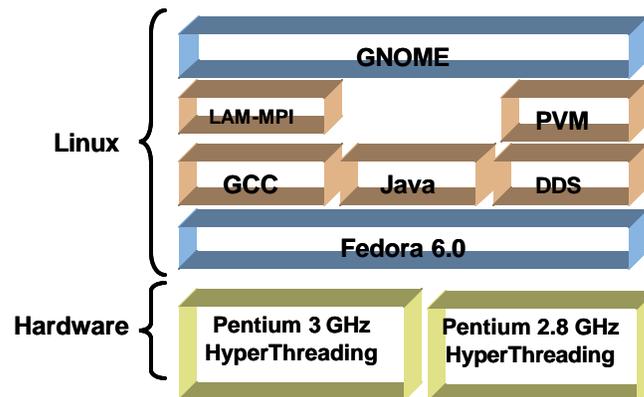


Fig. 2.10: Plataforma del cluster (software)

2.3.6 Ambientes de desarrollo

LAM-MPI. Es un ambiente de código abierto, el cual cuenta la implementación de las especificaciones del paradigma MPI (Message Passing Interface). LAM/MPI incluye un rico conjunto de características para administradores del sistema y también para investigaciones de computación paralela. Está diseñado para operar en *cluster's* heterogéneos [17], ayuda al desarrollo y depuración de programas en paralelo.

PVM (Parallel Virtual Machine). Es un paquete de código abierto, el cual ofrece un ambiente propicio para poder trabajar a través de una red, con una colección de

computadoras tanto Windows como UNIX. Con el **PVM** un conjunto de computadoras pueden ser usadas como una sola computadora paralela. Permite también a los usuarios explotar el hardware existente para resolver grandes problemas a un costo menor, es portable [18] y permite la creación de programas en paralelo.

GCC. Es un compilador de software libre para el lenguaje C, comúnmente utilizado en ambientes UNIX, su ejecución se realiza por medio de línea de comandos.

Fedora. Es un sistema operativo basado en Linux, el cual incluye lo último en software libre y de código abierto.

JAVA. Compilador que permite el desarrollo de aplicaciones orientadas a objetos.

GNOME. Ofrece un ambiente gráfico de trabajo para plataformas Linux o UNIX, es amigable para el usuario, contiene tecnología de componentes transparente a la red utilizando CORBA, extenso uso de XML, con uno de los más avanzados modelos de imágenes en cualquier plataforma. Está implementado en C lo que lo hace extremadamente eficiente, rápido, ligero y portable [14].

2.3.7 El hardware

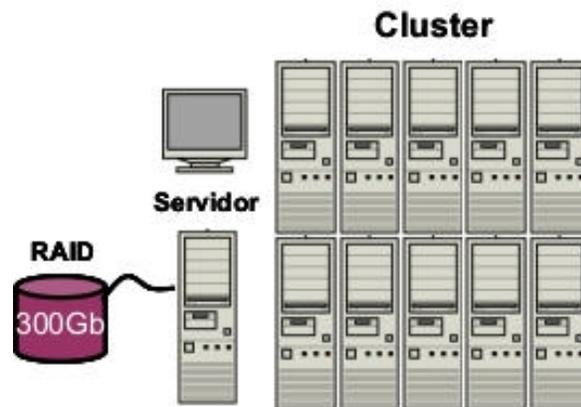


Fig 2.11: Arquitectura laboratorio cluster

En la figura 2.11 se muestra el esquema de la arquitectura del Laboratorio de Cluster. Cuenta con un conjunto de 32 máquinas Hyperthreading, Pentium de 3 GHz, con 1G en memoria RAM y un tamaño de palabra de 64 bits. Además de un servidor pentium de 2.8 GHz, con memoria RAM de 1G, en el cual se lleva acabo el control de acceso al cluster. Para el almacenamiento se tiene un RAID de 300 Gbytes.

2.4 Grid (Computación distribuida de área amplia)

La Grid es considerada una infraestructura de hardware y software que provee confiabilidad, un eficiente método de disponibilidad y coordinación de recursos de alta calidad y revolvedores. El uso de recursos compartidos es posible a través de una interfaz común, haciendo posible la conexión de dichos recursos, como por ejemplo: clusters y supercomputadoras, éstos distribuidos geográficamente. Sin embargo, el concepto Grid va más allá de compartir recursos en un ambiente distribuido, incluye acceso a información como: grandes bases de datos; instrumentos científicos tal como: repositorios, software de visualización [19] [20] [21]. La Grid es actualmente una herramienta poderosa que permite la colaboración entre individuos con un fin común.

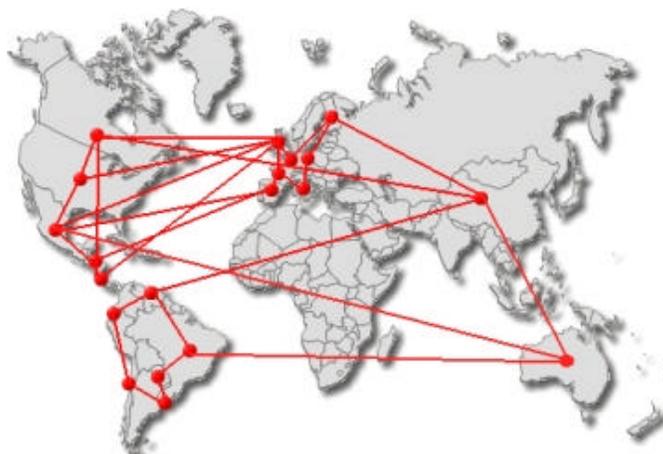


Fig. 2.12: Esquema de una Grid

La figura 2.12 muestra un esquema de una Grid. Se puede observar una red separada geográficamente, en donde cada nodo que compone la red puede ser: otra red, un cluster, supercomputadoras, una computadora personal, o una red hecha en base a computadoras con procesadores multi-núcleo.

La Grid es una herramienta para dar solución a los requerimientos de una relativamente nueva ciencia llamada “System Level Science” [43]. Esta ciencia requiere la creación de algún sistema que combine tanto recursos físicos como humanos. El fin de esta ciencia es hacer posible la integración de varias disciplinas, sistemas de software, datos, recursos computacionales y factor humano.

La “System Level Science” es una forma de investigación científica, en donde el objetivo no se limita solo al conocimiento a fondo de un fenómeno físico. Intenta entender el comportamiento complejo de un sistema: de que forma puede ser descompuesto en componentes modulares, la forma de interactuar entre los componentes y su interconexión.

En varios sistemas, los alcances computacionales no son suficientes, van más allá de la capacidad de recursos que están al alcance de los científicos o de las instituciones. Conocida ésta necesidad se presenta un problema computacional de alto rendimiento, en donde para poder resolver esa situación, fue necesario pensar en una solución alterna, como lo es la internet, la Grid y cyber-infraestructuras. De aquí nacen las Organizaciones Virtuales (OV), las cuales están compuestas por un conjunto de individuos, instituciones y recursos unidos por alguna tarea o interés.

Las organizaciones se encargan de asignar los atributos, roles y políticas que aplican a los participantes de la misma. Ahora, es posible observar proyectos de infraestructura de producción a gran escala, así como proyectos científicos organizados alrededor de conceptos y mecanismos de las OV.

2.4.1 Uso de la Grid

La Grid es un paso muy importante para el desarrollo de la ciencia de la computación y la ingeniería. Su uso comenzó en el área de la investigación, aunque actualmente se extiende a otras áreas como los son la industria y el comercio. El enfoque que se da en este trabajo de tesis es sobre la investigación. El uso de la Grid tiene un papel muy importante en el área de la investigación. Su infraestructura ha sido usada para analizar, simular y solucionar problemas científicos que anteriormente se consideraban imposibles de resolver, debido a la capacidad de cómputo que requerían, como por ejemplo: el estudio de sistemas que describen el comportamiento de los terremotos, con la finalidad de realizar simulaciones, para que algún día puedan ser predecibles; o la predicción de fenómenos meteorológicos.

La Grid y las OV son una parte muy importante para solucionar el problema de la demanda de recursos, cooperación y coordinación de proyectos de investigación, para compartir información entre científicos e instituciones, con el fin de contribuir al desarrollo de la ciencia en varias disciplinas, así como mejorar la calidad de sus investigaciones.

Con el creciente desarrollo de la Grid, se espera llegar a tener una gran colaboración entre diferentes instituciones y organizaciones, en la investigación.

2.4.2 La Grid y Servicios Web

Cuando se habla de computación científica, se viene a la mente un laboratorio con equipo centralizado corriendo programas desarrollados en Fortran, C y en otras épocas Q-Basic.

Actualmente el área de la investigación, junto con la “System-level Science”, han aprovechado los beneficios de tecnologías como: servicios Web, sistemas distribuidos, nuevas formas de interconexión de equipo, así como nuevos paradigmas como la Grid y la Internet; estas tecnologías han permitido un enfoque dirigido hacia arquitecturas orientadas a servicios. Pero a pesar de eso, no se ha dejado de aprovechar el uso de métodos convencionales; tampoco se han dejado de mejorar dichos métodos, como lo son: los “clusters”, supercomputadoras, estaciones de trabajo, ya que actualmente pueden trabajar como parte de la Grid o en forma independiente.

Lo anterior permite que la investigación obtenga un panorama más amplio sobre la forma de atacar, analizar y resolver problemas dentro de diferentes áreas, teniendo a su alcance y haciendo uso de recursos y tecnologías a las que anteriormente no se tenía acceso o se limitaba solo a algunos campos.

Con la combinación “Sistem Level Science”- Servicios Web (SW)- Grid o Servicios Web – Grid, científicos interesados en el estudio de fenómenos aislados, pueden enfocarse en dar una solución en niveles o componentes atómicos, programarlos y publicarlos como servicios.

La ventaja que se obtiene con el uso de servicios Web es que pueden ser manejados de forma independiente a la plataforma, son desarrollados con un lenguaje descriptor y publicados con un proveedor de servicios, con el fin de poder hacer uso de ellos a través de Internet. Ya publicados, científicos interesados en la composición de sistemas por niveles, pueden concentrarse en la integración de dichos servicios, para modelar el comportamiento de sistemas diferentes, en base a servicios que originalmente formaban parte de otro sistema.

Para la localización de un conjunto de servicios, existe lo que se le llama un solicitador de servicios; éste es capaz de localizar un conjunto de los mismos, a través de una solicitud. Un solicitador es la forma en que varios proyectos llevan a cabo la búsqueda de los SW disponibles. Para hacer posible la localización, es necesario que los servicios estén registrados en un catalogo independiente llamado UDDI (Universal Description Discovery and Integration), el cual es similar a una página amarilla.

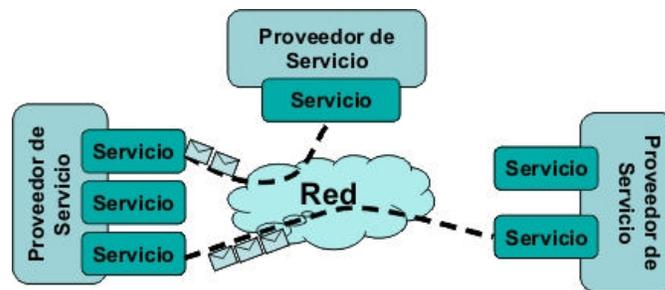


Fig. 2.13 : Esquema de comunicación entre servicios

La forma de comunicación entre los servicios Web es por medio de mensajes. El uso de mensajes permite tener servicios publicados, con proveedores en dominios diferentes y separados geográficamente. El protocolo SOAP (Simple Object Access Protocol) hace posible el manejo de mensajes en los servicios Web. Además de ese protocolo, otra parte importante para el manejo de servicios es el lenguaje descriptor, WSDL (Web Service Definition Language). En la figura 2.13, se muestra un esquema de comunicación, entre servicios. Un conjunto de servicios Web están localizados en diferentes dominios. Para la comunicación entre ellos se envía un mensaje, el cual pasa por la red Internet y llega a su destino. En caso de ser necesario que el servicio receptor emita una respuesta, ésta es enviada a través de otro mensaje. El paso de mensajes es una base de la Grid.

Un aspecto del uso de servicios Web dentro de la Grid, son los sistemas orientados a servicios. Estos sistemas están diseñados para conjuntar su uso y el uso de recursos disponibles a través de Internet, lo que hace posible la composición de sistemas potencialmente distribuidos, en una forma relativamente fácil el uso de nuevos servicios Web con los ya existentes [21]. Algunos de los sistemas que permiten esta forma de desarrollo son Kepler [22] [23], Triana [24] y Taverna [25].

2.4.3 Arquitectura Grid

Laszewski identifica un conjunto de capas fundamentales con las que realiza una arquitectura lógica de la Grid [25] de la siguiente forma:

Las capas identificadas son:

La capa de fabricación. Es la capa que contiene interfaces de aplicaciones, protocolos y herramientas que permiten el desarrollo de servicios y componentes.

La capa de conectividad y recursos. Incluye el núcleo necesario para la comunicación y autenticación, para la ejecución de transacciones seguras en la red. Esta capa incluye protocolos y servicios permitiendo intercambio de mensajes seguros y autorización.

La capa colectiva. Es la que concierne a la coordinación de múltiples recursos y define colecciones de recursos que son parte de una organización virtual. Por ejemplo los directorios de recursos, los planificadores de trabajos [12].

La capa de aplicación. Aplicaciones que son usadas dentro de una organización virtual.

Cada una de las capas contiene sus correspondientes protocolos, API's y su SDK (Software Development Kit).

2.4.4 Ventajas y desventajas del uso de la Grid

Ventajas que pueden obtenerse del uso de la Grid serían:

- Es posible integrar diferentes tipos de equipos y recursos, lo que hace posible que una Grid nunca quede obsoleta, debido a que es posible agregar equipo nuevo y más potente, sin dejar de hacer uso de los recursos del equipo con el que ya se contaba.
- Se puede tener al alcance gran capacidad de computo a un bajo costo. Así instituciones o centros de investigación, que no cuenten con los recursos necesarios para la obtención de una supercomputadora o simplemente equipo para la creación de un “cluster”, puede contar con una conexión a Internet y hacer uso de los recursos disponibles por medio de la Grid, a través de las organizaciones que se encargan de su desempeño y administración.
- Hace posible el intercambio, acceso y gestión de información entre diferentes instituciones, centros de investigación o científicos interesados en diferentes áreas y temas, lo que hace posible la colaboración de personas, organizaciones e instituciones a través de la Grid.
- La Grid con un enfoque en servicios Web, permite que personas y científicos que no están involucrados en la parte de programación, solo se concentren en ordenar el flujo de tareas que debe realizar un problema en específico, haciendo posible la creación de nuevas aplicaciones y nuevos servicios Web.

Las desventajas que se pueden presentar con el uso de la Grid realmente son pocas:

- No se tiene el control sobre llevar a buen término la ejecución de los servicios disponibles.
- No se tiene el control de la seguridad de la red que se está haciendo uso.
- La seguridad con la cuenta la Grid es la misma con la que cuentan las redes locales de las que está compuesta.

La computación de alto desempeño es uno de nuestros enfoques en el presente trabajo, debido a que se presenta un diseño de integrales de trayectoria y del núcleo de Monte Carlo en forma de hilos para un equipo multi-procesadores con memoria compartida. Así como también es usada la computación paralela, ya que el desarrollo fue echo en MPI para trabajar en forma distribuida y paralela. Asimismo se hace uso de la computación de red amplia, con el desarrollo del mismo trabajo en servicios web, para uso en Taverna, “*middleware*” que trabaja a través de una red Grid.

Capítulo 3

El método de Monte Carlo

El método de Monte Carlo, actualmente es considerado una herramienta estadística muy importante en el área de la investigación. No pertenece a un área específica, a pesar que sus inicios fueron dentro del área de la física. Posteriormente su aplicación es también en diferentes campos tales como: química, economía, bioinformática, por mencionar algunos. Debido a su base estadística lo hace un método muy flexible.

En este capítulo se presenta una breve descripción de la naturaleza del método, el conjunto de algoritmos para la generación de números pseudo-aleatorios, de los cuales se compone el núcleo desarrollado. Asimismo se presentan dos ejemplos de simulación en donde el método puede ser utilizado.

3.1 Naturaleza del método de Monte Carlo

La parte medular del método es la generación de números aleatorios. Se compone de un conjunto de métodos aparentemente simples, a través de los cuales es posible crear muestras con un determinado comportamiento, el cual se define por la función de distribución empleada para la generación de dichos números. La muestra creada pertenece a una población ficticia. La aproximación a resultados reales depende de la exactitud de los valores, el número de ensayos y la varianza de la muestra [1]. Los métodos de aceptación y rechazo, y la reducción de varianza son elementos importantes para la eficiencia del método en la solución de los problemas.

El desarrollo teórico se inicio en 1940, el método se utilizaba de forma implícita en el siglo XIX, nació antes de la llegada de la computadora, debido a esto en un tiempo se encontraron problemas que no fueron posibles de resolver manualmente, por la cantidad de operaciones necesarias y el tiempo que implicaban. Con el “boom” de la computación, el método renace con nuevos enfoques, uno es el uso del equipo de computo; el uso de métodos matemáticos para la generación de números aleatorios, dichos métodos fueron desarrollados en base a las diferentes funciones de probabilidad, a este tipo de número aleatorio se le dió el nombre de pseudo-aleatorio, debido a que se basan en un modelo matemático como ya fue mencionado, debido a esto es posible volver a generar en algún momento la misma secuencia de números.

La aparición de la computadora fue un factor muy importante para la aplicación del método. Las computadoras aumentaron enormemente el número y la fiabilidad de las operaciones aritméticas, que pueden ser ejecutadas en un experimento numérico. Actualmente estas son una herramienta muy importante para la aplicación del método de Monte Carlo, ya que con su llegada, la aplicación del muestreo estadístico para la investigación de problemas es una realidad [27]. Además los procesos para la solución de un problema por medio del método son más rápidos.

Stanislaw Marcin Ulam es el responsable de la reinención del método estadístico, fue el de la idea de calcular el efecto promedio de una frecuencia de un proceso repetido varias veces a través de la simulación en una computadora digital. Los tipos de problemas que pueden ser abordados por medio del método de Monte Carlo son: estocásticos y deterministas, según a si son o no afectados por el comportamiento y resultado de un proceso aleatorio. De acuerdo a un problema estocástico, la forma más simple del método que puede ser aplicada es la generación de números aleatorios aplicados a dicho problema. El tipo de distribución de los números aleatorios generados puede ser seleccionado en base a dos formas: una es la selección directa de cualquiera de los comportamientos, la otra consiste en seleccionar la distribución en base a observaciones o estudios del comportamiento físico del problema, realizados previamente. En el caso de un problema determinista, éste puede ser resuelto numéricamente por Monte Carlo [27].

Existe una técnica de resolver un problema por la simulación de otro. Dicha técnica es llamada Monte Carlo sofisticado: se comienza con la formulación teórica de un problema, se deriva un segundo, descrito por el resultado teórico, y finalmente es resuelto el primero en base a la simulación de este último [27].

Hay varias razones para utilizar el método Monte Carlo, la principal es por su naturaleza estadística, lo que permite realizar inferencias acerca de un problema, en base a observaciones realizadas. Las observaciones pueden ser más o menos representativas del problema. Es posible obtener resultados aceptables de buenos experimentos y buenas inferencias. Los resultados obtenidos mediante el uso del método de Monte Carlo no son exactos, porque las observaciones se hacen en base a números aleatorios, de aquí la importancia de reducción de varianza. El objetivo es hacer varias ejecuciones del problema y obtener una varianza lo más pequeña posible; entre más pequeña sea, el error disminuye y los resultados se apegan más a la realidad del problema en estudio.

Según el tipo de problema varias técnicas fueron desarrolladas en base al método de Monte Carlo: de muestreo simple, importancia de muestreo y secuencia, entre otros. Una de las técnicas de nuestro interés es la de la generación de camino aleatorio, ya que se trabajó sobre la generación de trayectorias por medio del uso de número aleatorios aplicado a tres

métodos para la generación de trayectorias, los cuales serán descritos en el siguiente capítulo.

3.1.1 Generación de camino aleatorio

El camino aleatorio representa una gran clase de problemas, cuya base es el desarrollo de trayectorias conectadas mediante el eslabonamiento de nuevas ligas, que cada vez se generan de manera aleatoria. Según la malla que sirve de movimiento [1]. El modelo de camino aleatorio esta íntimamente relacionado con la teoría de movimiento browniano, el cual considera un movimiento irregular de partículas que se encuentran suspendidas en un líquido, considerando la relación de este en su difusión [Einstein1956].

3.2 Distribuciones continuas

Una distribución continua puede tomar cualquier valor en un intervalo, no solo un número determinado de valores. Por ejemplo en un rango de 0 a 1, se pueden tomar valores como 0.34, 0.5784, 0.976543...

A continuación, se presentan los algoritmos para la generación de números aleatorios continuos, que forman parte del núcleo de Monte Carlo desarrollado en el trabajo de tesis. El desarrollo de dichos algoritmos se basa en las Funciones de Distribución de Probabilidad (F.D.P.) continuas, presentadas en esta sección, y un conjunto de FDP discretas, las cuales son descritas en la siguiente sección.

3.2.1 Uniforme continua

La Función de Probabilidad (F.D.P.) uniforme es la siguiente: $1/(b - a)$.

Para la generación de variables aleatorias con distribución uniforme, con valores entre 0 y 1, fue tomada la formula de la tabla 3.1 del libro de [26], a la cual previamente le fue aplicado el método de la transformada inversa [28]. A la formula antes mencionada se le aplico el algoritmo ITM tomado de [26].

Propósito: Generar variable aleatoria con comportamiento Uniforme

Z aleatorio que $\{F(z), a \leq z \leq b\}$ y $\{F^{-1}(u), 0 \leq u \leq 1\}$

Salida: Z

Entrada: alpha,beta

Fuente: Fishman(1999).

Método:

- 1 Generar U de ran3(idum)
- 2 Calcular $Z \leftarrow \text{alpha} + (\text{beta} - \text{alpha}) * U$
- 3 Si $Z > 0$; $Z = Z - 1$
- 4 Regresar Z

Fig 3.1: Algoritmo ITM

3.2.2 Exponencial

Una variable exponencial Z tiene una F.D.P.:

$$f_x(x) = \begin{cases} \frac{1}{b} e^{-x/b}, & 0 \leq x \leq \infty, b > 0 \\ 0 & \text{En otro caso} \end{cases} \quad (3.1.1)$$

denotada por d_exponencial(beta).

Por medio de la transformada inversa se tiene $U = F_x(x) = 1 - e^{-x/b}$ y

$$Z = -b \ln(1-U) \quad (3.1.2)$$

mientras 1-U es distribuido de la misma forma que U, se tiene:

$$Z = -b \ln U \quad (3.1.3)$$

El algoritmo utilizado es el siguiente:

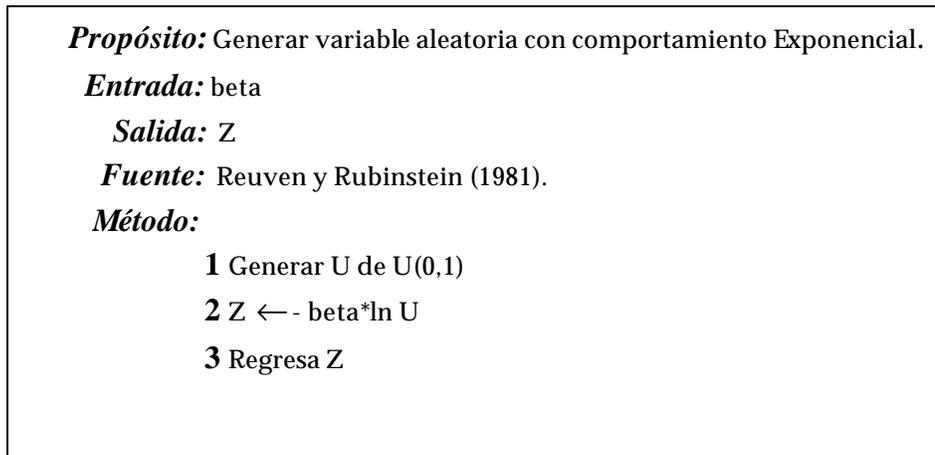


Fig 3.2 Algoritmo E-1

3.2.3 Gamma

Una variable aleatoria Z, tiene una distribución gamma si su F.D.P. se define como:

$$f_x(x) = \begin{cases} \frac{x^{a-1} e^{-x/b}}{b^a \Gamma(a)}, & 0 \leq x \leq \infty, a > 0, b > 0 \\ 0 & \text{En otro caso} \end{cases}$$

y dentro del núcleo se denota por d_gamma(alpha,beta). Hay que notar que para $a = 1$, d_gamma(1,beta) es d_exponencial(beta). El método de la transformada inversa no es aplicado en este caso. Para más detalle ver [28].

El procedimiento: permitir que $X_i, i=1, \dots, n$ sea una secuencia de variables aleatorias independientes de $d_gamma(alpha_i, beta)$. Entonces $X = \sum_{i=1}^n X_i$ es de $d_gamma(alpha, beta)$ donde $alpha = \sum_{i=1}^n alpha_i$. Si $alpha$ es un entero, como es el caso en la variable $alpha$ utilizada para el algoritmo. Si se tiene que $alpha = m$, se obtienen variables aleatorias de la distribución $d_gamma(m, beta)$, con la suma de m variables aleatorias de tipo exponencial obtenidas de $d_exponencial(1)$, esto es:

$$X = beta \sum_{i=1}^m (-\ln U_i) = -beta \ln \prod_{i=1}^m U_i \quad (3.1.4)$$

la cual es llamada distribución de Erlang.

Propósito: Generar variable aleatoria con comportamiento Gamma.

Entrada: alpha, beta

Salida: Z

Fuente: Reuven y Rubinstein (1981)

Método:

- 1** $Z \leftarrow 0$
- 2** Generar V de $d_exponencial(1)$
- 3** $Z \leftarrow Z + V$
- 4** If $alpha = 1$, $Z \leftarrow beta * Z$ y regresar Z
- 5** $alpha \leftarrow alpha - 1$
- 6** Ir al paso 2.

Fig 3.3: Algoritmo G-1

3.2.4 Beta

Una variable aleatoria X tiene una distribución beta si su F.D.P. es:

$$f_x(x) = \frac{\Gamma(\mathbf{a} + \mathbf{b})}{\Gamma(\mathbf{a})\Gamma(\mathbf{b})} x^{\mathbf{a}-1} (1-x)^{\mathbf{b}-1}, \quad \mathbf{a} > 0, \mathbf{b} > 0, 0 \leq x \leq 1 \quad (3.1.5)$$

y se denota por $d_beta(alpha, beta)$, el cual fue tomado del algoritmo Be-1. El procedimiento está basado en el resultado de la sección 3.5.2 de Reuven y Rubinstein [28], que dice: Si Y_1 y Y_2 son variables aleatorias independientes cada una obtenidas respectivamente de $d_gamma(alpha, 1)$ y $d_gamma(beta, 1)$, entonces:

$$X = \frac{Y_1}{Y_1 + Y_2} \quad (3.1.6)$$

de esta formula se obtiene una variable aleatoria con distribución beta.

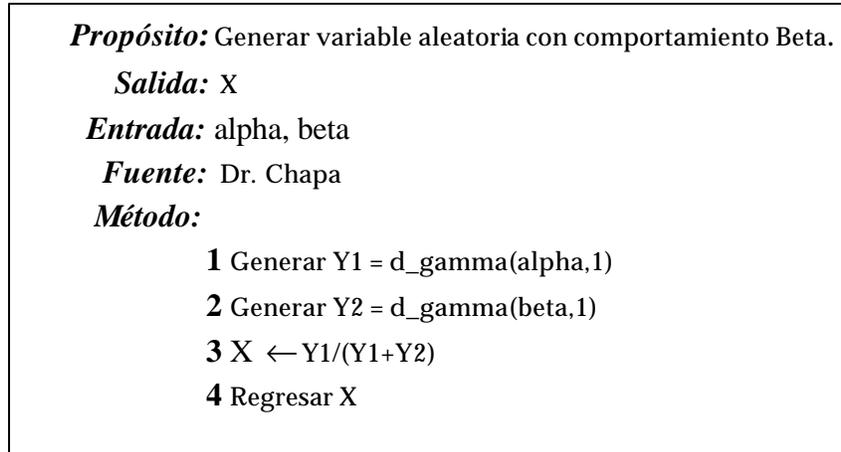


Fig 3.4: Algoritmo Be-1

3.2.5 Normal ó gaussiana

La distribución normal es la más importante y de mayor uso en cuanto a distribuciones continuas se refiere. Lo anterior es debido a que muchos de los eventos físicos estudiados estadísticamente, al momento de obtener muestras de ellos, el comportamiento de dichas muestras tiende hacia una distribución normal conforme va creciendo el tamaño de la muestra.

En el capítulo 4 hablamos sobre otra parte importante del trabajo de tesis, la construcción de trayectorias aleatorias. Para la construcción de las trayectorias se hace uso de la generación de números aleatorio, para la generación de estos números se utilizo el algoritmo presentado aquí, el de la distribución normal.

A continuación se presenta la función de distribución y el algoritmo correspondiente para la generación de números aleatorios con comportamiento normal:

Una variable aleatoria Z tiene una distribución normal si su f.d.p. es:

$$f_z(z) = \frac{1}{s\sqrt{2p}} \exp\left[-\frac{(z-m)^2}{2s^2}\right], \quad -\infty < z < \infty \quad (3.1.7)$$

y se denota por d_normal(miu, vanza). Donde *miu* es la media y *vanza* la varianza.

Se considera solo la generación de $d_normal(0,1)$ con $media = 0$ y $vanza = 1$.

La forma más básica para la generación de números aleatorios con una distribución normal es:

$$y_1 = \text{sqrt}(-2 * \ln(x1)) * \cos(2 * \pi * x2) \quad (3.1.8)$$

$$y_2 = \text{sqrt}(-2 * \ln(x1)) * \text{sen}(2 * \pi * x2) \quad (3.1.9)$$

En este caso el procedimiento para la generación de números esta basado en el método de la transformación de Box-Muller de la forma polar, este método es considerado veloz en cuanto a la generación de números aleatorios con comportamiento normal.

Propósito: Generar variable aleatoria con comportamiento Normal.

Salida: Z

Entrada: m, s (media y varianza)

Fuente: Reuven y Rubinstein (1981)

Método:

- 1** Generar $x1 = 2.0 * \text{ran3}(\text{idum}) - 1.0$;
- 2** Generar $x2 = 2.0 * \text{ran3}(\text{idum}) - 1.0$;
- 3** $w = x1 * x1 + x2 * x2$;
- 4** Si $w \geq 1.0$ regresar a paso 1;
- 5** $w = \text{sqrt}((-2.0 * \ln(w)) / w)$;
- 6** $yy1 = x1 * w$;
- 7** $yy2 = x2 * w$;
- 8** regresar ($m + yy1 * s$);

Fig 3.5: Algoritmo forma polar

3.2.6 Logonormal

Una variable X de la forma $d_normal(\text{miu}, \text{vanza}^2)$. Entonces $Y = e^x$ tiene una distribución logonormal con una F.D.P.

$$f_Y(y) = \begin{cases} \frac{1}{\sqrt{2\pi s} y} \exp\left[-\frac{(\ln y - m)^2}{2s^2}\right], & 0 \leq y \leq \infty \\ 0, & \text{En otro caso} \end{cases} \quad (3.1.10)$$

Propósito: Generar variable aleatoria con comportamiento Logonormal.

Salida: Y

Entrada: miu, vanza

Fuente: Reuven y Rubinstein (1981)

Método:

1 Generar Z d_normal()

2 $X \leftarrow \text{miu} + \text{vanza} * Z$

3 $Y = e^x$

4 Regresar Y

Fig 3.6: Algoritmo LN-1

3.2.7 Cauchy

Una variable aleatoria Z tiene una distribución denotada por d_cauchy(alpha,beta), si la F.D.P. es igual a

$$f_x(z) = \frac{b}{\pi [b^2 + (z-a)^2]}, \quad a > 0, b > 0, -\infty < z < \infty \quad (3.1.11)$$

El algoritmo utilizado para la generación de este tipo de variables, esta basado en la siguiente propiedad:

Si Y1 y Y2 son variables aleatorias independientes de U(-1/2,1/2) y $Y1^2 + Y2^2 \leq \frac{1}{4}$ entonces $Z = Y1/Y2$ es de tipo d_cauchy(0,1).

Propósito: Generar variable aleatoria con comportamiento Cauchy.

Salida: Z

Entrada: alpha, beta

Fuente: Reuven y Rubinstein (1981)

Método:

1 Generar U1 y U2 de d_uniforme_cont(0,1)

2 $Y1 \leftarrow U1 - 1/2$ y $Y2 \leftarrow U2 - 1/2$

3 Si $Y1^2 + Y2^2 > 1/4$ ir a paso 1

4 $Z \leftarrow \text{beta} * Y1/Y2 + \text{alpha}$

5 Regresar Z

Fig 3.7: Algoritmo C-3

3.2.8 Weibull

Una variable aleatoria tiene una distribución de Weibull si la F.D.P. es igual a

$$f_z(z) = \begin{cases} \frac{\mathbf{a}}{\mathbf{b}^{\mathbf{a}}} z^{\mathbf{a}-1} e^{-(z/\mathbf{b})^{\mathbf{a}}}, & 0 \leq z \leq \infty, \mathbf{a} > 0, \mathbf{b} > 0 \\ 0 & \text{En otro caso} \end{cases} \quad (3.1.12)$$

y se denota por $d_weibull(\alpha, \beta)$. Para generar Z aplicando el método de la transformada inversa, como observación

$$U = F_z(z) = 1 - e^{-(z/\mathbf{b})^{\mathbf{a}}} \quad (3.1.13)$$

y

$$Z = \mathbf{b}(-\ln(1-U))^{1/\mathbf{a}} \quad (3.1.14)$$

Mientras $1-U$ se genera de $d_uniforme_cont(0,1)$, se tiene

$$Z = \mathbf{b}(-\ln U)^{1/\mathbf{a}} \quad (3.1.15)$$

o

$$\left(\frac{Z}{\mathbf{b}}\right)^{\mathbf{a}} = -\ln U \quad (3.1.16)$$

Como se pudo observar anteriormente $-\ln(U)$ es una variable aleatoria de tipo $d_exponencial(1)$. De donde se obtiene el algoritmo utilizado.

Propósito: Generar variable aleatoria con comportamiento Weibull.

Salida: Z

Entrada: α, β

Fuente: Reuven y Rubinstein (1981)

Método:

- 1** Generar $V1$ de $d_exponencial(1)$
- 2** $Z \leftarrow \beta * V^{1/\alpha}$
- 3** Regresar Z

Fig 3.8: Algoritmo W-1

3.2.9 Chi-Cuadrada

Si se genera un conjunto de variables aleatorias Z_1, \dots, Z_k de $d_normal(0,1)$. Entonces:

$$Y = \sum_{i=1}^k Z_i^2 \quad (3.1.17)$$

tiene la distribución chi-cuadrada con k grados de libertad y se denota por $d_chi_square(k)$.

La formula (3.1.17) dice: “la suma del cuadrado de variables aleatorias tiene una distribución con grados de libertad igual al numero de términos en la suma”. Una aproximación para generar variables con distribución $d_chi_square(k)$, es generar k variables normales estándar y entonces aplicar (3.1.17). De ahí se obtiene el siguiente algoritmo:

Propósito: Generar variable aleatoria con comportamiento Chi-Cuadrada.

Salida: Y

Entrada: k

Fuente: Reuven y Rubinstein (1981)

Configurar: Y \leftarrow 0, i \leftarrow 0

Método:

- 1 Generar Z de $d_normal(0,1)$
- 2 Hacer $Z = Z^2$
- 3 Sumar Z a Y
- 4 Incrementar i \leftarrow i+1
- 5 if i < k ir a paso 1
- 6 Regresar Y

Fig 3.9: Algoritmo Chi-cuadrada

3.2.10 T de Student

Si se genera una variable aleatoria con distribución normal, una variable aleatoria con distribución chi-cuadrada con k grados de libertad, además las dos son independientes, entonces:

$$X = \frac{Z}{\sqrt{Y/k}} \quad (3.1.18)$$

tiene una distribución con T de Student con k grados de libertad. Para generar X, se genera Z con una distribución normal. En este caso se utilizó la formula: $Z \leftarrow (-2\ln U1)^{1/2} \cos 2\pi U2$, además se genera Y con una distribución chi-cuadrada y se aplica la formula 3.1.18, de donde se obtiene el algoritmo que se presenta a continuación.

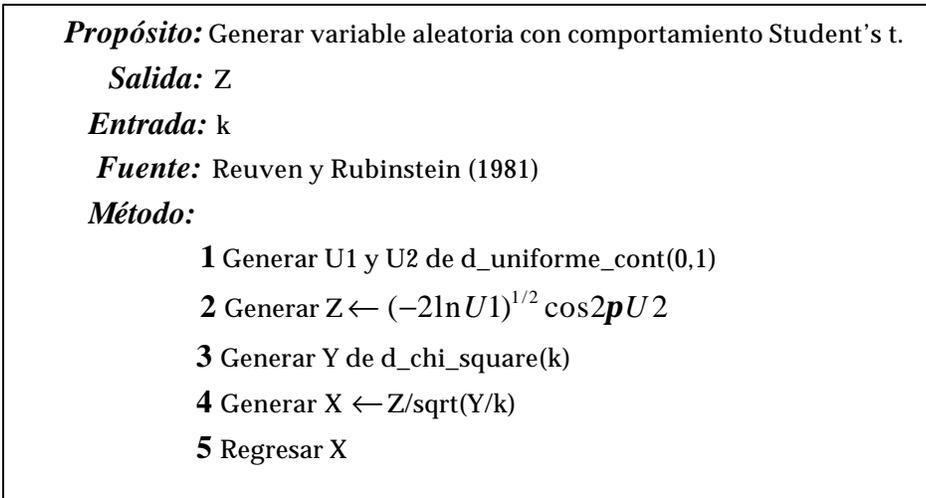


Fig 3.10: Student's t

3.3 Distribuciones discretas

Este tipo de distribuciones son aquellas en las que la variable aleatoria puede tomar un número determinado de valores.

A continuación, se presentan los algoritmos de distribuciones discretas incluidos en el núcleo de Monte Carlo implementado.

3.3.1 Binomial

Para la generación de variables aleatorias con comportamiento binomial, se utilizó el algoritmo IT-2, el cual puede ser usado para un conjunto de funciones a las que se les aplicó previamente el método de la transformada inversa, la tabla 3.7.1 de donde se tomó P_0 y A_{k+1} se encuentra en el libro de Reuven y Rubinstein [28].

A continuación se presenta el algoritmo IT-2 donde:

$$P_0 = (1 - p)^n$$

y

$$A_{k+1} = (n-k) * p / (k+1) * (1-p)$$

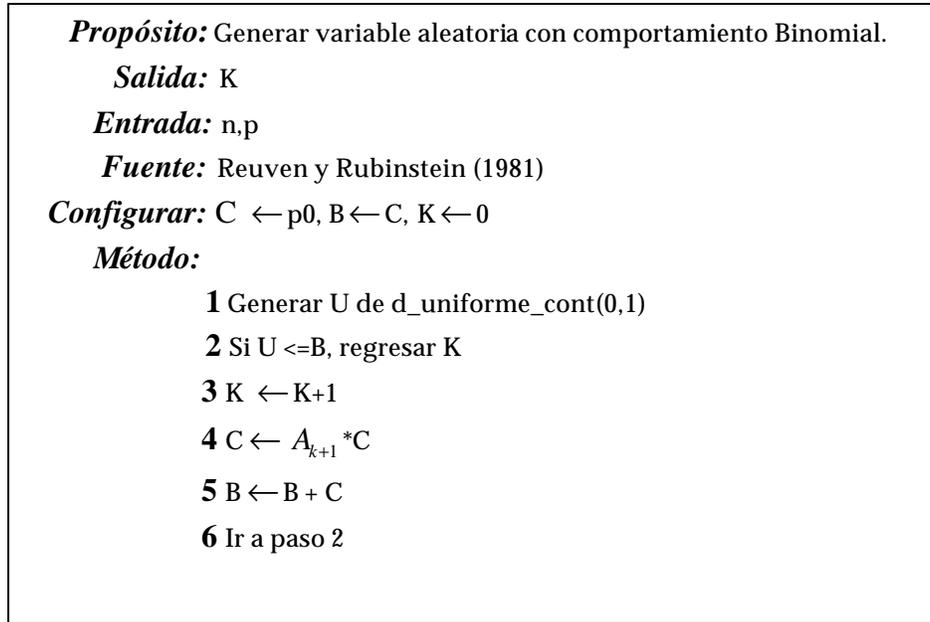


Fig 3.11: Algoritmo IT-2

3.3.2 Poisson

De la misma forma que en el caso anterior fue utilizado el algoritmo IT-2, pero en esta ocasión se tomó de la tabla 3.7.1 [28] P_0 y A_{k+1} correspondientes a la distribución de Poisson. Donde $p_0 = e^{-1}$ y $A_{k+1} = 1 / (k+1)$.

