



INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

**OPERADORES LINEALES EN GPUS APLICADOS A
LA EVALUACIÓN DE ENERGÍA POTENCIAL
PARA PROTEÍNAS**

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA
ERICKA GARCÍA BLANQUEL

DIRECTOR
DR. RENÉ LUNA GARCÍA



México D.F.

Diciembre 2012



**INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO**

SIP-14

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 10:15 horas del día 23 del mes de noviembre de 2012 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Operadores lineales en GPUs aplicados a la evaluación de energía potencial para proteínas"

Presentada por el alumno:

GARCÍA

Apellido paterno

BLANQUEL

Apellido materno

ERICKA

Nombre(s)

Con registro:

B	1	0	1	6	4	7
---	---	---	---	---	---	---

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de Tesis

Dr. René Luna García

Dra. Nareli Cruz Cortés

Dr. Ricardo Barrón Fernández

Dr. Marco Antonio Ramírez Salinas

Dr. Jesús Guillermo Figueroa Nazuno



PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN
EN COMPUTACIÓN
DIRECCIÓN



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de **México D.F** el día **30** del mes **noviembre** del año **2012**, la que suscribe **Ericka García Blanquel** alumna del Programa de **Maestría en Ciencias de la Computación** con número de registro **B101647**, adscrita al **Centro de Investigación en Computación**, manifiesta que es autora intelectual del presente trabajo de Tesis bajo la dirección del **Dr. René Luna García** y cede los derechos del trabajo intitulado **"Operadores lineales en GPUs aplicados a la evaluación de energía potencial para proteínas "**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección eri.gb07@gmail.com . Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ericka García Blanquel

Nombre y firma

RESUMEN

El presente trabajo se enmarca dentro de la línea de investigación del cómputo de alto rendimiento con una aplicación de propósito general en el área de dinámica molecular. En particular esta enfocado en realizar la evaluación de energía potencial en proteínas haciendo uso de las unidades gráficas de procesamiento (GPU) para aprovechar el paralelismo en el procesamiento de datos masivos que nos ofrece esta tecnología, esto se realizará mediante un modelo matemático que nos defina el comportamiento de la energía potencial y posteriormente la minimización de la energía molecular. Debido a que este proceso es de gran importancia dentro de cualquier aplicación de dinámica molecular, se propone implementar una nueva técnica para la minimización de energía molecular u optimización geométrica, por medio del uso de los operadores lineales, con los que se obtendrá la geometría final, que es la configuración de la molécula con sus valores naturales o ideales. Para el diseño e implementación del algoritmo se hizo en función de la arquitectura CUDA provista por NVIDIA, así como de su lenguaje de programación.

Palabras clave: Simulación, GPU, GPGPU, CUDA, cómputo paralelo, modelado molecular, energía potencial, operador lineal, transformaciones geométricas.

ABSTRACT

This work is focus on the research of high performance computing by developing a general purpose application in the field of molecular dynamics. In particular we computed the potential energy of proteins using graphics processing units (GPU) to exploit parallelism in processing data offered by this technology, we use a mathematical model to define the behavior for the potential energy and this is evaluated on the valid range. Due to the energy minimization is the most important part of the all molecular applications, thats why we use of linear operators like another proposal to get the final geometry with the natural values, the design and implementation of the algorithm is based on the CUDA architecture provided by NVIDIA

Key words: Simulation, GPU, GPGPU, CUDA, parallel computing, molecular modeling, potential energy, linear operator, geometric transformations.

DEDICATORIA

Esta tesis esta dedicada a la persona más importante en mi vida, la que me ha brindado la mayor motivación, paciencia, comprensión pero sobre todo amor mi hijo

Alexis Maximiliano Mendoza García

*Es increíble ver como alguien tan pequeñito,
puede hacer que las cosas
tengan un cambio tan gigantesco.*

AGRADECIMIENTOS

A Dios

Por haberme permitido llegar hasta este punto y haberme puesto en el lugar y momento correctos, para lograr mis objetivos.

A mis padres Juanita y Enrique

Porque nunca me han dejado sola y menos en los momentos más difíciles de mi vida, por apoyarme de forma incondicional, por sus consejos, sus valores y por la motivación constante que recibo día a día, todo eso que me ha permitido ser la persona que soy, pero más que nada por enseñarme que con el amor de una familia se pueden conseguir todos los sueños.

A mis hermanas Claudia y Nidia

Por permitirme compartir mis aciertos y debilidades, por confiar en mí, por sus consejos y por todos esos momentos que han estado presentes en mi vida. A mi sobrina Novaly por ser como una hermana para mi hijo, con la cual tiene la oportunidad de sentir lo que yo siento con las mías.

A Marco A. Mendoza Hernández

Por ser una persona muy importante en mi vida, pero sobre todo porque desde que inicie este proyecto ha tratado de entenderme y apoyarme en cada momento.

A mis amigos y amigas

Porque gracias a su apoyo, compañía y buenos consejos día a día motivaron mis deseos por avanzar hasta culminar este proyecto.

Al Dr. René Luna García

Primero que nada por haber aceptado participar conmigo en este reto personal el cual mediante su apoyo, enseñanzas y constante seguimiento así como su valioso tiempo, han hecho que se esto se hiciera realidad, también gracias por brindarme su amistad ya que esto facilitó el camino durante estos años.

Al Instituto Politécnico Nacional

Por ser mi alma mater y darme la oportunidad continuar con mi preparación como profesionista, para ser una mujer de bien y útil en una sociedad que exige cada día personas más preparadas. Por haberme proporcionado, las herramientas necesarias y de esta forma enfrentarme a las adversidades que presentan día a día.

A mis profesores y al comité tutorial

Por su apoyo, enseñanzas, consejos y señalamientos, ya que llevaron a que este trabajo culminará lo mejor posible en cuanto lo académico.

Al Centro de Investigación en Computación y al Consejo Nacional de Ciencia y Tecnología

Por su imprescindible apoyo en la realización de este trabajo.

ACRÓNIMOS

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FF	Force Field
GPU	Graphics processing unit
GPGPU	General Purpose Computing on Graphics Processing Unit
MD	Molecular Dynamics
MM	Molecular Mechanics
PDB	Data Bank Protein

GLOSARIO

Aminoácido: Son las unidades químicas o elementos constitutivos de las proteínas.

Campo de fuerza: Es la región del espacio donde se manifiesta una fuerza.

Dinámica molecular: La dinámica molecular es un campo multidisciplinario en el que sus leyes y teorías provienen de las matemáticas, física y química.

Energía potencial: En un sistema físico, es la energía que mide la capacidad que tiene dicho sistema para realizar un trabajo en función exclusivamente de su posición o configuración.

Interacción molecular: Son las uniones que se crean entre las moléculas y son de gran importancia, debido a que son las que cambian las características de un compuesto.

Mecánica molecular: Usa las ecuaciones de un campo de fuerzas de la mecánica clásica, para describir las superficies de energía potencial y propiedades físicas de la molécula.

Minimización u optimización geométrica: Es una técnica que busca encontrar un conjunto de coordenadas que minimizan la energía potencial del sistema de interés.

Modelado molecular: Es un término general que engloba métodos teóricos y técnicas computacionales para modelar, imitar y predecir el comportamiento de moléculas.

Operador lineal: Es una transformación lineal entre dos espacios vectoriales.

Proteína: Son moléculas formadas por cadenas lineales de aminoácidos.

Transformación geométrica: Son procesos de variación o movimiento de los puntos del plano, de forma que se establece una relación entre los elementos origen y los elementos transformados.

Unidad gráfica de procesamiento: Es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central.

ÍNDICE GENERAL

INTRODUCCIÓN	14
1.1 PLANTEAMIENTO DEL PROBLEMA	14
1.2 OBJETIVO GENERAL	17
1.3 OBJETIVOS ESPECÍFICOS	17
1.4 JUSTIFICACIÓN	18
1.5 DELIMITACIÓN Y LIMITACIONES DE LA INVESTIGACIÓN	19
ESTADO DEL ARTE	20
2.1 ANTECEDENTES	20
FUNDAMENTOS TEÓRICOS	23
3.1 UNIDADES GRÁFICAS DE PROCESAMIENTO (GPUS).....	23
3.1.1 <i>Historia de la GPU como procesador de Cálculo</i>	23
3.1.2 <i>Cómputo de Propósito General sobre Unidades Gráficas de Procesamiento (GPGPU)</i>	25
3.1.3 <i>CUDA (Compute Unified Device Architecture)</i>	26
3.1.4 <i>Arquitectura CUDA</i>	26
3.1.5 <i>Modelo de programación</i>	28
3.1.6 <i>Gestión de memoria y comunicación de datos entre CPU y GPU</i>	33
3.1.7 <i>Aplicaciones en entornos de alto rendimiento</i>	36
3.2 PROTEÍNAS.....	37
3.2.1 <i>Aminoácidos</i>	37
3.2.2 <i>Los aminoácidos esenciales</i>	38
3.2.3 <i>Enlace peptídico</i>	38
3.2.4 <i>Estructura de las proteínas</i>	39
3.2.5 <i>Propiedades de las proteínas</i>	41
3.2.6 <i>Funciones de las proteínas</i>	42
3.2.7 <i>Relación estructura – función de las proteínas</i>	42
3.2.8 <i>Determinación experimental de las proteínas</i>	43
3.2.9 <i>Banco de datos</i>	43
3.3 BIOINFORMÁTICA Y BIOLOGÍA COMPUTACIONAL	46
3.3.1 <i>Dinámica molecular computacional</i>	46
3.3.2 <i>Modelado Molecular</i>	47
3.3.3 <i>Mecánica molecular</i>	48
3.3.4 <i>Conformación y propiedades moleculares</i>	50
3.3.5 <i>Interacciones moleculares</i>	51
3.3.6 <i>Energía potencial</i>	52
3.3.7 <i>Potenciales y campos de fuerza</i>	53
3.3.8 <i>Desarrollo de los campos de fuerza</i>	53
3.3.9 <i>Proceso de optimización</i>	54
3.4 OPERADORES LINEALES.....	55
3.4.1 <i>Transformaciones geométricas</i>	55