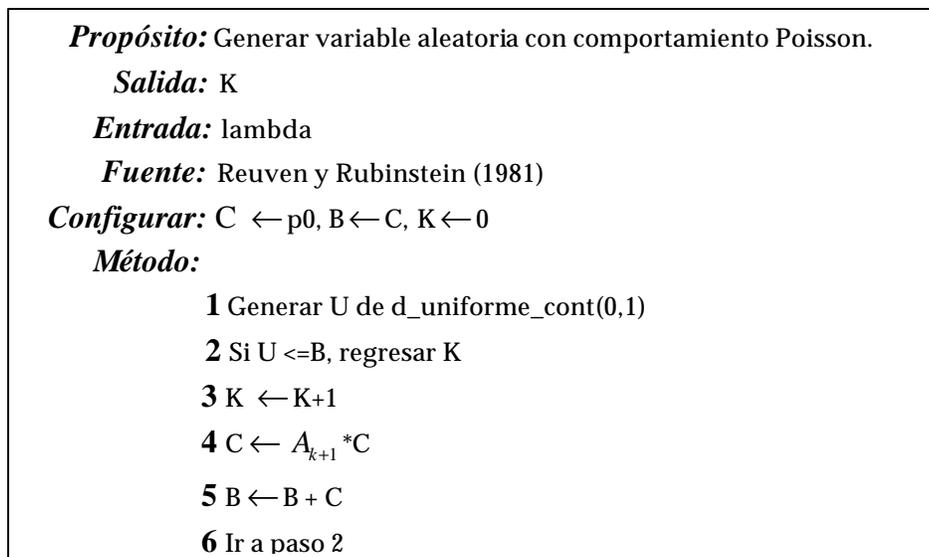


Fig 3.12: Algoritmo IT-2

3.3.3 Geométrica

Z tiene la F.D.P. geométrica

$$p_{i=(1-p)^i} \quad 0 < p < 1, i=0,1,\dots, \quad (3.1.19)$$

y se denota por $d_geometrica(p)$. Esta distribución tiene las propiedades siguientes:

1.- Y_1, Y_2, \dots , es una secuencia de pruebas de Bernoulli, variables aleatorias con probabilidad de éxito $1-p$. Entonces

$$Z = \min(i \geq 1 : Y_1 + \dots + Y_i = 1) - 1.$$

2.- $b = -1/\ln p$ y X es una variable aleatoria generada de $d_exponencial(1)$. Entonces $\lfloor X \rfloor$ tiene una F.D.P. como (3.1.19).

Para lo anterior se tomo el algoritmo GEO [26], para la generación de valores aleatorios con distribución geométrica

Propósito: Generar variable aleatoria con comportamiento Geométrico.
Salida: Z
Entrada: p
Fuente: Fishman(1999)
Configurar: beta ← -1/ln p
Método:

- 1 Generar Y de $d_exponencial(1)$
- 2 Si $U \leq B$, regresar K
- 3 $Z \leftarrow \lfloor beta * Y \rfloor$
- 4 Regresar Z

Fig 3.13: Algoritmo GEO

3.3.4 Hipergeométrica

De la misma forma que en el caso de la binomial fue utilizado el algoritmo IT-2, pero en esta ocasión se tomo de la tabla 3.7.1 [28] P_0 y A_{k+1} correspondientes a la distribución de Hipergeométrica. Donde

$$p_0 = \frac{(n-n_1)!(n-m)!}{(n-2n_1)!n!} \quad \text{y} \quad A_{k+1} = \frac{(n_1-k)(m-k)}{(k+1)(n-n_1-m+1+k)}$$

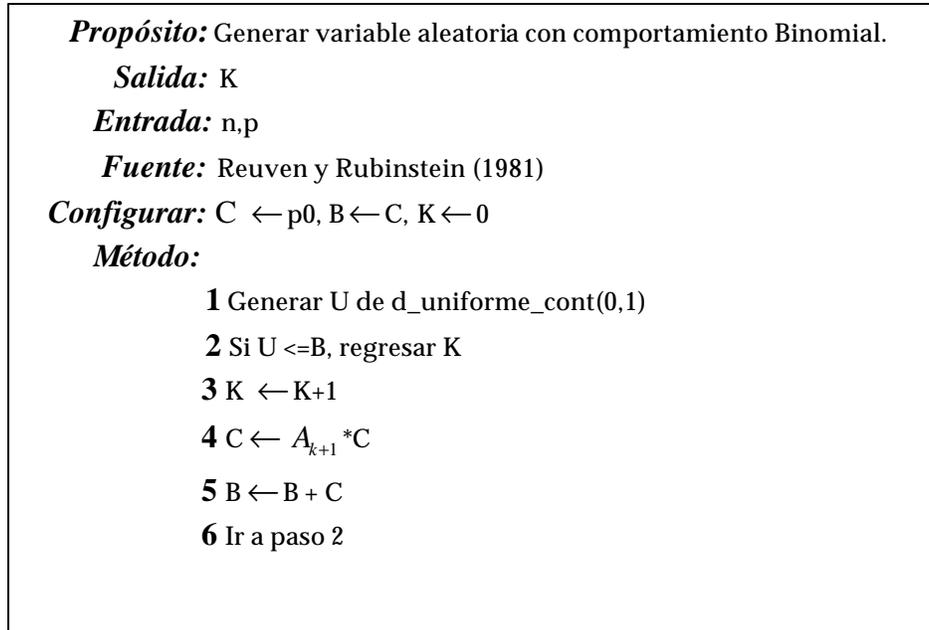


Fig 3.14: Algoritmo IT-2

3.3.5 Uniforme discreta

Para la generación de variables aleatorias con distribución uniforme discreta, de igual forma que en la continua, fue tomada la formula de la tabla 3.1 del libro de Fishman [26], a la que previamente se le aplicó el método de la transformada inversa. Según la referencia, para la generación de variables con este tipo de distribución únicamente se aplica p_0 , debido a que no se cuenta con la parte $c(z+1)$. Obteniendo el algoritmo que se presenta a continuación.

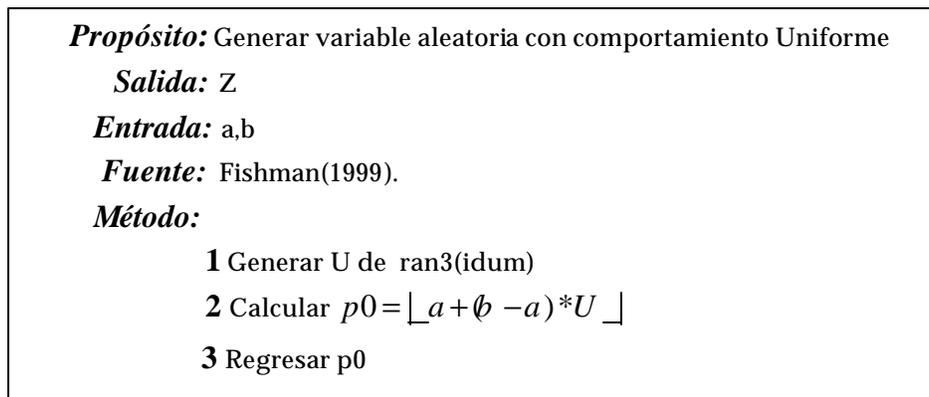


Fig 3.15: Algoritmo uniforme discreta

3.4 Problemas de simulación

La simulación es una técnica numérica para realizar experimentos en una computadora, la cual envuelve cierto tipo de modelos matemáticos y lógicos que describen el comportamiento de un sistema sobre periodos largos de tiempo real. El tiempo es un factor muy importante en simulación estocástica.

La inquietud de conocer el futuro y el deseo de predecirlo son dos factores que al paso del tiempo se han modificado. Anteriormente las predicciones estaban limitadas a los métodos deductivos tal como los usados por los filósofos Platón [28], Aristóteles y otros. Actualmente se cuenta con la computadora y el método de Monte Carlo que juntos, son unas herramientas muy poderosas que han permitido la simulación y con buenos resultados, la predicción de algunos sucesos. Dada la naturaleza probabilística del método de Monte Carlo, los resultados son obtenidos con un cierto grado de confianza.

Dos aplicaciones de la simulación que pueden ser mencionadas son: la creación de juegos de video o simuladores de vuelo.

El objetivo de la simulación, haciendo uso de la computación y del método de Monte Carlo es crear un proceso el cual represente un problema físico, dicho problema va estar compuesto de propiedades físicas entendibles; hacer uso del Método de Monte Carlo de la mejor manera posible, para el buen uso de las condiciones experimentales y de la herramienta computacional; por ultimo la posibilidad de examinar cada aspecto de la configuración del sistema con detalle. Entonces se observa la relación entre teoría, experimento y simulación, como se muestra en la figura 3.16 [27].

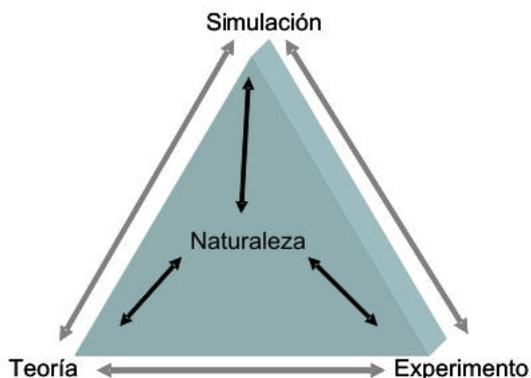


Fig 3.16: Relación entre teoría, experimento y simulación [27].

Son descritos a continuación.

3.4.1 Historial de un neutrón

En los 40's Von Newman describe este problema [31] a Stanislaw Marcin Ulam en una carta, en el cual Von Newman asume que los neutrones fueron generados en forma isotrópica, que el espectro de velocidad es conocido y que la absorción, dispersión y fisión en el material fisionable pueden ser descritos como una velocidad del neutrón. Además asume un relato apropiado del carácter estadístico del número de fisión de neutrones con probabilidades especificadas para la generación de 2, 3 y 4 neutrones en cada proceso de fisión.

La idea es entonces trazar la historia de un neutrón, haciendo uso de números aleatorios, para la selección de los resultados de varias interacciones a través del trayecto. Por ejemplo, Von Newman sugirió en ese entonces que, el calculo de cada neutrón “fuese representado por una carta”, la cual contiene sus características, esto es como: la zona del material en donde se encuentra el neutron, su posición radial, si se mueve hacia adentro o hacia fuera, su velocidad y el tiempo. La carta solo lleva “los valores aleatorios necesarios”. Eso fue usado para determinar el siguiente paso en la historia, el tamaño de la trayectoria y su dirección, tipo de colisión y velocidad después de la dispersión. Un nuevo neutrón es iniciado (análogamente asignando nuevos valores a la carta) cuando el neutrón bajo consideración fue dispersado o cualquier cosa que haya pasado dentro de otro núcleo; las cartas son iniciadas para varios neutrones si el neutrón original inicia la fisión. Una de las principales cantidades de interés fue la tasa de multiplicación por cada 100 de los neutrones inicializados.

3.4.2 Cálculo de área sobre un plano

Un ejemplo sencillo del método de Monte Carlo es el del cálculo de áreas sobre un plano:

Hay que imaginarse el siguiente experimento: una figura agrandada de la imagen anterior cuelga de una pared como un blanco. A cierta distancia de la pared se realizan N tiros en diferente tiempo, apuntando al centro del cuadro. Se asume que la distancia es considerablemente larga y no se tiene una buena puntería. Se asume también que todos los tiros no pegan en el centro; los tiros caen en N puntos aleatorios ¿Es posible estimar el área señalada como S en el plano, en base a los tiros realizados?.

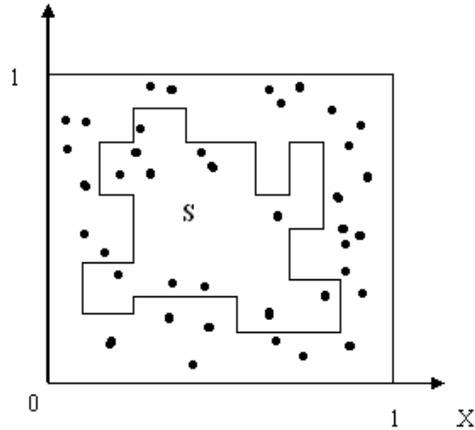


Fig 3.17: Calculo de área

El resultado del experimento es mostrado en la figura 3.17. En este experimento, N es el total de tiros y tiene un valor de 40, N' son los tiros acertados en el área S y tiene un valor de 24 y la proporción de $N'/N = 0.60$, la cual es prácticamente el doble de la actual (0.35). Es obvio que los tiros fueron realizados en una forma hábil. El resultado del experimento podría ser engañoso porque todos los tiros pegaron en el blanco cerca del centro y por lo tanto dentro del área S . Podemos ver que el método para calcular el área es válido cuando los tiros tienen un comportamiento uniforme sobre toda la área del cuadro. No basta con que sean aleatorios [30].

Capítulo 4

Integrales funcionales y de Feynman

Este tipo de integrales es usado en diferentes áreas como: finanzas, mecánica cuántica, bioinformática, etc, para la solución de procesos de tipo estocástico. En este capítulo se presentan cada una de las integrales funcionales de Wiener, Ito y Feynman, así como los algoritmos utilizados para la construcción de trayectorias, según sus propiedades. Además se presenta el algoritmo utilizado para la generación de trayectorias en paralelo, en base a los algoritmos de cada integral.

Trayectorias aleatorias

Las integrales funcionales presentadas en este capítulo serán implementadas para la construcción de trayectorias aleatorias, para dicho fin va ser necesario el uso del núcleo de Monte Carlo.

Las variables aleatorias generadas para la construcción de las trayectorias, van a ser generadas por medio del algoritmo para la generación de variables con comportamiento de tipo normal.

4.1 Integral de Wiener

El trabajo realizado sobre la *integral de Wiener* fue la implementación en C del programa original, además de su diseño e implementación en paralelo y como *servicio Web*. La primera versión del programa, para la generación de trayectorias con la *integral de Wiener*, fue diseñada y desarrollada en Fortran en el año de 1973 por el Dr. Sergio V. Chapa Vergara [32].

La integral de Wiener es una integral estocástica sobre un espacio de funciones continuas medibles o sobre un funcional. La integral es la siguiente:

$$\int_x F[x]dw \quad (4.1.1)$$

Calcular la *integral de Wiener* por métodos analíticos resulta muy complicado, por lo que la integral es calculada por un método numérico. El método numérico utilizado fue el de interpolación de trayectoria Browniana dado por Levy [32], el cual se muestra a continuación [32]:

Supongase que se tiene $x(t_0)$ y $x(t_1)$ tal que $t_0 < t_1$; entonces $x(t_2)$ para $t_0 < t_2 < t_1$ podrá ser encontrado por la siguiente fórmula de interpolación:

$$x_2(t_2) = \frac{x_0(t_1 - t_2) + x_1(t_2 - t_0)}{t_1 - t_0} + \mathbf{z} \sqrt{\frac{(t_2 - t_0)(t_1 - t_2)}{(t_2 - t_0)}} \quad (4.1.2)$$

donde:

$$x_0 = x_0(t) ; x_1 = x_1(t_1)$$

La letra \mathbf{z} es una *variable aleatoria* gaussiana con esperanza cero y varianza 1.

Como el intervalo $[0,1]$ es dividido en partes iguales entonces, las formulas de computación son:

$$\begin{aligned} x(1) &= \mathbf{z}_1 \sqrt{\frac{1}{2}} \\ x(1/2) &= \frac{1}{2} [x(0) + x(1)] + \mathbf{z}_2 \sqrt{\frac{1}{4}} \\ x(1/4) &= \frac{1}{2} [x(0) + x(1/2)] + \mathbf{z}_3 \sqrt{\frac{1}{8}} \\ x(3/4) &= \frac{1}{2} [x(1/2) + x(1)] + \mathbf{z}_4 \sqrt{\frac{1}{8}} \\ x(1/8) &= \frac{1}{2} [x(0) + x(1/4)] + \mathbf{z}_5 \sqrt{\frac{1}{16}} \\ &\dots \end{aligned}$$

En el capítulo 6 de resultados y conclusiones se presentarán algunas trayectorias realizadas por este método. A continuación se muestra el algoritmo principal utilizado, para la generación de trayectorias.

4.1.1 Algoritmo de Wiener

Propósito: Contruir trayectoria Browniana en el intervalo 0,1.

Entrada: NP (Número de puntos por trayectoria)

Salida: Arreglo X con los puntos de la trayectoria generada.

Fuente: Dr. Chapa 1973 [32].

Configurar: $I \leftarrow 0$, $T \leftarrow 1.0$, $XC \leftarrow 0.0$, $A \leftarrow 0.5$, $C1 \leftarrow \sqrt{T/4.0}$, $C2 \leftarrow \sqrt{T/8.0}$,
 $C3 \leftarrow \sqrt{T/16.0}$, $C4 \leftarrow \sqrt{T/32.0}$, $C5 \leftarrow \sqrt{T/64.0}$

Método:

- 1 Generar Y[I] con distribución normal(0,1).
- 2 $I \leftarrow I+1$
- 3 Si I menor a NP; ir a paso 1
- 4 $X1 \leftarrow \sqrt{1/2} * \text{punto 1}$
- 5 $X2 \leftarrow A * (XC + X1) + C1 * \text{punto 2}$
- 6 Si IB igual a 16; terminar
- 7 $X3 \leftarrow A * (XC + X2) + C2 * \text{punto 3}$
- 8 $X4 \leftarrow A * (XC + X1) + C2 * \text{punto 4}$
- 9 Si IB igual a 8; terminar
- 10 $X5 \leftarrow A * (XC + X3) + C3 * \text{punto 5}$
- 11 $X6 \leftarrow A * (X3 + X2) + C3 * \text{punto 6}$
- 12 $X7 \leftarrow A * (X2 + X4) + C3 * \text{punto 7}$
- 13 $X8 \leftarrow A * (X4 + X1) + C3 * \text{punto 8}$
- 14 Si IB igual a 4; terminar
- 15 $X9 \leftarrow A * (XC + X5) + C4 * \text{punto 9}$
- 16 $X10 \leftarrow A * (X5 + X3) + C4 * \text{punto 10}$
- 17 $X11 \leftarrow A * (X3 + X6) + C4 * \text{punto 11}$
- 18 $X12 \leftarrow A * (X6 + X2) + C4 * \text{punto 12}$
- 19 $X13 \leftarrow A * (X2 + X7) + C4 * \text{punto 13}$
- 20 $X14 \leftarrow A * (X7 + X4) + C4 * \text{punto 14}$
- 21 $X15 \leftarrow A * (X4 + X8) + C4 * \text{punto 15}$
- 22 $X16 \leftarrow A * (X8 + X1) + C4 * \text{punto 16}$
- 23 Si IB igual a 2; terminar
- 24 $X17 \leftarrow A * (XC + X9) + C5 * \text{punto 17}$
- 25 $X18 \leftarrow A * (X9 + X5) + C5 * \text{punto 18}$
- 26 $X19 \leftarrow A * (X5 + X10) + C5 * \text{punto 19}$
- 27 $X20 \leftarrow A * (X10 + X3) + C5 * \text{punto 20}$
- 28 $X21 \leftarrow A * (X3 + X11) + C5 * \text{punto 21}$
- 29 $X22 \leftarrow A * (X11 + X6) + C5 * \text{punto 22}$

```

30 X23 ← A * (X6 + X12) + C5 * punto 23
31 X24 ← A * (X12 + X2) + C5 * punto 24
32 X25 ← A * (X2 + X13) + C5 * punto 25
33 X26 ← A * (X13 + X7) + C5 * punto 26
34 X27 ← A * (X7 + X14) + C5 * punto 27
35 X28 ← A * (X14 + X4) + C5 * punto 28
36 X29 ← A * (X4 + X15) + C5 * punto 29
37 X30 ← A * (X15 + X8) + C5 * punto 30
38 X31 ← A * (X8 + X16) + C5 * punto 31
39 X32 ← A * (X16 + X1) + C5 * punto 32

```

Fig 4.1: Algoritmo Wiener

Con el programa, que hace uso del algoritmo para la generación de trayectorias, es posible la creación de las mismas en un espacio de funciones continuas $C_{[0,1]}$ en el intervalo $[0,1]$ y un número máximo de 32 puntos por trayectoria. La condición inicial es que $X(0) = 0$, entonces la medida de Wiener en dicho espacio, da la densidad de distribución de funciones continuas lo que es lo mismo: la distribución de trayectorias definidas en los tiempos $0 = t_0 < t_1 \dots < t_{n-1} < t_n = 1$. Por lo tanto la medida de Wiener expresa la probabilidad que una partícula recorra alguna de las posibles trayectorias, en el caso del ejemplo descrito en la sección 3.4.1 sobre el historial de una partícula. Debido a que el funcional se integra sobre un conjunto de trayectorias $\{x_1(t), \dots, x_n(t)\}$, las cuales forman el funcional $F[x]$, entonces es posible sacar el promedio de dicho funcional $F[x(t)]$ sobre un conjunto de trayectorias $x(t)$, de donde [32]:

$$\int_x F[x] dw \approx \sum_{i=1}^N \frac{F[x_i(t)]}{N} \tag{4.1.3}$$

se puede obtener el promedio del funcional para la estimación de la esperanza matemática.

4.2 Integral de Ito

El trabajo realizado sobre esta integral fue el de su análisis para el diseño, y su implementación en C, MPI, Java, además como servicio Web. Con su implementación en MPI se hace uso de una computación tanto distribuida como paralela; con la implementación en Java y servicio Web, se hace uso de una computación distribuida y de

red amplia como lo es la Grid. Lo anterior con la finalidad de aprovechar todos los beneficios obtenidos a través de estos tipos de computación.

Para la creación de trayectorias con la integral funcional de Ito, se utilizó otra forma de generación de trayectorias de Wiener, lo anterior debido a que la integral de Ito se basa en un proceso de Wiener.

La integral de Ito es otro tipo de integral funcional, para su análisis se tomó como referencia [33], de donde se obtuvieron las principales propiedades de un proceso de Wiener, que son:

- 1.- $W_0 \equiv 0$;
- 2.- Las trayectorias de procesos de Wiener son funciones continuas de $t \in [0, \infty)$;
- 3.- El valor esperado de $EW_t = 0$;
- 4.- La función de correlación $E(W_t W_s) \equiv t \wedge s, (a \wedge b = \min(a, b))$
- 5.- Para algún intervalo t_1, \dots, t_n el vector aleatorio $(W_{t_1}, \dots, W_{t_n})$ es de comportamiento gaussiano.
- 6.- Para algún s, t

$$EW_t^2 \equiv t$$

$$E[W_t - W_s] \equiv 0$$

$$E[W_t - W_s]^2 = |t - s|$$
- 7.- Los incrementos en un proceso de Wiener, en intervalos que no se interceptan son independientes, por ejemplo: $(s_1, t_1) \cap (s_2, t_2) = \emptyset$, las variables aleatorias $W_{t_2} - W_{s_2}, W_{t_1} - W_{s_1}$ son independientes
- 8.- Las trayectorias de procesos de Wiener no son diferenciables.

Tomando en cuenta las propiedades anteriores, para la construcción de una trayectoria de Ito en un intervalo $[a, b]$, se consideró la condición inicial en donde $W_0 = 0$ y los incrementos en base a un proceso de Wiener son $\Delta W_t = W_t - W_{t-1}$, las variables aleatorias con las que se definen los incrementos en X, son generadas de una distribución normal $(0, h) = \sqrt{h} * \text{normal}(0, 1)$, en donde:

$$h = \frac{b-a}{n} \quad (4.1.4)$$

de donde se obtiene la varianza de las variables aleatorias generadas, h son los incrementos definidos para t en el procesos de Wiener.

Las variables para la construcción de los segmentos para la construcción de las trayectorias en Ito son obtenidas por medio de:

$$\begin{aligned}
 W_1 &= W_0 + \Delta W_{t_1} \\
 W_2 &= W_1 + \Delta W_{t_2} \\
 &\cdot \\
 &\cdot \\
 &\cdot \\
 W_n &= W_{n-1} + \Delta W_{t_n}
 \end{aligned}$$

La integral estocástica de Ito para un proceso simple, que se puede denotar por $I(X)$, es la siguiente [34]:

$$\int_0^b X_t dB_t = \sum_{k=0}^{n-1} X^k (B_{t_{k+1}} - B_{t_k}) = I(X) \quad (4.1.5)$$

donde $B_{t_{k+1}} - B_{t_k}$ son los segmentos generados en base a un proceso de Wiener, con la B se hace referencia a un movimiento Browniano, de la misma forma que se le conoce a un proceso estocástico de Wiener y por ultimo la variable X^k es una variable aleatoria de tipo normal, con media 0 y varianza $2h$. Solo por notación, la integral anterior es posible representarla de la siguiente forma:

$$\int_0^b X_t dW_t = \sum_{k=0}^{n-1} X^k (W_{t_{k+1}} - W_{t_k}) = I(X) \quad (4.1.6)$$

Las variables X_t y ΔW_t son variables aleatorias independientes.

4.2.1 Algoritmo de Ito

A continuación se presenta el algoritmo utilizado para la generación de las trayectorias en base a la integral de Ito.

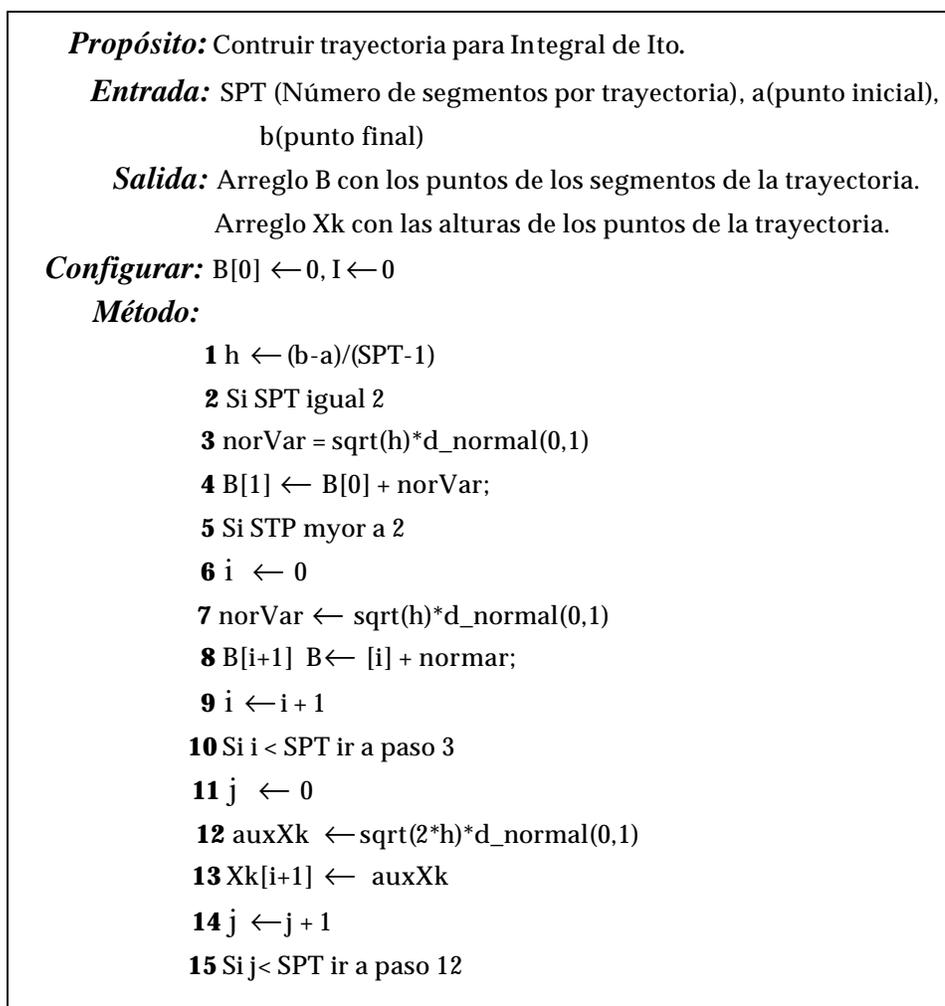


Fig 4.2: Algoritmo de Ito

De igual forma que para la integral de Wiener, ejemplos de trayectorias generadas con este algoritmo se presentan en el capítulo de resultados y conclusiones.

4.3 Integral de Feynman

En la presente sección se hará un pequeño análisis sobre la Integral de Feynman. Dicho análisis permitirá realizar el diseño e implementación en C y MPI. La implementación en MPI hace uso de una computación tanto distribuida como paralela. Lo anterior con la

finalidad de aprovechar todos los beneficios obtenidos a través de estos tipos de computación.

El uso de la integral de Feynman se da en diferentes campos como: bioinformática, física, mecánica cuántica, matemáticas, etc. El término “Integración de un funcional” se refiere a la suma de todas las trayectorias posibles encontradas entre un punto inicial y uno final. Dicha integral es una integral funcional.

Una integral funcional, análogamente a la definición de una función, su dominio puede estar compuesto ya sea de un conjunto de valores o funciones y su contradominio está compuesto de un conjunto de funciones, en donde a un elemento del dominio le corresponde solo un elemento del contradominio.

La principal idea de la integral de trayectoria de Feynman es la ley de superposición. Para calcular la amplitud de transición de un sistema en un estado inicial T' a un estado final T'' , se consideran todas las posibles trayectorias para llegar de un punto a otro. En la integral se considera la superposición de todas las posibles trayectorias [36].

La figura 4.3 muestra un ejemplo de 5 trayectorias, las cuales son presentadas esquemáticamente.

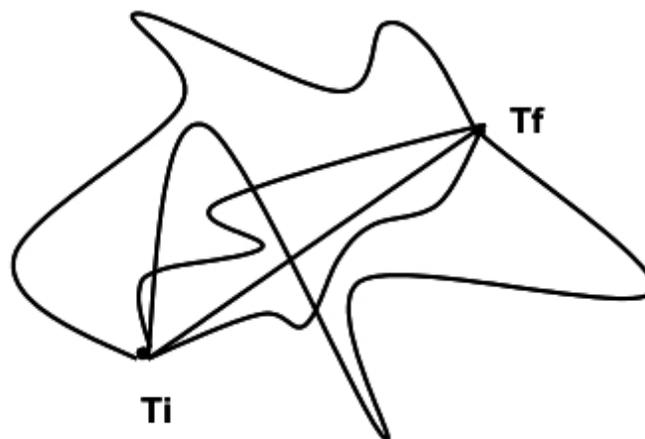


Fig 4.3: Ejemplo trayectorias

Es común que la trayectoria dependa del tiempo, en este caso es conveniente realizar una discretización en el tiempo, posteriormente realizar una parametrización, lo cual es una manera de identificar como un parámetro a cada segmento de la discretización realizada sobre el tiempo, lo que permite tratar como funciones a curvas que no lo son.

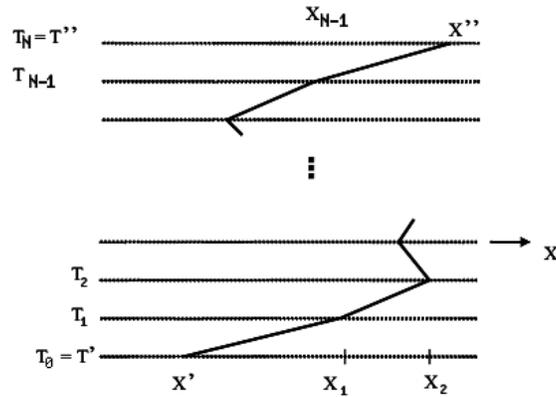


Fig 4.4: Trayectoria [36]

La figura 4.4 muestra el ejemplo de una trayectoria que va desde x' a x'' en un intervalo de tiempo T' a T'' , después de la discretización y parametrización sobre el tiempo.

Para encontrar la amplitud de probabilidad de un proceso dado, son integradas todas las posibles trayectorias de un sistema entre los estados inicial y final. Sin embargo, de entrada no se descarta ninguna trayectoria que tenga una probabilidad no nula de ocurrir, aunque esta probabilidad sea muy baja [37].

Inicialmente en el caso de la integral de Feynman, se trabaja con un operador de evolución y un Hamiltoniano, en donde su representación es la siguiente [36]:

La amplitud mencionada esta dada por:

$$\langle \mathbf{y}(x'') | U(T'', T') | \mathbf{y}(x') \rangle = \int \Psi^*(x'') e^{-iH(t''-t')} \mathbf{y}(x') \quad (4.1.7)$$

en donde:

$|\mathbf{y}(x')\rangle$ representa un estado inicial, $\langle \mathbf{y}(x'')|$ un estado final y $U(T'', T')$ representa todas las posibles trayectorias para llegar desde el estado inicial al final.

$$U(T'', T') = e^{-iH(t''-t')/\hbar} \quad (4.1.8)$$

el cual es el operador de evolución y H el Hamiltoniano del sistema.

Posteriormente introduciendo la discretización y parametrización en el intervalo temporal es posible reformular el problema como en la siguiente expresión:

$$\langle \mathbf{y}(x'') | U(T'', T') | \mathbf{y}(x') \rangle = \int_{x(t')=x'}^{x(t'')=x''} [Dx(t)] e^{iS[x]/\hbar} \quad (4.1.9)$$

en donde $S[x]$ es la acción:

$$S[x] = \int_{t'}^{t''} L(x, \dot{x}) dt \quad (4.1.10)$$

y la ec. 4.1.10 es el Lagrangiano de el sistema. $x(t)$ es la trayectoria entre x' y x'' , $[Dx(t)]$ es la medida del funcional, entonces $\int_{x'}^{x''} [Dx(t)]$ denota la integral sobre todas las trayectorias entre $x(t') = x'$ y $x(t'') = x''$.

Gracias a los análisis y estudios realizados sobre la integral y un cambio de un tiempo euclideo a un tiempo real $t \rightarrow it$ [36] (se recomienda también consultar para el desarrollo de los cambios realizados), la integral anterior puede ser trabajada como en la ecuación (4.1.11), otra de las ventajas que se obtiene al realizar dicho cambio en el tiempo es que se permite manejar un sistema cuántico como un sistema de física estadística.

$$\langle y(x'') | U(T'', T') | y(x') \rangle = \int [Df(x, t)] e^{-S[f]/\hbar} \quad (4.1.11)$$

En donde $S[f]$ sigue siendo un Lagrangiano y la acción del sistema, pero ahora trabajado en forma real.

Las trayectorias que más contribuyen a la ec. (4.1.11) son aquellas que minimizan la acción $S[f]$. Note además que si $S[f] \gg \hbar$ entonces se recupera el principio de mínima acción.

En nuestra propuesta sobre la integral, el conjunto de trayectorias generadas para el cálculo de la integral, va ser dividido en conjuntos más pequeños. La cantidad de trayectorias por conjunto es generado en diferentes procesadores. En este caso, el conjunto de trayectorias generadas por procesador va ser considerado una muestra, de la cual se obtendrá su media y varianza. El detalle del desarrollo de este programa es descrito más adelante en la sección 4.1.3.

Para mostrar el uso de la integral de Feynman, se considero como acción $S[f]$ el caso del oscilador armónico simple en donde su notación Lagrangiana esta dada por:

$$L = T - V$$

en donde T denota a la *Energía Cinética* y $V = \text{Energía Potencial}$, entonces para el oscilador armónico:

$$L = \frac{P^2}{2m} - \frac{1}{2} m \omega^2 x^2 \quad (4.1.12)$$

en donde:

m = masa

x denota posición

$w = \frac{1}{2}kx^2$ frecuencia de oscilación

$P = m \cdot v$, v es la velocidad, la cual puede ser representada por $\frac{dx}{dt}$ para masa etc.

entonces:

$$L = \frac{P^2}{2m} - \frac{1}{2}mw^2x^2 = \frac{1}{2m}m^2\left(\frac{dx}{dt}\right)^2 - \frac{mw^2}{2}x^2 \quad (4.1.13)$$

Por medio de la definición de derivada:

$$\left.\frac{dx}{dt}\right|_{t_0} = \lim_{\Delta t \rightarrow 0} \frac{x(t_0 + \Delta t) - x(t_0)}{\Delta t} \text{ cuando } \Delta t \rightarrow 0$$

se puede obtener una versión discreta en el intervalo $[x_i, x_{i+1}]$ como $\frac{dx}{dt} \approx \frac{x_{i+1} - x_i}{\Delta t}$ donde

$x_i = x[t_i]$ y $\Delta t = t_{i+1} - t_i$

entonces el Lagrangiano utilizado en la acción $S[\mathbf{f}]$ por la integral de Riemann es:

$$S[\mathbf{f}] = \int L dt \approx \sum_{i=1}^n L_i \Delta t \quad (4.1.14)$$

por lo tanto una versión discretizada de la acción es la siguiente:

$$S[\mathbf{f}] = \sum_{i=1}^n \left[\frac{m}{2} \frac{(x_{i+1} - x_i)^2}{\Delta t^2} - \frac{mw^2}{2} x_i^2 \right] \Delta t \quad (4.1.15)$$

Como ya fue mencionado, es posible tener una gran cantidad de trayectorias para poder llegar de un punto a otro, lo que hace al problema, un problema cuántico. Por lo cual, el cálculo de la integral de Feynman es muy pesado para hacerse manualmente, por esta razón se consideró su desarrollo en paralelo, para hacer uso del computo distribuido y así reducir tiempos de ejecución, además de aprovechar los recursos computacionales de un cluster y poner a disposición una herramienta computacional que, pueda ser usada por personas interesadas en el tema, aunque no se encuentren muy relacionadas con el mundo de la programación.

4.3.1 Algoritmo de Feynman

A continuación se presenta el algoritmo para la generación de trayectorias según la integral de Feynman y el algoritmo que corresponde a la acción:

Propósito: Cálculo de la integral de Feynman, en base a la construcción de trayectorias sobre una acción $S[f]$.

Entrada: NT (Número de trayectorias).

Salida: Archivo con el historial de la acción.

Configurar: $\text{auxI} \leftarrow 0.0, \text{ValIntTr} \leftarrow 0.0$

Método:

- 1 $i \leftarrow 0$
- 2 $\text{apFuncion}(\text{PTM}, \text{ITiem}, m, w)$
- 3 $\text{valorIT_Accion}(\text{PTM})$
- 4 $\text{param} \leftarrow (-1 * \text{ar_aux}[i])$
- 5 $\text{tempE} \leftarrow \text{exp_neg}(\text{param})$
- 6 $\text{ar_exp}[i] \leftarrow \text{tempE}$
- 7 $\text{auxI} \leftarrow \text{tempE} * (i+1-i)$ //Incrementos en 1 sobre con respecto a la integral de Rienman
- 8 $\text{ValIntTr} \leftarrow \text{ValIntTr} + \text{auxI}$
- 9 $i \leftarrow i + 1$
- 10 Si i menor a NT ir a paso 2
- 11 regresar ValIntTr

Fig 4.5: Algoritmo Feynman

La propuesta presentada sobre la integral de Feynman, se representa por medio del algoritmo valorIT_Expo , el cual llama a dos funciones importantes que son: apFuncion y valorITaccion .

En nuestro caso, en el paso 3 se llama a la función apFuncion , sin embargo en el paso 3, dentro del diseño general, se llama al apuntador de la función entrante de la acción.

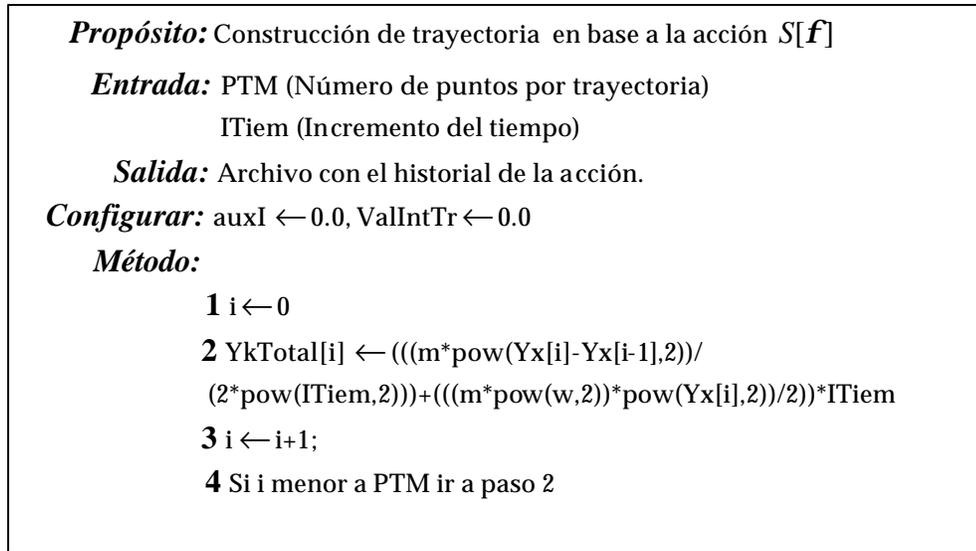


Fig 4.6 Algoritmo apFuncion

El algoritmo *apFuncion* es el que genera los puntos sobre el eje Y de la trayectoria. Dichos puntos son guardados en un arreglo.

La *acción* es el Lagrangiano del oscilador armónico, el cual corresponde al ejemplo presentado en la tesis, con el propósito de mostrar el funcionamiento para la generación de trayectorias.

En el algoritmo *acción*, Yx es un arreglo de puntos aleatorios generados con una distribución $d_normal(0,1)$, los cuales simulan la posición del objeto.

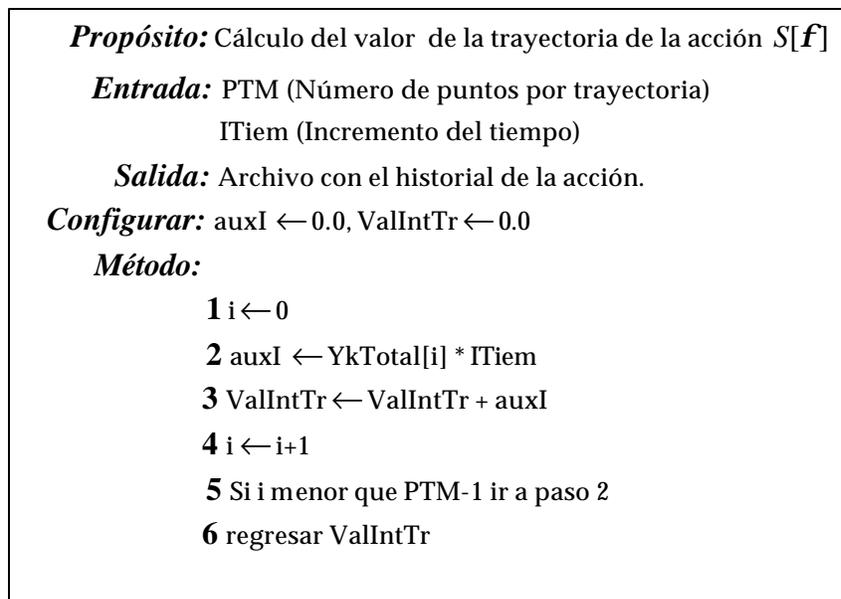


Fig 4.7: Algoritmo valorIT_accion

El algoritmo *valorIT_accion* calcula el valor de la trayectoria (generada por el algoritmo *apFuncion*). Se promedian los puntos aleatorios generados y el resultado se multiplica por el intervalo del tiempo total de la trayectoria desde un punto inicial al un punto final.

4.4 Algoritmo paralelo para la generación de trayectorias

Propósito: Algoritmo en paralelo para el envío y recepción de datos entre los diferentes procesadores, se calcula la media muestral, varianza muestral, media poblacional y varianza poblacional, de la generación de trayectorias, ya sea de Wiener, Ito o Feynman.

Salida: Archivo con la información de cada procesador

Configurar: $k \leftarrow 0$

Método:

Leer NPT \leftarrow número de puntos por trayectoria

Leer NT \leftarrow número de trayectorias

if (NPT > 0) && (NT > 0) **then**

 aux_numtrays = NT/nps; //saca trayectorias a generar por procesador

else

 Finalizar ámbito paralelo

 Terminar programa

end

if (NPT > 0) && (aux_numtrays > 0) **then**

while k < aux_numtrays **do**

 Trayectorias de Wiener o

 Trayectorias de Ito o

 Trayectorias de Feynman

end

for lm=0 to num_valores_trays **do**

 proc = proc + ar_aux(lm)

end

 mediaM = proc/aux_numtrays //se calcula la media del conjunto de trayectorias generadas por procesador

for lm=0 to num_valores_trays **do**

 Calcula diferencia entre cada valor de las trayectorias con la media

 Elevar al cuadrado cada uno de los valores resultantes del paso anterior y sumarlos

end

Fig 4.8 a: Algoritmo paralelo

```

Con los resultados del ciclo calcular varianza muestral
if (proa_id!=0) then
    Enviar mediaM al procesador 0
else
    MedMuesP(0) = mediaM
    if (num_procs>1) then
        for i=1 to num_procs do
            El procesador 0 recibe los valores de la media del resto de los procesadores
            MedMuesP(i) = mediaM
        end
    end
end

```

```

if proa_id!=0 then
    Enviar vanzaM al procesador 0
else
    MedMuesP(0) = mediaM
    if (num_procs>1) then
        for i=1 to num_procs do
            El procesador 0 recibe los valores de la varianza del resto de los procesadores
            MedMuesP(i) = mediaM
        end
    end
end

```

```

if proc_id=0 then
    for i=0 to num_procs do
        sumMedP=sumMedP + MedMuestP(i) //Calcular suma de las medias de cada
                                                procesador
    end
    promDproms = sumMedP/num_procs //Sacar promedio de los promedios de cada
                                                procesador
end

```

```

MPI_Allreduce(proc,totalProc,1MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD); //Se suman
//todos los valores de proc para posteriormente obtener la media poblacional

```

```

mediaTotTry = totalProc/totalG_trayectorias //Sacar media poblacional

```

Fig 4.8 b: Algoritmo paralelo

```

/*En forma paralela cada uno de los procesadores ejecuta el siguiente for */
for i=1 to aux_numtrays do //Ciclo hasta en número de trayectorias por procesador
    cuadDif = ar_aux(i) - mediaTotTray; //Diferencia de cada uno de los valores de
                                     la acción con respecto a la media global

    aux_vanza2 = aux_vanza2 + (cuadDif*cuadDif) //Sumar los cuadrados de la dieferencia
                                               anterior para posteriormente obtener la
                                               varianza poblacional

end

MPI_Reduce(aux_vanza,vanzaParci,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

if proc_id==0
    vanzaP = vanzaParci/totalG_trayectorias //Sacar varianza poblacional
end //else

Finalizar programa
end

```

Fig 4.8 c: Algoritmo paralelo

El algoritmo de la figura 4.10 es utilizado para la generación de trayectorias en forma paralela. La generación de trayectorias se realiza en base a cualquiera de los tres métodos descritos en el presente capítulo, ya sea de Wiener, Ito o Feynman. En el algoritmo, dentro del ciclo while se incluyen los programas que involucran el desarrollo de los métodos correspondientes.

Además de la generación de trayectorias en forma distribuida, se recaba la información de cada procesador de su media muestral y su varianza muestral, para su almacenamiento en un archivo.

Para el cálculo de la media poblacional y la varianza poblacional, se hace uso de dos instrucciones (***MPI_Allreduce*** y ***MPI_Reduce***) de MPI, las cuales fueron de utilidad para reducir los tiempo de ejecución, evitando la transferencia de información al procesador principal, lo que se vio reflejado en la disminución de los tiempos de ejecución. Para mayor detalle del diseño se recomienda ver el capítulo 5.

Capítulo 5

Arquitectura y Diseño

A lo largo de los capítulos anteriores, fueron tratados conceptos que se consideran relevantes para el presente trabajo de tesis tales como: cómputo local, cómputo paralelo y de red amplia, método de Monte Carlo, integrales de Wiener, Ito y Feynman, etc. En el presente capítulo se describe la arquitectura general enfocada a computación local, su diseño en OpenMP. Posteriormente se presenta una arquitectura general para dar solución al problema, un diseño en paralelo para la generación de trayectorias y la implementación de dicha solución, lo anterior enfocado a computación paralela distribuida. Posteriormente, se aborda la arquitectura para una computación en red amplia o Grid, en donde se hace uso de un software llamado Taverna, del cual inicialmente se da una descripción, además de presentar el diseño e implementación para la herramienta antes mencionada.

5.1 Arquitectura para computación local

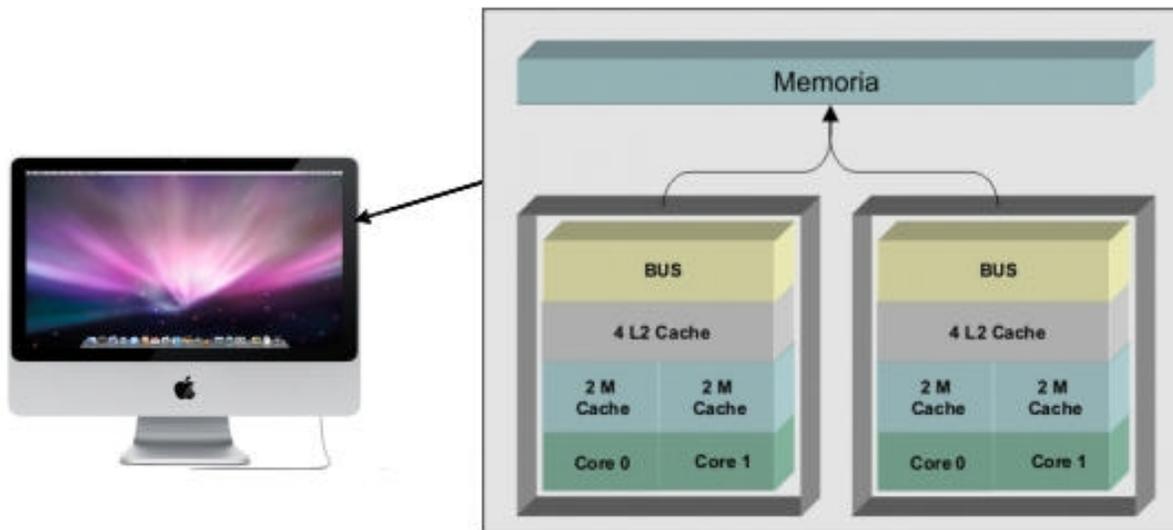


Fig 5.1: Arquitectura computación local

La figura 5.1 muestra la arquitectura considerada para la propuesta del diseño en computación local, es una computadora multi-núcleo, en este tipo de arquitecturas, para aprovechar al máximo sus recursos computacionales, la programación se realiza mediante hilos. Para la programación en una arquitectura de memoria compartida existen dos opciones: una es *Pthreads* y la segunda OpenMP, las cuales difieren en la programación y la forma de manejar los hilos. El uso de *Pthreads* permite tener más control sobre el manejo de los hilos, lo cual lo hace un poco más complejo de programa. En cuanto a OpenMP, la

distribución y en ocasiones la “paralelización” en hilos depende no tanto del programador, sino del mismo OpenMP.

La propuesta para la computación local se realizó en OpenMP. Se selecciono de esta manera porque OpenMP tiene la ventaja de encargarse de la distribución del trabajo entre los diferentes hilos y del hecho de ser una herramienta de programación un poco más sencilla de manejar, se considera una buena opción para su comprensión en áreas diferentes a las de la computación. Por lo que la propuesta para la computación local se realizo en OpenMP.

5.2 Diseño en OpenMP

La biblioteca desarrollada en el trabajo de tesis, la cual contiene el conjunto de algoritmos que forman el núcleo de Monte Carlo, para la generación de números aleatorios, puede ser usada en una arquitectura multi-núcleo a través de programas desarrollados en C, con el uso de la biblioteca OpenMP. Esta es utilizada para la programación en equipo con memoria compartida, simplemente se hace el llamado de la función deseada dentro del programa en C y dentro del ámbito de OpenMP. En caso de generar un conjunto de números aleatorios, la distribución del trabajo necesario para su generación queda a cargo de OpenMP, a través de la paralelización de ciclos.

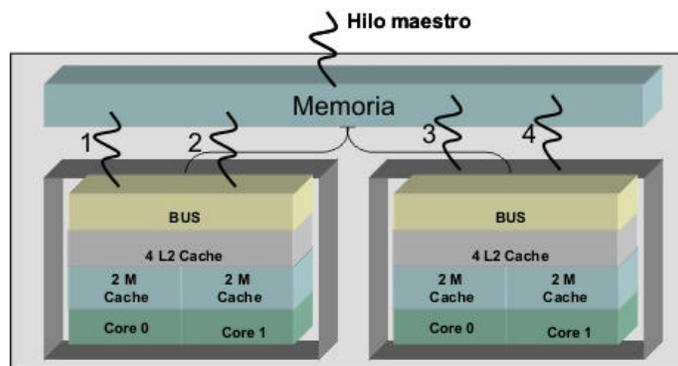


Fig 5.2: Hilos con OpenMP

Un bucle es fácilmente paralelizable en OpenMP, solo es necesario seguir algunas reglas como son:

- No dependencia entre iteraciones
- Prohibidas instrucciones `break`, `exit()`, `goto`,...

Se tiene a continuación un ejemplo:

```
#pragma omp parallel for
for ( i= primero; i< 1000000; i++ )
    b[i] = d_uniforme_cont(0,1);
```

El hilo maestro se encarga de generar nuevos hilos para cubrir las iteraciones del ciclo. El ciclo es dividido por la biblioteca, enviado a cada núcleo y ejecutado a través de hilos. La figura 5.2 ilustra un procesador multi-núcleo con la distribución de hilos, después de la ejecución del ciclo dentro del ámbito de OpenMP.

Para el caso de construcción de trayectorias propone considerar dos bibliotecas que son *distribuciones.h* y *trayectorias.h*. La primera corresponde al conjunto de funciones implementadas, tanto continuas como discretas. La biblioteca *trayectorias.h* se compone de las funciones correspondientes a la parte de integración, dichas bibliotecas son descritas con más detalle en las siguientes secciones. Cabe mencionar que la construcción de *trayectorias.h* sería considerando únicamente la parte secuencial de las funciones encargadas de generar las trayectorias. Teniendo en cuenta estos dos archivos de cabecera, el diseño se describe a continuación:.

De entrada se solicita el número de puntos por trayectoria (NP) y el número de trayectorias, OpenMP por medio de un fork crea el número de hilos solicitado. La construcción de trayectorias se realiza dentro de un ciclo for. OpenMP se encarga de distribuir el trabajo del ciclo entre los hilos, lo anterior es posible gracias a que no se cuenta con dependencias. Cuando concluye el ciclo son calculadas la media y varianza de los datos generados por cada hilo. Posteriormente el hilo principal espera los resultados y se calcula $mediaG$ y $varianzaG$, que corresponden a la media global y varianza global. La figura 5.3 muestra un esquema del diseño descrito anteriormente.

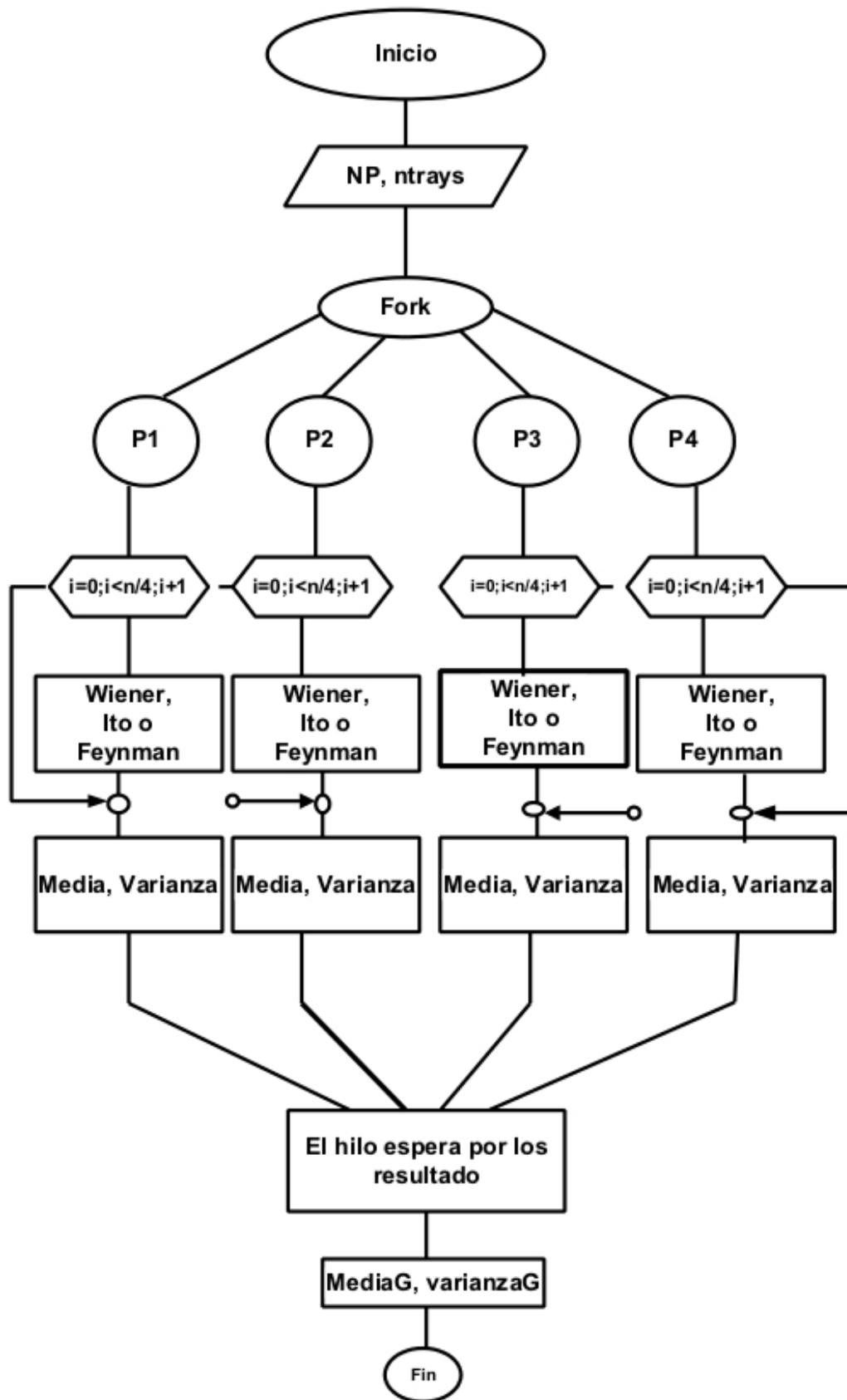


Fig 5.3: Diseño computación local

5.3 Diseño y arquitectura en computación paralela distribuida

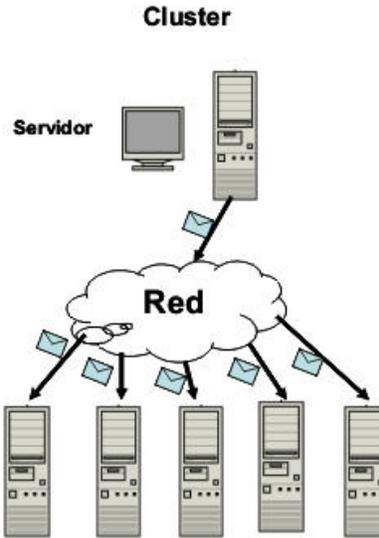


Fig 5.4: Distribución de datos en un cluster

La biblioteca del conjunto de algoritmos, que conforman el núcleo de Monte Carlo, de igual forma que en el diseño multi-núcleo, puede ser utilizada en los programas secuenciales de C, pero en este caso para el efecto de su uso en paralelo, es utilizada la biblioteca MPI. MPI se utiliza en equipo con memoria distribuida, como lo son los clusters, esta biblioteca -como ya se ha mencionado anteriormente-, hace uso del paso de mensajes para la distribución del trabajo entre los diferentes procesadores.

Dentro del ámbito MPI, el procesador principal se encarga de la distribución del trabajo, divide la cantidad de números aleatorios generados, entre la cantidad disponible de procesadores y envía los datos al resto de los procesadores, indicando la cantidad correspondiente de números a generar de los procesadores.

La ejecución de la función seleccionada, se hace a través de cómputo redundante, en donde todos los procesadores ejecutan el mismo código, lo cual es posible por el uso de MPI. La figura 5.4 muestra la distribución de los datos necesarios para la ejecución de las funciones.

En cuestión de “*hardware*”, para el desarrollo y ejecución de los programas en MPI, se utilizó un cluster -el cual fue descrito en la sección (2.3.7)-, con el fin de aprovechar la disponibilidad de los recursos computacionales del mismo, además de poder realizar la ejecución de las aplicaciones en tiempos más cortos. El cluster utilizado cuenta con 32

nodos de los cuales fueron usados 20, cada uno de ellos es de 64 bits, éstos están interconectados a través de una red local.

Para el desarrollo de los programas en MPI fue utilizado lam-MPI, “*software*” con el que cuenta el cluster para el desarrollo de aplicaciones distribuidas.

La aplicación que hace uso del cluster, corre sobre un procesador principal, con un identificador de 0. El envío de la información de este nodo hacia el resto de los procesadores se realiza a través del paso de mensajes. La figura 18 muestra el envío de mensajes del procesador principal, a través de la red, hacia el resto de los nodos que componen el cluster.

Arquitectura general de solución

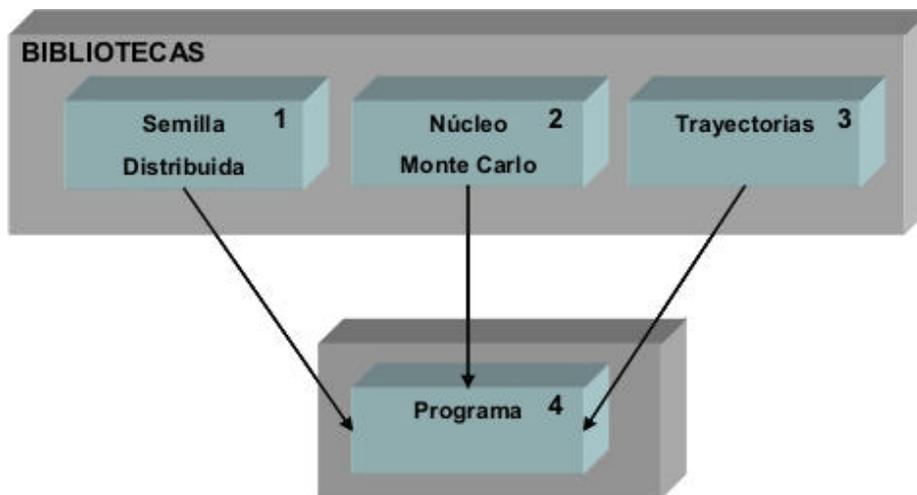


Fig 5.5: Arquitectura general

La figura 5.5 presenta la arquitectura general propuesta para la resolución del problema, cada uno de sus componentes es descrito a continuación:

Dicha arquitectura consta de 3 partes principales que son:

- 1) *SemillaMPI*.- debido a que se va a trabajar sobre una plataforma paralela y distribuida es necesaria la generación de una semilla diferente para cada procesador. En el caso de ser la misma en cada nodo, los datos a procesar dependientes de ésta serían iguales en cada procesador, debido a que se generaría la misma secuencia de números aleatorios. Por esta razón se consideró

necesaria la generación de una *semilla distribuida*, la cual se encargará de crear al mismo tiempo semillas diferentes en cada uno de los nodos utilizados.

- 2) **Núcleo Monte Carlo.**- En este módulo se consideró la creación del núcleo de Monte Carlo basado en dos grupos de distribuciones: uno de continuas y el otro de discretas. Este módulo hará uso de la *semillaMPI*, ya que cada grupo utiliza la distribución uniforme para la generación de variables aleatorias con diferentes comportamientos. Sin embargo, también será posible el uso de dicho núcleo en programas secuenciales.
- 3) **Trayectorias.**- En este módulo fue considerada la creación de trayectorias de camino aleatorio, haciendo uso del núcleo de Monte Carlo propuesto anteriormente.
- 4) **Programa principal.**- Fue considerado que un programa sea capaz de hacer uso de los módulos anteriormente mencionados.

A continuación, se presenta el diseño realizado para la solución propuesta.

5.4 Diseño en MPI

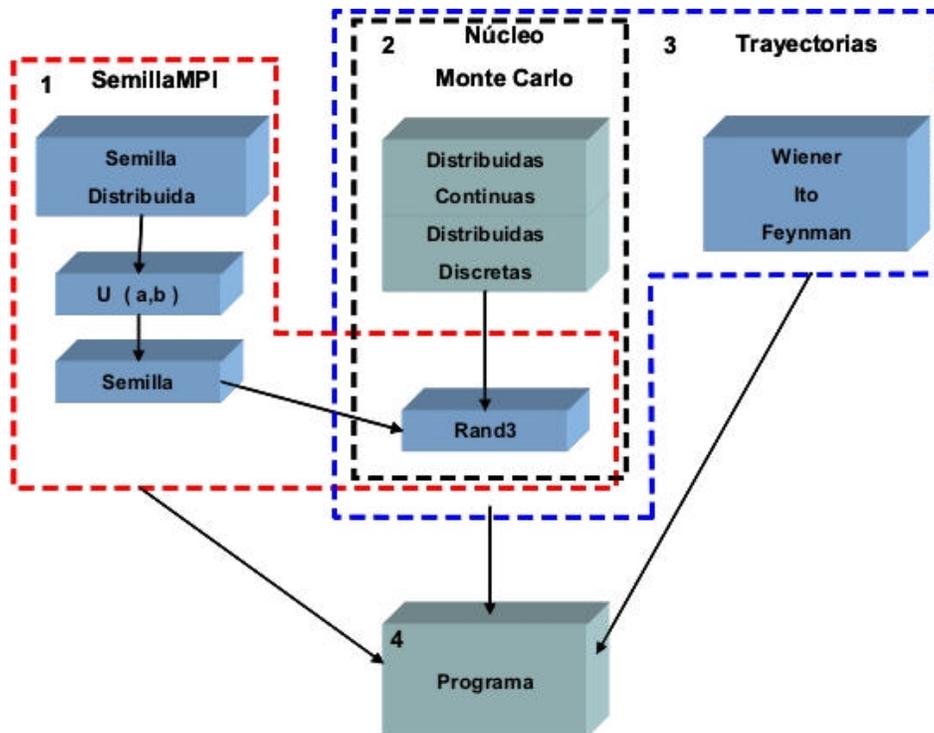


Fig 5.6: Diseño de la solución

En esta sección se describe el diseño detallado para cada uno de los módulos que involucran la arquitectura general. La figura 5.6 muestra el diseño.

- 1) **SemillaMPI**.- Como ya fue mencionado anteriormente, este módulo es el encargado de generar una semilla diferente en cada procesador, para ésto en nuestro diseño se involucra a su vez varios componentes que son:
 - (a) **Semilla distribuida**.- En este componente para la generación de la semilla, es necesario el uso de una distribución $U(a,b)$, con la finalidad de generar semillas diferentes en forma uniforme en cada procesador.
 - (b) **$U(a,b)$** .- Este módulo hace uso de una distribución uniforme, la cual se encarga de generar un número discreto, distribuido en forma uniforme aleatoriamente en un rango a-b.
 - (c) **Semilla**.- El componente semilla de nuestro diseño es el encargado de generar una semilla inicial para la distribución uniforme utilizada en el componente **semilla distribuida**. Para lograr lo anterior, se hará uso del módulo **Rand3**.

- 2) **Núcleo Monte Carlo**.- Los conjuntos de distribuciones que componen el núcleo son:
 - (a) **Distribuciones continuas**: exponencial, gamma, beta, normal, lognormal, cauchy, weibull, ji-cuadrada, t de student, uniforme.
 - (b) **Distribuciones discretas**: binomial, poisson, geométrica, hipergeométrica, uniforme.

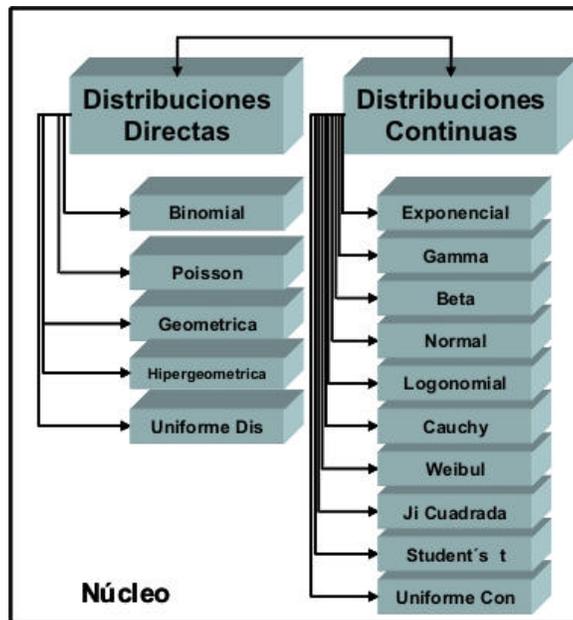


Fig 5.7: Componentes de módulo Núcleo Monte Carlo

La figura 5.7 muestra el diseño del núcleo de Monte Carlo y sus respectivos componentes.

Rand3.- Es un módulo común entre *Núcleo Monte Carlo* y *Semilla MPI*. Se encarga de la generación de variables aleatorias continuas en un rango de 0-1. El rand3 [38] es un método bastante estudiado.

3) **Trayectorias.**- Este módulo comprende al conjunto de métodos para la generación de trayectorias. Los métodos utilizados serán los de: Wiener, Ito y Feynman, los cuales fueron descritos en el capítulo 4.

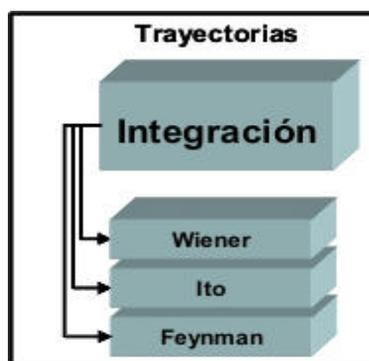


Fig 5.8: Diseño de módulo trayectorias

La figura 5.8 muestra el diseño general del módulo de *trayectorias*. En la próxima sección es presentado el diseño en MPI de cada uno de los componentes que forman el módulo.

5.5 Diseño paralelo para generación de trayectorias

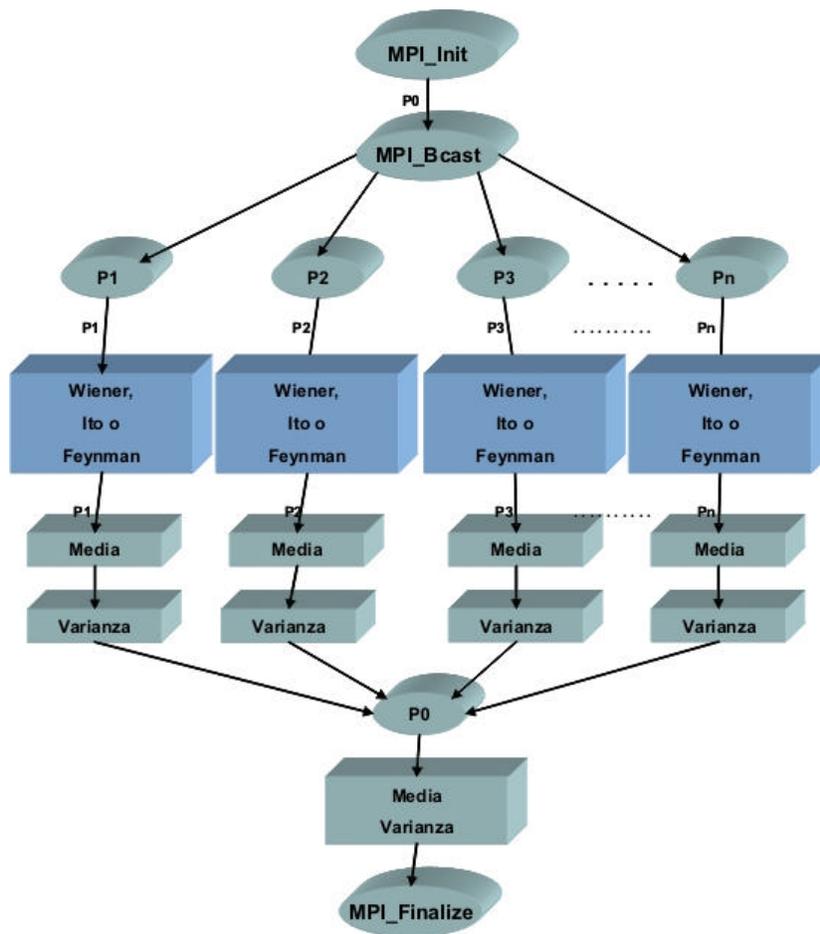


Fig 5.9: Diseño paralelo para generación de trayectorias

La figura 5.9 muestra el diseño en paralelo para la construcción de trayectorias sobre un cluster.

En el diseño paralelo, después de realizar la inicialización del ámbito de MPI, los datos recibidos del usuario son enviados (con un “cast”) a todos los procesadores involucrados en la ejecución, con el fin de que sea realizada la generación de una cierta cantidad de trayectorias por procesador. La creación de las trayectorias se realiza en base a la generación de números aleatorios.

Cada procesador se encarga de calcular el valor de cada una de las trayectorias generadas, esos valores son guardados. El conjunto de valores generados por procesador serán considerados una muestra, con la finalidad de calcular por cada muestra su respectiva

media y varianza. Estos datos se utilizarán para obtener finalmente la media y varianza globales, todo esto con el propósito de llevar a cabo un análisis de varianza e identificar la muestra más representativa del conjunto de trayectorias generadas.

En la figura 5.8 se resalta con un tono diferente, los módulos en donde varía la forma de construcción y cálculo de las trayectorias que pueden ser construidas por Wiener, Ito o Feynman.

Según la taxonomía de Flynn [39], el modelo de programación a utilizar es “*Simple Program Multiple Data*” (por sus siglas en inglés SPMD), en el cual cada nodo se encarga de ejecutar la misma instrucción, trabajando simultáneamente con distinto conjunto de datos.

En nuestro diseño, de la forma en que trabaja el modelo SPMD es como se realizará la ejecución del programa, en donde la parte de generación de trayectorias son las funciones comunes a ejecutar en los diferentes nodos, solo que los datos generados en cada procesador serán diferentes.

5.6 Implementación de solución MPI

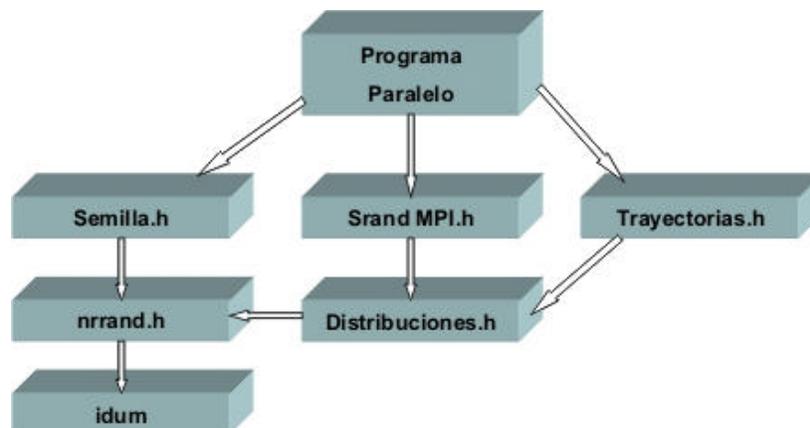


Fig 5.10: Conjunto de bibliotecas y su relación

La implementación de la solución en MPI consta de un conjunto de bibliotecas. La figura 5.10 muestra dicho conjunto y la relación entre ellas.

Es posible que un programa secuencial o paralelo llame a las bibliotecas *semilla.h*, *nrrand.h* y *distribuciones.h*. Las bibliotecas *srandMPI.h* y *trayectorias.h* solo pueden ser llamadas por un programa paralelo desarrollado en MPI. A continuación, se describen las funciones que componen cada biblioteca:

semilla.h.- se compone por la función *long semilla(int my_rank)*, la cual se encarga de generar una semilla en base al tiempo del sistema. El tiempo obtenido del sistema es representado por segundos, dichos segundos son los transcurridos desde el año 1970 hasta la fecha, debido a que más de una instrucción puede ser procesada en un segundo, para asegurar que no es tomada la misma semilla, la función *semilla(int my_rank)* recibe el identificador del procesador en donde se ejecuta la función y es sumado a la variable de retorno.

La función de *semilla(int my_rank)* se limita a un valor de retorno de tipo long, debido a que la estructura que guarda el tiempo obtenido es de ese tipo, por lo que no se puede obtener un número mayor a 2147483647, que es el tamaño máximo a una variable declarada como long.

Además la biblioteca *semilla.h* consta de dos variables importantes definidas en forma global, una es un apuntador llamado *idum*, en el cual se guarda la semilla, valor regresado por la función anteriormente descrita. *idum* es utilizado por la función *ran3* definida en *nrrand.h*. La otra variable definida es *my_rank* valor que se toma del programa principal, para ser enviado a la función *semilla(my_rank)*.

narrand.h.- Se compone de las funciones *ran0(long *idum)*, *ran1(long *idum)*, *ran2(long *idum)* y *ran3(long *idum)*, las cuales se encarga de generar valores aleatorios entre 0-1. El código de estas funciones fue tomado de [38].

Para la generación de números aleatorios en forma uniforme continua, en el trabajo de tesis fue utilizado el *ran3*, debido a que es la base de la generación de números con diferentes comportamientos y a que es considerado un buen generador de números aleatorios.

srandMPI.h.- Esta compuesta por la función *void srandMPI(int my_rank, int nprocs)*, esta función genera las semillas a ser enviadas entre los nodos, en base a la función *d_uniforme_dis(a,b)*, la cual se encuentra en la biblioteca *distribuciones.h*. Los valores enviados a la función son $a = 1$ y $b = 2147483647$, los números regresados por la función van a ser dentro de este rango $a-b$. El valor de b es el número máximo para una variable de tipo long.

distribuciones.h.- Es la biblioteca que compone el núcleo del método de Mote Carlo, consta como ya se había mencionado de dos tipos de funciones, un conjunto de funciones que genera números aleatorios de tipo continuo y el otro conjunto que genera números aleatorios de tipo discreto.

Funciones continuas:

float d_exponencial(int beta).- Regresa un número aleatorio de tipo *float*, con un comportamiento exponencial. El parámetro que recibe corresponde a la duración medida de cierto evento.

float d_gamma(int alpha, int beta).- Regresa un número aleatorio de tipo *float* con comportamiento gamma, dependiendo de los datos de entrada, los parámetros alpha y beta son factores: uno de forma y el otro de escala respectivamente.

float d_beta(float alpha, float beta).- Regresa un número aleatorio tipo *float* con comportamiento beta, las variables de entrada alpha y beta son parámetros de perfil.

float d_normal(float miu, float varza).- Regresa un número aleatorio de tipo *float* con un comportamiento normal. Los parámetros de entrada *miu* y *varza* corresponden a la media y varianza para la generación del número.

float d_logonormal(float miu, float vanza).- Regresa un número aleatorio de tipo *float* con comportamiento logonormal, los parámetros de entrada *miu* y *vanza*, corresponden a la media y varianza respectivamente.

float d_cauchy(float alpha, float beta).- Regresa un número aleatorio tipo *float* con comportamiento cauchy

float d_weibull(float alpha, float beta).- Regresa un número aleatorio tipo *float* con comportamiento weibull

float d_chi_cuadrada(int k).- Regresa un número aleatorio tipo *float* con comportamiento chi-cuadrada. Su parámetro de entrada corresponde a los grados de libertad de la función.

float d_students_t(int k).- Regresa un número aleatorio tipo *float* con comportamiento t de "Student". Su parámetro de entrada corresponde a los grados de libertad, como en el caso anterior.

float d_uniforme_cont(int a, int b).- Regresa un número aleatorio de tipo *float*. Sus parámetros de entrada corresponden a un intervalo en el que va a ser generado el número donde: *a* es el valor mínimo y *b* es el valor máximo del intervalo.