3.5 OPENGL	64
3.5.1 ¿Qué es OpenGL?	64
3.5.2 OpenGL como una máquina de estados	64
3.5.3 Bibliotecas relacionadas con OpenGL	65
3.5.4 Transformaciones geométricas con OpenGL	65
3.5.5 Aplicaciones	66
METODOLOGÍA	67
RESULTADOS	80
CONCLUSIONES Y TRABAJOS FUTUROS	83
6.1 CONCLUSIONES	83
6.2 TRABAJOS FUTUROS	83
APÉNDICE A	85
APÉNDICE B	87

ÍNDICE DE FIGURAS

FIGURA 1: CO-PROCESAMIENTO ENTRE CPU Y GPU	25
FIGURA 2: COMPARACIÓN ENTRE CPU Y GPU	27
FIGURA 3: ARQUITECTURA CUDA	28
FIGURA 4: ARQUITECTURA CPU-GPU	29
FIGURA 5: JERARQUÍA DE HILOS	30
FIGURA 6: DEFINICIÓN E INVOCACIÓN DE UN KERNEL	32
FIGURA 7: EJECUCIÓN ALTERNADA, ENTRE EL CPU Y EL GPU	35
FIGURA 8 COMPOSICIÓN DE AMINOÁCIDOS	37
FIGURA 9: ENLACE PEPTÍDICO	39
FIGURA 10: ESTRUCTURAS DE LAS PROTEÍNAS	40
FIGURA 11: CAMPO DE FUERZAS	49
FIGURA 12: REGLAS DE UN OPERADOR LINEAL	55
FIGURA 13: EJEMPLO DE ESCALAMIENTO 3D	57
FIGURA 14: EJEMPLO DE TRASLACIÓN 3D	58
FIGURA 15: REGLA DE LA MANO DERECHA PARA OBTENER LA DIRECCIÓN DE UN GIRO POSITIVO EN 3D	59
FIGURA 16: EJEMPLO DE ROTACIÓN 3D SOBRE EL EJE X_3	60
FIGURA 17: PERMUTACIONES CÍCLICAS DE LOS EJES COORDENADOS	61
FIGURA 18: FUERZAS DE INTERACCIÓN ENTRE PARTÍCULAS	68
FIGURA 19: ESTRUCTURA TRIDIMENSIONAL DE UNA PROTEÍNA	70
FIGURA 20: ALGORITMO GENERAL DE EVALUACIÓN Y MINIMIZACIÓN DE ENERGÍA	71
FIGURA 21: ESTRUCTURA PRIMARIA DE LA PROTEÍNA	72

FIGURA 22: REPRESENTACIÓN DE TIPOS DE AMINOÁCIDOS.....	73
FIGURA 23: ESTRUCTURA QUÍMICA DE LOS AMINOÁCIDOS	73
FIGURA 24: CAMPO DE FUERZAS POR CADA ÁTOMO DE LA MOLÉCULA	74
FIGURA 25: LONGITUDES Y ENERGÍAS DE ENLACE	76
FIGURA 26: ALGORITMO DE MINIMIZACIÓN DE ENERGÍA	77
FIGURA 27: MOLÉCULA EN EQUILIBRIO.....	79

ÍNDICE DE TABLAS

TABLA 1: APLICACIONES DE DINÁMICA MOLECULAR USANDO GPUS	22
TABLA 2 AMINOÁCIDOS ESENCIALES.....	38
TABLA 3: ENCABEZADOS DE UN ARCHIVO (.PDB).....	45

ÍNDICE DE ECUACIONES

ECUACIÓN 1: EXPRESIÓN MATRICIAL PARA EL ESCALAMIENTO 3D	56
ECUACIÓN 2: EXPRESIÓN MATRICIAL PARA LA TRASLACIÓN 3D	58
ECUACIÓN 3: FÓRMULAS PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_3	60
ECUACIÓN 4: EXPRESIÓN MATRICIAL PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_3	60
ECUACIÓN 5: FÓRMULAS PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_1	61
ECUACIÓN 6: EXPRESIÓN MATRICIAL PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_1	62
ECUACIÓN 7: FÓRMULAS PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_2	62
ECUACIÓN 8: EXPRESIÓN MATRICIAL PARA LA ROTACIÓN 3D ALREDEDOR DEL EJE X_2	62
ECUACIÓN 9: CAMPO DE FUERZAS.....	68
ECUACIÓN 10: MODELO MATEMÁTICO DE LA ENERGÍA POTENCIAL	69

Capítulo 1

Introducción

1.1 Planteamiento del problema

Dentro del ámbito científico existe una gran variedad de problemas que pueden ser tratados por algoritmos en forma vectorial o matricial. Al poder adoptar esta forma, la ejecución de este tipo de algoritmos en un ordenador es paralelizable en la mayoría de los casos de forma relativamente sencilla.

La historia de la paralelización es muy larga comenzando en los años 70 con la aparición de sistemas multiprocesador y los procesadores vectoriales. En aquellos momentos, la posibilidad de emplear el cálculo paralelo se podía desarrollar en pocos lugares sólo aquellos donde se disponía de los recursos. Con el rápido desarrollo tecnológico que se ha tenido en el área de cómputo, ahora se tienen más alternativas a un costo mucho menor, lo que nos permite hacer uso de forma más general de la paralelización, incluso en entornos domésticos.

Actualmente, la tecnología empleada para la fabricación de tarjetas gráficas o GPUs (Graphics Processing Unit) han evolucionado de una manera muy significativa, de tal forma que dejaron de ser procesadores exclusivos para el procesamiento de gráficos y se convirtieron sofisticados co-procesadores de bajo precio y alto rendimiento, que permitieron aumentar la capacidad de una computadora personal de escritorio o portátil con las mejoras en su arquitectura y un modelo flexible de programación para el manejo masivo de datos en paralelo [1-3], lo que los convierte en una atractiva alternativa de cómputo de alto rendimiento (High Performance Computing en inglés HPC).

La idea de usar esa potencia para aplicaciones diferentes al manejo de gráficos se le denomina GPGPU (General Purpose computation on Graphics Processing Units) o GPU Computing. En el momento en que las tarjetas gráficas permiten

que se programen funciones sobre su hardware se empieza a hacer uso de GPGPU.

Razón por la cual la comunidad de Dinámica Molecular fue una de las primeras que adoptó el desarrollo de aplicaciones sobre GPUs [4-9]. Este tipo de aplicaciones son particularmente aptas para la arquitectura de procesamiento paralelo masivo de las GPUs NVIDIA, por la cantidad de datos a procesar, con los que se pueden realizar simulaciones complejas que antes sólo podían realizarse utilizando recursos de supercomputación, actualmente pueden ejecutarse en una estación de trabajo personal, debido a que con esta tecnología se puede convertir esta en un "laboratorio computacional", capaz de ejecutar códigos complejos.

Para facilitar el empleo de las tarjetas gráficas para aplicaciones de propósito general, NVIDIA desarrolló toda una tecnología en donde la nueva arquitectura de hardware y software de las GPU es una arquitectura unificada llamada (CUDA acrónimo en inglés de The Compute Unified Device Architecture) [10], la cual se apoya del uso del lenguaje C, para el procesamiento en paralelo y OpenGL o CG para el manejo de gráficos, lo cual hace más accesible programación y que permitía utilizarla para cualquier tarea.

Como por ejemplo tenemos las simulaciones de dinámica molecular computacionales las cuales son de gran utilidad para simular y comprender aspectos de los experimentos reales [11] y son de gran ayuda cuando la investigación de laboratorio sea inapropiada, impracticable o imposible, ya que el comportamiento del fenómeno se puede modelar mediante el uso de las matemáticas, física y química teórica y son de gran ayuda especialmente cuando el estudio de forma experimental se tiene que realizar varias veces o el proceso tarda grandes cantidades de tiempo por lo que a fin de que estas simulaciones sean más útiles, es necesario desarrollar algoritmos paralelos eficientes, de tal forma que se distribuya el procesamiento de los datos entre los procesadores de la GPU, con la finalidad de reducir el tiempo computacional.

De esta forma se ha tenido una notable contribución en el campo de la investigación experimental, tanto para la interpretación de resultados obtenidos como para la planificación de trabajos futuros así como para la producción de información no asequible experimentalmente, debido a que las moléculas estudiadas de forma experimental han sido calculadas a niveles confiables de teoría.

Por tal motivo el propósito de este trabajo es mostrar la importancia de hacer uso de las capacidades de la GPU de NVIDIA en equipos de cómputo personales, al implementar un proceso de dinámica molecular.

En específico se realizará la evaluación y minimización de energía potencial para proteínas, ya que es un proceso fundamental para el análisis de sus estructuras que nos lleva a la creación de nuevos fármacos, descubrimiento de enfermedades o bien la secuencia del ADN, este proceso se implementará haciendo uso de un modelo matemático el cual nos permitirá caracterizar el comportamiento de la energía potencial de la molécula y el cual posteriormente será evaluado para ver si este comportamiento esta dentro de los parámetros deseados, es decir, que los parámetros de la molécula se encuentren dentro de sus parámetros ideales o naturales, caso de no ser así se procederá con la minimización que es la modificación de la geometría de la molécula tantas veces como sea necesaria para encontrar los valores deseados. La simulación se implementará en la arquitectura CUDA de las GPU de NVIDIA y los resultados que se esperan es analizar la aceleración del proceso utilizando esta tecnología.