Funciones discretas:

long d_binomial(int n, float p).- Regresa un número de forma aleatoria de tipo *long*, con comportamiento binomial. Sus parámetros de entrada *n* y *p* corresponden al número de pruebas realizadas y a la probabilidad de éxito, respectivamente.

long d_poisson(float lambda).- Regresa un número aleatorio de tipo long, con comportamiento Poisson, en donde su parámetro ***lambda*** corresponde a un valor esperado de sucesos en un intervalo dado.

long d_geometrica(float p).- Regresa un número aleatorio de tipo long, con un comportamiento geométrico, en donde su parámetro ***p*** corresponde a una probabilidad hasta la aparición de un primer éxito.

long d_hyper_geo(int n, int m, int n1).-

long d_uniforme_dis(int a, int b).- Regresa un número aleatorio, con comportamiento uniforme, el número es generado del intervalo a-b, en donde ***a*** corresponde al límite inferior y ***b*** al límite superior de dicho intervalo.

trayectorias.h.- Se compone de las funciones para la creación de trayectorias:

tray_wiener(int NP, int ntrayects ,int nprocs).- Esta función se encarga de generar un número ***n*** de trayectorias brownianas por medio del método de Wiener descrito en la sección (4.1). Los parámetros corresponden al número de puntos por trayectoria y el número de trayectorias respectivamente, además del número de procesos (***nprocs***). Para la construcción de las trayectorias, a su vez manda llamar internamente a otro conjunto de funciones, las cuales son: ***srandMPI(int my_rank, int nprocs)***, ***void browt(int NP, in N)***, ***void funal(int NP)***, ***void orvar(), float d_normal(0,1)***.

tray_Ito(int NP, int ntrayects ,int a, int b,int nprocs).- Esta función se encarga de generar un número ***n*** de trayectorias por medio del método de Ito descrito en la sección (4.2). Los parámetros corresponden al número de puntos por trayectoria (***NP***), número de trayectorias (***ntrayects***), punto inicial (***a***), punto final (***b***) y el número de procesos (***nprocs***). Para la construcción de las trayectorias, a su vez manda llamar internamente a otro conjunto de funciones, las cuales son: ***srandMPI(int my_rank, int nprocs)***, ***void asignaXkB(int NP)***, ***float valorIT(int NP)***.

tray_Feynman(int NP, int ntrayects, float ITiem, int nprocs, float (*accion)(int NP ,float ITiem)).- Esta función se encarga de generar un número ***n*** de trayectorias por medio del método de Feynman descrito en la sección (4.3). Los parámetros corresponden al número de puntos por trayectoria (***NP***), el número de trayectorias (***ntrayects***), el incremento del tiempo (***ITiem***), número de procesos (***nprocs***), un apuntador a una función (****accion***), la cual corresponde a la ***acción*** en la integral de Feynman. Para la construcción de las trayectorias, se llama internamente a otro conjunto de funciones, las cuales son:

srandMPI(int my_rank, int nprocs), float valorIT_Expo(ntrayects), double valorIT_Accion(int NP), void generaPuntos(int NP,float ITiem), void asignaTiem(int NP,float ITiem).

En nuestro caso se creo una función para la **acción**. Para efecto de pruebas, la función implementada es: ***void apFuncion(int NP,float ITiem,float m,float w)***, que corresponde al oscilador armónico. Recibe como parámetros el número de puntos, el incremento del tiempo, la masa y la frecuencia.

Para uso de la función ***tray_Feynman***, el parámetro que recibe la función que corresponde a la acción, únicamente tiene de entrada los parámetros: número de puntos e incremento del tiempo, forma en que debe ser representada la función de entrada (posición, tiempo), lo anterior con el propósito de no limitarla a cierto número de datos. Por lo tanto, la función que recibe ***tray_Feynman***, debe tener bien definidos internamente los parámetros correspondientes al problema tratado.

Además de la creación de trayectorias, como fue descrito en el diseño: las funciones se encargan de calcular la media y varianza del conjunto de trayectorias generadas por procesador. Se envían al final de sus cálculos dichos valores al modo principal, para que este a su vez (con el uso de los datos recibidos) calcule la media y varianza globales. Dos archivos son generados, un archivo en el que es guardado el conjunto de trayectorias creadas, en el segundo son almacenados los datos correspondientes a cada nodo como son sus medias y varianzas muestrales, además de incluir en dicho archivo la media y varianza globales.

Las funciones están implementadas en C y MPI, para su ejecución en un cluster en forma paralela.

5.7 Arquitectura para computación Grid

Para el diseño en computación de red amplia, se hace uso de una herramienta llamada Taverna, en la cual es aplicado el concepto de computación distribuida en red amplia. A continuación se describe Taverna con un poco más de detalle:

Taverna.- Una iniciativa relacionada con los inicios de Taverna es la e-Science [40] en el 2001, en aquel tiempo pretendía crear una nueva forma de colaboración, haciendo uso de los recursos computacionales entre diferentes instituciones. Lo que se planea lograr creando una “Grid” mundial que permita compartir recursos e intercambiar información entre redes heterogéneas. Con la aparición de la e-Science, el uso de servicios es muy útil y muy importante en el área de la investigación.

En el proyecto *myGrid*, se hace uso de servicios Web para el intercambio de información. Este tipo de servicios permite integrar sistemas heterogéneos y desarrollar sistemas distribuidos.

Taverna es un sistema para el desarrollo de aplicaciones combinando componentes y repositorios de información locales y/o remotos. El área de trabajo de *Taverna* maneja los servicios Web como componentes, soporta la creación y edición de flujos de trabajo (workflows), haciendo uso de su explorador.

El “workbench enactor” provee una interfaz para la ejecución de los flujos de trabajo [41]. La interacción con el usuario es de forma gráfica, lo cual facilita a los desarrolladores su uso para la creación de aplicaciones (flujos de trabajo) [42]; sólo se tiene que seleccionar y combinar con el “mouse” los componentes de acuerdo a la lógica de la aplicación deseada. Los resultados de dicha aplicación son presentados de forma transparente al usuario

Actualmente, se requiere de otro tipo de servicios para la resolución de problemas. Nuestro diseño se compone del mismo núcleo de Monte Carlo presentado en la sección 5.6, pero en este caso desarrollado como servicios Web, para su incorporación como componentes dentro de Taverna, además del desarrollo de dos servicios más que se componen de los métodos para la generación de trayectorias de Wiener e Ito.

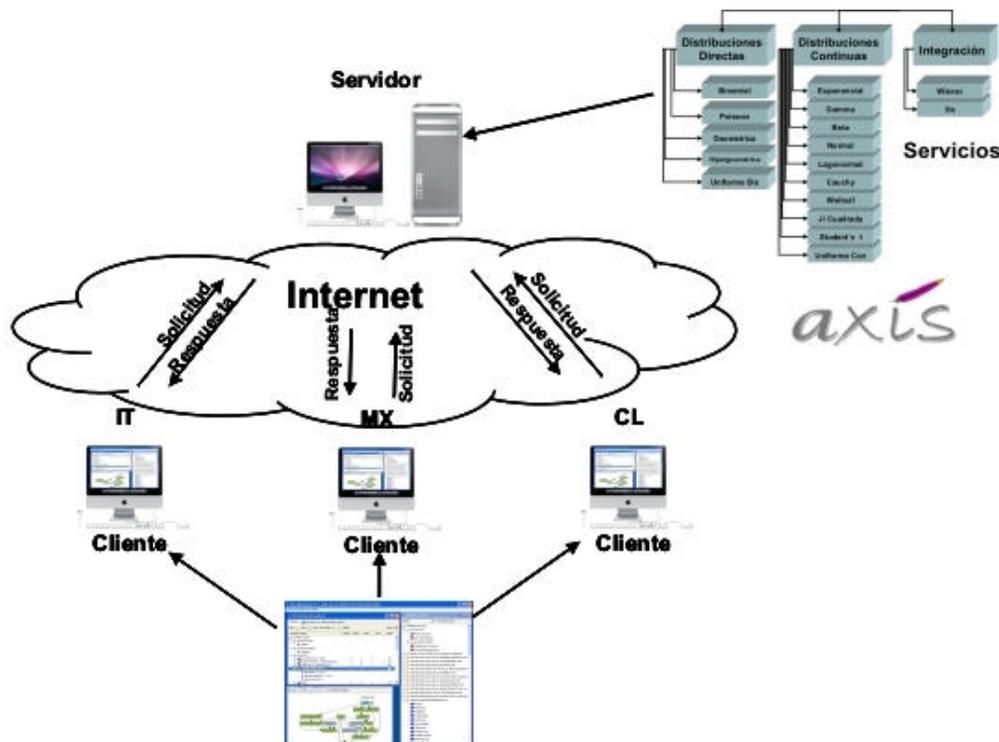


Fig 5.11: Arquitectura para el diseño en Taverna

La figura 5.11 muestra la arquitectura necesaria para el diseño en Taverna, para la creación de aplicaciones potencialmente distribuidas en base a servicios Web. Es necesario el uso de Internet, ya que permite el uso de redes heterogéneas a través de protocolos como SOAP, TCP/IP y lenguajes como XML, WSDL, los cuales permiten la transferencia de información entre diferentes redes y sistemas.

Servidor.- Es el equipo en donde van a estar disponibles a través de Internet los servicios Web a desarrollar, de los cuales está compuesto el núcleo de Monte Carlo y la parte de integración. El servidor es el encargado de recibir las solicitudes de los clientes, la ejecución de dicha solicitud y al final se encarga del envío de la respuesta al cliente. La respuesta se compone del resultado de la ejecución del servicio solicitado.

Apache Axis.- Es un ambiente de código abierto basado en SOAP que permite el desarrollo, la publicación, la ejecución y la administración de servicios Web. Es indispensable tenerlo disponible en el *servidor*.

Servicios.- Son los servicios Web desarrollados, los cuales son tratados como componentes dentro de Taverna. El conjunto de servicios disponibles se componen de un grupo de distribuciones discretas, otro grupo que corresponde a distribuciones continuas, además de dos componentes de la parte de Integración.

Cliente.- La máquina cliente es la encargada de realizar la solicitud al servidor, para ello, es necesario que el equipo este conectado a la Internet y tener instalado Taverna, el cual va permitir a la máquina cliente realizar las solicitudes de los servicios a los diferentes servidores que compongan un flujo de trabajo.

Taverna.- Es la herramienta que a su vez funciona como un software cliente. Como fue descrito anteriormente, permite la creación y ejecución de flujos de trabajo. Es el software diseñado para encargarse de las solicitudes al servidor de los componentes (servicios) que conforman una aplicación desarrollada en base a servicios. Todo esto a través de Internet.

Internet.- Es muy conocida la red Internet, algunas definiciones que se le han dado desde su aparición es como: método de interconexión descentralizado de redes de computadoras, garantiza que redes heterogéneas funcionen como una red lógica única, de alcance mundial.

En la arquitectura presentada anteriormente, podría verse como el ambiente que hace propicia la comunicación entre cliente y servidor, a través de la cual se realizan las solicitudes por parte de los clientes, y el servidor puede enviar su respuesta (resultados de la ejecución de los servicios).

5.8 Diseño en Taverna

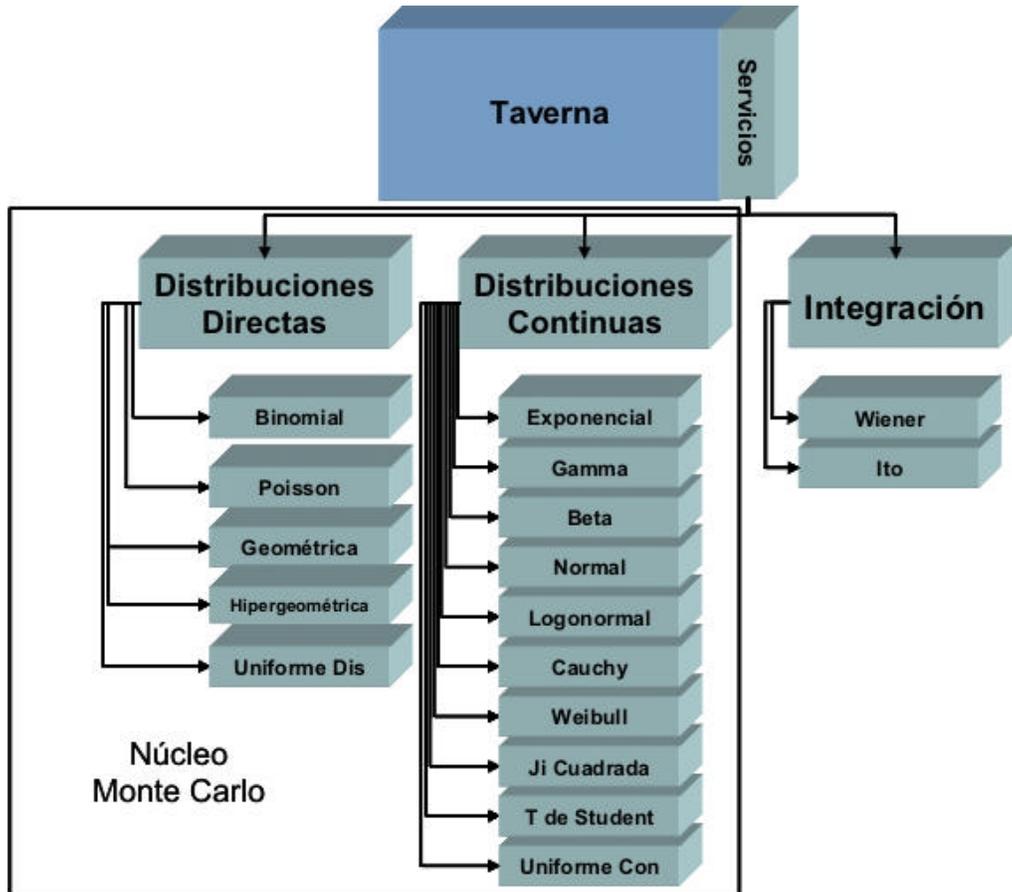


Fig 5.12: Diseño del conjunto de servicios a desarrollar

La figura 5.12 muestra el diseño del núcleo del método de Monte Carlo para ser utilizado como componentes dentro de Taverna.

De igual forma que se hizo en el diseño en paralelo, el núcleo de Monte Carlo en Taverna, va a estar compuesto de distribuciones continuas, discretas y una parte de integración.

Serán desarrollados cada uno de los siguientes servicios:

- (a) **Distribuciones continuas.**- Exponencial, gamma, beta, normal, logonormal, cauchy, weibull chi-cuadrada, t de student y uniforme continua.

- (b) **Distribuciones discretas.**- Binomial, poisson, geométrica, hipergeométrica y uniforme discreta.
- (c) **Integración.**- Wiener e Ito, los cuales corresponden al conjunto de servicios que componen la parte de integración. Estos módulos hacen uso de los métodos para la generación de trayectorias, descritos en las secciones (4.1 y 4.2) respectivamente.

Los servicios, serán incorporados a Taverna, para hacer posible su uso en problemas modelados como flujos de trabajo que hagan uso del núcleo.

Para la representación de un componente dentro de Taverna se da un ejemplo gráfico a continuación:

Cada uno de los componentes cuenta ya sea con una entrada o un conjunto de entradas y una salida. La salida puede ser de un tipo primitivo o un arreglo. En caso de que sea un arreglo es posible tener de salida un conjunto de resultados de dicho componente.

Para su representación gráfica se ve al componente parecido a una caja negra, en donde, resaltan sus entradas y su salida, debido a que las entradas pueden ser conectadas a otros servicios y ser resultado de un componente ejecutado anteriormente.

En nuestro diseño, las entradas corresponden a los parámetros necesarios para cada componente según sus características, con la finalidad de generar números aleatorios o la creación de una trayectoria. Como ejemplo se presenta un esquema del componente para la distribución binomial:

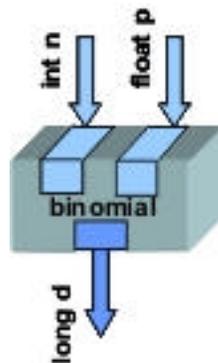


Fig 5.13: Ejemplo de Componente en Taverna.

La figura 5.13 muestra la representación de un componente en nuestro diseño, las flechas representan las entradas y salidas de datos de los componentes.

Donde:

n.- Número de ensayos.

p.- Probabilidad constante de éxito.

d.- Número aleatorio generado.

La salida de las distribuciones será conectada a otros componentes, con la finalidad de enviar números aleatorios o el valor de una trayectoria generada. Estos componentes, después de recibir los datos, van a poder hacer uso de los resultados.

5.9 Implementación en Taverna

Para la creación de los servicios Web, fue necesaria su previa implementación en Java y el apoyo en Apache Axis, con el cual se generaron los archivos WSDL necesarios para el manejo de los servicios Web. Posteriormente al tener los servicios listos en el servidor, es posible que sean llamados desde Taverna por medio del URL en donde se localizan.

La clase generada que contiene los métodos correspondientes al núcleo de Monte Carlo se llama *Distribuciones*, para la generación de los números aleatorios, algunos de los métodos se apoyan en la clase *rand3*, la cual se encarga de generar números aleatorios uniformemente distribuidos, el código utilizado para esta clase es el mismo que se obtuvo en el libro “Numerical Recipes” para *rand3*. La clase con los métodos Wiener e Ito, fue llamada *Integracion*.

Una clase en un diagrama es representada por un solo elemento indicando sus métodos y atributos. En este caso para mostrar con más detalle los métodos contenidos en cada una de las clases, fue considerado, dentro de un diagrama representar a cada método como una clase en sí. A continuación se presenta el diagrama que muestra el conjunto de métodos que componen la parte de integración.

Integración

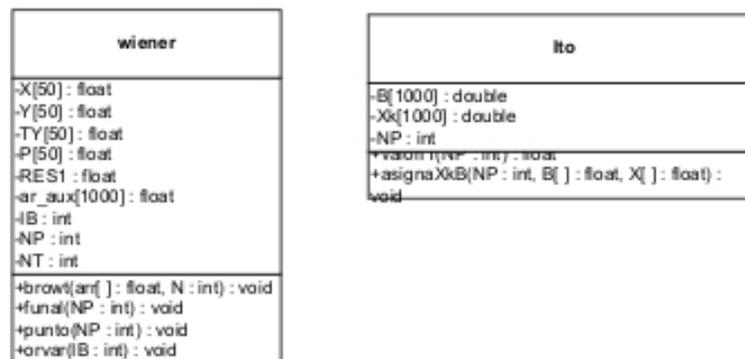


Fig 5.14: Diagrama de los métodos Wiener e Ito

La figura 5.14 muestra los diagramas correspondientes a los métodos Wiener e Ito, que comprenden a la parte de integración en el diseño.

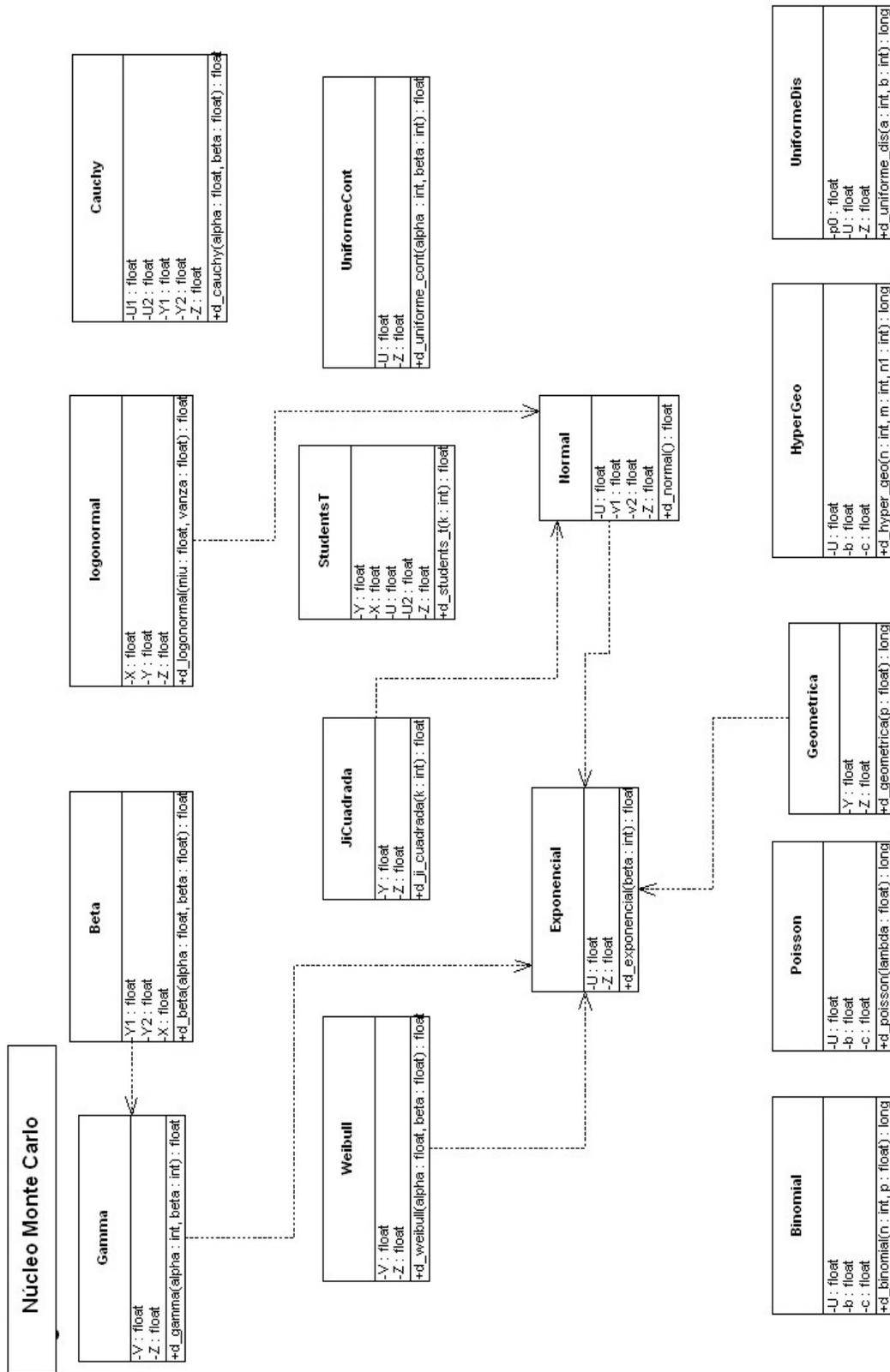


Fig 5.15 Diagrama núcleo Monte Carlo

La figura 5.15 muestra el diagrama de los métodos implementados en Java, necesarios para generar los servicios Web.

La figura 5.16 a) y b) muestran el conjunto de servicios creados con sus respectivas entradas y salidas, los cuales se incorporarán a Taverna.

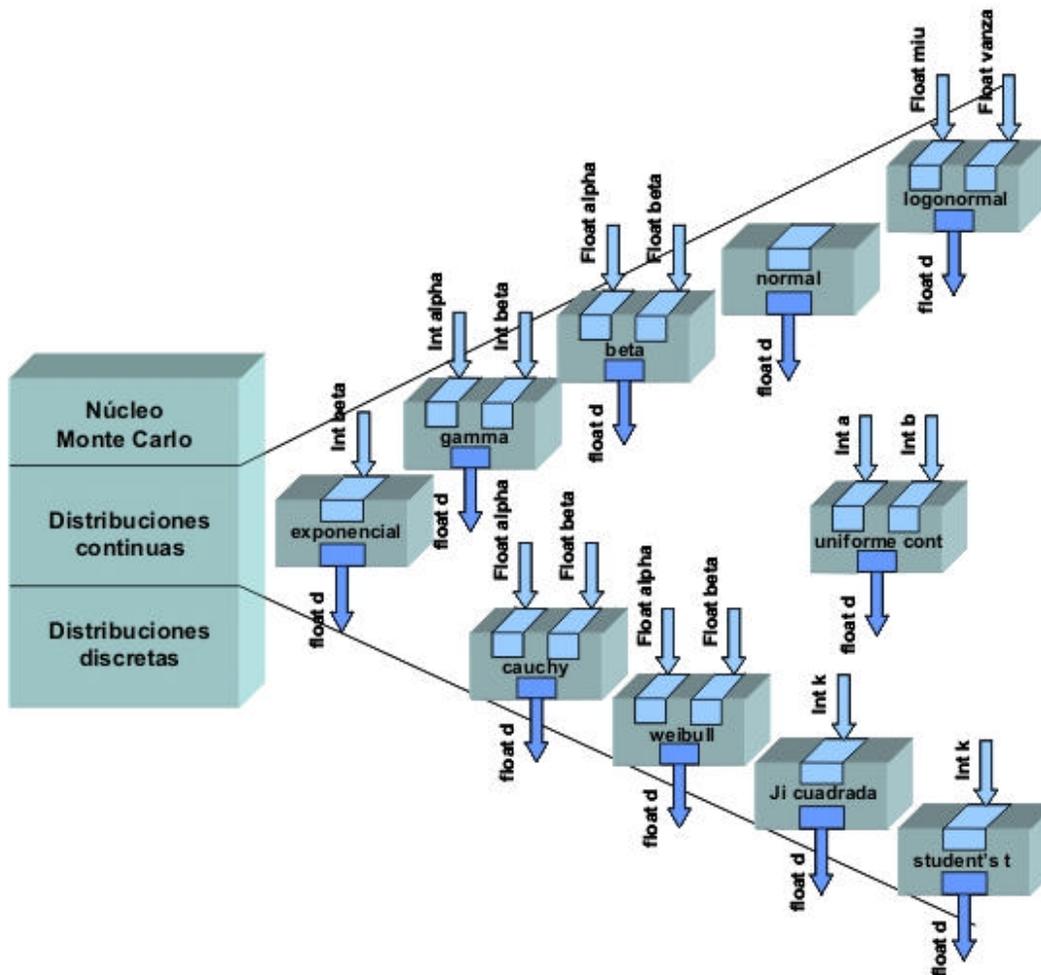


Fig 5.16: a) Servicios del núcleo Monte Carlo

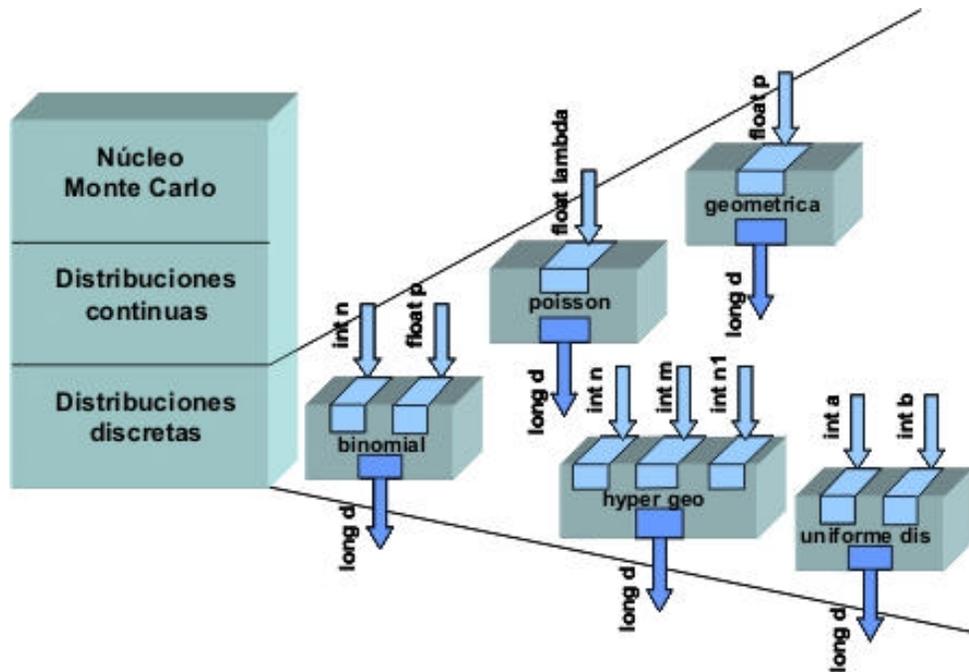


Fig 5.16: b) Servicios del núcleo Monte Carlo

Las figuras 5.16 a) y b) muestran a detalle los servicios Web desarrollados, tanto para generación de números aleatorios continuos como discretos, de los cuales se compone el núcleo de Monte Carlo en Taverna. Para mayor detalle del servicio, las imágenes a su vez muestran los tipos de datos definidos para las entradas y salidas.

La descripción realizada en cada una de las funciones de la sección (5.6) corresponde análogamente a cada uno de los servicios expuestos en la presente sección.

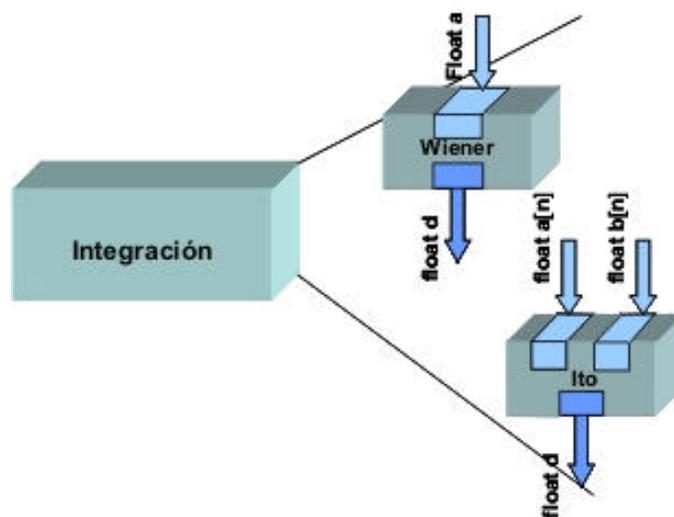


Fig 5.17: Servicios Web módulo integración

La figura 5.17 muestra el conjunto de servicios Web implementados para la parte de integración en Tavera. La imagen muestra los tipos de datos de entrada y salida. Para el desarrollo de estos servicios únicamente se tomó en cuenta la parte secuencial de las funciones descritas en la sección.

Capítulo 6

Resultados

En el presente capítulo se muestran los resultados de rendimiento para la generación de trayectorias utilizando ambientes de alto desempeño como computación local y computación de red amplia. Se incluyen tablas que muestran el número de puntos y las trayectorias generadas por ejecución y por procesador. Posteriormente para ilustrar de una mejor forma los resultados obtenidos, se incluyen gráficas de tiempos y trayectorias generadas por cada uno de los métodos implementados en la presente tesis. Lo descrito anteriormente está enfocado a una computación paralela.

Con respecto a la computación de red amplia, se muestra una lista de los componentes (servicios Web) desarrollados para Taverna. Cada uno de los componentes son presentados en una tabla con sus respectivas entradas y salidas. Además se incluye una pequeña descripción de su función dentro del área de trabajo de Taverna.

6.1 Resultados de cómputo paralelo

Los productos obtenidos con el diseño e implementación realizados para la parte de cómputo paralelo son las siguientes bibliotecas.

semilla.h.- permite obtener la semilla por medio del tiempo de una computadora. Es posible su uso tanto en forma secuencial como paralela.

nrrand.h.- contiene la función `ran3` (código obtenido del libro “Numerical Recipes” [38]), la cual genera números aleatorios uniformes entre 0 - 1. Este código fue seleccionado debido a que se requería de un buen generador de números aleatorios uniformes, ya que estos son la base de los diferentes algoritmos que componen el núcleo desarrollado y descrito en el capítulo 5. El algoritmo que permitió implementar `ran3` es bastante estudiado y analizado, lo cual garantiza un buen generador.

distribuciones.h.- contiene las funciones para la generación de números aleatorios con diferentes comportamientos, dependiendo de la distribución de la función.

srandMPI.h.- destinada a ser utilizada en un ambiente distribuido para la generación de semillas diferentes, en varios procesadores, en forma paralela. Esta biblioteca permite obtener números aleatorios diferentes en cada procesador, y por ende el conjunto de trayectorias aleatorias generadas por procesador es diferente. En el caso de generar una sola

semilla, nos llevaría a obtener trayectorias iguales en los diferentes procesadores, ya que los números aleatorios obtenidos no son precisamente aleatorios, sino pseudo-aleatorios, debido a que su generación se basa en un método matemático, haciendo uso de una semilla inicial.

trayectorias.h.- contiene las funciones para la generación de trayectorias por medio de los métodos de Wiener, Ito y Feynman. Esta biblioteca hace uso de las bibliotecas anteriormente mencionadas.

Fueron realizadas varias pruebas con el fin de obtener resultados de la implementación en paralelo de los métodos mencionados.

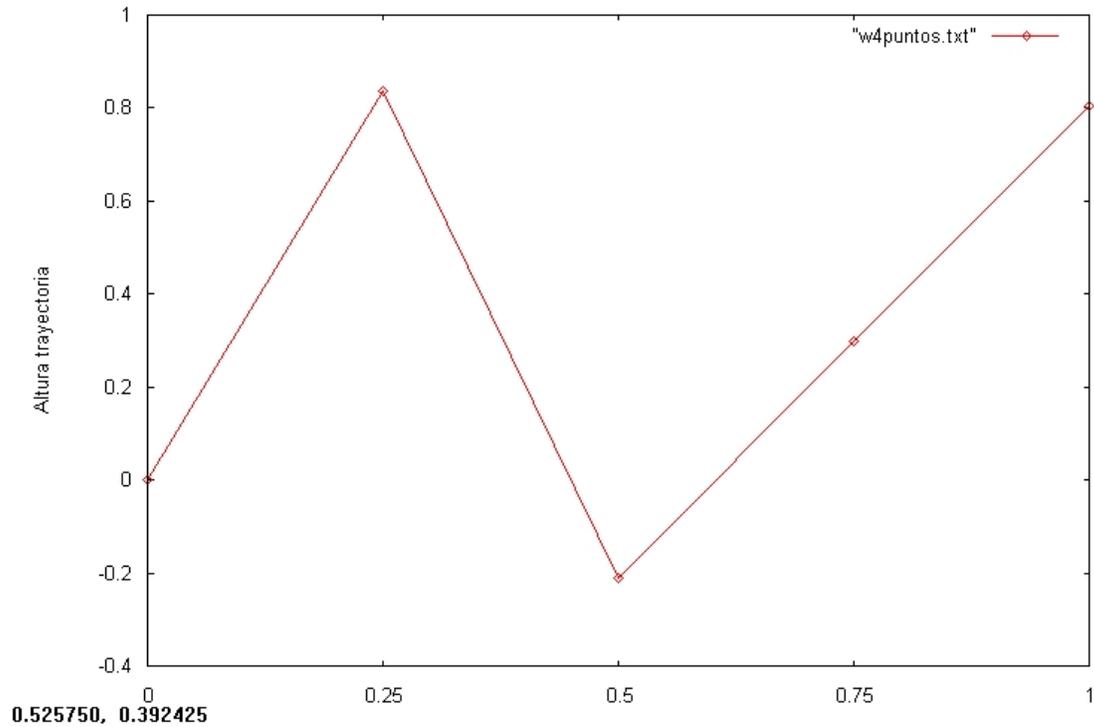
6.2 Resultado de las pruebas realizadas a Wiener.

Para la obtención de resultados, se realizaron varias ejecuciones, haciendo una combinación en la cantidad de puntos por trayectoria, número de trayectorias y cantidad de procesadores. A continuación, se presentan varias tablas con las ejecuciones detalladas para este método. Además se muestran gráficamente algunas de las trayectorias y sus tiempos de ejecución.

Tabla para trayectorias con 4 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	4000	4000	1000
1000	5	4000	800	200
1000	10	4000	400	100
1000	20	4000	200	50
10000	1	40000	40000	10000
10000	5	40000	8000	2000
10000	10	40000	4000	1000
10000	20	40000	2000	500
100000	1	400000	400000	100000
100000	5	400000	80000	20000
100000	10	400000	40000	10000
100000	20	400000	20000	5000
1000000	1	4000000	4000000	1000000
1000000	5	4000000	800000	200000
1000000	10	4000000	400000	100000

Tabla 6.1: Ejecuciones realizadas para Wiener con 4 puntos

La tabla 6.1 muestra las pruebas realizadas para el método de Wiener. La prueba consistió en la generación de 1000, 10000, 100000 y 1000000 trayectorias con 4 puntos para 1, 5, 10 y 20 procesadores. A continuación, se muestra la trayectoria resultante de estas pruebas.



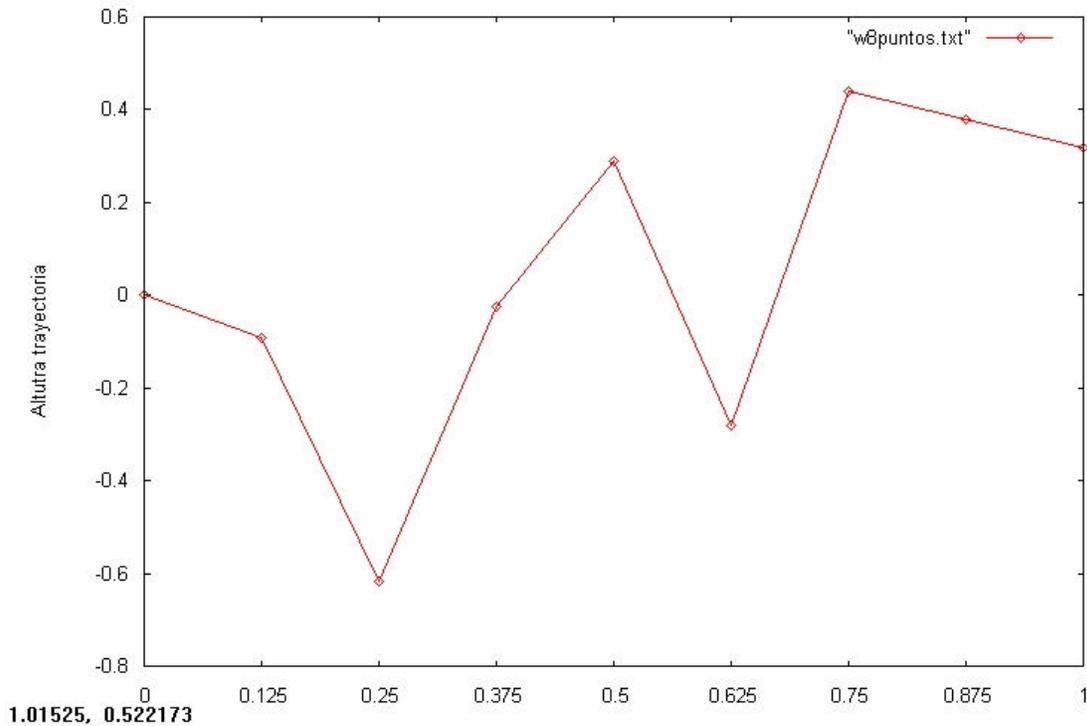
Gráfica 6.1: Trayectoria con 4 puntos de Wiener

La gráfica 6.1 muestra un ejemplo de una trayectoria aleatoria generada por el método de Wiener.

Tabla para trayectorias con 8 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	8000	8000	1000
1000	5	8000	1600	200
1000	10	8000	800	100
1000	20	8000	400	50
10000	1	80000	80000	10000
10000	5	80000	16000	2000
10000	10	80000	8000	1000
10000	20	80000	4000	500
100000	1	800000	800000	100000
100000	5	800000	160000	20000
100000	10	800000	80000	10000
100000	20	800000	40000	5000
1000000	1	8000000	8000000	1000000
1000000	5	8000000	1600000	200000
1000000	10	8000000	800000	100000

Tabla 6.2: Ejecuciones realizadas para Wiener con 8 puntos

La tabla 6.2 muestra las pruebas realizadas para el método de Wiener. La prueba consistió en la generación de 1000, 10000, 100000 y 1000000 trayectorias, pero ahora con 8 puntos para 1, 5, 10 y 20 procesadores. A continuación, se muestra la trayectoria resultante de estas pruebas.



Gráfica 6.2: Trayectoria con 8 puntos de Wiener

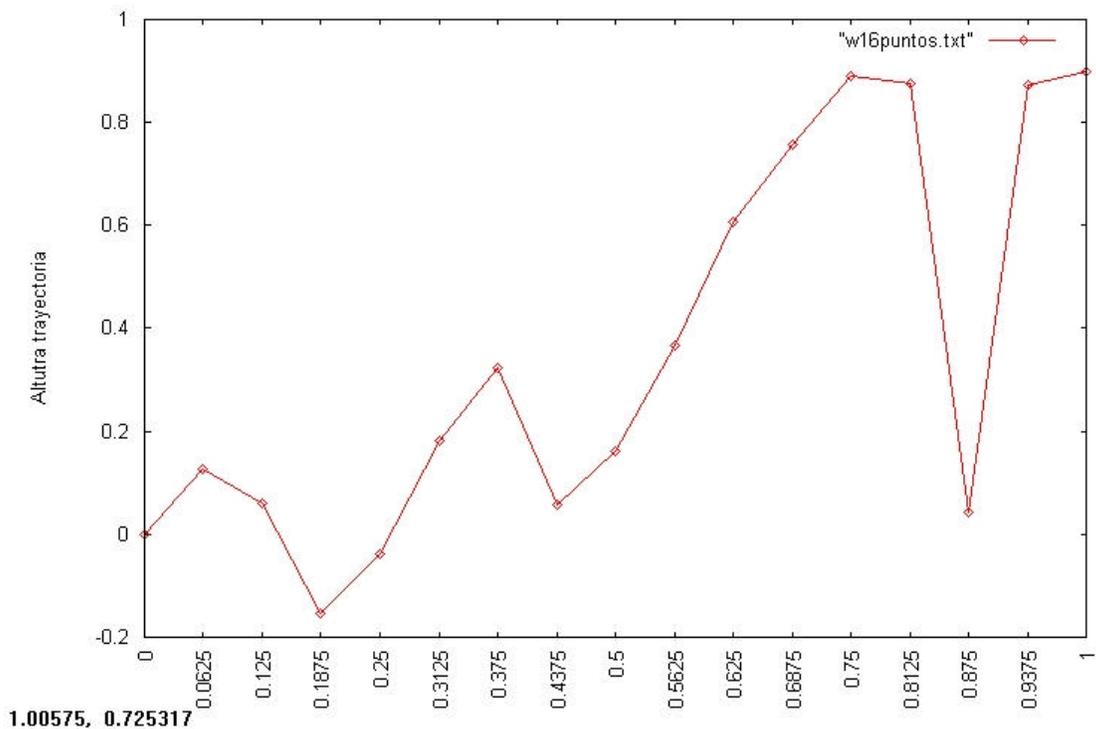
La gráfica 6.2 muestra el ejemplo de una trayectoria aleatoria generada por el método de Wiener con 8 puntos.