1.2 Objetivo General

Acelerar el procesamiento de una aplicación que simule la estabilidad de energía potencial de las proteínas ya que es un proceso fundamental para cualquier programa de dinámica molecular, debido a que de este depende el análisis correcto de sus estructuras y el cual por sus características puede ser aprovechado para implementarlo con la tecnología CUDA de las GPU de NVIDIA.

1.3 Objetivos Específicos

- Análisis y diseño del problema planteado para hacer uso de la arquitectura CUDA de las GPUs.
- Hacer uso de un modelo matemático para diseñar un algoritmo paralelo que simule el comportamiento de la energía potencial.
- Realizar la minimización de energía por medio de operadores lineales
- Análisis de resultados
- Mostrar los resultados obtenidos de forma numérica y mostrar la interoperabilidad que se tiene con la API de OpenGL, además de que aprovecha la aceleración de la GPU.

1.4 Justificación

El desarrollo y la implementación del campo de fuerzas es uno de los mayores problemas en la expansión del espectro de aplicación a ciertas áreas como la química médica, la química organometálica, la espectroscopia vibracional y el cálculo de estructuras cristalinas debido a las siguientes razones:

La falta de algoritmos que resuelvan satisfactoriamente alguna de las funciones de energía potencial.

El hecho de que no todos los parámetros y constantes de fuerza son realmente transferibles de un programa a otro.

La insuficiencia de parámetros. Las moléculas que contienen grupos funcionales o elementos que no están parametrizados en los campos de fuerza existentes requieren la estimación de nuevos parámetros específicos para cada tipo de enlace, ángulo de enlace o ángulo de torsión.

Esto puede ser resuelto con una herramienta hecha a la medida como la que se propone de bajo costo y con buenos resultados, con la que se puede obtener el análisis de estructuras moleculares, mediante cálculos teóricos como los de mecánica molecular ya que es fundamental cuando la aplicación de los métodos experimentales NO proporcionan la información deseada.

1.5 Delimitación y limitaciones de la investigación

Dado que el objetivo principal de la tesis es mostrar que el uso de las GPU acelera los procesos que requieren procesar una gran cantidad de datos, el trabajo se centra exclusivamente en el proceso de evaluación y minimización de energía potencial de las proteínas.

El uso de métodos de mecánica molecular se utiliza, para realizar cálculos sólo de moléculas estables.

En caso de requerirlo es conveniente modificar en cuanto a la cantidad de términos utilizados el modelo matemático.

Capítulo 2

Estado del arte

2.1 Antecedentes

Actualmente existen una variedad de técnicas utilizadas para acelerar las simulaciones de Dinámica Molecular en arquitecturas que nos permiten el cómputo en paralelo. Esto va desde el uso de los típicos equipos de HPC ó el uso de nuevas arquitecturas de procesamiento.

En este capítulo se discuten algunas estrategias de acuerdo a la arquitectura utilizada en la aceleración de procesos de dinámica molecular y las aplicaciones actualmente realizan ya simulaciones en esta área mediante GPUs.

En las arquitecturas de grano grueso se incluyen las supercomputadoras de propósito general, los clúster de PCs y las grids. El uso de supercomputadoras para simulaciones de DM puede proporcionar un rendimiento tremendo a costa de ser excesivamente caros e inaccesibles para la mayoría de los investigadores. Por lo que el uso de clúster o grids [12], se han utilizado para proporcionar un alto rendimiento a un menor costo, sin embargo estos arreglos de PCs sufren problemas de escalabilidad para un creciente número de procesadores, debido a las altas latencias de comunicación entre ordenadores.

Los proyectos basados en Grids como Folding@Home5 y Predictor@Home han atraído a miles de usuarios voluntarios en el mundo para cooperar de forma voluntaria con el proyecto, como equipos clientes, pero desafortunadamente la comunicación sólo se puede hacer entre el servidor y el cliente, por lo que no existe comunicación alguna entre todas las máquinas clientes y por tanto este método sólo es adecuado para simulaciones MD con gran número de trayectorias separadas con escalas de tiempos cortos.

Las arquitecturas de grano fino, son arquitecturas de propósito especial, arquitecturas reconfigurables y GPUs. Para usos especiales como las arquitecturas de Anton[13], FASTRUN[14], MDGRAPE[15] y el motor de MD[16], pueden proporcionar el medio más rápido de ejecutar un algoritmo de MD en particular con una densidad aritmética muy alta. Sin embargo tales arquitecturas se limitan a un solo algoritmo y por tanto no puede suministrar la flexibilidad para ejecutar una variedad de algoritmos necesarios para distintas simulaciones. Estos pueden ser vistos como aceleradores que satisfacen la demanda de bajo costo.

La ventaja principal de las GPU en comparación con las arquitecturas mencionadas es que son componentes básicos. En particular, la mayoría de los usuarios ya tienen acceso a ordenadores con tarjetas gráficas modernas. Para estos usuarios esta dirección proporciona una solución de costo cero. Incluso en caso de tener que comprar la tarjeta gráfica, la instalación de esta es muy fácil (plug and play), en cuanto a la programación se tiene un modelo de alto nivel llamado CUDA, el cual ofrece un entorno de programación simplificado basado en el lenguaje C.

Con el lanzamiento de CUDA y OpenGL, la programación de las GPU se ha convertido en mucho más accesible y los científicos computacionales no necesitan tener una amplia experiencia en gráficos por computadora para aprovechar las capacidades de cómputo de las GPU.

En los últimos años se ha tenido un enorme progreso en el desarrollo de implementaciones con GPUs, de algoritmos fundamentales haciendo uso de bibliotecas, que han servido para el desarrollo de las aplicaciones más exigentes.

Una de las aplicaciones más convenientes para el cómputo con GPUs ha sido la aceleración de las simulaciones de dinámica molecular basados en la mecánica clásica. A pesar de años de investigación en algoritmos eficientes y técnicas de procesamiento en paralelo, las demandas en investigación biomédica requiere cada vez simulaciones más complejas, de gran tamaño y demandas de tiempo más grandes, con una mejor toma de muestras y con mejora en los campos de fuerza, cada una de estas dimensiones genera una demanda de más cómputo.

Antes de la introducción de CUDA en 2007, ninguna de las simulaciones de dinámica molecular había sido diseñada para aprovechar la aceleración de las GPUs, pero actualmente se encuentra una cantidad considerable de aplicaciones que están haciendo uso de estas unidades gráficas.

En la TABLA 1 se detalla la aplicación y el proceso en el cual se hace uso de las GPU.

APLICACIÓN	FUNCIONES Y CARACTERÍSTICAS
NAMD	<ul style="list-style-type: none"> ○ Fue el primer paquete en incorporar la aceleración de GPU. ○ Cálculo de fuerzas intermoleculares sin enlace. ○ Usa el programa de gráficos moleculares VMD, para configurar la simulación y el análisis de la trayectoria y es compatible con AMBER, CHARMM y X-PLOR. ○ Diseño de simulaciones para grandes sistemas moleculares ○ Se distribuye de forma gratuita
HOOMD-Blue	<ul style="list-style-type: none"> ○ Simulaciones en una sola estación de trabajo ○ Maneja varios tipos de potenciales ○ Escrito en python ○ Distribución gratuita
ACEMD	<ul style="list-style-type: none"> ○ Compatible con NAMD ○ Estructura molecular creada con VMD o Amber
AMBER	<ul style="list-style-type: none"> ○ Simulación de los campos de fuerza mecánicos para solvente implícito y explícito
GROMACS	<ul style="list-style-type: none"> ○ Simulación de moléculas bioquímicas con interacciones complejas en los enlaces
VMD	<ul style="list-style-type: none"> ○ Visualización y análisis de grandes sistemas moleculares con gráficos en 3D

Tabla 1: Aplicaciones de dinámica molecular usando GPUs

En todas la aplicaciones que se encuentran en la Tabla 1

- Sólo algunos procesos están disponibles en las versiones de CUDA
- Se recomienda probar antes de mandarlos a producción, ya que difieren entre las versiones de la GPU y CPU
- Requieren GPU con arquitectura "Fermi", como las Tesla que son muy costosas

Capítulo 3

Fundamentos teóricos

A lo largo de la historia el hombre ha desarrollado, mecanismos que le ayuden a realizar tareas de distinta índole de una forma más rápida. El cómputo de alto rendimiento proporciona mayor capacidad de cómputo que la que se puede obtener de computadoras individuales, dentro de este tipo de cómputo encontramos a los sistemas masivamente paralelos y clústeres.

Un sistema paralelo es una máquina que tiene más de un procesador y por lo tanto tiene la capacidad de ejecutar más de un programa al mismo tiempo como las GPU de NVIDIA.

3.1 Unidades Gráficas de Procesamiento (GPUs)

La unidad de procesamiento gráfico o GPU (*Graphics Processing Unit*) es un coprocesador dedicado al procesamiento de gráficos y operaciones de coma flotante, que ayuda a aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y aplicaciones 3D interactivas.

Las GPUs actuales están compuestas por cientos de núcleos de procesamiento en paralelo con un sistema de memoria distribuido bastante complejo, por lo que pueden procesar un alto número de tareas en paralelo y su rendimiento es superior a un sistema con un procesador general multinúcleo, lo cual nos da una solución de alto rendimiento no sólo para el procesamiento gráfico para el que fueron diseñadas, si no para aplicaciones de tipo GPGPU.

3.1.1 Historia de la GPU como procesador de Cálculo

Las primeras GPU fueron diseñadas como aceleradoras de gráficos y admitían apenas procesos específicos de funcionamiento fijo. Desde finales de los

noventas, el hardware cada vez se volvió más programable, lo que culminó con la primera GPU de NVIDIA en 1999. Menos de un año después de que NVIDIA acuñara el término GPU, otros profesionales se aliaron a los artistas y desarrolladores de juegos para hacer trabajos revolucionarios con la tecnología, los investigadores empezaron a aprovechar su excelente rendimiento de punto flotante. Surgía de esta forma el movimiento de la GPU para fines generales (GPGPU).

Pero en ese entonces, la GPGPU estaba lejos de ser fácil, incluso para quienes conocían los lenguajes de programación de gráficos como OpenGL. Los desarrolladores tenían que mapear cálculos científicos en problemas que podían representarse por triángulos y polígonos. La GPGPU era prácticamente inalcanzable para los que no hubieran memorizado las últimas API gráficas, hasta que un grupo de investigadores en la Universidad de Stanford se propuso reimaginar la GPU como un "coprocesador de flujos."

En 2003, un equipo de investigadores dirigidos por Ian Buck lanzó "Brook", el primer modelo de programación ampliamente adoptado para extender C con datos paralelos. Usando conceptos como los flujos, los *kernels* y los operadores de reducción, el compilador Brook y el sistema de tiempo de ejecución expusieron la GPU como un procesador con fines generales en un lenguaje de alto nivel, pero lo más importante es que los programas de Brook no sólo eran más fáciles de escribir que el código de la GPU ajustado a mano, sino que eran siete veces más rápidos que el código similar existente.

NVIDIA sabía que un hardware impresionantemente rápido tenía que combinarse con herramientas de hardware y software intuitivas e invitó a Ian Buck a unirse a la compañía y a empezar a hacer evolucionar una solución que ejecutara C a la perfección en la GPU. Al reunir el software y el hardware, NVIDIA lanzó CUDA en 2006, la primera solución del mundo para computación general en las GPU

En NVIDIA se dieron cuenta de las ventajas que supondría poner todo este rendimiento al alcance de la comunidad científica y decidieron invertir en modificar la GPU a fin de hacerla totalmente programable para aplicaciones científicas y añadir soporte para lenguajes de alto nivel como C, C++ y Fortran. El resultado fue la plataforma de cálculo paralelo CUDA para la GPU. [17]

3.1.2 Cómputo de Propósito General sobre Unidades Gráficas de Procesamiento (GPGPU)

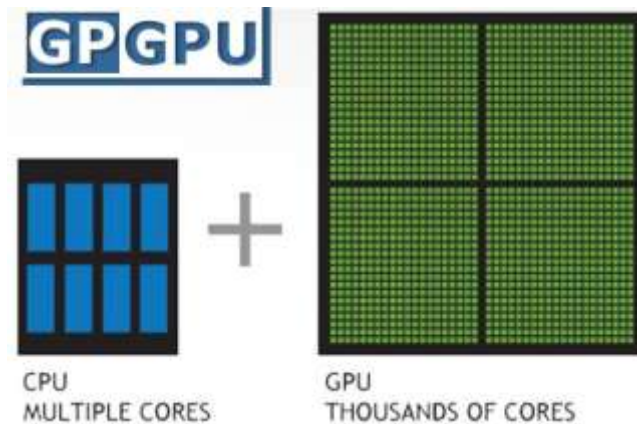


Figura 1: Co-procesamiento entre CPU y GPU

La computación de la GPU es el uso de una GPU junto con una CPU para acelerar las aplicaciones científicas y de ingeniería con fines generales. Resultado de los trabajos pioneros de NVIDIA, la computación de la GPU se ha convertido rápidamente en un estándar del sector, ha sido aprovechada por millones de usuarios en todo el mundo y la han adoptado prácticamente todos los proveedores de computación.

La computación de la GPU ofrece un rendimiento sin precedentes en las aplicaciones al descargar a la GPU las partes de las aplicaciones con uso intensivo de computación, mientras que el resto del código se sigue ejecutando en la CPU. Desde la perspectiva del usuario, las aplicaciones simplemente se ejecutan mucho más rápido.

La combinación de la CPU + la GPU resulta muy poderosa debido a que las CPU consisten en algunos núcleos optimizados para el procesamiento en serie, mientras que las GPU se refieren a miles de núcleos más pequeños y más eficientes diseñados para el rendimiento paralelo. Las partes en serie del código se ejecutan en la CPU mientras que las partes paralelas se ejecutan en la GPU.

Todas las GPU NVIDIA (GeForce®, Quadro® y Tesla®) admiten la computación de la GPU y el modelo de programación paralela CUDA®. Los desarrolladores tienen acceso a las GPU NVIDIA en prácticamente cualquier plataforma de su elección, incluyendo la más reciente Apple MacBook Pro.

3.1.3 CUDA (Compute Unified Device Architecture)

El desarrollo de aplicaciones en paralelo ya no está restringido sólo a aplicaciones que se realizan en grandes y costosos equipos de cómputo. Actualmente todos los modelos de computadoras personales cuentan con al menos un procesador con dos núcleos y en algunos casos poseen tarjetas gráficas con poder computacional suficiente para llevar a cabo operaciones de propósito general.

La arquitectura CUDA surgió en 2006, cuando NVIDIA lanzó sus tarjetas GeForce 8800 GTX las que por primera vez incluyeron elementos dirigidos específicamente a posibilitar la solución de problemas de propósito general. Poco después, NVIDIA lanzó el compilador CUDA C, el primer lenguaje de programación para desarrollar aplicaciones de propósito general sobre la GPU; con la finalidad de captar la mayor cantidad de programadores posibles que adoptasen esta arquitectura, CUDA C es una extensión al familiar ambiente de C/C++, al cual se le agregaron más instrucciones.

Este conjunto de instrucciones permite que los programadores desarrollen aplicaciones que comunican a la CPU con la GPU para utilizar de esta última sus múltiples multiprocesadores, sobre los cuales pueden ser ejecutadas simultáneamente varias tareas.

CUDA Permite aumentos impresionantes en el rendimiento de la computación al aprovechar la potencia de la Unidad de Procesamiento de Gráficos (GPU).

3.1.4 Arquitectura CUDA

Las diferencias entre la capacidad de la CPU y la GPU, es que la GPU está diseñada para realizar computación-intensiva, altamente paralela y por ello está dotada de mayor cantidad de transistores que se dedican al procesamiento de datos en las unidades aritmético-lógicas (ALU) en lugar de almacenar datos en cache o controlar el flujo de información como se muestra en la Figura 2.

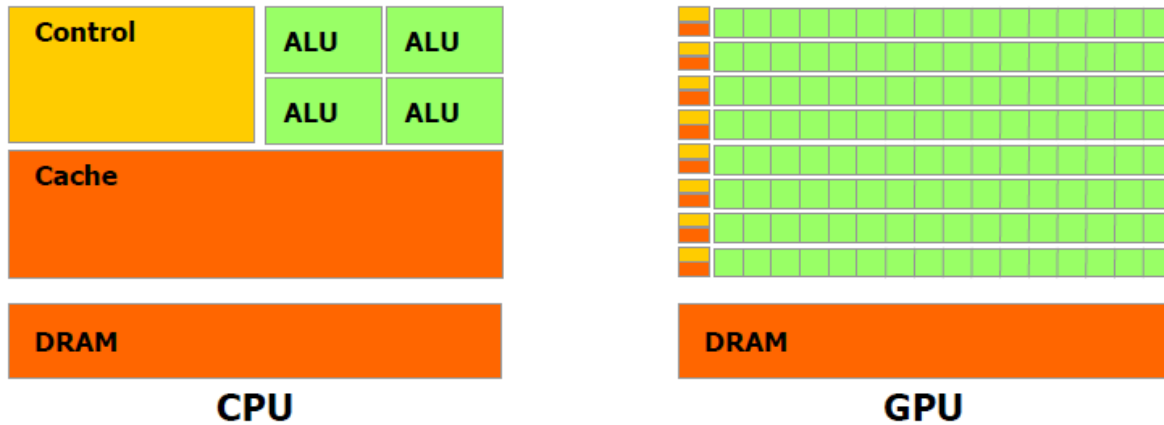


Figura 2: Comparación entre CPU y GPU

Esto quiere decir que las GPU están especialmente diseñadas para llevar a cabo gran cantidad de operaciones en paralelo. Puesto que cada elemento de procesamiento posee sus propias localidades de memoria, no se requiere habilitar un control de flujo sofisticado.

En la arquitectura CUDA, se manejan los siguientes elementos:

- **Threads, hilos:** El hilo es la unidad básica de ejecución, los hilos pueden procesarse simultáneamente lo que es equivalente a tener una iteración del bucle en el caso serializado, estos cuentan con memoria local para almacenar sus datos.
- **Blocks o bloques:** Un conjunto de hilos que se ejecutan en un multiprocesador. Cada bloque cuenta con una zona de memoria, denominada memoria compartida accesible para todos sus hilos.
- **Grid o malla:** Un conjunto de bloques. Cada malla se ejecuta sobre un GPU distinto y cuenta con memoria accesible para todos los bloques que la componen.