Tabla para trayectorias con 16 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	16000	16000	1000
1000	5	16000	3200	200
1000	10	16000	1600	100
1000	20	16000	800	50
10000	1	160000	160000	10000
10000	5	160000	32000	2000
10000	10	160000	16000	1000
10000	20	160000	8000	500
100000	1	1600000	1600000	100000
100000	5	1600000	320000	20000
100000	10	1600000	160000	10000
100000	20	1600000	80000	5000
1000000	1	16000000	16000000	1000000
1000000	5	16000000	3200000	200000
1000000	10	16000000	1600000	100000

Tabla 6.3: Ejecuciones realizadas para Wiener con 16 puntos

La tabla 6.3 muestra las pruebas realizadas para el método de Wiener. La prueba en esta ocasión consistió en la generación de 1000, 10000, 100000 y 1000000 trayectorias con 16

puntos cada una, para 1, 5, 10 y 20 procesadores. A continuación se muestra la trayectoria resultante de estas pruebas.



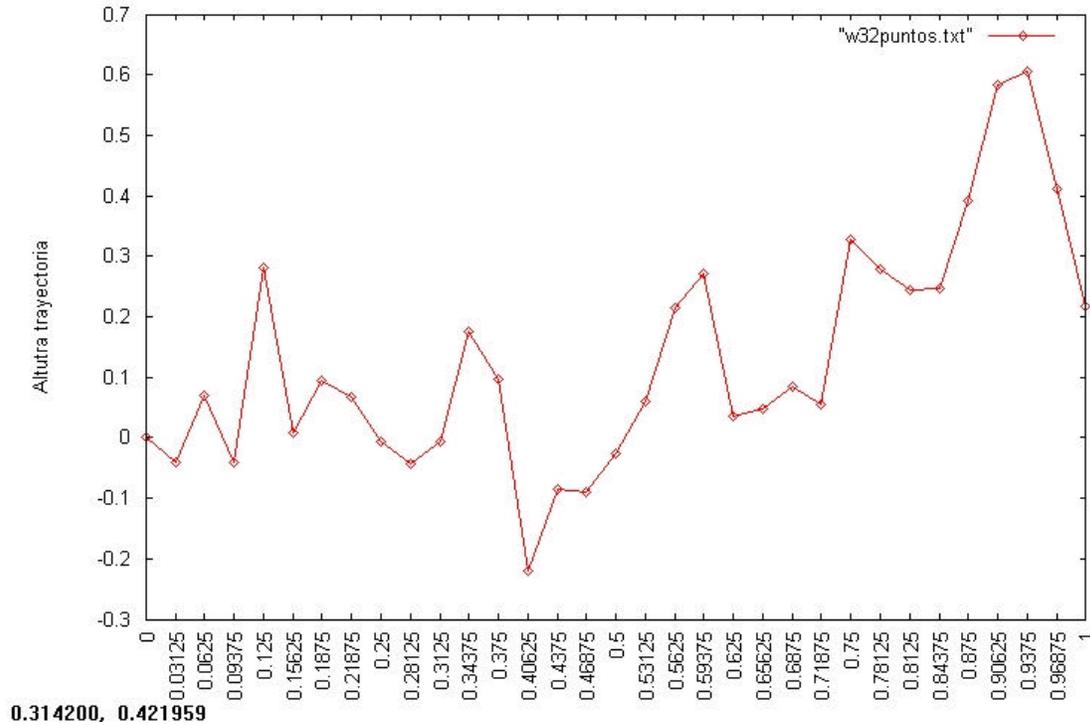
Gráfica 6.3: Trayectoria con 16 puntos de Wiener

La gráfica 6.3 muestra el ejemplo de una trayectoria aleatoria generada por el método de Wiener con 16 puntos.

Tabla para trayectorias con 32 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	32000	32000	1000
1000	5	32000	6400	200
1000	10	32000	3200	100
1000	20	32000	1600	50
10000	1	320000	320000	10000
10000	5	320000	64000	2000
10000	10	320000	32000	1000
10000	20	320000	16000	500
100000	1	3200000	3200000	100000
100000	5	3200000	640000	20000
100000	10	3200000	320000	10000
100000	20	3200000	160000	5000
1000000	1	32000000	32000000	1000000
1000000	5	32000000	6400000	200000
1000000	10	32000000	3200000	100000

Tabla 6.4: Ejecuciones realizadas para Wiener con 32 puntos

La tabla 6.4 muestra las pruebas realizadas para el método de Wiener. La prueba en esta ocasión consistió en la generación de 1000, 10000, 100000 y 1000000 trayectorias con 32 puntos cada una, para 1, 5, 10 y 20 procesadores. A continuación se muestra la trayectoria resultante de estas pruebas.

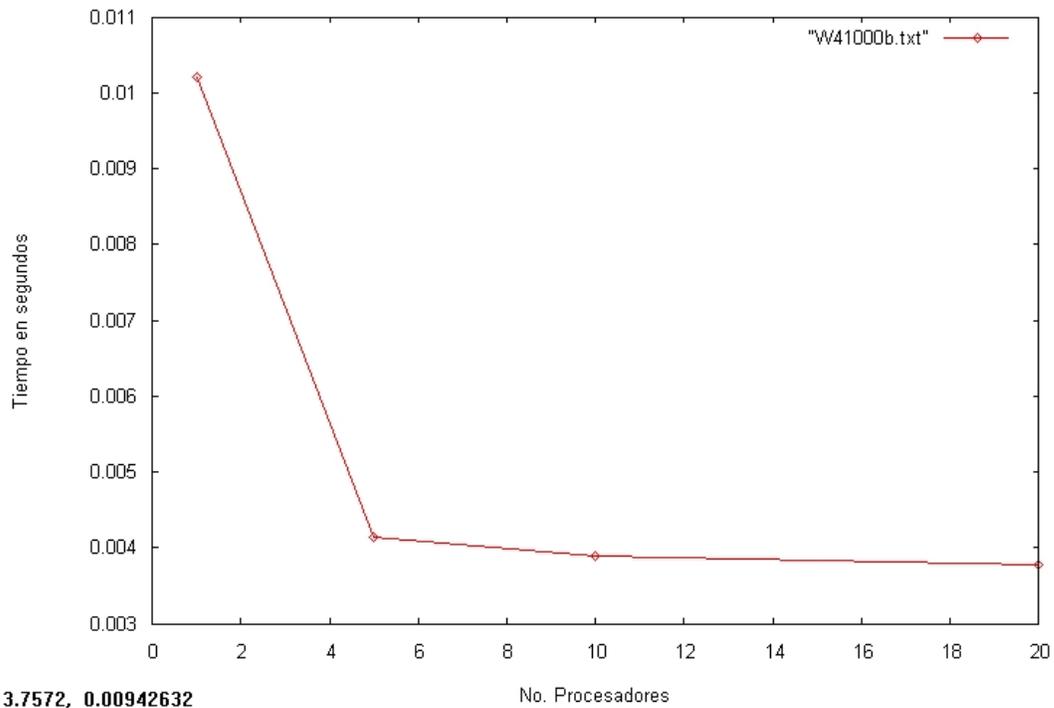


Gráfica 6.4: Trayectoria con 32 puntos de Wiener

La gráfica 6.4 muestra el ejemplo de una trayectoria aleatoria generada por el método de Wiener con 32 puntos.

En las figuras 6.1, 6.2, 6.3 y 6.4, en donde se muestran ejemplos de trayectorias aleatorias generadas con diferente número de puntos por el método de Wiener, se puede observar que conforme aumenta el número de puntos de la trayectoria, ésta se va suavizando, lo cual se puede ver en la definición que da a la trayectoria el aumento de puntos.

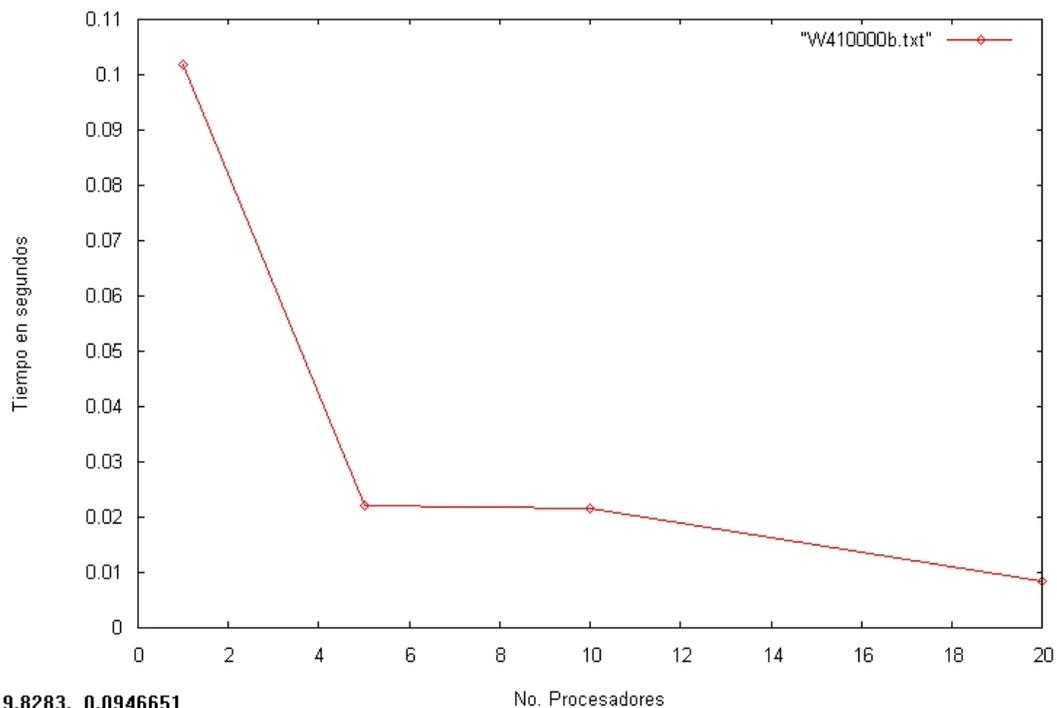
6.2.1 Tiempos de ejecución para la integral de Wiener con 4 puntos



13.7572, 0.00942632

Gráfica 6.5: Tiempos de ejecución para 1000 trayectorias de Wiener

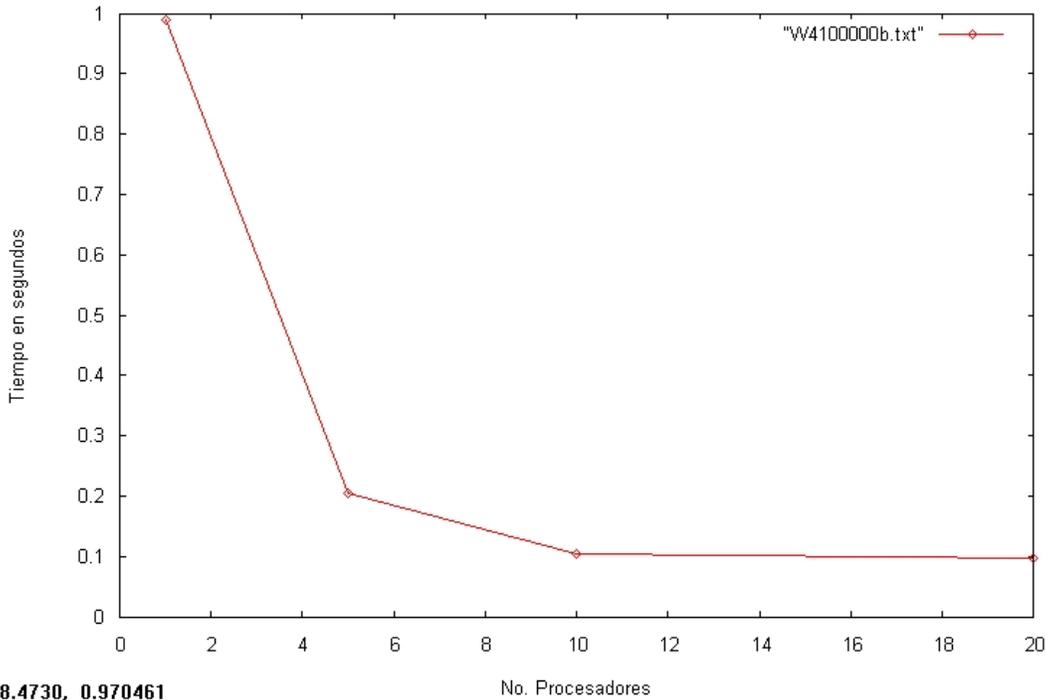
La gráfica 6.5 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias con 4 puntos. Los tiempos graficados son los correspondientes a las ejecuciones presentadas en la tabla 1.



19.8283, 0.0946651

Gráfica 6.6: Tiempos de ejecución para 10000 trayectorias de Wiener

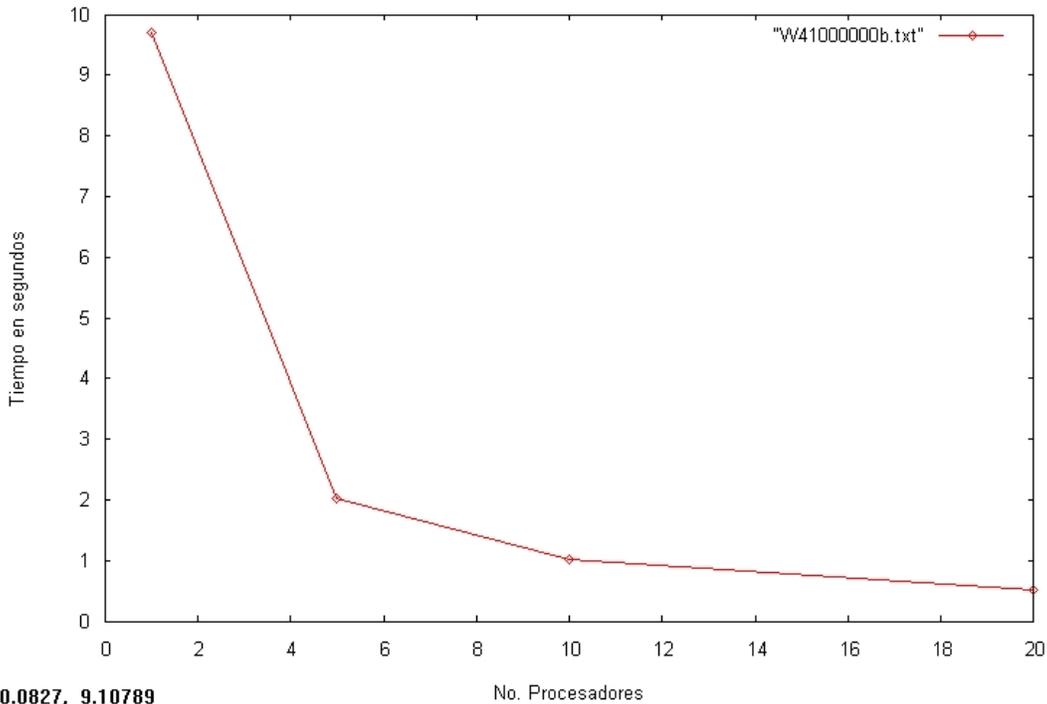
La gráfica 6.6, muestra los tiempos obtenidos de las ejecuciones para la generación de 10000, trayectorias aleatorias con 4 puntos, los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 1.



18.4730, 0.970461

Gráfica 6.7: Tiempos de ejecución para 100000 trayectorias de Wiener

La gráfica 6.7, muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias con 4 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 1.

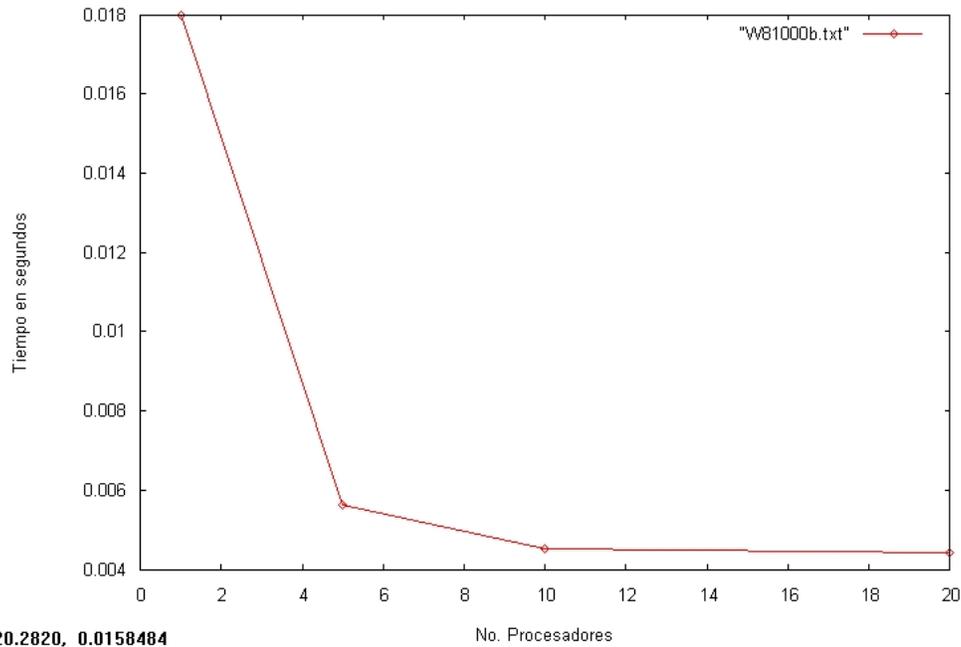


20.0827, 9.10789

Gráfica 6.8: Tiempos de ejecución para 1000000 de trayectorias de Wiener

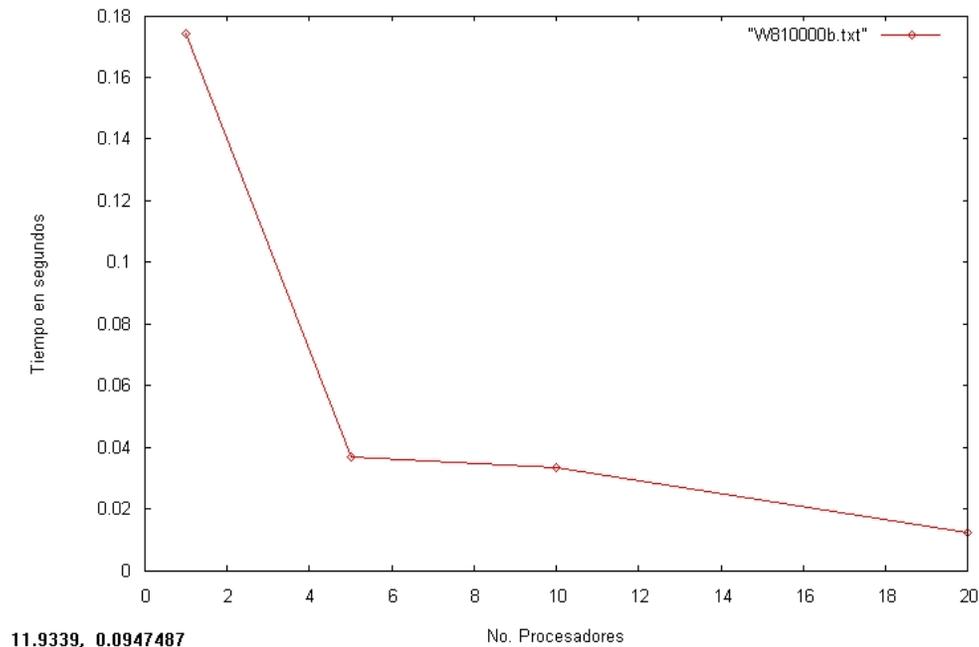
La gráfica 6.8 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 4 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 1.

6.2.2 Tiempos de ejecución para la integral de Wiener 8 puntos



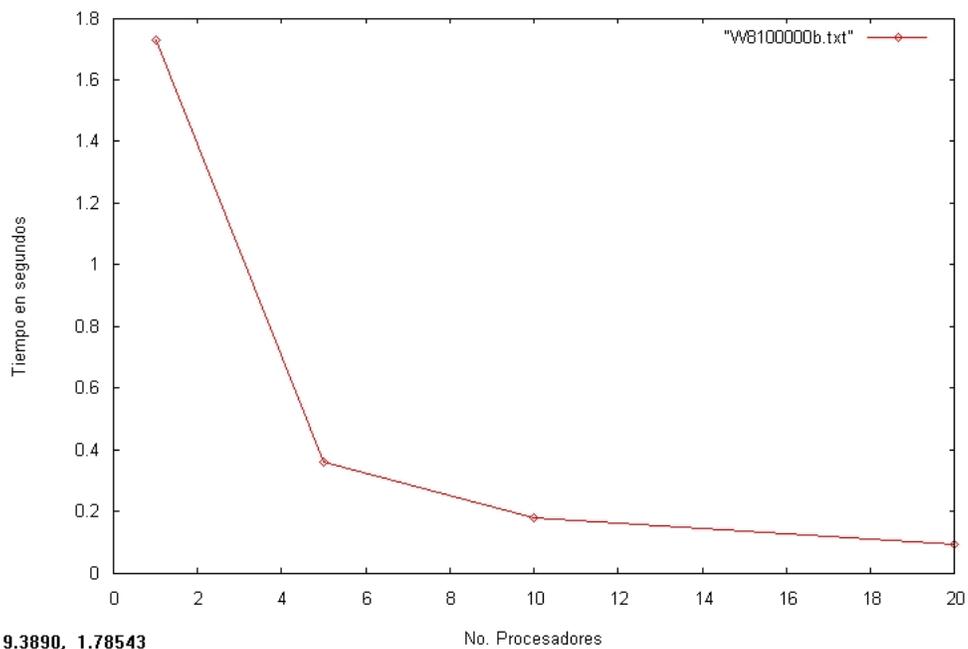
Gráfica 6.9: Tiempos de ejecución para 1000 trayectorias de Wiener

La gráfica 6.9 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias con 8 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 2.



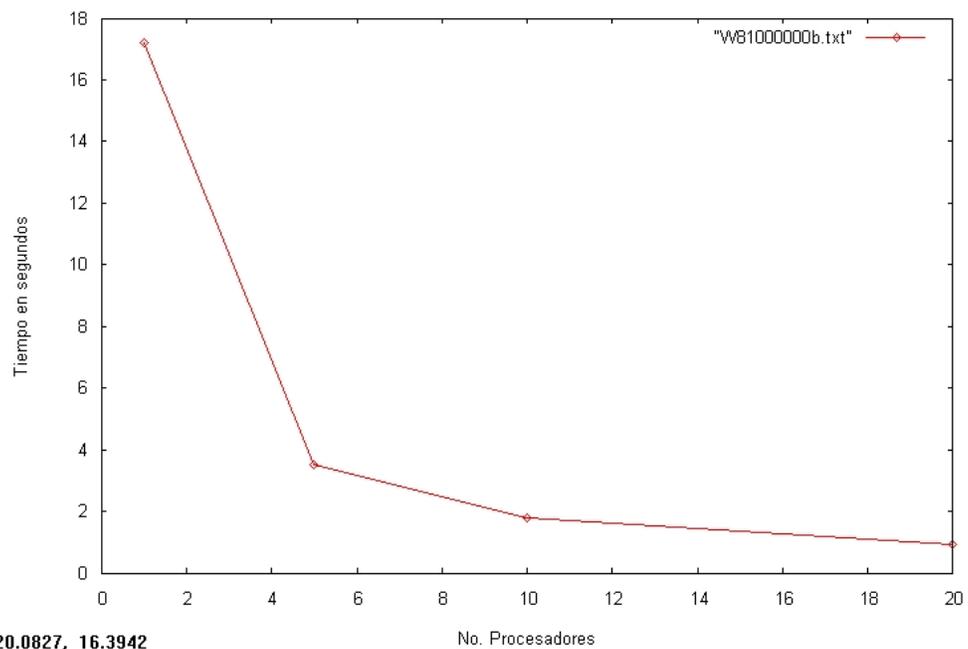
Gráfica 6.10: Tiempos de ejecución para 10000 trayectorias de Wiener

La gráfica 6.10 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias con 8 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 3.



19.3890, 1.78543
Gráfica 6.11: Tiempos de ejecución para 100000 trayectorias de Wiener

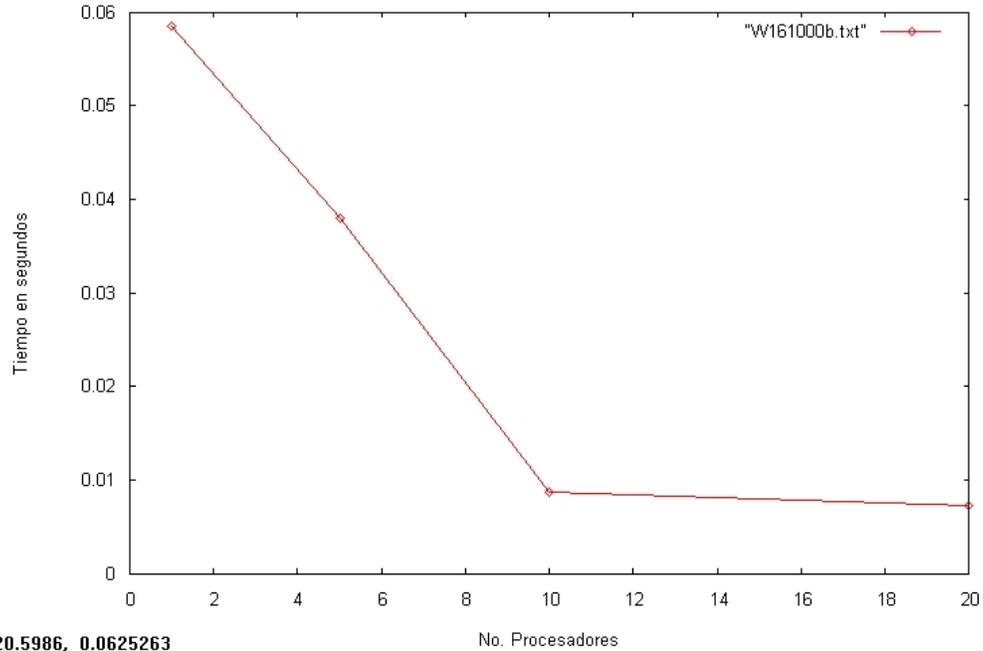
La gráfica 6.11 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias con 8 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 2.



20.0827, 16.3942
Gráfica 6.12: Tiempos de ejecución para 1000000 de trayectorias de Wiener

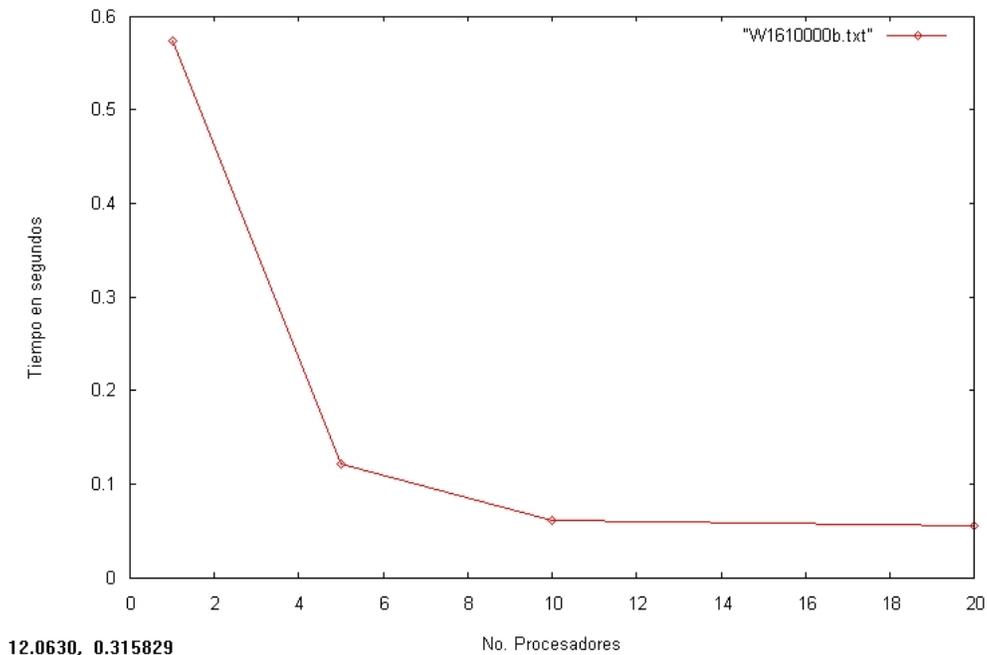
La gráfica 6.12 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 8 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 2.

6.2.3 Tiempos de ejecución para la integral de Wiener 16 puntos



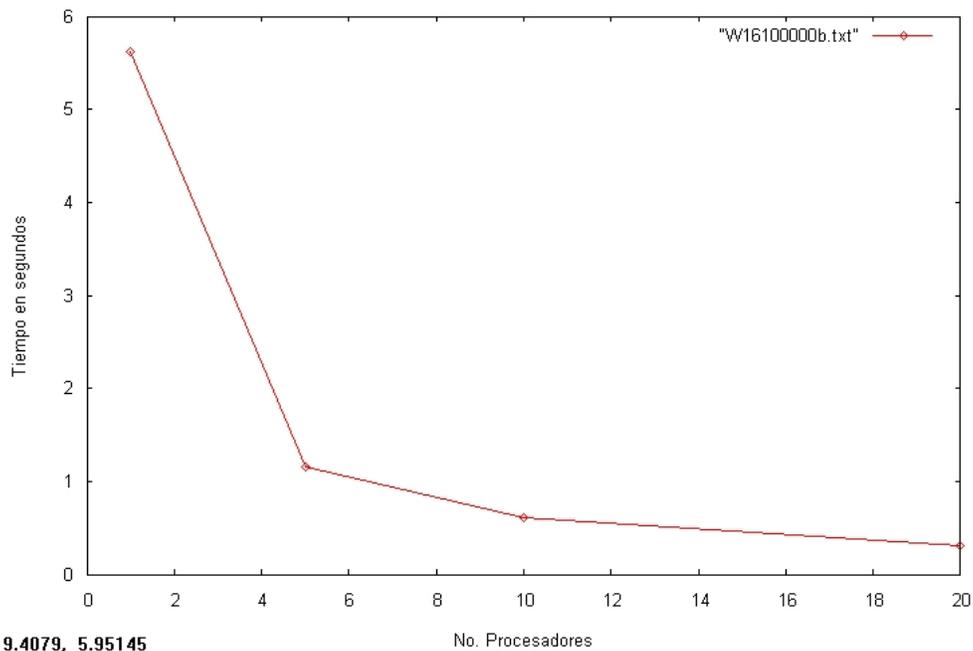
Gráfica 6.13: Tiempos de ejecución para 1000 trayectorias de Wiener

La gráfica 6.14 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias con 16 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 3.



Gráfica 6.14: Tiempos de ejecución para 10000 trayectorias de Wiener

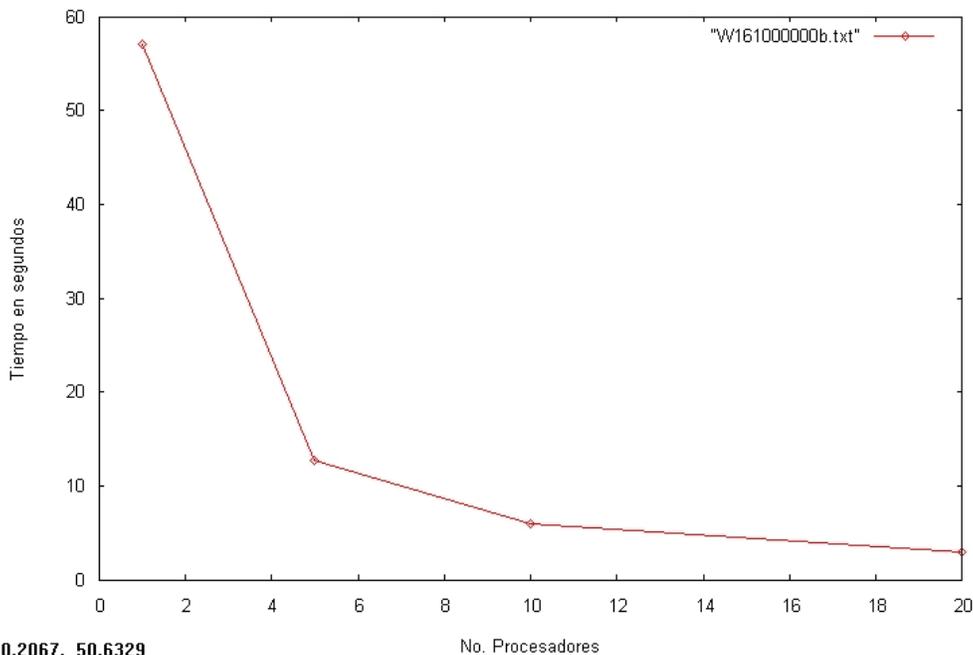
La gráfica 6.14 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias con 16 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 3.



19.4079, 5.95145

Gráfica 6.15: Tiempos de ejecución para 100000 trayectorias de Wiener

La gráfica 6.15 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias con 16 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 3.

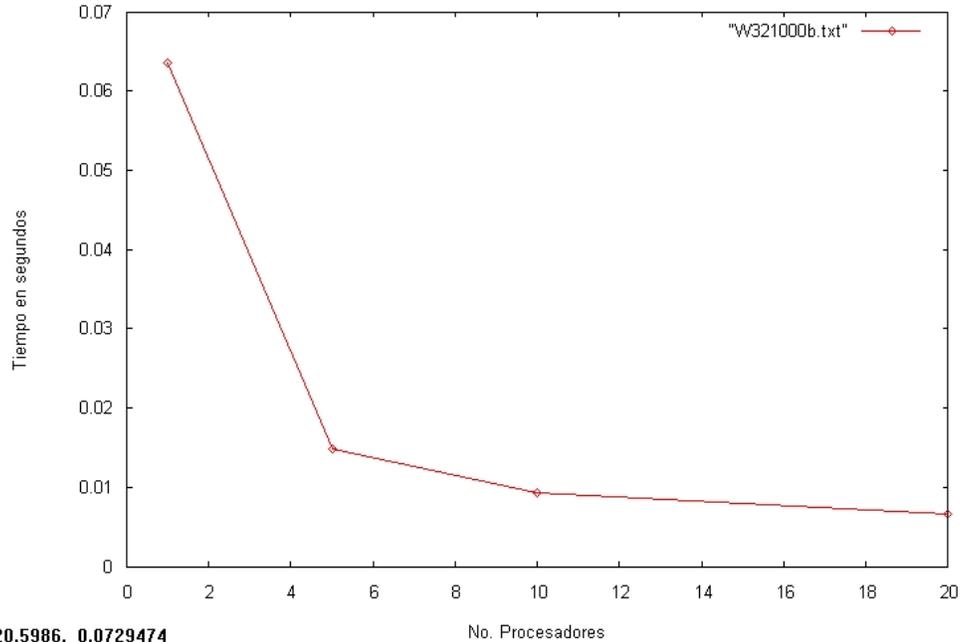


20.2067, 50.6329

Gráfica 6.16: Tiempos de ejecución para 1000000 de trayectorias de Wiener

La gráfica 6.16 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 16 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 2.

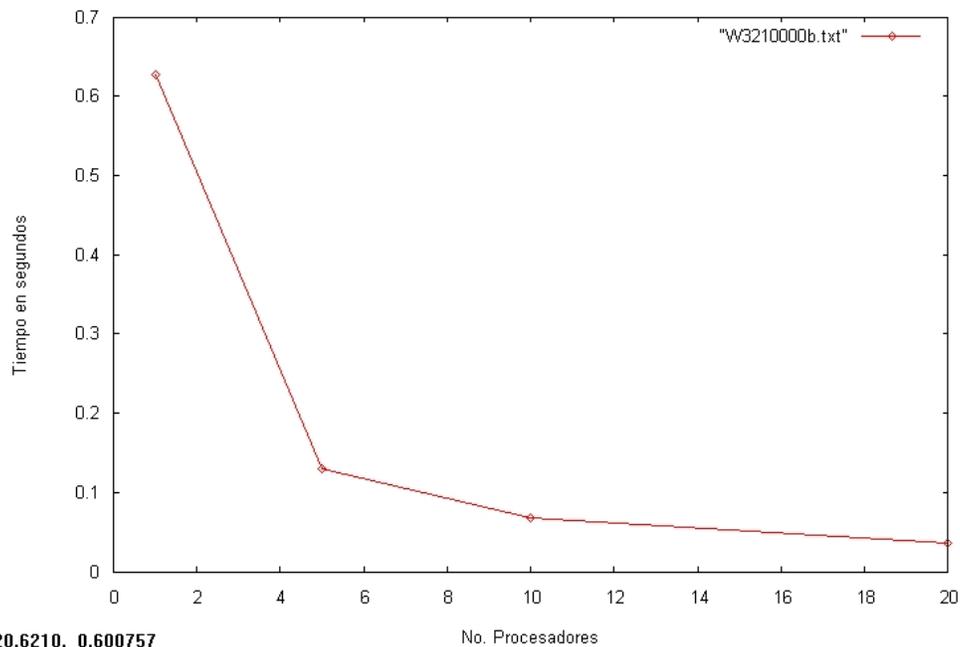
6.2.4 Tiempos de ejecución para la integral de Wiener 32 puntos



20.5986, 0.0729474

Gráfica 6.17: Tiempos de ejecución para 1000 trayectorias de Wiener

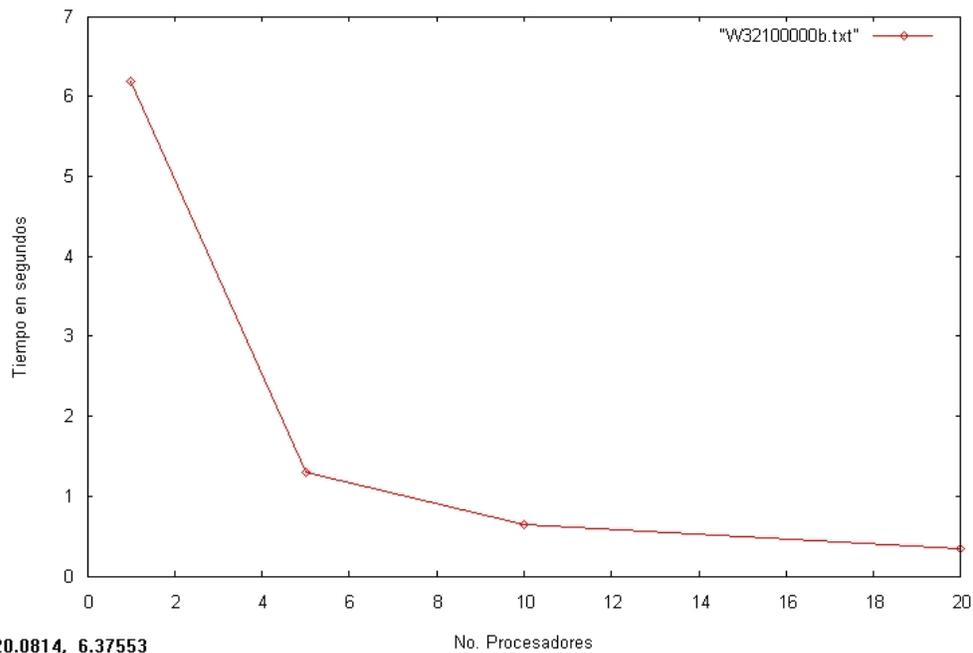
La gráfica 6.17 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias con 32 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 4.