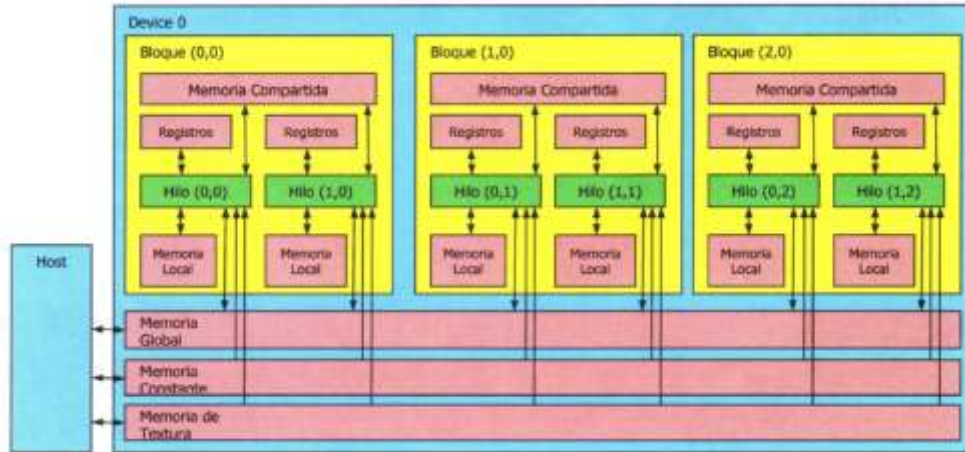


Figura 3: Arquitectura CUDA

3.1.5 Modelo de programación

CUDA se diseñó de tal manera que tiene una curva de aprendizaje baja para los programadores de C, que sólo deberán familiarizarse con los siguientes aspectos: La forma en que cooperan los hilos, la jerarquía de la memoria, el uso de la memoria compartida y las instrucciones necesarias para sincronizar las tareas que se realizan simultáneamente.

Esas pequeñas adiciones al estándar de C permiten paralelizar el problema ya sea con una **paralelización de grano fino** orientada a los datos; ó una **paralelización de grano grueso** orientada a las tareas.

El reto para el programador es decidir de que manera atacar el problema para dividirlo en sub-problemas que se puedan agrupar en hilos y bloques que lo resuelvan cooperativamente y en paralelo.

Para ello hay que considerar que el primer rasgo característico de CUDA es la existencia de dos entornos de ejecución diferenciados:

- **Device:** corresponde con el lado de la GPU y es donde se ejecutarán los *kernels*.
- **Host:** corresponde con el lado de la CPU y es donde se llamarán a dichos *kernels*

Por otro lado, dado que pretendemos sustituir un bucle de muchas iteraciones por un número equivalente de ejecuciones paralelas, se necesita algún tipo de

control en cuanto a la relación entre ejecuciones. Esto lo obtenemos gracias a los denominados *Grid*, *Block* y *Thread*.

La ejecución de un *kernel* se lleva a cabo, por un conjunto numerado de *threads*, ordenados dentro de *blocks*, que a su vez estarán contenidos dentro del *Grid*. De esta manera, a través de la posición del *thread* dentro de su *block* y este dentro del *Grid*, hará posible la identificación del hilo de forma única.

El modelo de programación CUDA asume que los hilos CUDA se ejecutan en una unidad física distinta que actúa como coprocesador (*device*) al procesador (*host*) donde se ejecuta el programa como se aprecia en la Figura 4. CUDA C es una extensión del lenguaje de programación C, que permite al programador definir funciones C, llamadas *kernels*, que al ser llamadas se ejecutan en paralelo por N hilos diferentes. Los *kernels* se ejecutan de forma secuencial en el *device*.

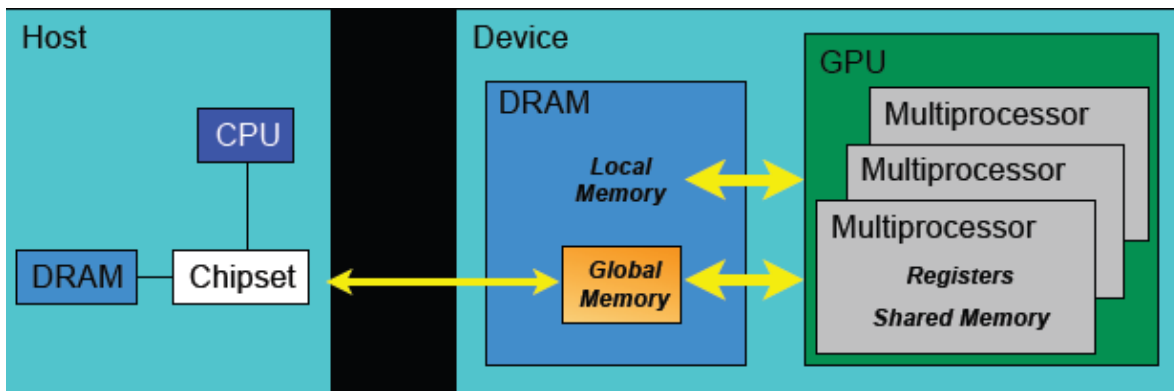


Figura 4: Arquitectura CPU-GPU

Existe una jerarquía perfectamente definida sobre los hilos de CUDA. Los hilos se agrupan en vectores a los que se les llama bloques, estos vectores pueden ser de una, dos o tres dimensiones, de forma que definen bloques de hilos de una, dos o tres dimensiones.

Hilos del mismo bloque pueden cooperar entre sí, compartiendo datos y sincronizando sus ejecuciones. Sin embargo, hilos de distintos bloques no pueden cooperar.

Los bloques a su vez, se organizan en un *grid* de bloques. Este *grid*, de nuevo puede ser de una o dos dimensiones. Los valores entre <<< ... >>> se

conocen como la configuración del *kernel*, y definen la dimensión del *grid* y el número de hilos de cada bloque. La Figura 5 muestra como se organizan los hilos en un *grid* de 2x3 bloques y 3x4 hilos cada uno.

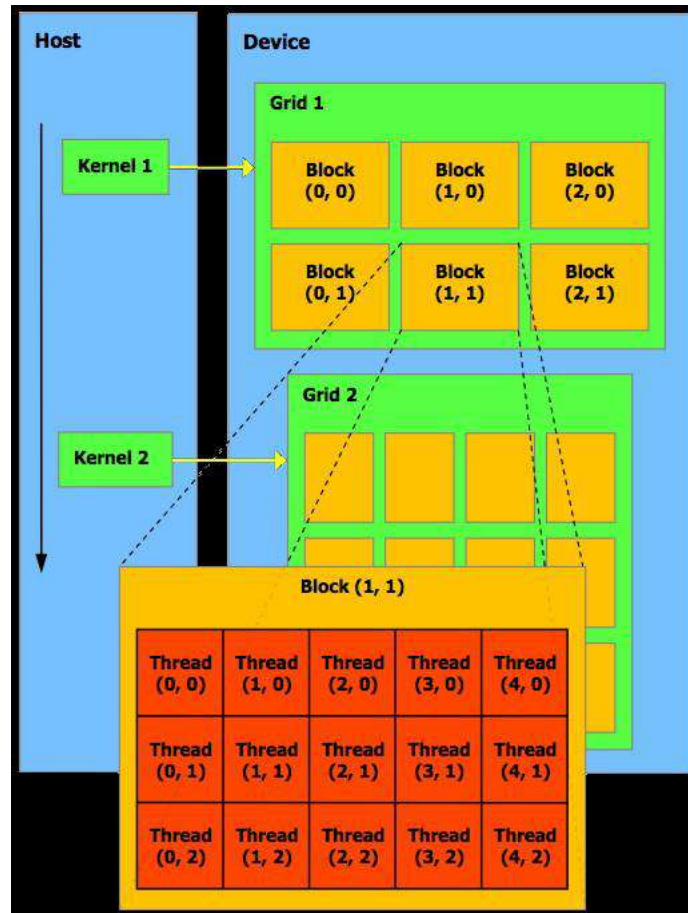


Figura 5: Jerarquía de hilos

Para la programación de aplicaciones con CUDA, podemos aprovechar el rendimiento de las GPUs por medio del Kit de herramientas de CUDA, que brinda un entorno de desarrollo integral para los desarrolladores de C y C++, ya que incluye un compilador, bibliotecas matemáticas y herramientas para corregir y optimizar el rendimiento de las aplicaciones, además de que NVIDIA ofrece, códigos de ejemplos, guías de programación, manuales de usuario y todo tipo de documentación necesaria para el desarrollo de las aplicaciones.

3.1.5.1 Ejecución de funciones en el device y el host

Al haber dos entornos de trabajo diferenciados, tendremos un duplicado de la memoria de la CPU en la GPU. Sin embargo todo el código de la programación de CUDA puede ir en el mismo fichero, por lo que será necesario utilizar distintos cualificadores de acceso para clasificar el espacio de trabajo de las funciones y las variables.

Son de especial relevancia los atributos de funciones:

- **__device__**: Función que se llama y se ejecuta en el *device*
- **__host__**: Función que se llama y se ejecuta desde el *host* (CPU) y es equivalente a no usar el atributo.
- **__global__**: Función que se llama desde el host pero se ejecuta en el *device* su tipo de retorno debe de ser de tipo *void*. Todos los *kernels* utilizan este atributo.

Los parámetros y el cuerpo de las funciones respetan las reglas del C estándar.

Una función con cualificador **__global__** se invoca como sigue

```
nombre_kernel __global__ <<<dim_grid, dim_block>>(*arg*);
```

Donde *nombre_kernel* es el nombre con que habremos declarado la función, *dim_grid* y *dim_block* son las dimensiones del *Grid* y del *Block*, pasadas como variables del tipo *dim3* y entre paréntesis los argumentos propiamente dichos de la función.

Una vez que hemos declarado el *kernel*, con toda seguridad necesitaremos saber de alguna forma con que *thread* estamos trabajando. Esto lo obtendremos mediante las variables internas de tipo *dim3* con que trabaja el *kernel*.

- **gridDim**: Devuelve las dimensiones del *Grid* en número de bloques.
- **blockDim**: Devuelve las dimensiones del *Block* en *threads*
- **blockIdx**: Identifica el bloque en el que se encuentra el *thread*
- **threadIdx**: Identifica el *thread* en que se encuentra, refiriéndose al bloque.

Mediante estas variables podemos identificar de forma única el *thread* de trabajo, como se aprecia en la Figura 6 y por lo tanto se puede diferenciar a que espacio de memoria de un puntero debe acceder el determinado *thread*.

Como ejemplo el siguiente código muestra como se define un *kernel* y como es llamado desde un programa:

```
// definición de un Kernel  
  
__global__ void VecAdd( f loat _ A, f loat _ B, f loat _ C)  
{  
  int i = threadIdx . x ;  
  C[ i ] = A[ i ] + B[ i ] ;  
}  
  
int main ()  
{  
  ...  
  // Invocación de un Kernel con N threads  
  
  VecAdd<<<1, N>>>(A, B, C) ;  
}
```

Figura 6: Definición e invocación de un kernel

3.1.6 Gestión de memoria y comunicación de datos entre CPU y GPU

Los tipos de memoria que se pueden manejar son los siguientes:

Memoria Global: Es la más abundante de la GPU, su alta latencia pese a ser inferior en la CPU, suele orientar el estilo de programación y es el punto más importante a optimizar ya que es un factor determinante en el tipo de ejecución del código.

Registros: También son un elemento clave, dado que cuantos más registros necesite un *kernel* para su ejecución, menor será el tamaño de los bloques que se podrán ejecutar en el multiprocesador. Porque si no hay muchos registros en un multiprocesador para todos los *threads* de al menos un bloque, el compilador lanzará un mensaje de error.

Memoria compartida: Es local al multiprocesador y su acceso tiene un coste equivalente al de los registros.

Memoria constante: Dispone de una pequeña caché a nivel de multiprocesador desde la que su acceso es muy ágil. En caso de no disponer del dato en caché, el coste de acceso es idéntico al de la memoria global.

Memoria de textura: De la misma forma que la anterior, dispone de una pequeña caché cuyo acceso será muy rápido.

Memoria local: En realidad es un espacio de la memoria global que se dedica en exclusiva a cada *thread* en casos donde el compilador decide que las estructuras utilizadas a nivel de registros son demasiado grandes. Debe evitarse a toda costa, pues su acceso no tiene el coste de los registros sino el de la memoria global.

Para gestionar memoria en el *device* (GPU) se utiliza la función:

```
cudaMalloc((void**)*nombre_variable, cantidad_de_bytes);
```

Esta función se invoca desde la CPU y gestiona la cantidad de bytes indicados en la GPU, los cuales serán referenciados por medio de una variable dinámica específica.

Ejemplo:

```
float *v;
```

```
cudaMalloc((void **)&v, 3 * sizeof(float));
```

Se gestiona memoria para un arreglo de tres números de tipo float, a los que se hará referencia mediante la variable v.

Para liberar la memoria gestionada, se utiliza la función:

```
cudaFree(nombre_variable);
```

El contenido de las variables gestionadas con **cudaMalloc**, no puede modificarse ni visualizarse directamente desde el host(CPU), ni los datos que residen en el CPU son accesibles desde el GPU, por ello se hace necesario copiar los datos desde y hacia la GPU, utilizando la siguiente función.

```
cudaMemcpy(destino, fuente, cant, tipo);
```

Donde:

destino: Es la dirección de memoria donde se almacenarán los datos copiados.

fuentes: Es la dirección de memoria donde residen los datos que van a copiarse.

cant: Es la cantidad de bytes que van a copiarse.

tipo: Indica el tipo de transferencia que se realizará.

El tipo de transferencia puede tomar los siguientes valores:

- **cudaMemcpyHostToDevice:** Los datos se copian desde el host hacia el device.
- **CudaMemcpyDeviceToHost :** Los datos se copian desde el device hacia el host.
- **CudaMemcpyDeviceToDevice :** Los datos se copian desde el device hacia el device.
- **CudaMemcpyHostToHost :** Los datos se copian desde el host hacia el host. En este caso, la función es equivalente a utilizar la función
- **memcpy** del C estándar.

Al usar cualquiera de estos tipos de transferencia, hay que tener especial cuidado en que la ubicación de las variables fuente y destino sea la correcta, de acuerdo al sentido de la copia.

Al desarrollar una aplicación, puede ejecutarse código alternadamente en el host y el *device*, como se muestra en la Figura 7.

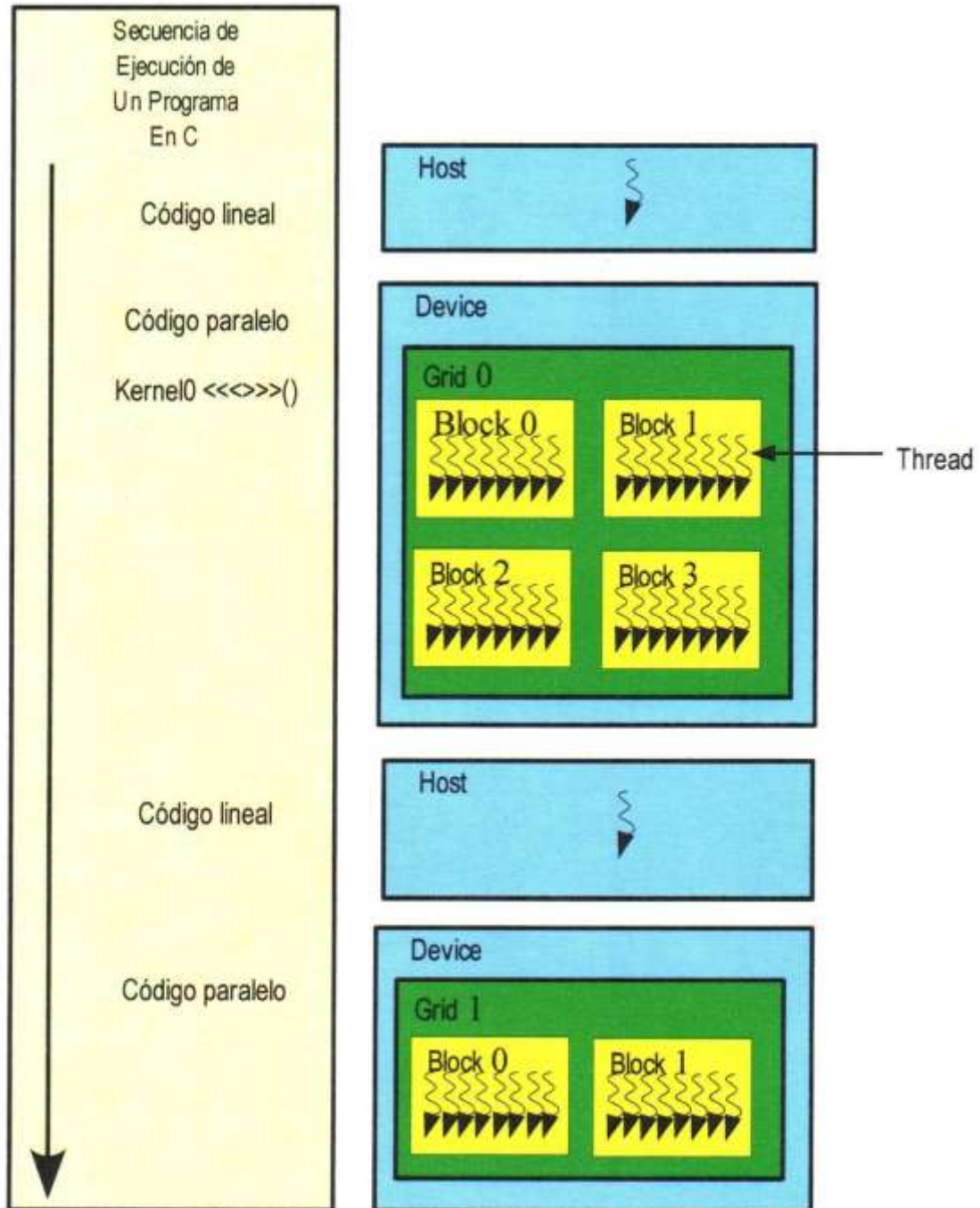


Figura 7: Ejecución alternada, entre el CPU y el GPU

3.1.7 Aplicaciones en entornos de alto rendimiento

Los científicos e investigadores cada vez encuentran usos más amplios para la computación de la GPU con CUDA, en la página www.nvidia.com, podemos encontrar una larga lista de aplicaciones específicas y modificaciones de programas para aprovechar esta tecnología, para diferentes ámbitos de aplicaciones, tales como:

- Automatización de diseño electrónico
- Animación
- Bioinformática, química computacional y dinámica molecular
- Cálculo financiero
- Defensa e Inteligencia
- Dinámica de fluidos
- Diseño automatizado
- Imágenes para medicina
- Mecánica estructural
- Minería de datos
- Procesamiento de imágenes y visión computarizada
- Etc.

3.2 Proteínas

Las proteínas son sustancias de gran importancia biológica que constituyen los seres vivos cumpliendo diversas funciones, estas definen la identidad de cada ser vivo y juegan un papel muy importante en la mayoría de los procesos fisiológicos.