20.6210, 0.600757

Gráfica 6.18: Tiempos de ejecución para 10000 trayectorias de Wiener

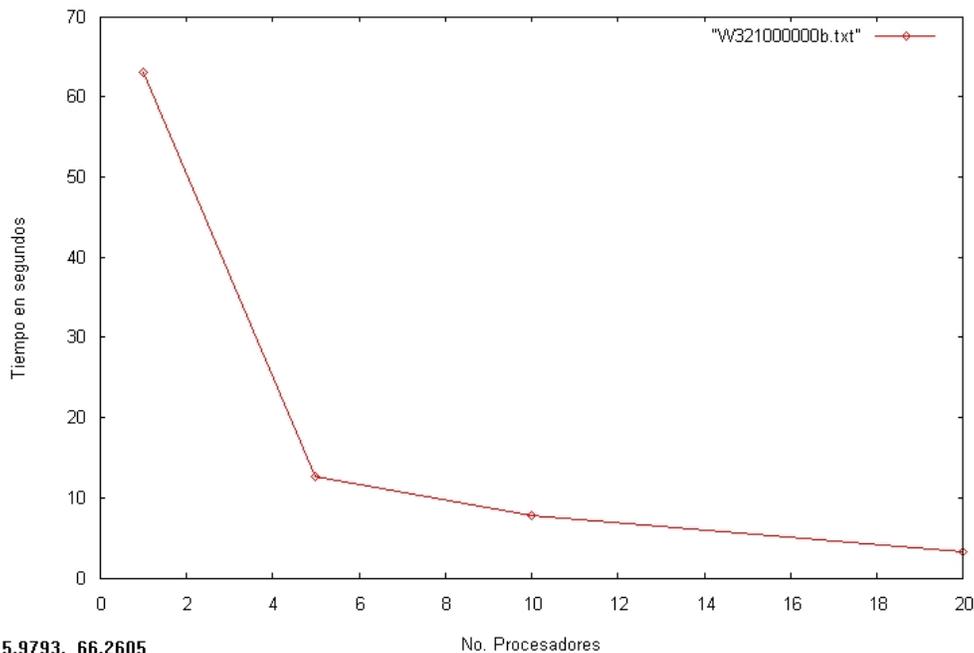
La gráfica 6.18 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias con 32 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 4.



20.0814, 6.37553

Gráfica 6.19: Tiempos de ejecución para 100000 trayectorias de Wiener

La gráfica 6.19 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias con 32 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 4.



15.9793, 66.2605

Gráfica 6.20: Tiempos de ejecución para 1000000 de trayectorias de Wiener

La gráfica 6.20 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 32 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 4.

El objetivo de las figuras que corresponden a las gráficas de tiempos de ejecución, es mostrar el desempeño de la función de paralelización de Wiener para generación de trayectorias aleatorias. En las imágenes se puede observar que conforme aumenta el número de puntos en las trayectorias, también aumenta el tiempo de ejecución. Sin embargo, conforme el número de procesadores es aumentado, es posible apreciar que se obtiene un mejor desempeño en cuanto al tiempo de ejecución para la misma cantidad de trayectorias.

6.3 Resultado de las pruebas realizadas a Ito y Feynman

Como se describió en capítulo 4, la integral de Ito se construye en base a un movimiento aleatorio de Wiener. En nuestras pruebas, para la construcción de este movimiento, se ingresa un intervalo entre **a** a **b**, el cual es utilizado para la obtención de los incrementos en Wiener.

Para las pruebas realizadas a la implementación de Ito en paralelo, como ejemplo se tomó un punto inicial **a = 10** y un punto final **b = 41** para el intervalo. Los números de puntos generados por trayectoria fueron:

- a) 10 para 1000, 10000, 100000 y 1000000 trayectorias aleatorias,
- b) 20 para 1000, 10000, 100000 y 1000000 trayectorias aleatorias,
- c) 30 para 1000, 10000, 100000 y 1000000 trayectorias aleatorias,

para cada inciso de los anteriores, se utilizaron 1, 5, 10 y 20 procesadores. A continuación son presentadas las tablas generales 6.5, 6.6 y 6.7.

En el caso de la integral de Feynman (como ya se mencionó en el capítulo 4), las trayectorias construidas son en base a una acción, la cual corresponde a un sistema en su forma Lagrangiana. El sistema que se tomó como ejemplo es el oscilador armónico. Este sistema tiene como parámetros de entrada la *masa* y su *frecuencia*. Las posiciones de la partícula u objeto con movimiento oscilatorio armónico son obtenidas en base a números aleatorios.

Para la generación de las variables aleatorias se usó la distribución normal con una media 0 y varianza 1.

Como ejemplo para las pruebas, los valores para la *masa* y la *frecuencia* utilizadas para la generación de las trayectorias fueron **4** y **4** respectivamente.

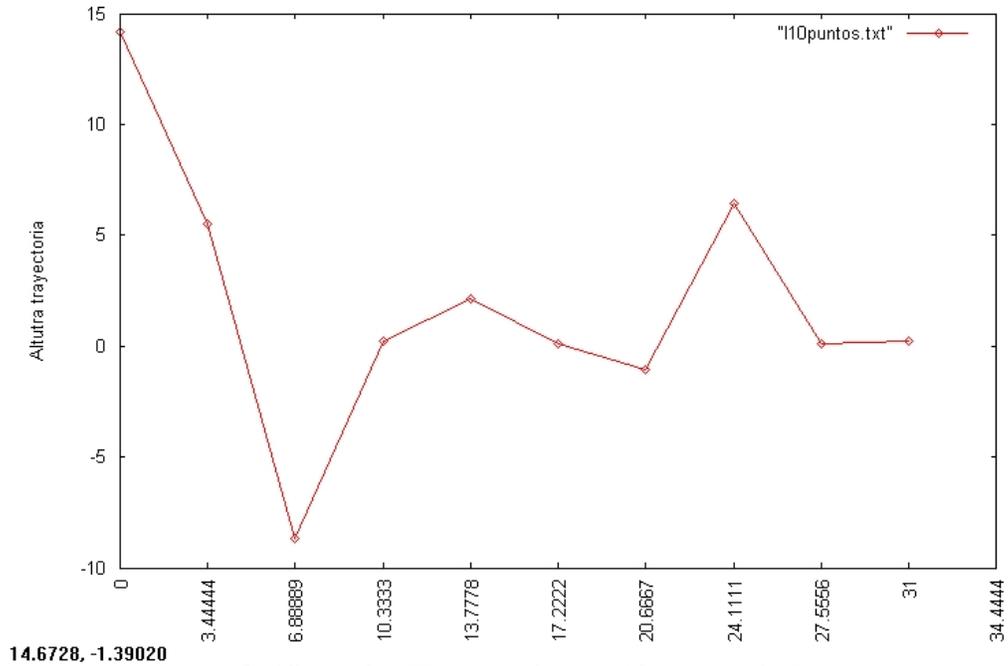
En esta sección se presentan las ejecuciones realizadas como pruebas, las gráficas de las trayectorias generadas como ejemplos y los tiempos de ejecución del los procesadores para la integral de Feynman.

Las ejecuciones realizadas son las mismas que se llevaron a cabo para la integral de Ito. Por lo cual, las tablas presentadas muestran las pruebas realizadas tanto para Ito como para Feynman.

Tabla para trayectorias con 10 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	10000	10000	1000
1000	5	10000	2000	200
1000	10	10000	1000	100
1000	20	10000	500	50
10000	1	100000	100000	10000
10000	5	100000	20000	2000
10000	10	100000	10000	1000
10000	20	100000	5000	500
100000	1	1000000	1000000	100000
100000	5	1000000	200000	20000
100000	10	1000000	100000	10000
100000	20	1000000	50000	5000
1000000	1	10000000	10000000	1000000
1000000	5	10000000	2000000	200000
1000000	10	10000000	1000000	100000

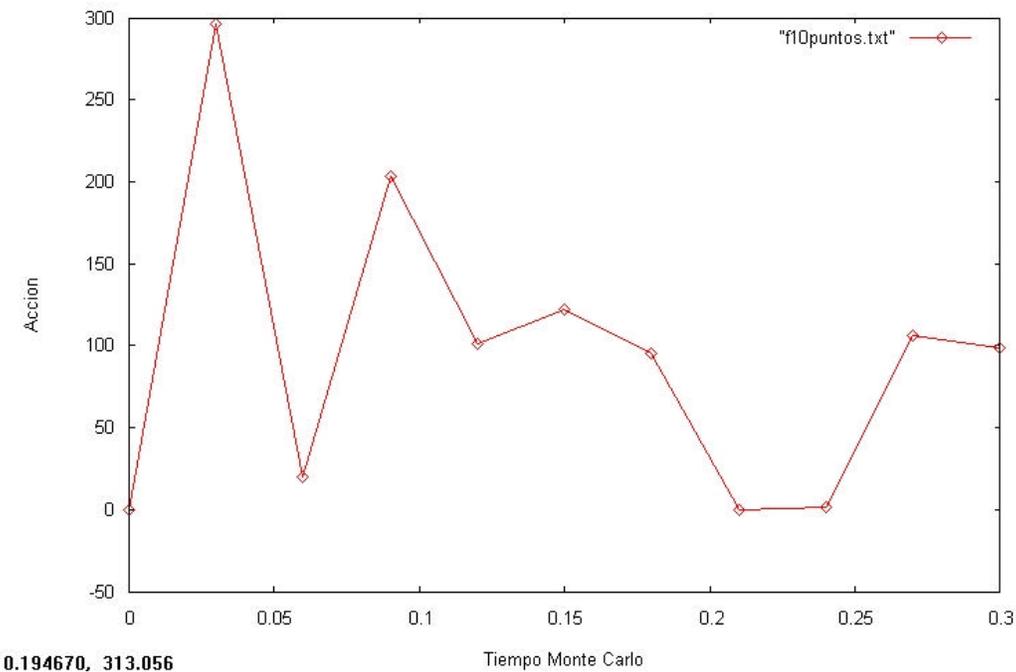
Tabla 6.5: Ejecuciones realizas para Ito y Feynman con 10 puntos

La tabla 6.5 muestra las pruebas realizadas para las integrales de Ito y Feynman implementadas en paralelo. La prueba consistió en la generación de 1000, 10000, 100000 trayectorias con 10 puntos cada una, para 1, 5, 10 y 20 procesadores. A continuación se muestran las trayectorias de Ito y una de Feynman, resultantes de las pruebas realizadas.



Gráfica 6.21: Trayectoria con 10 puntos de Ito

La gráfica 6.21 muestra el ejemplo de una trayectoria aleatoria generada por el método de Ito con 10 puntos.



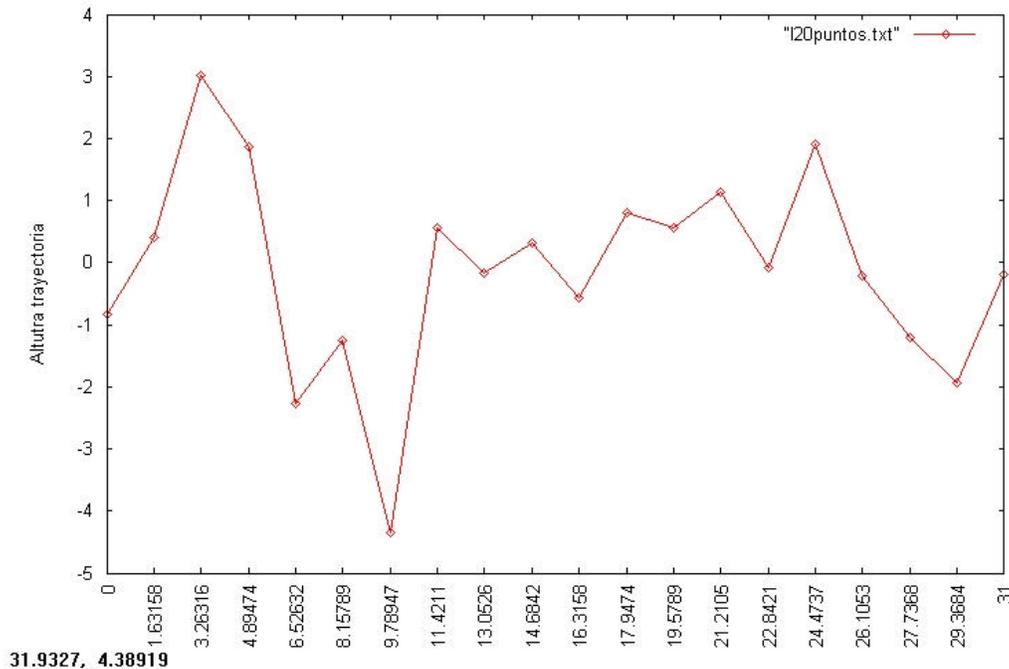
Gráfica 6.22: Trayectoria con 10 puntos de Feynman

La gráfica 6.22 muestra el ejemplo de una trayectoria aleatoria generada por el método de Feynman con 10 puntos.

Tabla para trayectorias con 20 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	20000	20000	1000
1000	5	20000	4000	200
1000	10	20000	2000	100
1000	20	20000	1000	50
10000	1	200000	200000	10000
10000	5	200000	40000	2000
10000	10	200000	20000	1000
10000	20	200000	10000	500
100000	1	2000000	2000000	100000
100000	5	2000000	400000	20000
100000	10	2000000	200000	10000
100000	20	2000000	100000	5000
1000000	1	20000000	20000000	1000000
1000000	5	20000000	4000000	200000
1000000	10	20000000	2000000	100000

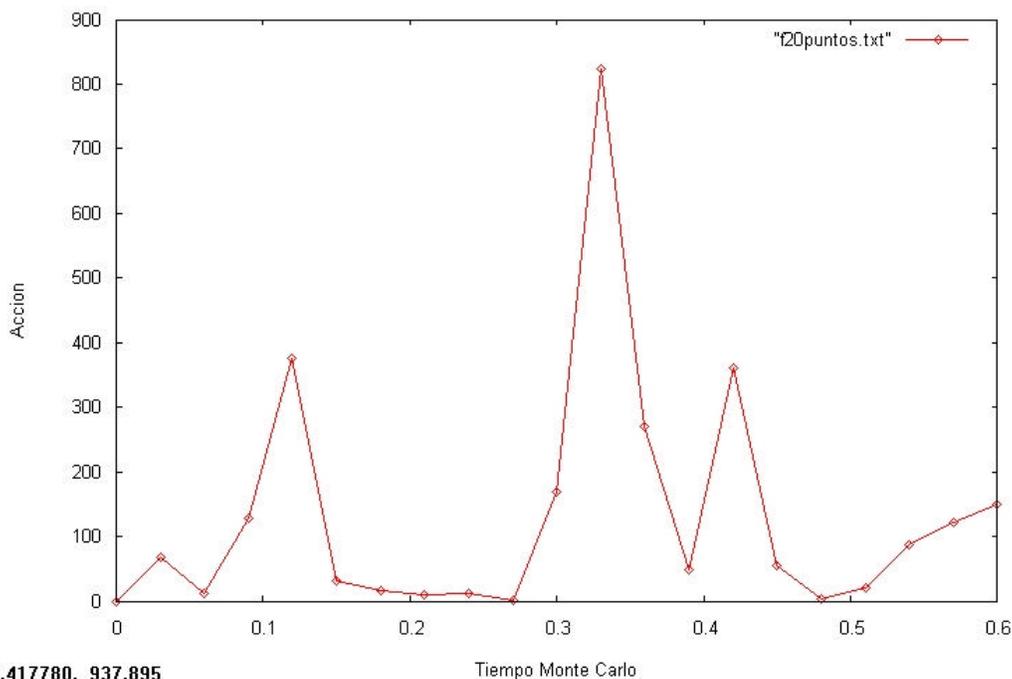
Tabla 6.6: Ejecuciones realizadas para Ito y Feynman con 20 puntos

La tabla 6.6 muestra las pruebas realizadas para las integrales de Ito y Feynman implementadas en paralelo. La prueba consistió en la generación de 1000, 10000, 100000 trayectorias con 20 puntos cada una para 1, 5, 10 y 20 procesadores. A continuación se muestran las trayectorias de Ito y de Feynman, resultantes de las pruebas realizadas.



Gráfica 6.23: Trayectoria con 20 puntos de Ito

La gráfica 6.23 muestra el ejemplo de una trayectoria aleatoria generada por el método de Ito con 20 puntos.



0.417780, 937.895

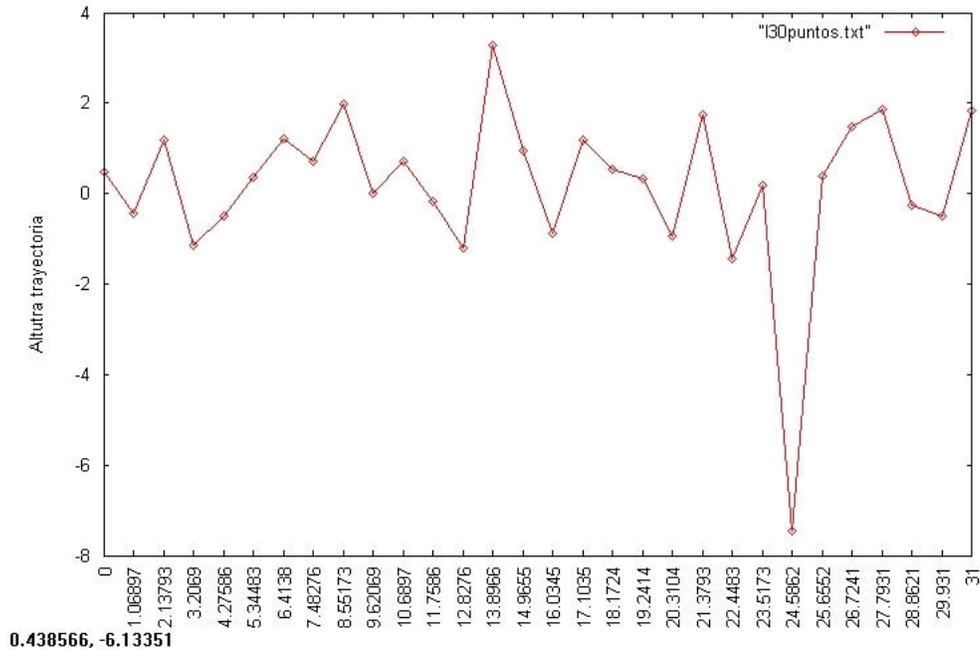
Gráfica 6.24: Trayectoria con 20 puntos de Feynman

La gráfica 6.24 muestra el ejemplo de una trayectoria aleatoria generada por el método de Feynman con 20 puntos.

Tabla para trayectorias con 30 puntos				
Número trayectorias	Número de nodos	Número de puntos totales	Número de puntos por procesador	Número de trayectorias por procesador
1000	1	30000	30000	1000
1000	5	30000	6000	200
1000	10	30000	3000	100
1000	20	30000	1500	50
10000	1	300000	300000	10000
10000	5	300000	60000	2000
10000	10	300000	30000	1000
10000	20	300000	15000	500
100000	1	3000000	3000000	100000
100000	5	3000000	600000	20000
100000	10	3000000	300000	10000
100000	20	3000000	150000	5000
1000000	1	30000000	30000000	1000000
1000000	5	30000000	6000000	200000
1000000	10	30000000	3000000	100000

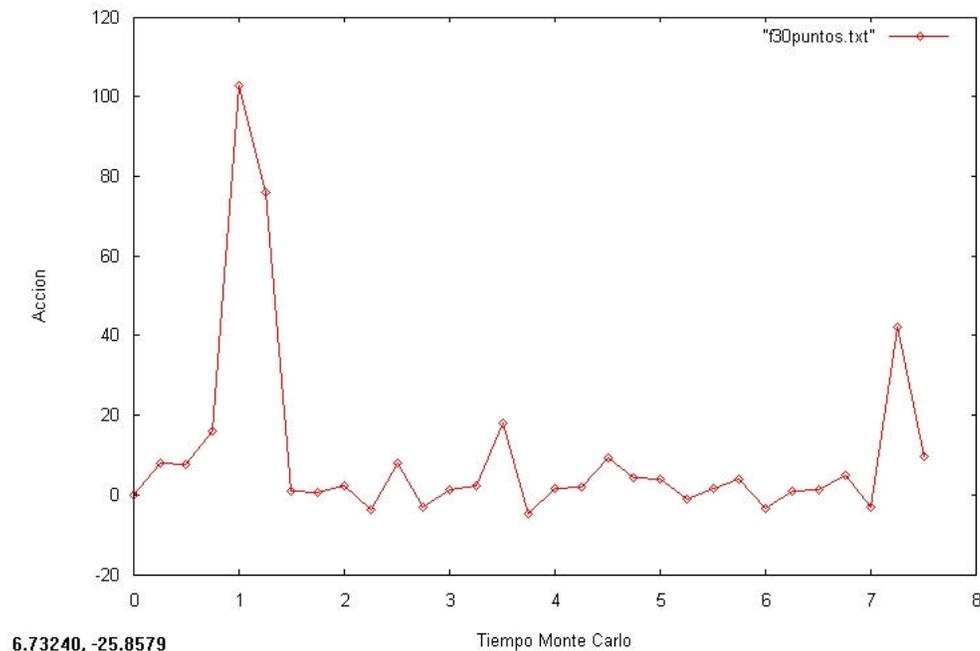
Tabla 6.7: Ejecuciones realiza para Ito y Feynman con 30 puntos

La tabla 6.7 muestra las pruebas realizadas para las integrales de Ito y Feynman implementadas en paralelo. La prueba en este caso consistió en la generación de 1000, 10000, 100000 trayectorias, con 30 puntos cada una, para 1, 5, 10 y 20 procesadores. A continuación se muestran las trayectorias de Ito y de de Feynman, resultantes de las pruebas realizadas.



Gráfica 6.25: Trayectoria con 30 puntos de Ito

La gráfica 6.25 muestra el ejemplo de una trayectoria aleatoria generada por el método de Ito con 30 puntos.

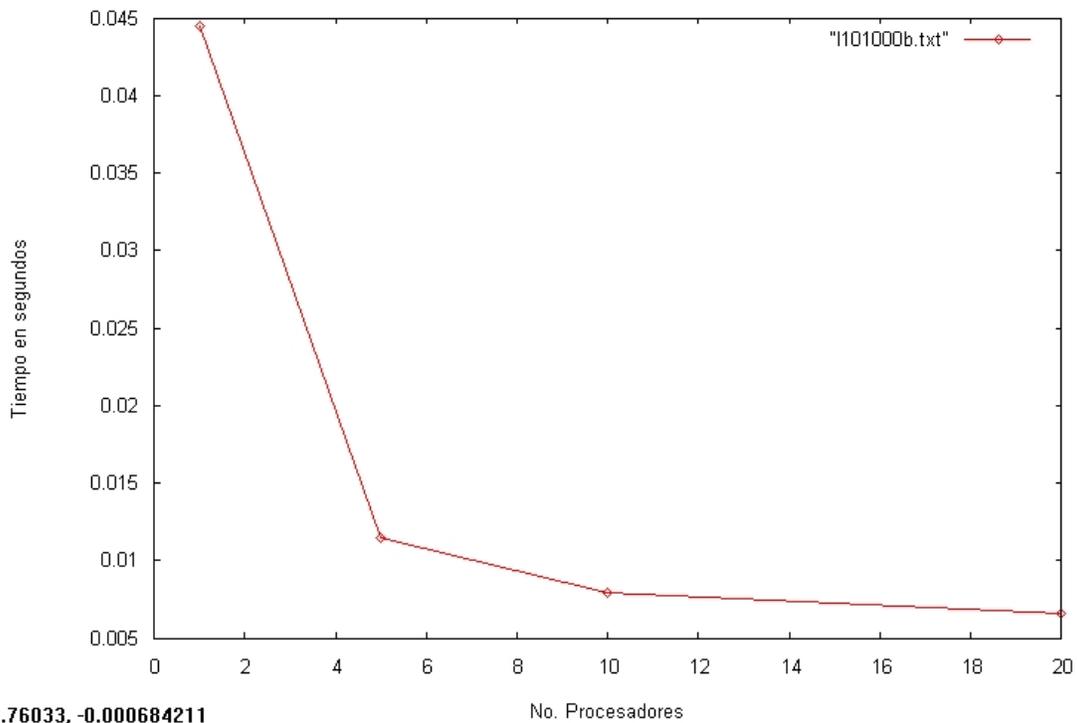


Gráfica 6.26: Trayectoria con 30 puntos de Feynman

La gráfica 6.26 muestra el ejemplo de una trayectoria aleatoria generada por el método de Feynman con 30 puntos.

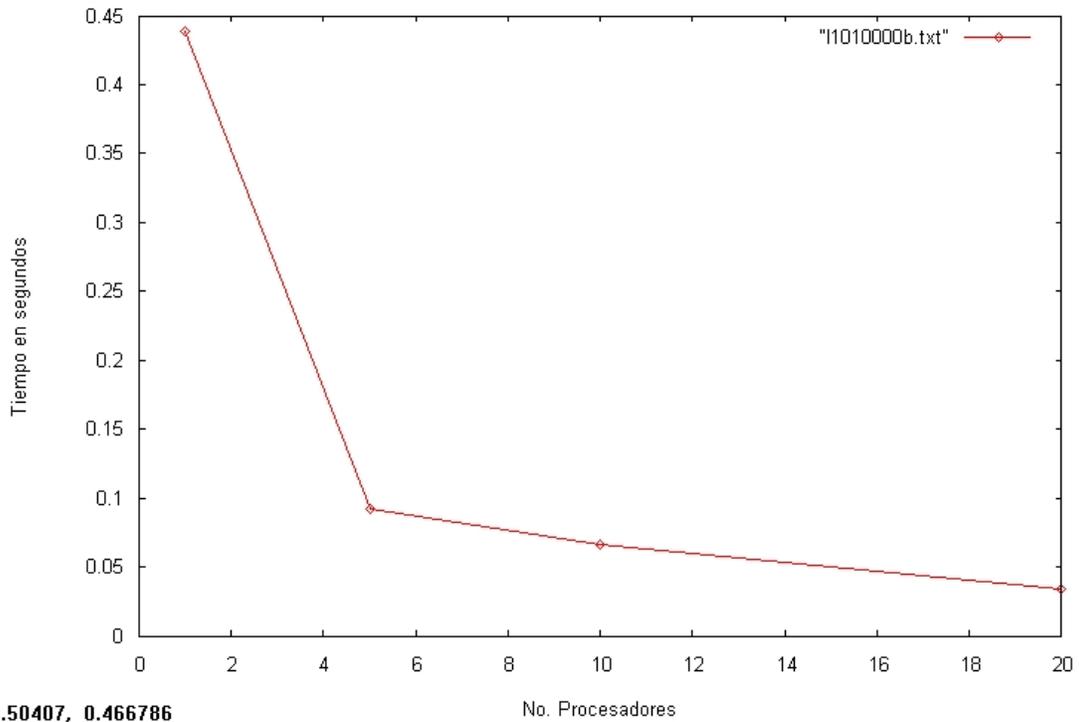
Siendo los métodos de integración de Ito y Feynman para la generación de trayectorias aleatorias, dos métodos diferentes al de Wiener, al igual como ocurre en el método implementado de Wiener en forma paralela. En las figuras 6.21, 6.22, 6.23, 6.24, 6.25 y 6.26 es posible observar que: conforme aumenta el número de puntos de la trayectoria (tanto en Ito como en Feynman), ésta se suaviza, lo cual puede ser apreciado en la definición que da el aumento de puntos a la trayectoria en dichas trayectorias.

6.3.1 Tiempos de ejecución para la integral de Ito con 10 puntos



Gráfica 6.27: Tiempos de ejecución para 1000 trayectorias de Ito

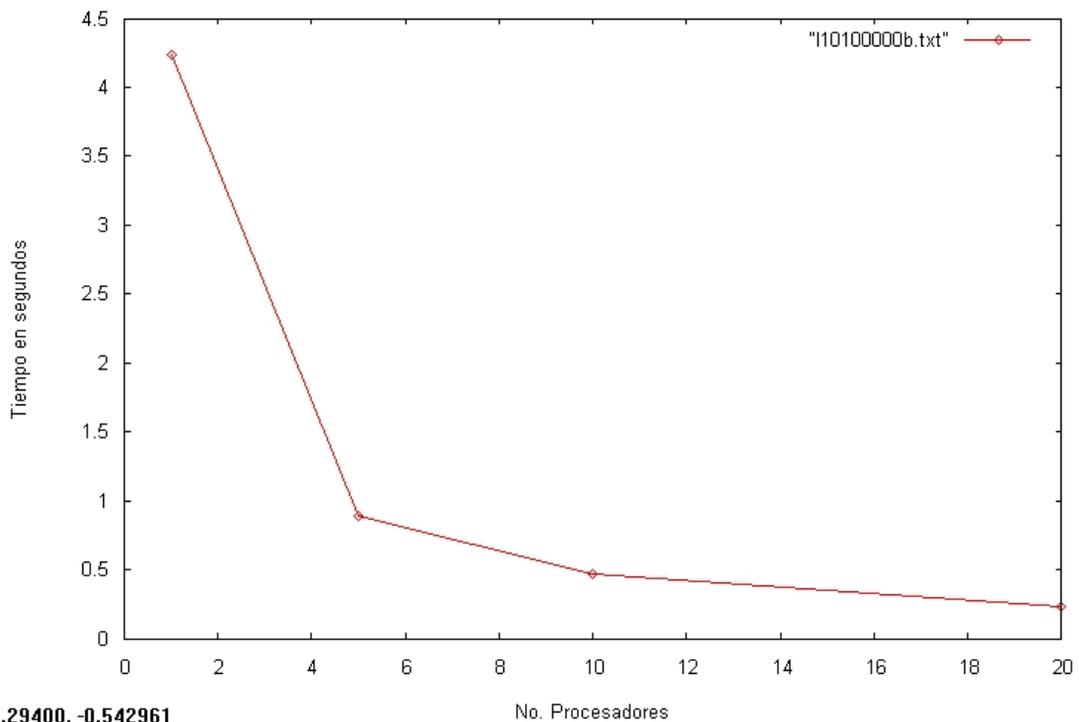
La gráfica 6.27 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Ito con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.



1.50407, 0.466786

Gráfica 6.28: Tiempos de ejecución para 10000 trayectorias de Ito

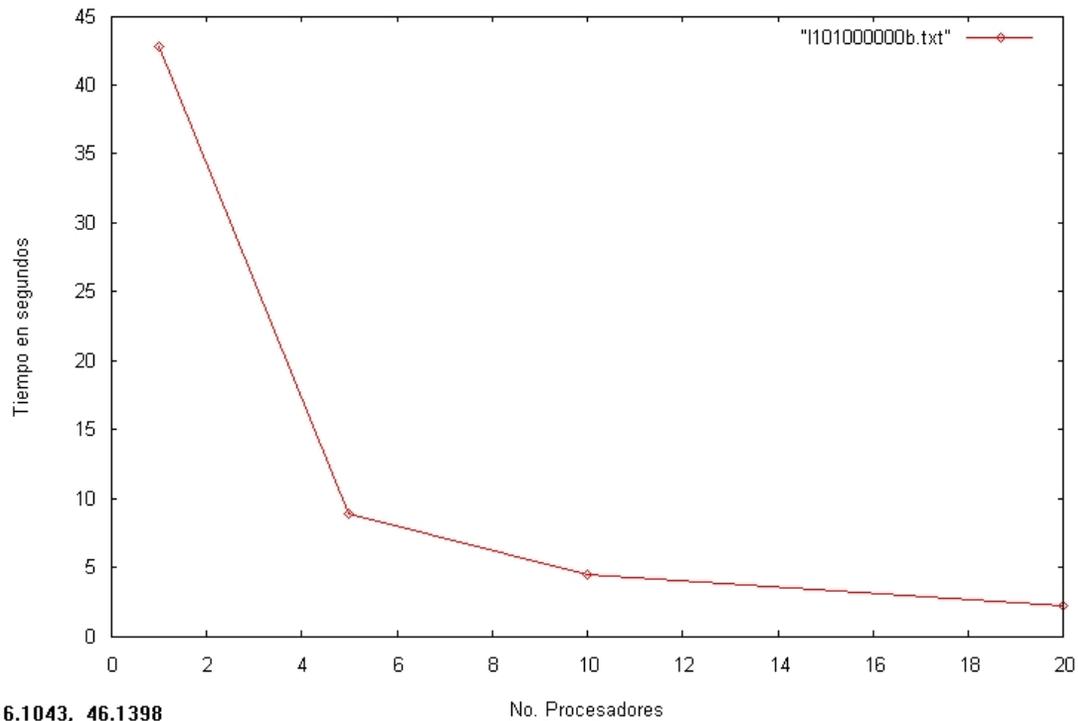
La gráfica 6.28 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Ito con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.



4.29400, -0.542961

Gráfica 6.29: Tiempos de ejecución para 100000 trayectorias de Ito

La gráfica 6.29 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Ito con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.

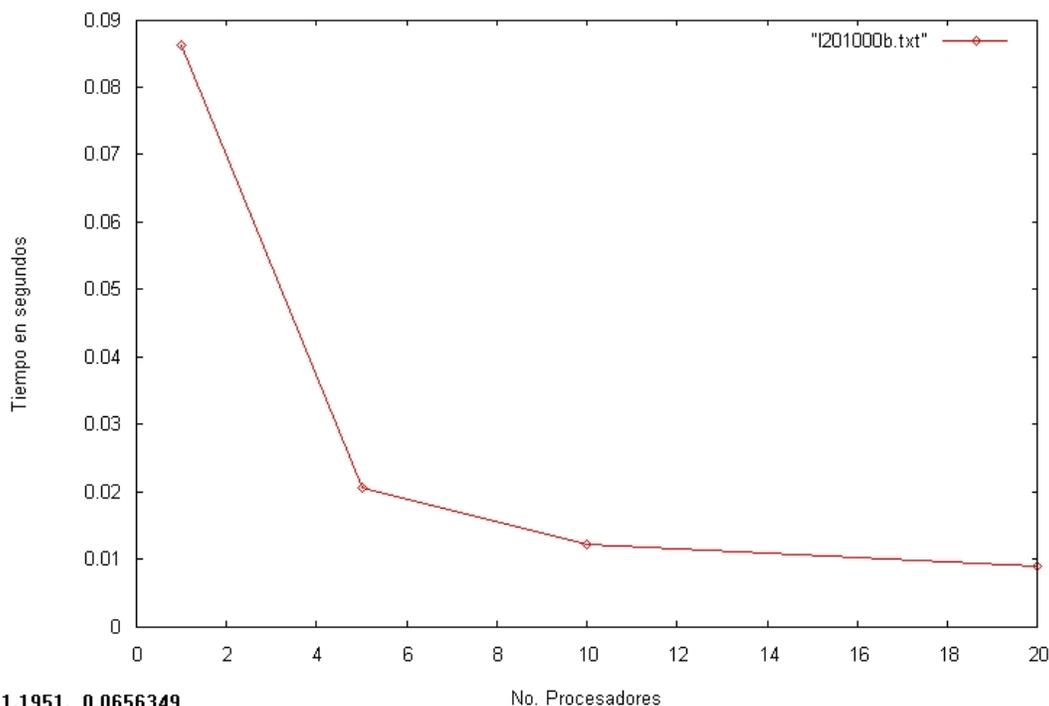


16.1043, 46.1398

Gráfica 6.30: Tiempos de ejecución para 1000000 de trayectorias de Ito

La gráfica 6.30 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 5.

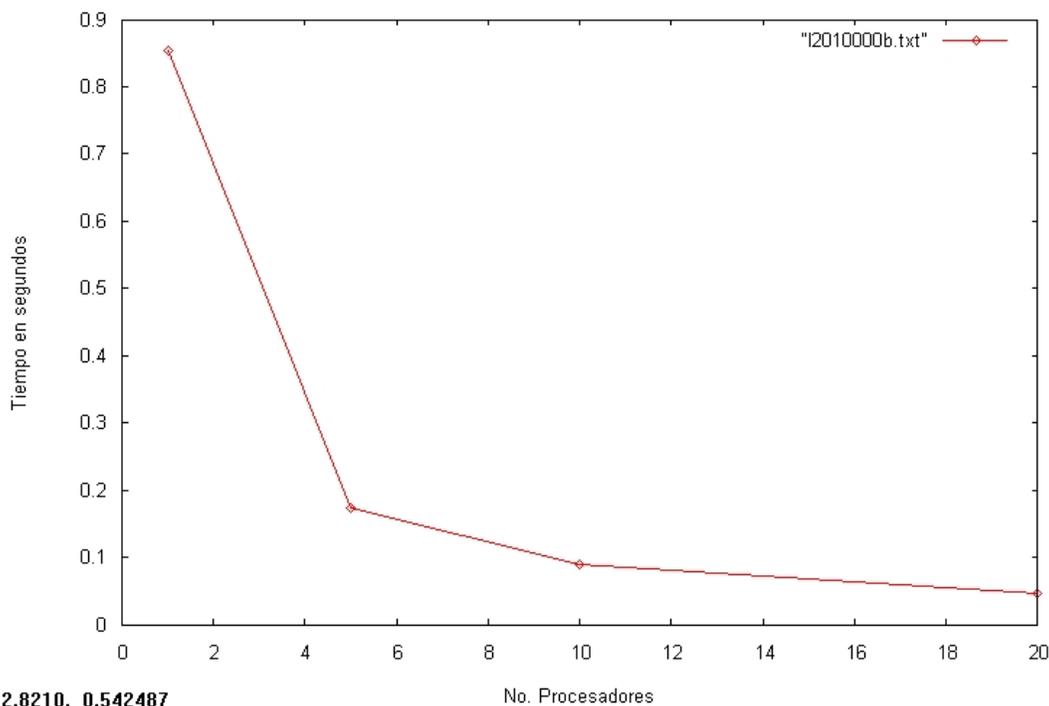
6.3.2 Tiempos de ejecución para la integral de Ito con 20 puntos



11.1951, 0.0656349

Gráfica 6.31: Tiempos de ejecución para 1000 trayectorias de Ito

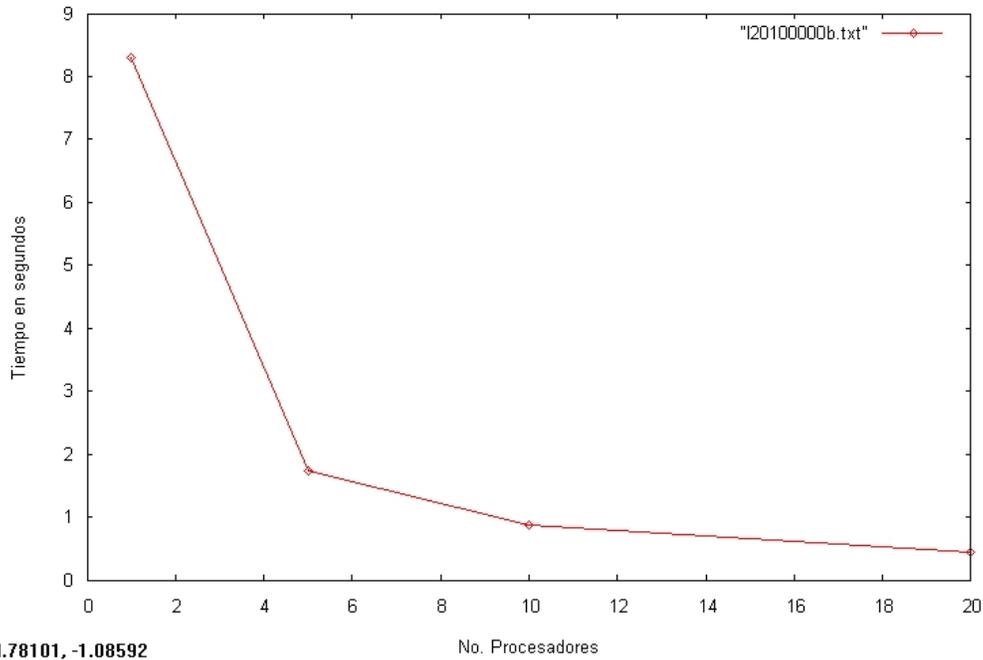
La gráfica 6.31 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Ito con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.



12.8210, 0.542487

Gráfica 6.32: Tiempos de ejecución para 10000 trayectorias de Ito

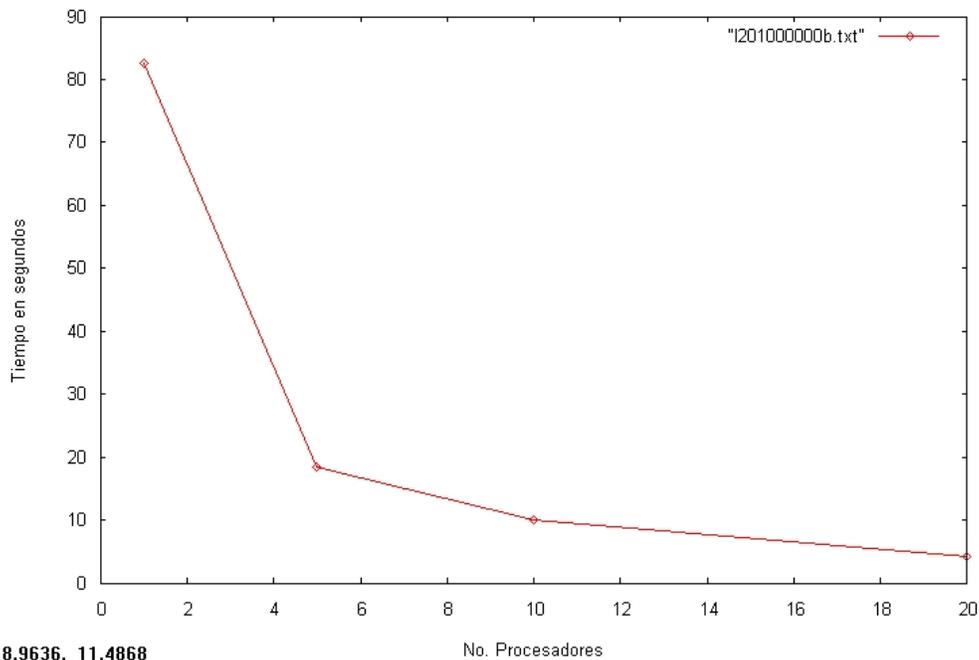
La gráfica 6.32 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Ito con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.



4.78101, -1.08592

Gráfica 6.33: Tiempos de ejecución para 100000 trayectorias de Ito

La gráfica 6.33 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Ito con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.

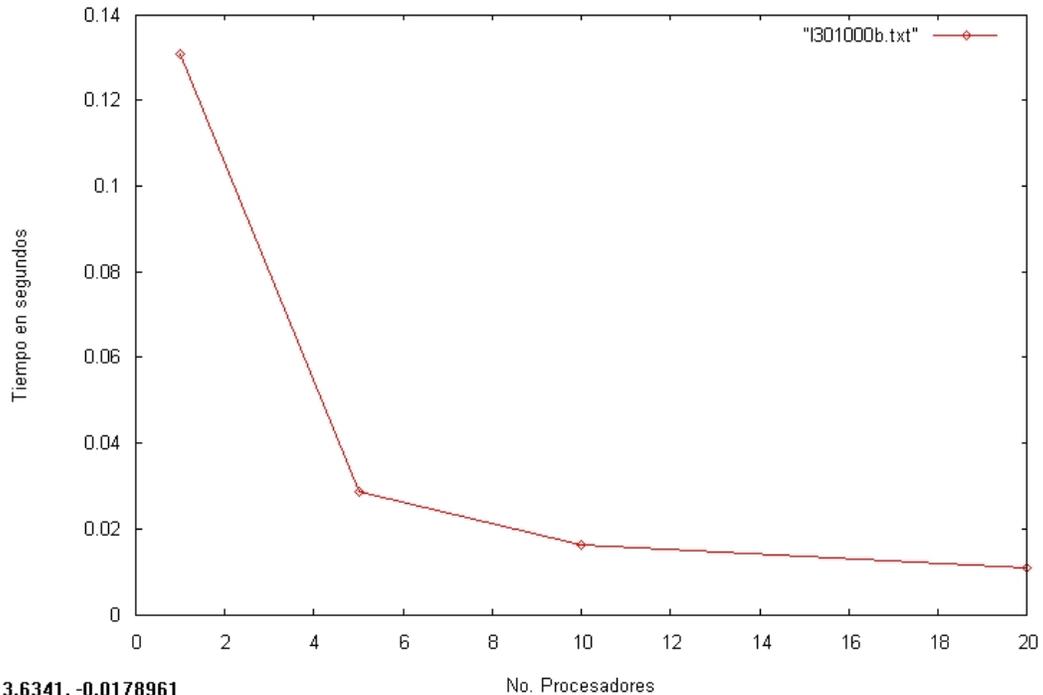


18.9636, 11.4868

Gráfica 6.34: Tiempos de ejecución para 1000000 de trayectorias de Ito

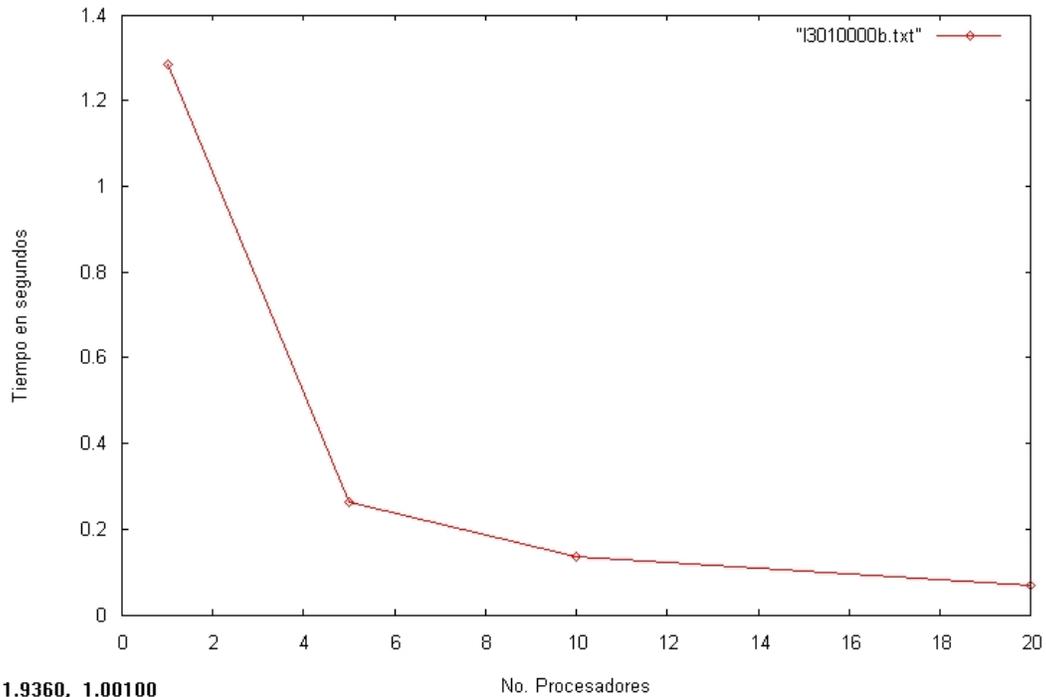
La gráfica 6.34 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 6.6.

6.3.3 Tiempos de ejecución para la integral de Ito con 30 puntos



Gráfica 6.35: Tiempos de ejecución para 1000 trayectorias de Ito

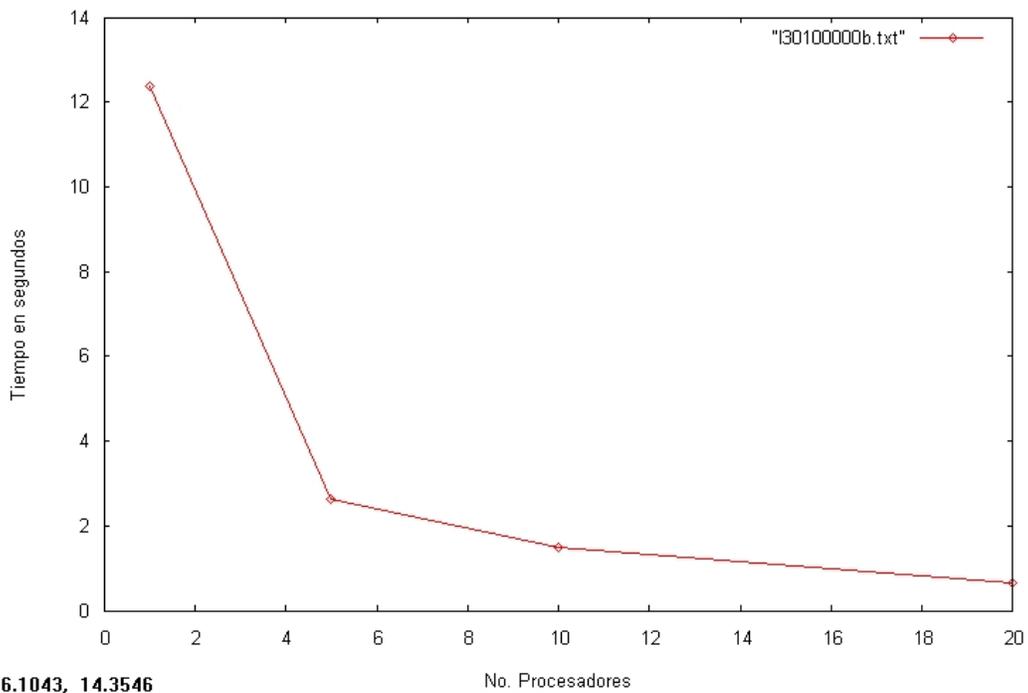
La gráfica 6.35 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Ito con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



11.9360, 1.00100

Gráfica 6.36: Tiempos de ejecución para 10000 trayectorias de Ito

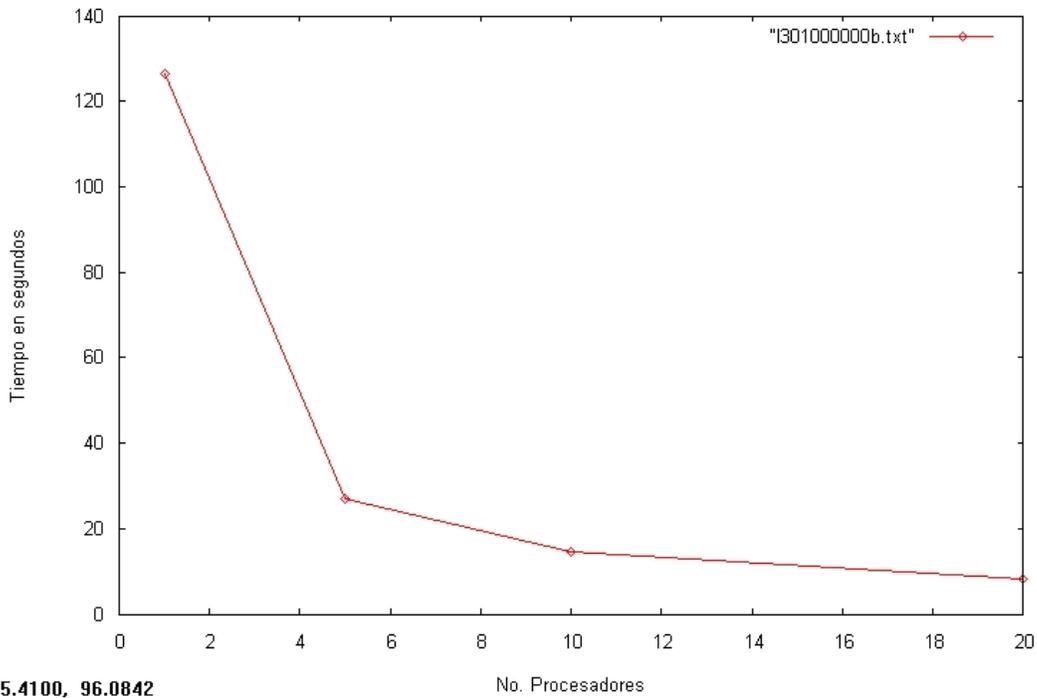
La gráfica 6.36 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Ito con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



16.1043, 14.3546

Gráfica 6.37: Tiempos de ejecución para 100000 trayectorias de Ito

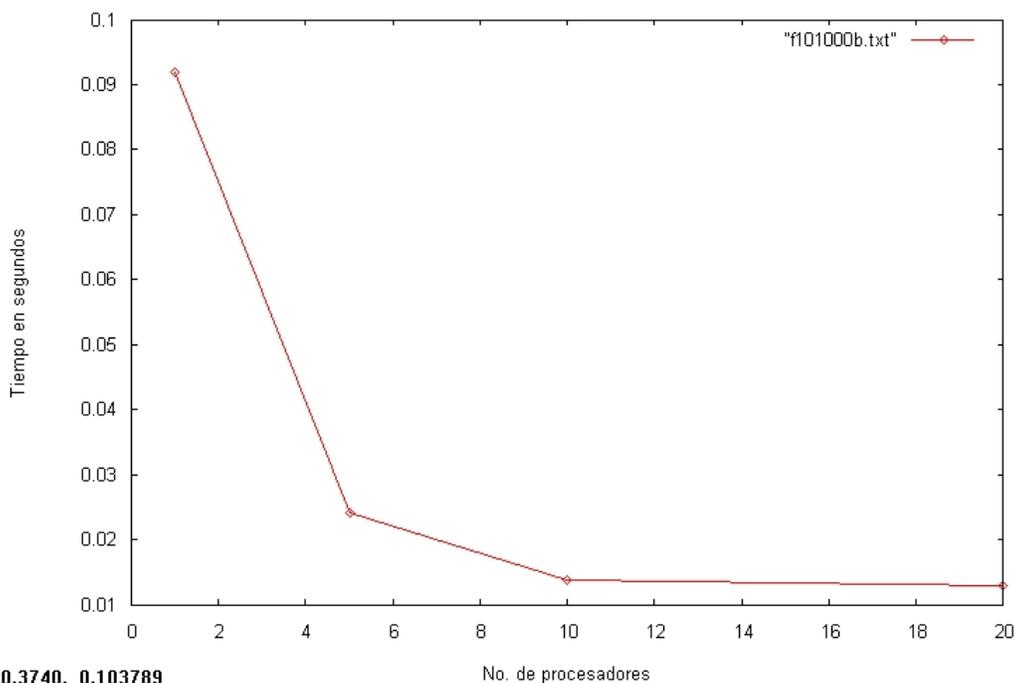
La gráfica 6.37 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Ito con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



Gráfica 6.38: Tiempos de ejecución para 1000000 de trayectorias de Ito

La gráfica 6.38 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Ito con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones mostradas en la tabla 6.7.

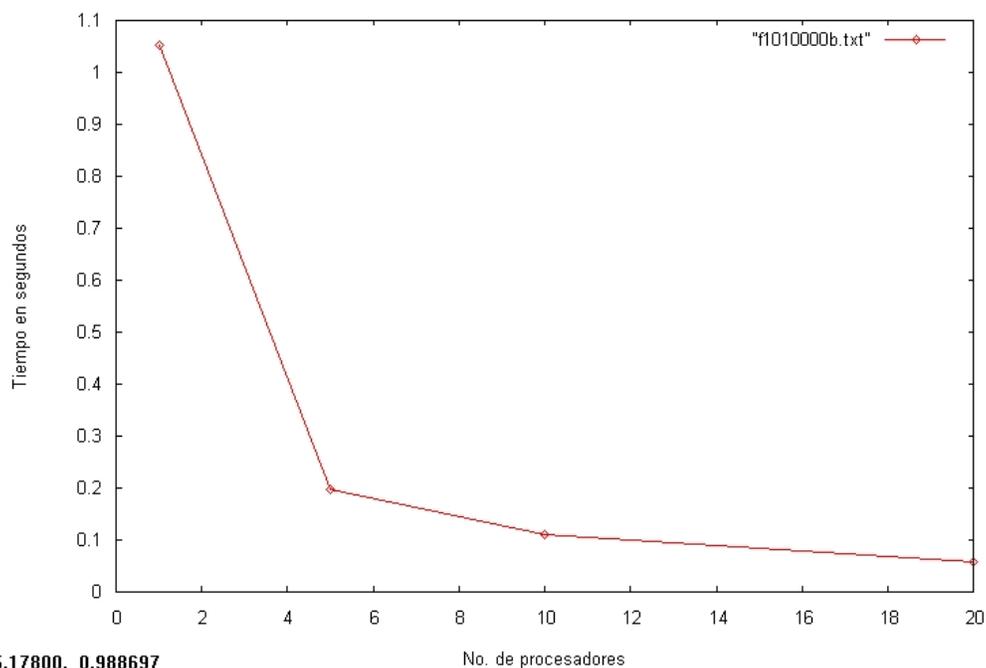
6.3.4 Tiempos de ejecución para la integral de Feynman con 10 puntos



20.3740, 0.103789

Gráfica 6.39: Tiempos de ejecución para 1000 trayectorias de Feynman

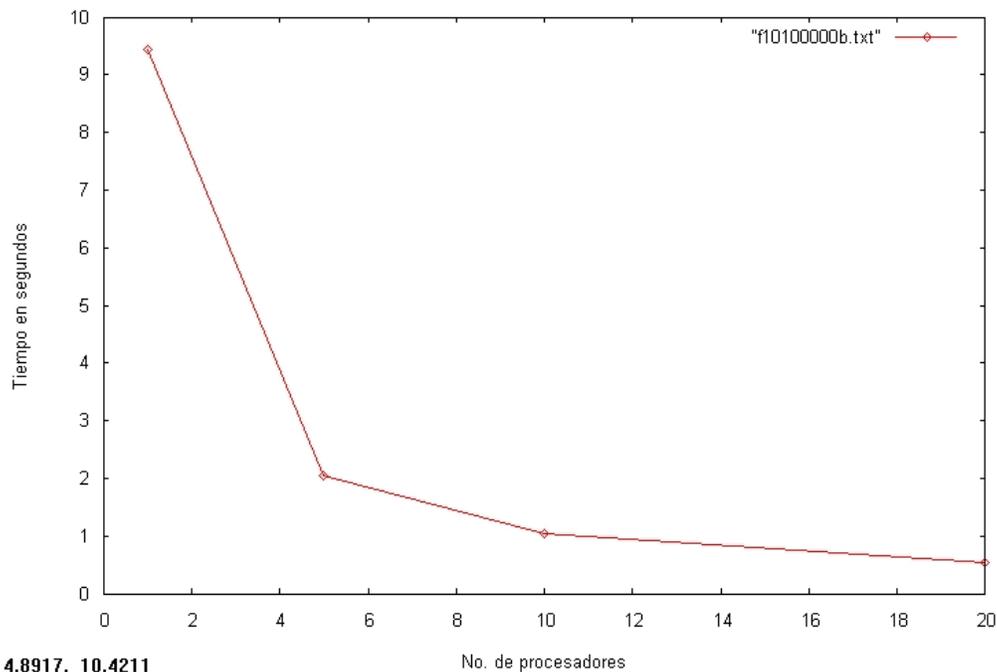
La gráfica 6.39 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Feynman con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.



5.17800, 0.988697

Gráfica 6.40: Tiempos de ejecución para 10000 trayectorias de Feynman

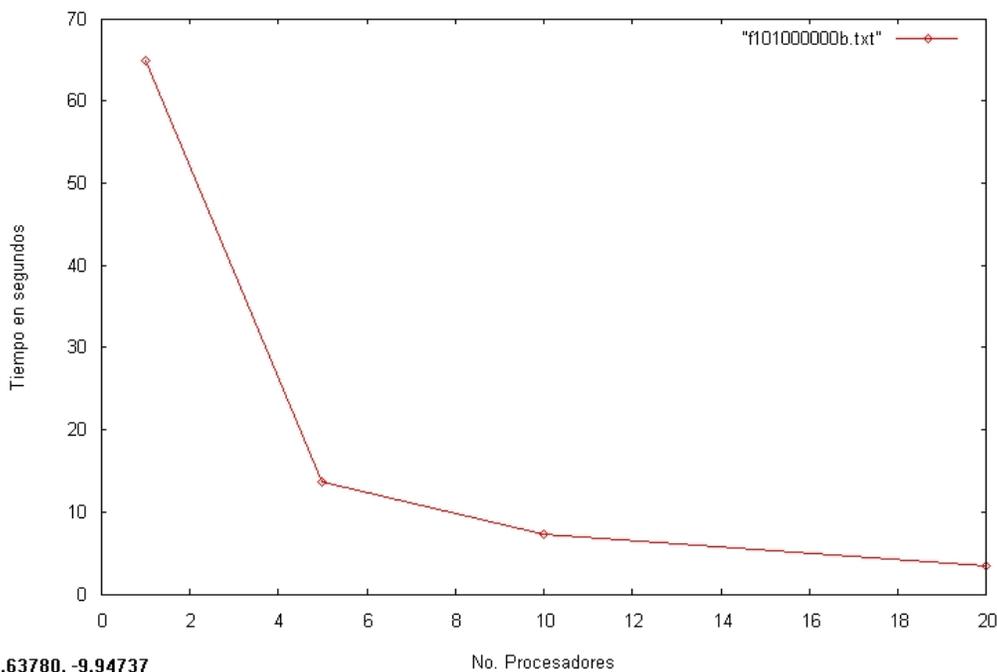
La gráfica 6.40 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Feynman con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.



14.8917, 10.4211

Gráfica 6.41: Tiempos de ejecución para 100000 trayectorias de Feynman

La gráfica 6.41 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Feynman con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.

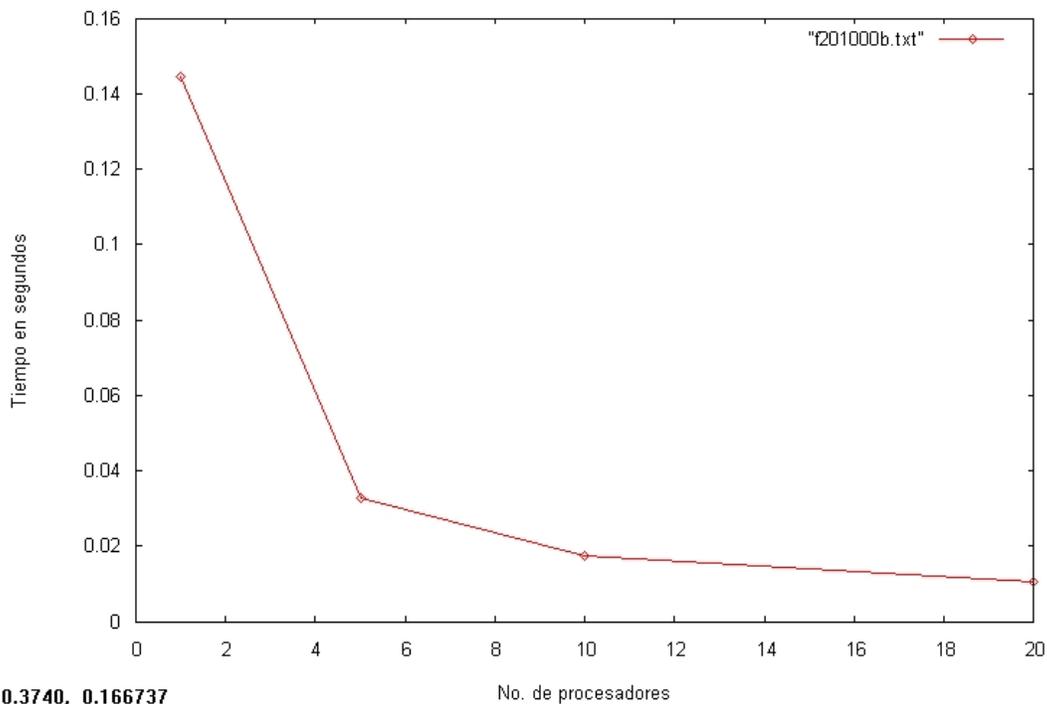


2.63780, -9.94737

Gráfica 6.42: Tiempos de ejecución para 1000000 de trayectorias de Feynman

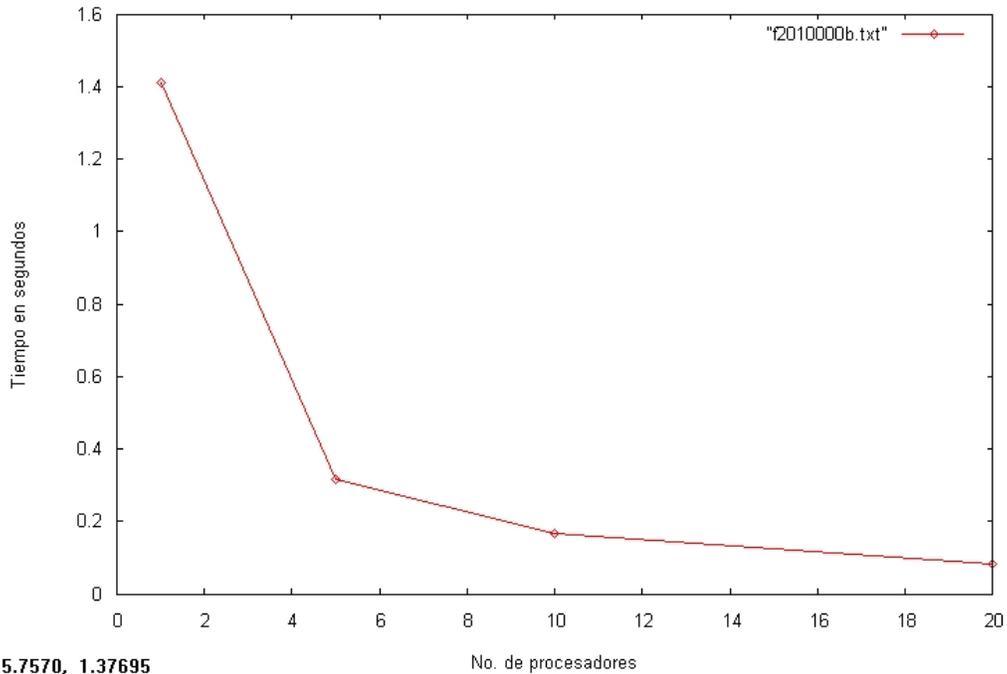
La gráfica 6.42 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 de trayectorias aleatorias de Feynman con 10 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 5.

6.3.5 Tiempos de ejecución para la integral de Feynman con 20 puntos



Gráfica 6.43: Tiempos de ejecución para 1000 trayectorias de Feynman

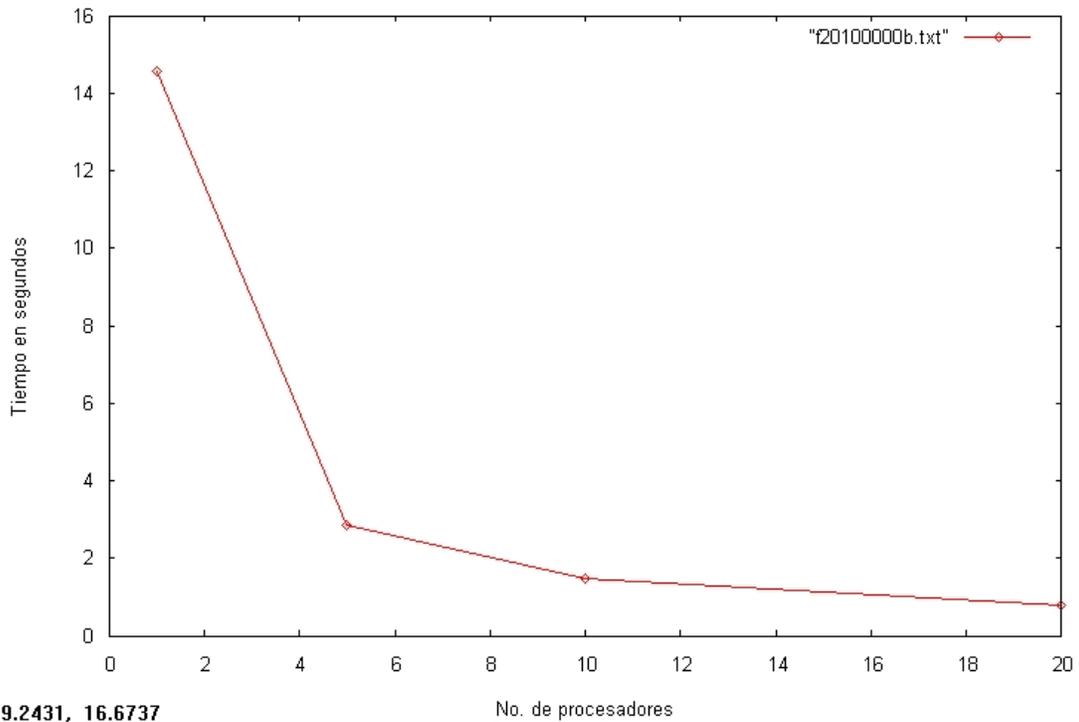
La gráfica 6.43 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Feynman con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.



15.7570, 1.37695

Gráfica 6.44: Tiempos de ejecución para 10000 trayectorias de Feynman

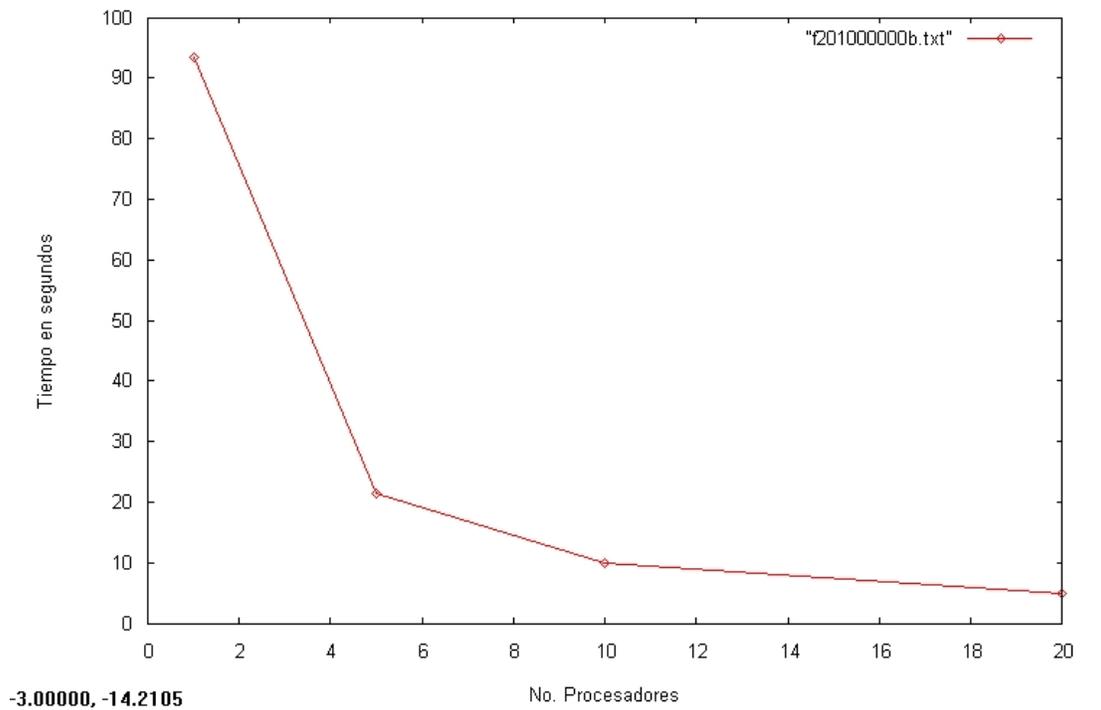
La gráfica 6.44 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Feynman con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.



19.2431, 16.6737

Gráfica 6.45 Tiempos de ejecución para 100000 trayectorias de Feynman

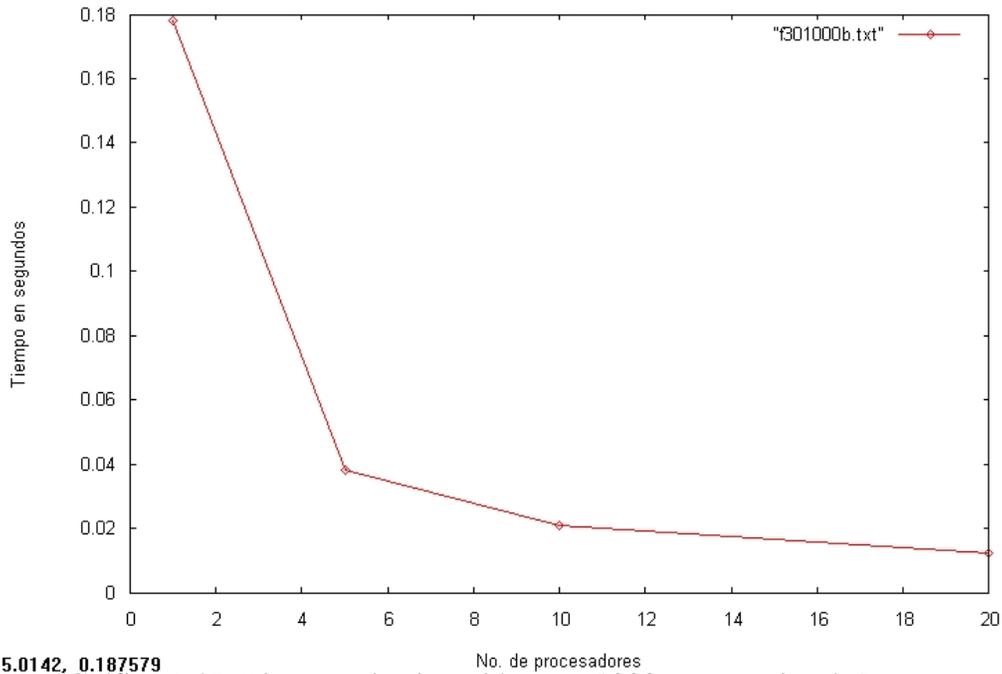
La gráfica 6.45 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Feynman con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.



Gráfica 6.46: Tiempos de ejecución para 1000000 de trayectorias de Feynman

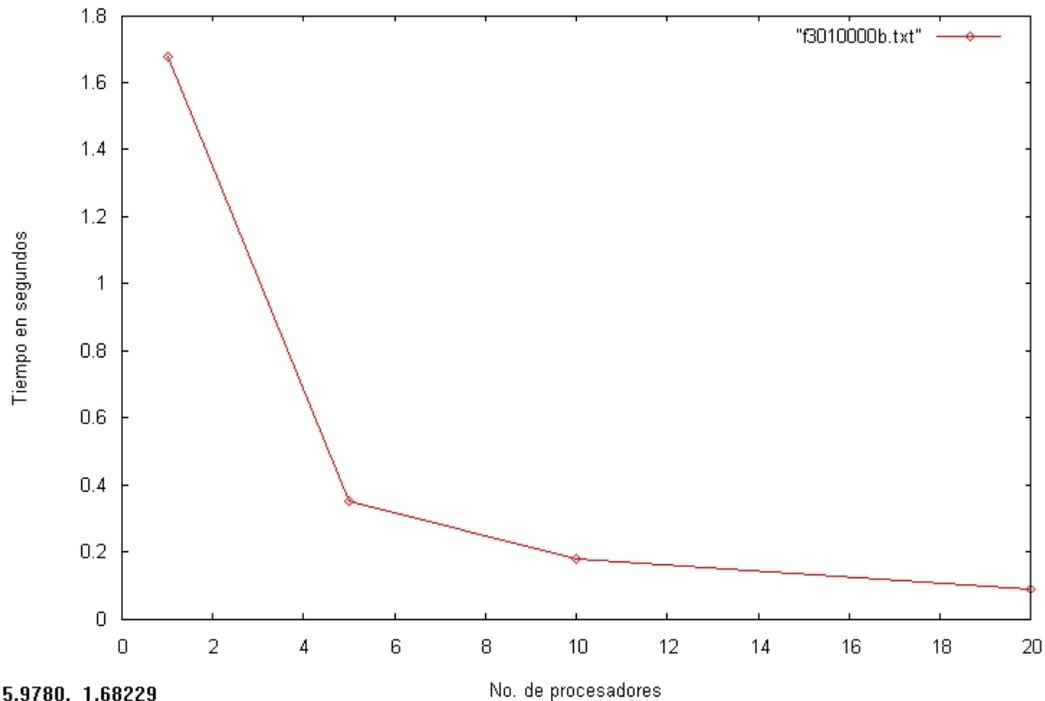
La gráfica 6.46 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 de trayectorias aleatorias de Feynman con 20 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 6.

6.3.6 Tiempos de ejecución para la integral de Feynman con 30 puntos



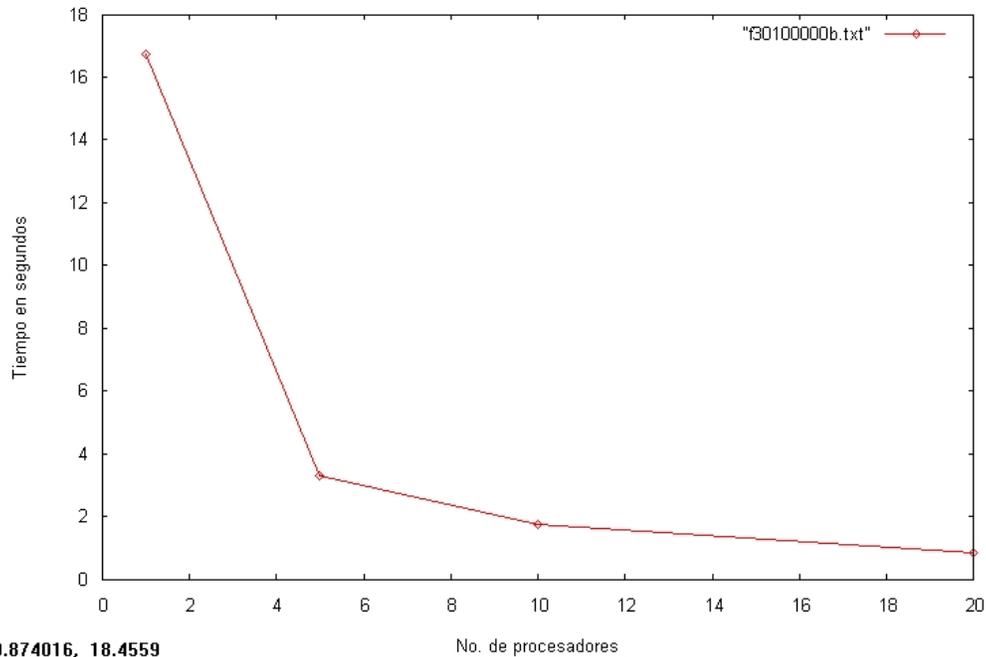
15.0142, 0.187579
Gráfica 6.47: Tiempos de ejecución para 1000 trayectorias de Feynman

La gráfica 6.47 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000 trayectorias aleatorias de Feynman con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



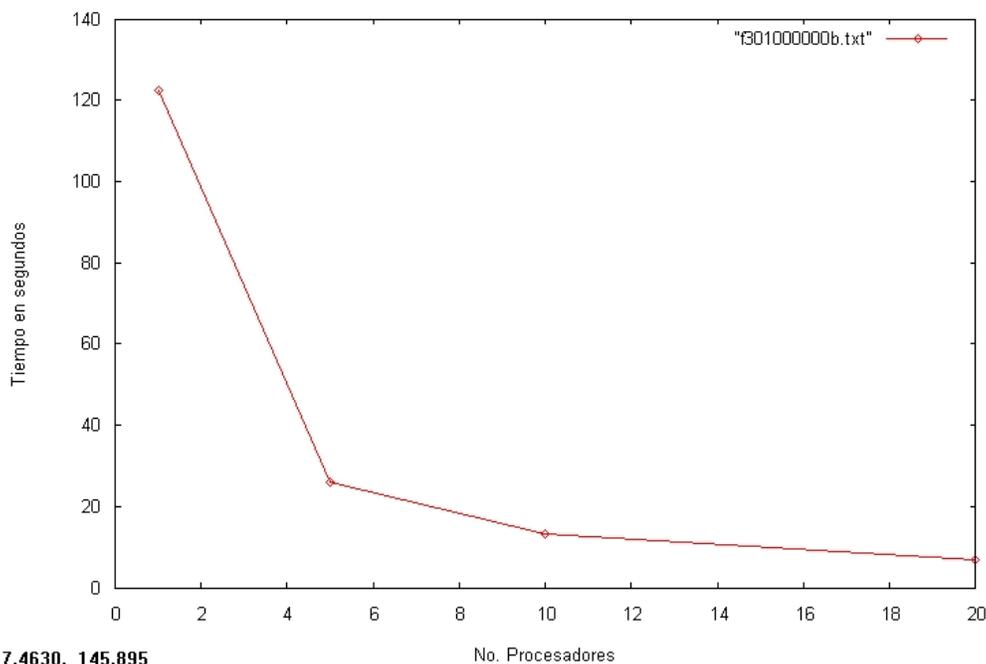
15.9780, 1.68229
Gráfica 6.48: Tiempos de ejecución para 10000 trayectorias de Feynman

La gráfica 6.48 muestra los tiempos obtenidos de las ejecuciones para la generación de 10000 trayectorias aleatorias de Feynman con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



0.874016, 18.4559
Gráfica 6.49: Tiempos de ejecución para 100000 trayectorias de Feynman

La gráfica 6.49 muestra los tiempos obtenidos de las ejecuciones para la generación de 100000 trayectorias aleatorias de Feynman con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.



17.4630, 145.895
Gráfica 6.50: Tiempos de ejecución para 1000000 de trayectorias de Feynman

La gráfica 6.50 muestra los tiempos obtenidos de las ejecuciones para la generación de 1000000 trayectorias aleatorias de Feynman con 30 puntos. Los tiempos graficados son los correspondientes a las ejecuciones de la tabla 7.

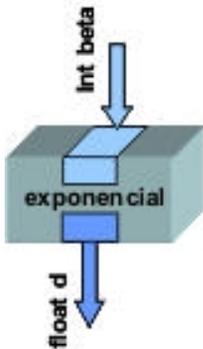
El objetivo de las figuras que corresponden a las gráficas de tiempos para Ito y Feynman, es mostrar el desempeño de paralelización de las funciones de Ito y Feynman para la generación de trayectorias aleatorias. En las imágenes se puede observar que conforme se aumenta el número de puntos en las trayectorias (de igual forma que sucede en Wiener) aumenta el tiempo de ejecución, pero conforme el número de procesadores es aumentado, fue posible disminuir los tiempos para el mismo número de trayectorias con la misma cantidad de puntos, lo que es posible apreciar en las gráficas mostradas.

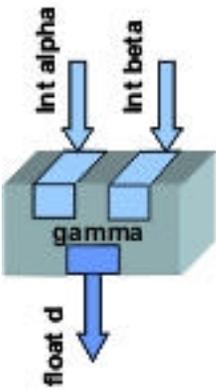
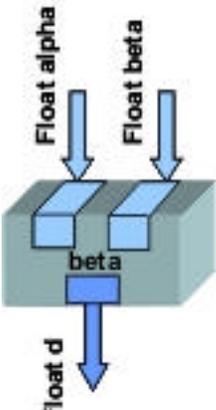
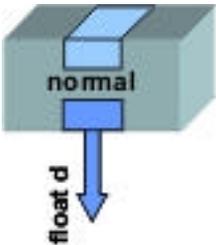
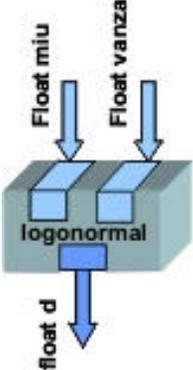
6.4 Resultados computación red amplia

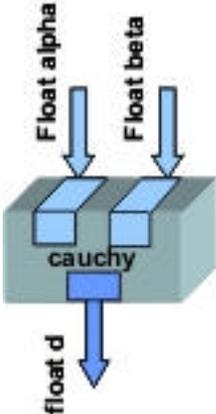
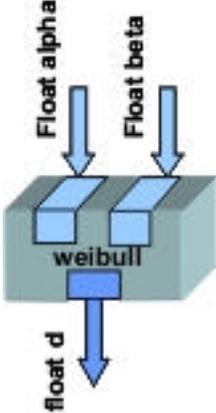
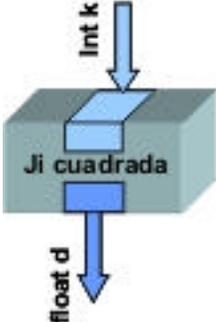
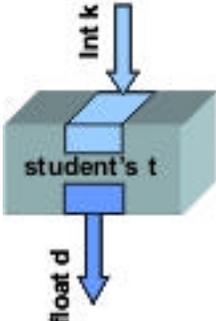
Los resultados obtenidos para la computación en red amplia, fueron el conjunto de componentes (servicios Web), de los cuales está compuesto el núcleo de Monte Carlo para la generación de números aleatorios con diferentes distribuciones, los servicios para la generación de trayectorias por los métodos de Wiener e Ito, los cuales se encargan de regresar los valores de las trayectorias generadas.

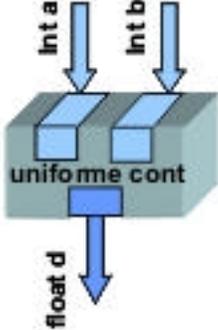
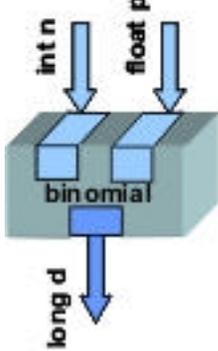
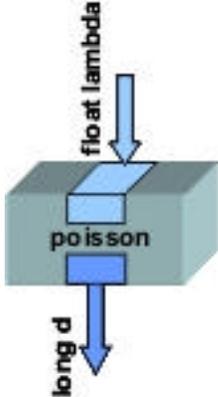
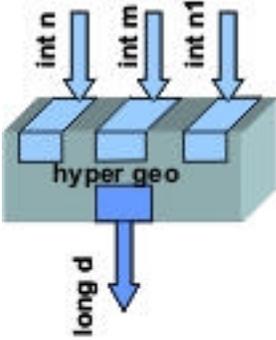
El objetivo inicial de la presente tesis fue desarrollar el núcleo de Monte Carlo y los métodos de las integrales de Wiener e Ito para computación paralela en servicios Web, para posteriormente ser incorporados a Taverna como componentes. Después de haber estudiado y analizado Taverna nos encontramos con algunas limitantes como son: la no existencia de ciclos, lo que no permite regresar a un componente anterior para seguir generando números. Los ciclos van a ser incorporados en una versión 2.0 de Taverna. Por el momento, los componentes quedarán dentro del área de trabajo de Taverna, si se quiere hacer uso de ellos es posible, pero el componente solo generará un solo número aleatorio de salida.

A continuación se presenta una tabla con los componentes implementados:

Método	Conexiones del componente	Entradas y salidas
Exponencial	 <p>El diagrama muestra un componente rectangular con el texto 'exponencial' en su interior. Una flecha azul apunta hacia abajo desde el texto 'Int beta' hacia la parte superior del componente. Otra flecha azul apunta hacia abajo desde la parte inferior del componente hacia el texto 'float d'.</p>	<p>Entrada:</p> <p>Beta: variable de tipo entero.</p> <p>d: variable de tipo float</p> <p>Salida: número aleatorio con comportamiento exponencial</p>

Método	Conexiones del componente	Entradas y salidas
Gamma	 <p>The diagram shows a 3D block labeled 'gamma' with two input ports on top labeled 'Int alpha' and 'Int beta'. A single output port on the bottom is labeled 'float d'.</p>	<p>Entrada:</p> <p>Alpha: variable de tipo entero.</p> <p>Beta: variable de tipo entero.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento gamma.</p>
Beta	 <p>The diagram shows a 3D block labeled 'beta' with two input ports on top labeled 'Float alpha' and 'Float beta'. A single output port on the bottom is labeled 'float d'.</p>	<p>Entrada:</p> <p>Alpha: variable de tipo float.</p> <p>Beta: variable de tipo float.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento beta.</p>
Normal	 <p>The diagram shows a 3D block labeled 'normal' with one input port on top. A single output port on the bottom is labeled 'float d'.</p>	<p>Entrada:</p> <p>Ninguna.</p> <p>d: variable de tipo float</p> <p>Salida: número aleatorio con comportamiento normal.</p>
Logonormal	 <p>The diagram shows a 3D block labeled 'logonormal' with two input ports on top labeled 'Float miu' and 'Float vanza'. A single output port on the bottom is labeled 'float d'.</p>	<p>Entrada:</p> <p>Miu: variable de tipo float.</p> <p>Vanza: variable de tipo float.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento logonormal.</p>

Método	Conexiones del componente	Entadas y salidas
Cauchy		<p>Entrada:</p> <p>Alpha: variable de tipo float.</p> <p>Beta: variable de tipo float.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento cauchy.</p>
Weibull		<p>Entrada:</p> <p>Alpha: variable de tipo float.</p> <p>Beta: variable de tipo float.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento weibull.</p>
Ji cuadrada		<p>Entrada:</p> <p>K: variable de tipo entero.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento chi-cuadrada.</p>
T de Student		<p>Entrada:</p> <p>K: variable de tipo entero.</p> <p>d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento t de student.</p>

Método	Conexiones del componente	Entadas y salidas
Uniforme continua	 <p>The diagram shows a 3D block labeled 'uniforme cont'. Two blue arrows labeled 'int a' and 'int b' point down into the top of the block. A blue arrow labeled 'float d' points down from the bottom of the block.</p>	<p>Entrada:</p> <p>a: variable de tipo entero. b: variable de tipo entero. d: variable de tipo float.</p> <p>Salida: número aleatorio con comportamiento uniforme continuo.</p>
Binomial	 <p>The diagram shows a 3D block labeled 'binomial'. Two blue arrows labeled 'int n' and 'float p' point down into the top of the block. A blue arrow labeled 'long d' points down from the bottom of the block.</p>	<p>Entrada:</p> <p>n: variable de tipo entero. p: variable de tipo float. d: variable de tipo long.</p> <p>Salida: número aleatorio con comportamiento binomial.</p>
Poisson	 <p>The diagram shows a 3D block labeled 'poisson'. One blue arrow labeled 'float lambda' points down into the top of the block. A blue arrow labeled 'long d' points down from the bottom of the block.</p>	<p>Entrada:</p> <p>lambda: variable de tipo float. d: variable de tipo long.</p> <p>Salida: número aleatorio con comportamiento poisson.</p>
Hipergeométrica	 <p>The diagram shows a 3D block labeled 'hyper geo'. Three blue arrows labeled 'int n', 'int m', and 'int n1' point down into the top of the block. A blue arrow labeled 'long d' points down from the bottom of the block.</p>	<p>Entrada:</p> <p>n: variable de tipo entero. m: variable de tipo entero. n1: variable de tipo entero. d: variable de tipo long.</p> <p>Salida: número aleatorio con comportamiento hipergeometrico.</p>

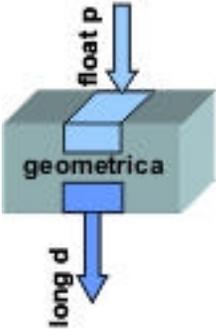
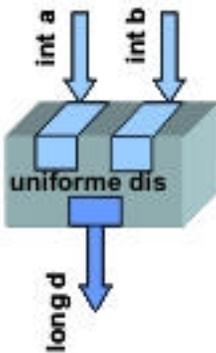
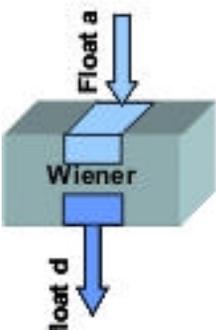
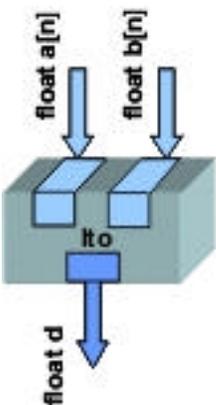
Método	Conexiones del componente	Entadas y salidas
Geométrica		<p>Entrada:</p> <p>p: variable de tipo float.</p> <p>d: variable de tipo long.</p> <p>Salida: número aleatorio con comportamiento geométrico.</p>
Uniforme discreta		<p>Entrada:</p> <p>a: variable de tipo entero.</p> <p>b: variable de tipo entero.</p> <p>d: variable de tipo long.</p> <p>Salida: número aleatorio con comportamiento uniforme discreto.</p>
Wiener		<p>Entrada:</p> <p>a: arreglo tipo float.</p> <p>d: variable de tipo long.</p> <p>Salida: valor trayectoria.</p>
Ito		<p>Entrada:</p> <p>a: arreglo tipo float.</p> <p>b: arreglo tipo float.</p> <p>d: variable de tipo float.</p> <p>Salida: valor trayectoria.</p>

Tabla 6.8: Servicios Taverna

6.5 Conclusiones

En este trabajo de tesis se describió el diseño e implementación de dos bibliotecas: una que comprende el núcleo de Monte Carlo y otra que se refiere a la generación de trayectorias aleatorias. A estas bibliotecas se les denominó como “Núcleo de Monte Carlo y Camino Aleatorio (NNMYCA)”.

En el diseño del proyecto fueron considerados varios factores como los ambientes en donde iban a ser utilizadas las bibliotecas y los tipos de problemas que podían hacer uso de la misma. También fue considerado que algunos problemas iban a requerir de una HPC (High Performance Computing), por lo que se pensó en el diseño para un cluster y el desarrollo del núcleo y los métodos de generación de trayectorias aleatorias de Wiener e Ito como servicios Web para un ambiente Grid.

En un ambiente paralelo es posible el uso de NNMYCA en programas que requieran de la generación de números aleatorios, así como de su uso en programas que impliquen un problema el que requiera de alguno de los métodos de Wiener, Ito y Feynman para generación de trayectorias en forma aleatoria.

El NNMYCA ofrece un conjunto de funciones las que permiten al usuario no preocuparse por la forma en que son obtenidos los números o las trayectorias, sino poder concentrarse en su problema a resolver. El núcleo de Monte Carlo es posible ejecutarlo tanto en programas en forma secuencial como en forma paralela, debido a que los algoritmos para la generación de números aleatorios con diferente distribución de probabilidad, podrían considerarse como una operación atómica, en donde su resultado es únicamente un número, debido a esto no se considero necesaria su paralelización. Para el uso de las funciones del núcleo de MC es posible su llamado dentro del ámbito de un programa en paralelo, como se hace en las funciones de generación de trayectorias. El llamado de las funciones en forma secuencial dentro de un ámbito paralelo, se considera un modelo SPMD (Simple Program Multiple Data), por lo que la paralelización se considera implícita.

Además el NNMYCA también fue desarrollado como servicios Web para su uso en un ambiente Grid, lo cual fue posible por su desarrollo como servicio Web e incorporación a Taverna como componente. Como ya se había mencionado en la sección 6.4, nos encontramos con algunas limitantes en Taverna como son: la no existencia de ciclos, lo que no permite regresar a un componente anterior para seguir generando números. Los ciclos van a ser incorporados en una versión 2.0 de Taverna. Por el momento, los componentes

quedarán dentro del área de trabajo de Taverna de manera que si se quiere hacer uso de ellos es posible, pero el componente solo generará un solo número aleatorio de salida.

Para probar el funcionamiento de los métodos de Wiener e Ito, fueron generadas muestras de una gran cantidad de trayectorias aleatorias, para obtener sus medias y varianzas tanto muestral como poblacional. Para cada ejecución, las muestras de trayectorias fueron creadas con la misma cantidad de números por trayectoria y la misma cantidad de trayectorias a generar, variando el número de procesadores. Con lo que fue posible constatar que con el paralelismo implementado en los métodos fue posible obtener mejores tiempos de ejecución, aumentando el número de procesadores.

Para la prueba de la función de Feynman fue necesario disponer de algún problema para la generación de trayectorias, ya que con este método las trayectorias generadas en forma aleatoria corresponden al comportamiento de un sistema en un intervalo de tiempo dado. Por lo que como ejemplo se tomó el oscilador armónico, en el cual se hace una simulación de su comportamiento a través de la generación de trayectorias, haciendo uso de su diseño en forma matemática.

Con las pruebas realizadas a la función de Feynman, fue posible corroborar la misma observación realizada con los métodos de Wiener e Ito: la creación de muestras con la misma cantidad de trayectorias con igual número de puntos, con el paralelismo aplicado y el aumento de procesadores fue posible obtener mejores tiempos para la ejecución.

La principal aportación es el diseño e implementación de una biblioteca de generación de números aleatorios, así como de una biblioteca de generación de trayectorias para los ambientes de alto desempeño.

Las aportaciones particulares de nuestro trabajo son listadas de la siguiente forma:

- Diseño e implementación de una biblioteca de generación de números que obedecen a una cierta distribución de probabilidad. Se basa en los principios de Monte Carlo, a la cual se le denomina núcleo de Monte Carlo.
- Diseño e implementación de una biblioteca de generación de trayectorias basada en los métodos de Ito y Feynman.
- Implementación del método de Wiener para la generación de trayectorias en forma paralela.

- Diseño e implementación de una función que permite generar semillas en forma paralela.
- La extensión a Taverna con el desarrollo de servicios Web de la biblioteca del núcleo de Monte Carlo y de los métodos de Wiener e Ito para la generación de trayectorias aleatorias.

6.6 Trabajo futuro

Como trabajo futuro es posibles hacer las siguientes recomendaciones:

- Cálculo de las trayectorias a través del método de MC para los diferentes métodos de construcción de trayectorias de Wiener, Ito y Feynman.
- Implementación de las integrales de Wiener, Ito y Feynman para aplicaciones financieras.
- Pruebas de los módulos de Taverna con ejemplos prácticos.
- Implementación de la Integral de Feynman y sus aplicaciones físicas como servicio para su uso en Taverna dentro de la e-science.

Apéndice A

Pruebas distribución uniforme y normal

En el presente trabajo de tesis fueron utilizadas dos distribuciones: una la distribución uniforme y la distribución norma, ya que son de suma importancia para el desarrollo de dicho trabajo.

A.1 Uniforme

La distribución uniforme es la base del núcleo de Monte Carlo para la generación de números aleatorios en base a diferentes comportamientos (distribución de probabilidad). Por esta razón fue utilizado el ran3, el cual ya ha sido desarrollado y estudiado anteriormente, este se encarga de generar números aleatorios entre 0 y 1 con comportamiento uniforme. Con el fin de mostrar su eficiencia en el trabajo de tesis, fue realizada una prueba, la cual consiste en generar una cantidad de números aleatorios y graficarla en pares ordenados, en la cual es posible observar y comprobar que tan bueno es el generador ran3.

La forma de observar su eficiencia es en la distribución de dichos puntos en un plano cartesiano de dos dimensiones. Si estos pares son distribuidos uniformemente sobre la superficie (lo que implica que no se concentren en ciertos puntos), es posible concluir que es un buen generador.

A continuación, en la Figura A.1, se presenta una imagen con 25000 pares ordenados. Como se puede observar los puntos son distribuidos uniformemente.

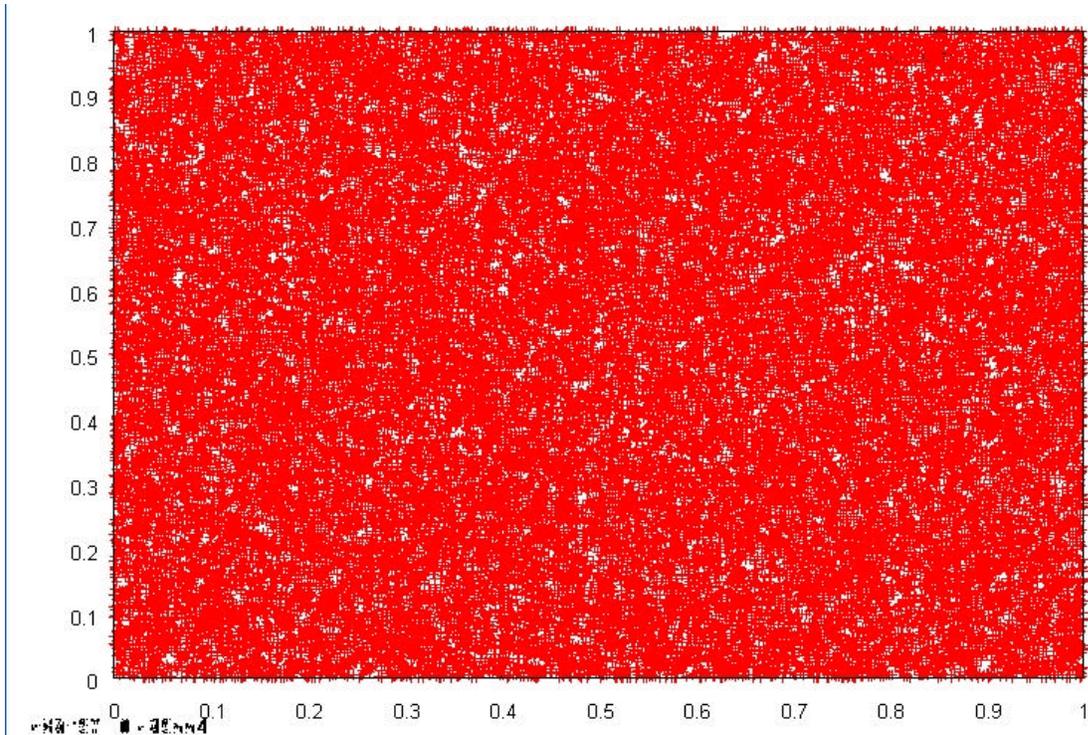


Figura A.1: Generador ran3 para 50000 puntos

A continuación, en la Figura A.2, se muestra otra imagen con 1000000 de puntos. En esta imagen se puede observar que el plano es cubierto completamente de forma uniforme por los números aleatorios.

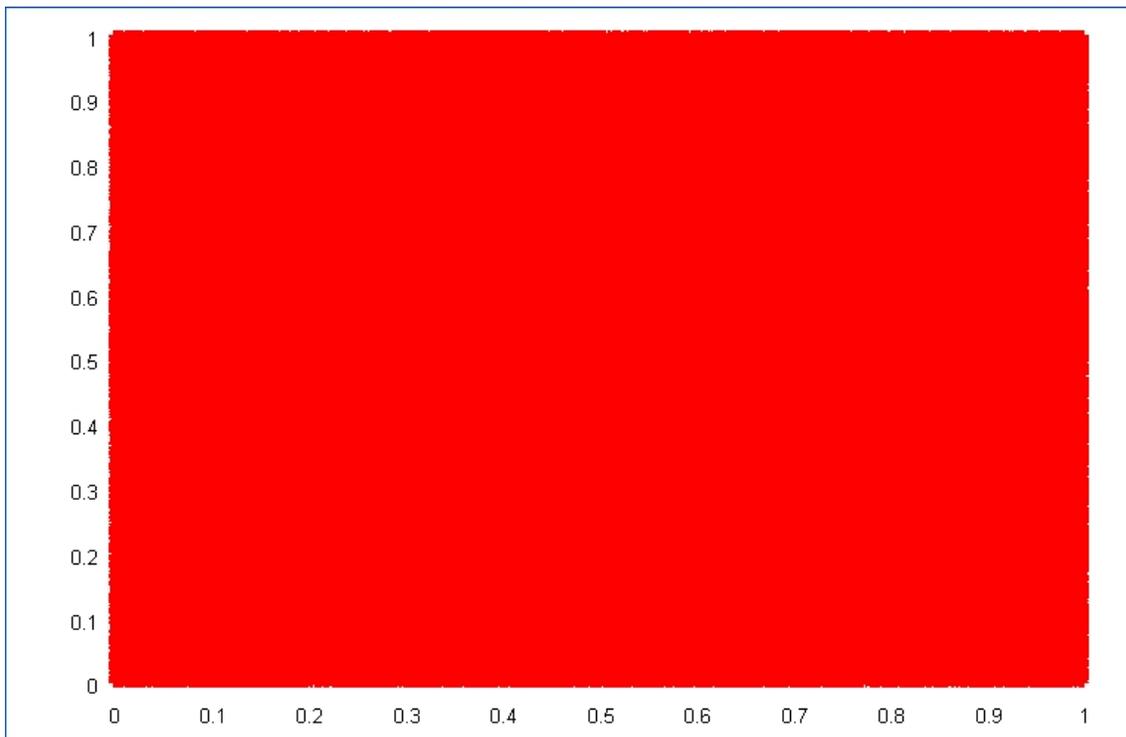


Figura A.2 Generador ran3 para 1000000

A.2 Normal

El algoritmo de Box Muller fue utilizado para su implementación e integración al núcleo de Monte Carlo de este trabajo, pero además, ya integrado al núcleo, fue utilizado para la generación de números aleatorios, con lo que fueron construidas las trayectorias aleatorias en los métodos de Wiener, Ito y Feynman. Lo anterior debido a que la mayoría de los fenómenos naturales tiene un comportamiento normal.

El comportamiento de la distribución normal o de Gauss es en forma de una campana como se muestra en la Figura A.3.

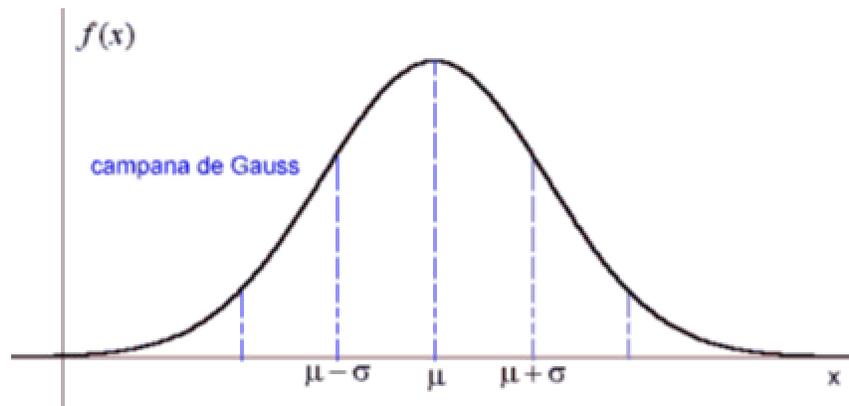


Figura A.3: Comportamiento de la distribución normal

Para probar el buen funcionamiento del algoritmo de Box Muller, se realizó un programa para generar números aleatorios con comportamiento normal, de dicho programa se obtiene un archivo con los rangos y valores correspondientes a un histograma. Fueron generados diferentes cantidades de números aleatorios y graficados sus histogramas a través de gnuplot. A continuación son presentados algunos de los resultados obtenidos:

En la Figura A.4 se muestra el histograma con 50000 datos generados y la semilla igual a 34567688, con una cantidad de 9 intervalos.

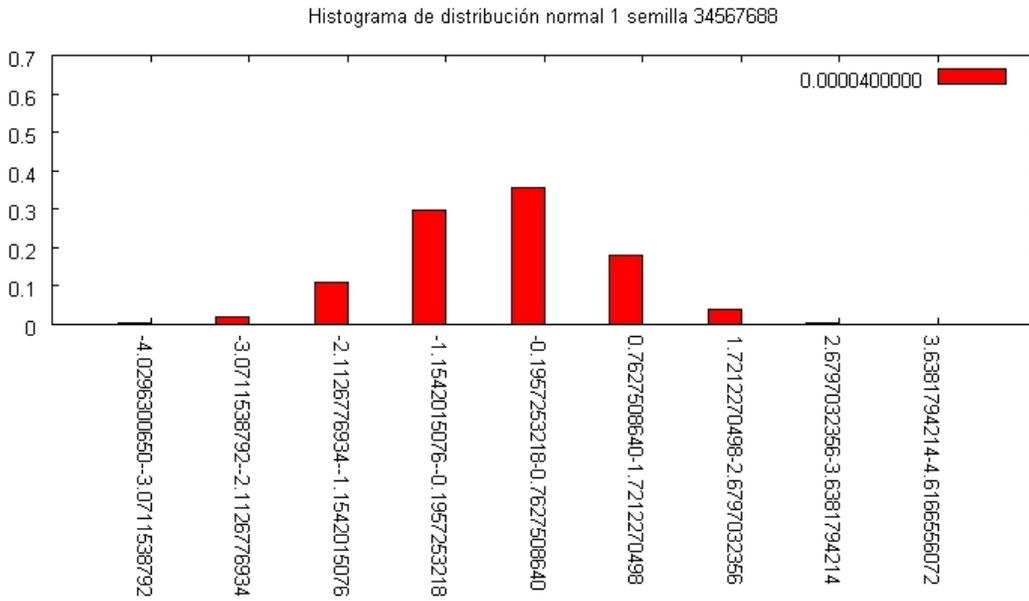


Figura A.4: Histograma de distribución normal para 50000 datos

Como se puede observar, el resultado del histograma es una campana, aproximadamente entre los rangos -4 a 3, con lo que es posible asumir que los números aleatorios generados son buenos y su comportamiento corresponde a una distribución normal.

Para obtener una histograma mejor definido fueron generados 500000 datos para 19 intervalos, con una semilla igual a 1234567890. La Figura A.5 muestra el histograma resultante.

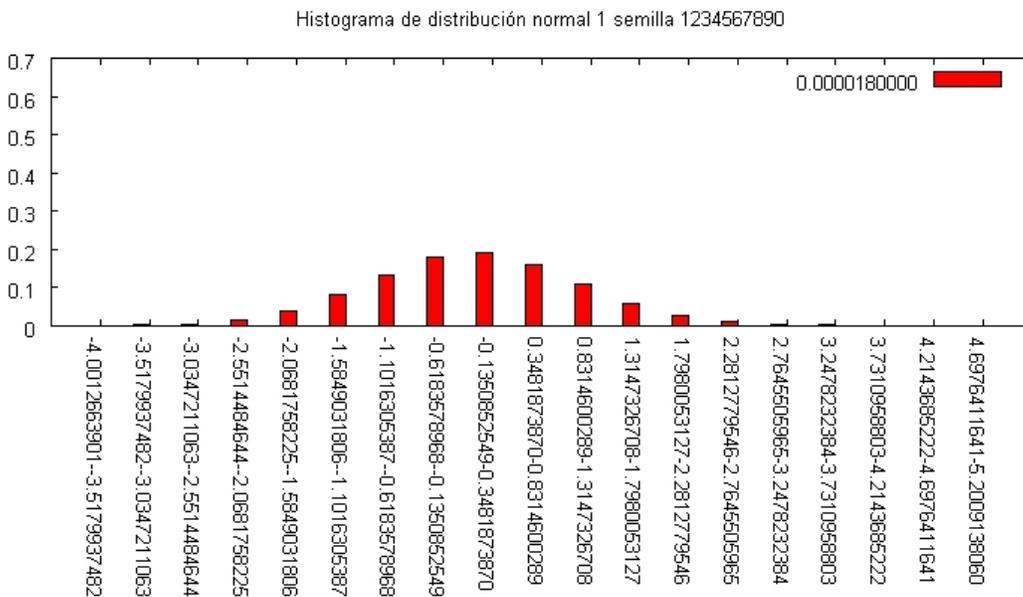


Figura A.5 Histograma de distribución normal para 500000 datos

Como se puede ver en la figura A.5 fueron obtenidos buenos resultados aplicando el algoritmo de Box Muller. Se obtuvo un comportamiento normal, lo cual puede ser visto por el comportamiento del histograma en forma de campana, de igual forma que en la imagen anterior, los resultados están entre -3 y 3.

Se toma como referencia el rango -3 a 3, ya que los rangos de probabilidad de la distribución normal están definidos de la siguiente forma:

$$P(\mathbf{m}-\mathbf{s} < X < \mathbf{m}+\mathbf{s}) = 0.6827$$

$$P(\mathbf{m}-2\mathbf{s} < X < \mathbf{m}+2\mathbf{s}) = 0.9545$$

$$P(\mathbf{m}-3\mathbf{s} < X < \mathbf{m}+3\mathbf{s}) = 0.9973$$

A continuación se presenta uno de los histogramas resultante del programa anteriormente mencionado, el histograma fue ajustado a una función con comportamiento normal, la Figura A.6 muestra el resultado.

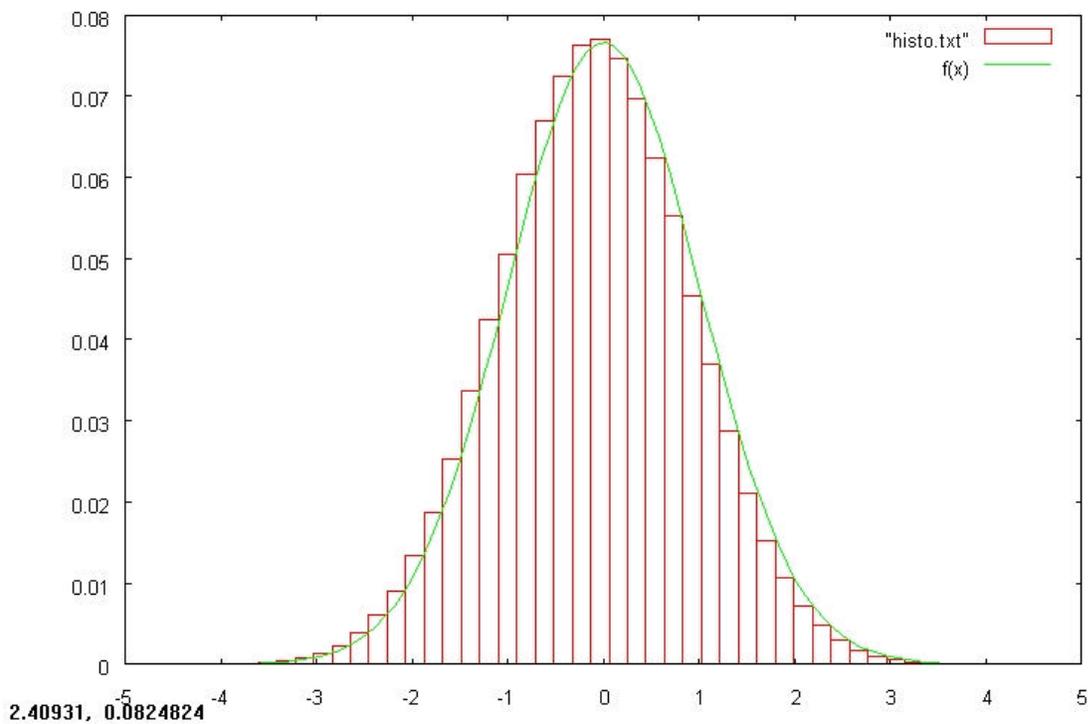


Figura A.6: Histograma ajustado a una función con comportamiento normal.

En la figura es posible observar que el resultado de la comparación es bueno, debido a que prácticamente tanto el histograma como el ajuste coinciden por completo. Además es posible observar la forma de campana del histograma, la misma forma de un histograma con comportamiento normal.

REFERENCIAS

- [1] Dr. Sergio V. Chapa Vergara; “*Laboratorio de Computación Científica: Un Ambiente Computacional para Ingeniería Eléctrica y Solución de Problemas Científicos*”, Centro de Investigación y de Estudios Avanzados, Sección Computación. México, D.F., mayo de 2005.
- [2] Frost Gorder P.; “*Multicore Processors For Science and Engineering*”, vol. 9, pp. 3-7. March-April 2007. IEEE.
- [3] Ramanathan R.M.; “*Intel Multi-core Processors: Making the Move to Quad-Core and Beyond*”, Intel,
- [4] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc. San Francisco California. 1997.
- [5] Cheng A., Folk M., Li R. “*HDF Support For Network Of Workstations*”. National Center for Supercomputing Applications. University of Illinois. Abril 1996. http://hdf.ncsa.uiuc.edu/Parallel_HDF/HDFNOW/hdfnow-whitepaper.pdf
- [6] Lastovetsky L. A lexey. *Parallel Computing on Heterogeneous Networks*. Wiley Interscience. 2003.
- [7] Plataform Brief, “*Boost Performance and Energy Efficiency with New 45nm Multicore Intel Xeon Processors*”. Intel Xeon Processor Technology. http://www.intel.com/products/processor/xeon5000/index.htm?iid=technology_quadcore_index+body_xeon5400
- [8] Geer, D.; “*For Programmers, Multicore Chips Mean Multiple Challenges*”, vol. 40, pp. 17-19. Sep 2007. IEEE.
- [9] Holohan A. “*Collaboration Online: The Example of Distributed Computing*”. <http://jcmc.indiana.edu/vol11/issue4/holahan.html>
- [10] Coulouris, George; Dollimore, Jean; Kindberg, Tim. “*Sistemas Distribuidos: Conceptos y Diseño*”. 3a ed. Madrid: Pearson Educación S.A. 2001.
- [11] Scott R. L., Clarck T., Bagheri B. “*Scientific Parallel Computing*”. ISBN: 0-691-11935-X. Princeton University Press. 2005.
- [12] Página Web: *Computadora Central*. http://es.wikipedia.org/wiki/Computadora_central
- [13] Página Web: *Introduction to Parallel Computer*.

https://computing.llnl.gov/tutorials/parallel_comp.

- [14] Hernandez Hernandez Carlos Alberto. “*Algoritmos de ordenamiento y teoría de gráficas en una computadora paralela de tipo hipercubo*”. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 1995.
- [15] Sato M. OpenMP: “*Parallel programming API for shared memory multiprocessors and on-chip multiprocessors*”. International Symposium System Synthesis. pp. 109-111. 2002. IEEE.
- [16] Geist G. A., Kohl J. A., Papadopoulos P. M.; “*PVM and MPI: a Comparison of Features*”, Mayo 1996, <http://Citeset.ist.psu.edu/geist96pvm.html>
- [17] <http://www.lam-mpi.org/>
- [18] http://www.csm.ornl.gov/pvm/pvm_home.html
- [19] Gilfeather F., Kovatch P. “*Superclusters: TeraClass Computing*”. High Performance Computing in the Asia-Pacific Region, The Fourth International Conference, vol. 2, pp. 874-883. 14-17 May 2000. IEEE.
- [20] Mukherjee S., Mustafi J. Chaudhuri A.; “*Grid Computing: The Future of Distributed Computing for High Performance Scientific and Business Applications*”. Institute of Information Technology, Kolkata, INDIA. 2002.
- [21] Laszewski G.V., “*Grid Computing: Enabling a Vision for Collaborative Research*”. Argonne National Laboratory, 2002.
<http://citeseer.ist.psu.edu/vonlaszewski02grid.html>
- [22] UC Berkeley; “*Kepler Project*”,
<http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm>, 2003.
- [23] Beltram Ludäscher; Ilkay Altinas; Chad Berkey; Dan Higgs; Efrat Jaeger; Matthew Jones; Edward A. Lee; Jing Tao; Yang Zhao; “*Scientific Workflow Management and the Kepler System, Concurrency and Computation: Practice and Experience*”. 2005. <http://citeseer.ist.psu.edu/ludscher05scientific.html>
- [24] I. Taylor; Matthew Shields; Ian Wang; Roger Philp; “*Grid Enabling Applications Using Triana, In Workshop on Grid Applications and Programming Tools*”, Settle 2003. <http://citeseer.ist.psu.edu/taylor03grid.html>
- [25] T. Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter

- Li, Phillip Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, Chris Wroe. *“Taverna: Lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and experience”*, April 27 2005.
- [26] Fishman George S. *Monte Carlo: Concepts, Algorithms and Applications*. Springer. 1995.
- [27] Newman M. E. J.; Barkema G.T. *Monte Carlo Methods in Statistical Physics*. OXFORD University Press. 2001.
- [28] Rubistein R. Y. *Simulation and the Monte Carlo Method*. John Wiley & Sons, United States of America. 1981.
- [29] Landau D. P.; Binder K. *Monte Carlo Simulations in Statistical Physics*. Cambridge University Press. 2002.
- [30] I. M. Sobol, Alya Sobol, I. M. Sobol, *A Primer for the Monte Carlo Method*. Sch. of Eng, Editorial: CRC Press Inc, ISBN: 084938673X, página(s):1-10, Año 1994.
- [31] No 15. Special Issue, Stanislaw Ulman 1909-1984, *Los Alamos Science*, Los Alamos National Laboratory. 1987.
- [32] Dr. Sergio V. Chapa Vergara, *“Tesis: Integral de Wiener y su estimación numérica”*, Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional, 1973.
- [33] Liptser R., Lecture 3, *“Wiener Process. Stochastic Ito Integral”*, Department of Electrical Engineering-Systems, www.eng.tau.ac.il/~liptser/lectures1/lect3.pdf
- [34] Luis Rincón, *“Construyendo la integral estocástica de Ito”*, Sociedad Matemática de México. www.smm.org.mx/SMMP/html/modules/Publicaciones/AM/Cm/35/artExp12.pdf
- [35] Wiegand I.W. *“Introduction to Path-Integral Methods in Physics and Polymer Science”*, World Scientific, 1986.
- [36] Dra. Julieta Medina García, *“Tesis: Teorías de Campo Escalares Difusas: Investigaciones Analíticas y Numéricas”*, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Abril, 2006.
- [37] Equipo Krüss, *“Path Integral Formulation”*, <http://www.answers.com/topic/feynman-integral?cat=technology>
- [38] H. Press William, Flannery Brian P., Saul A. Teukolsky, Vetterling T. William,

“Numerical Recipes: The Art of scientific in C Computing”, Second Edition, 1992.

- [39] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc. San Francisco California.1997.
- [40] Perrot R. H. *”e-Science and it’s application: life science and finance”*. Computer Architecture and High Performance Computing. Octubre 2005.
- [41] T. Oinn, Greenwood M. Addis M., Nedim A., Ferris J., Glover K., Goble C., Goderis A., Hull D., Marvin D., Li P., Lord P., Pocock R. M., Stevens R., Wipat A., Wroe C. *“Taverna: Lessons in creating a workflow environment for the life sciences”*. Concurrency and Computation: Practice and Experience, 27 April 2005.
- [42] Zhao A., Belloum, A., Wibisono A. *“Scientific workflow, management: between generality and applicability”*, Fifth International Conference on Quality Software, 2005.(QSIC), 2005 September.
- [43] Foster I., Kesselman C. *“Scaling System-Level Science: Scientific Exploration and IT Implications”*, Inovative Technology for Computer Professionals Computer, vol. 39, IEEE, 2006 November.