Si se pudiera entender todos y cada uno de los pliegues en el plano tridimensional, se podría tener la lista completa de las partes de un organismo y así poder entender como esas partes componen las células, lo cual tiene muchas implicaciones, ya que el mal funcionamiento de las proteínas es la causa más común de enfermedades endógenas y la acción de patógenos es usualmente mediante esas proteínas.

Químicamente las proteínas son polímeros lineales de aminoácidos compuestos básicamente por elementos químicos como el carbono (C), hidrógeno (H), oxígeno (O), nitrógeno (N), hay algunos elementos que sólo aparecen en algunas proteínas como (fósforo, cobre, zinc, hierro, etc.). Estos elementos se agrupan para formar unidades estructurales llamados **aminoácidos**.

3.2.1 Aminoácidos

Los aminoácidos son las unidades básicas que forman las proteínas, se pueden considerar como los "ladrillos de los edificios moleculares proteicos" y se caracterizan por poseer un grupo carboxilo (-COOH) y un grupo amino (-NH₂), estas son las unidades que constituyen las proteínas.

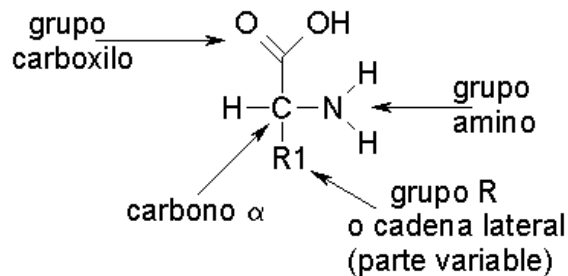


Figura 8 Composición de aminoácidos

3.2.2 Los aminoácidos esenciales

Los aminoácidos esenciales son aquellos que el cuerpo humano no puede generar por sí sólo. Esto implica que la única fuente de estos aminoácidos en los organismos es la ingesta directa a través de la dieta.

Existen 20 aminoácidos esenciales que dan lugar a las proteínas y forman un conjunto de moléculas con características más restringidas. En la Tabla 2 se muestra el nombre, la abreviatura y el símbolo para cada uno de los aminoácidos esenciales.

Alanina (Ala, A)	Cisteína (Cys,C)	Aspártico (Asp,D)	Glutámico (Glu,E)
Fenilalanina (Phe,F)	Glicina (Gly,G)	Histidina (His,H)	Isoleucina (Ile,I)
Lisina (Lys,k)	Leucina (Leu,L)	Metionina (Met,M)	Asparagina (Asn,N)
Prolina (Pro,P)	Glutamina (Gln,Q)	Arginina (Arg,R)	Serina (Ser,S)
Treonina (Thr,T)	Valina (Val,V)	Triptofano (Trp,W)	Tirosina (Tyr,Y)

Tabla 2 Aminoácidos esenciales

3.2.3 Enlace peptídico

Los aminoácidos se encuentran unidos linealmente por medio de uniones peptídicas. Estas uniones se forman por la reacción de síntesis entre el grupo carboxilo del primer aminoácido con el grupo amino del segundo aminoácido.

La formación del enlace peptídico entre dos aminoácidos es un ejemplo de la reacción de condensación. Dos moléculas se unen mediante un enlace de tipo covalente CO-NH con la pérdida de una molécula de agua y el producto de esta unión es un dipéptido. El grupo carboxilo libre del dipéptido reacciona de modo similar con el grupo amino de un tercer aminoácido y así sucesivamente hasta formar una larga cadena. Podemos seguir añadiendo aminoácidos al péptido, porque siempre hay un extremo NH₂ terminal y un COOH terminal, como se puede apreciar en la Figura 9.

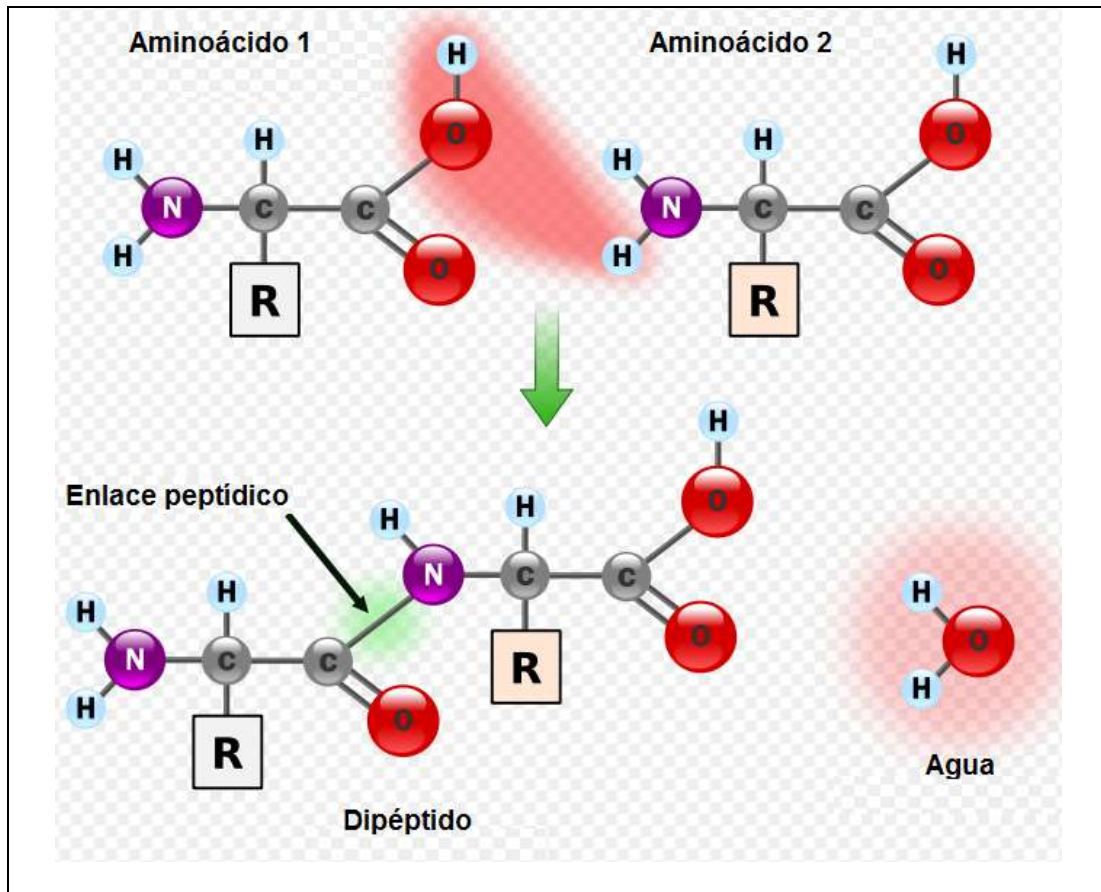


Figura 9: Enlace peptídico

Por otra parte, el carácter parcial de doble enlace peptídico ($-C-N-$) determina la disposición espacial de éste en un mismo plano, con distancias y ángulos fijos. Como consecuencia, el enlace peptídico presenta cierta rigidez e inmoviliza el plano a los átomos que lo forman.

3.2.4 Estructura de las proteínas

Las proteínas no son polímeros al azar de longitud indefinida, si no que cada una de ellas tienen una determinada composición en aminoácidos y estos están ordenados en una determinada secuencia y su estructura se encuentra plegada de un modo característico, que es igual para todas las moléculas de una misma proteína y que recibe el nombre de **estructura** o **conformación tridimensional nativa** de la proteína.

La secuencia lineal de aminoácidos puede adoptar múltiples conformaciones en el espacio que se forma mediante el plegamiento del polímero lineal. Tal plegamiento se desarrolla en parte espontáneamente por la repulsión de los aminoácidos hidrófobos por el agua, la atracción de aminoácidos cargados y la formación de puentes de disulfuro y también en parte es ayudado por otras proteínas.

A continuación se presenta una breve descripción de los cuatro niveles de estructuración en las proteínas:

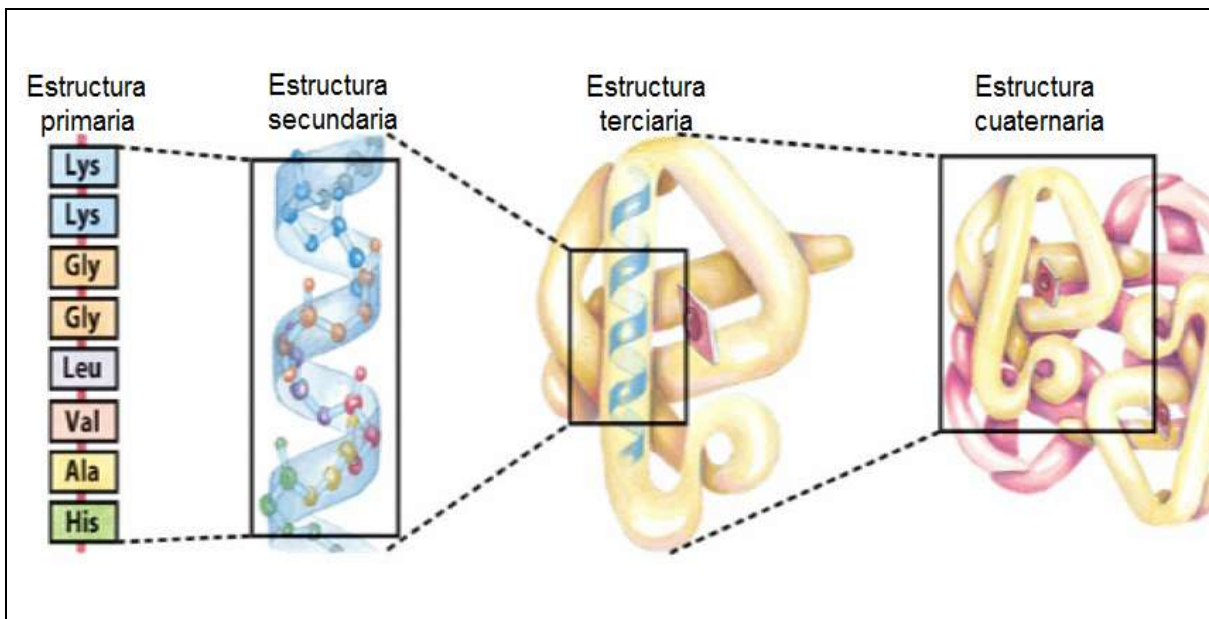


Figura 10: Estructuras de las proteínas

- **Estructura primaria:** viene determinada por la secuencia de aminoácidos en la cadena proteica, es decir, el número de aminoácidos presentes y el orden en que están enlazados, las posibilidades de estructuración a nivel primario son prácticamente ilimitadas. Como en casi todas las proteínas existen 20 aminoácidos diferentes. Conocer la estructura primaria de una proteína no sólo es importante para entender su función (ya que esta depende de la secuencia de aminoácidos y de la forma que adopte), si no que también en el estudio de las enfermedades. Es posible que el origen de una enfermedad genética radique en una secuencia anormal. Generalmente el número de aminoácidos que forma una proteína oscila entre 80 y 300.

- **Estructura secundaria:** La estructura secundaria de las proteínas es el plegamiento que la cadena polipeptídica adopta gracias a la formación de puentes de hidrógeno entre los átomos que forman el enlace peptídico. Los puentes de hidrógeno se establecen entre los grupos -CO- y -NH- del enlace peptídico (el primero como aceptor de H y el segundo como donador de H). De esta forma la cadena polipeptídica es capaz de adoptar conformaciones de menor energía libre y por tanto más estables.
- **Estructura terciaria:** Se le llama estructura terciaria a la disposición tridimensional de todos los átomos que componen la proteína. Esta estructura es la responsable directa de sus propiedades biológicas, ya que la disposición espacial de los distintos grupos funcionales determina su interacción con los diversos ligandos. Para las proteínas que constan de una sola cadena polipeptídica, estas carecen de estructura cuaternaria y la terciaria es la máxima información estructural que se puede obtener.
- **Estructura cuaternaria:** Cuando una proteína consta de más de una cadena polipeptídica, decimos que tiene estructura cuaternaria. Esta estructura deriva de la conjunción de varias cadenas peptídicas que asociadas conforman un multímero, que posee propiedades distintas a las de sus monómeros componentes.

3.2.5 Propiedades de las proteínas

Las proteínas cuentan con las siguientes propiedades:

- **Especificidad:** Se refiere a que cada proteína lleva a cabo una determinada función y esta es dada porque posee una determinada estructura primaria y una conformación espacial propia; por lo que un cambio en la estructura de la proteína puede significar una pérdida de la función.
- **Desnaturalización:** Consiste en la pérdida de la estructura terciaria y se puede producir por cambios de temperatura o variaciones del pH.

3.2.6 Funciones de las proteínas

Existen varias funciones de las proteínas todas de gran importancia y bien diferenciadas. Las proteínas determinan la forma y la estructura de las células y dirigen casi todos los procesos vitales.

Las funciones de las proteínas son específicas por cada tipo de proteína y permite a las células defenderse de agentes externos, mantener su integridad, controlar y regular funciones así como la reparación de daños.

Las funciones principales son las siguientes:

- **Estructural:** Forman tejido de sostén y relleno que confieren elasticidad y resistencia a órganos y tejidos.
- **Enzimática:** Actúan como biocatalizadores acelerando las reacciones químicas del metabolismo.
- **Hormonal:** Funcionan como mensajeros de señales hormonales, generando una respuesta en determinados órganos.
- **Defensiva:** Crean anticuerpos y regulan factores contra agentes extraños o infecciones.
- **Transporte:** Algunas proteínas tienen la capacidad de transportar sustancias como oxígeno, lípidos, electrones, etc.
- Etc.

3.2.7 Relación estructura – función de las proteínas

Las proteínas desempeñan un elevado número de funciones biológicas diferentes y cada una esta especializada en llevar a cabo una determinada función.

Entre las funciones más importantes encontramos las siguientes: catalíticas, estructurales, de transporte, nutrientes y de reserva, de defensa, reguladoras del metabolismo entre otras. La función de una proteína depende de la interacción de la misma con una molécula a la que llamamos ligando y este es específico de cada proteína (el centro activo).

3.2.8 Determinación experimental de las proteínas

Hay dos técnicas experimentales que pueden ser usadas para determinar la estructura tridimensional de las macromoléculas a nivel atómico. Cristalografía de Rayos X y Espectroscopia con Resonancia Magnética Nuclear (NMR), a pesar que son bastante complejos es necesario entender algunos aspectos básicos de los resultados, porque como se verá la mayoría de los métodos para la predicción de la estructura de la proteína son basados en datos de estructuras existentes.

3.2.9 Banco de datos

En años recientes hay un crecimiento explosivo de datos biológicos, las cuales se han depositado en bases de datos con un identificador único, dentro de las cuales se encuentran las de **proteínas**.

Las bases de datos de proteínas son las fuentes de información más exhaustiva y se clasifican en dos categorías: Datos de secuencias de archivos simples y bases de datos anotadas donde la información adicional ha sido añadido al registro de secuencia.

- Base de datos de secuencia primaria de proteínas tal como UniprotKB
- Bases de datos de secuencia de proteínas especializadas como ENZIME
- Bases de datos secundaria tal como InterPro
- **Bases de datos de estructura como PDB**

3.2.9.1 Propiedades de los archivos PDB

Los archivos de coordenadas moleculares contienen información precisa sobre la estructura tridimensional de una molécula. Estos archivos son de acceso público y pueden conseguirse en diferentes sitios a través de la WEB, uno de los sitios mas utilizados es el Protein Data Bank (PDB), aquí se encuentran los archivos de coordenadas moleculares para la mayoría de las moléculas cuya estructura terciaria ha sido dilucidada por las técnicas de difracción de Rayos X y Resonancia Magnética Nuclear, como también los archivos de coordenadas construidos mediante la aplicación de modelos matemáticos y debido a las

mejoras en estas técnicas el número de estructuras depositadas ha crecido exponencialmente.

Las técnicas computacionales nos sirven para analizar, caracterizar y visualizar estructuras proteicas, así como sus interacciones con otras proteínas, péptidos o ligandos.

3.2.9.2 Características de un archivo PDB

La información de un archivo PDB se encuentra distribuida en diferentes líneas o encabezados que contienen propiedades particulares de la molécula, las cuales se muestran en la Tabla 3.

HEADER	<i>Nombre y fecha de creación del archivo PDB.</i>
COMPND	<i>Nombre de la molécula.</i>
SOURCE	<i>Organismo del que se obtuvo la proteína.</i>
AUTHOR	<i>Lista de los Autores que proporcionaron la estructura molecular al PDB</i>
REVDAT	<i>Datos de revisión de la estructura de la proteína.</i>
REMARK	<i>Comentarios en relación a los artículos en donde se publicó la estructura molecular o sobre las características de la molécula.</i>
SPRSDE	<i>Lista de archivos de coordenadas para la misma estructura.</i>
SEQRES	<i>Secuencia de los aminoácidos de la proteína.</i>
FTNOTE	<i>Notas de pie de página. No todos los archivos lo tienen.</i>
HET & FORMUL	<i>Lista de cofactores, grupos prostéticos, inhibidores y otras sustancias no proteicas presentes.</i>
HELIX, SHEET & TURN	<i>Lista de los residuos con estructura secundaria en la proteína.</i>
CRYST1, ORIG & SCALE	<i>Información general sobre los cristales con los que se</i>

	<i>obtuvo la estructura por Rayos X</i>
ATOM & HETATM	<i>Contiene la información de la posición espacial en los ejes X,Y y Z de cada uno de los átomos, especificando el residuo y la cadena.</i>
CONNECT	<i>Enlaces formados entre los átomos no proteicos presentes en el PDB.</i>
MASTER & END	<i>Indican la finalización del archivo.</i>

Tabla 3: Encabezados de un archivo (.pdb)

Las líneas del tipo **ATOM**, tienen la siguiente estructura:

ATOM 29 CA PRO A 5 -2.551 -37.351 4.539 1.00 0.00 1CLG 79	
Número secuencial del átomo dentro del archivo:	29
Tipo de átomo	CA
Nombre del residuo, código de tres letras	PRO
Número del residuo y cadena	A-5
Coordenadas en angstroms del átomo en los ejes X,Y y Z de la celda unitaria	-2.551 -37.351 4.539
Ocupancia	1.00
Factor de temperatura	0.00

3.3 Bioinformática y biología computacional

La bioinformática es el campo de las ciencias en el que la biología, las ciencias de la computación y la tecnología de la información convergen para formar una sola disciplina. Abarca cualquier herramienta o método computacional usado para administrar, analizar y manipular grandes conjuntos de datos biológicos y su meta final es permitir el descubrimiento de nuevo conocimiento biológico así como crear una perspectiva global desde la cual unificar los principios de la biología [28]. Actualmente, el proceso de analizar e interpretar datos se conoce como biología computacional [29].

Con el progreso del proyecto Genoma Humano, cada vez más investigadores están involucrados en el dominio de tópicos de la bioinformática, como la predicción de secuencias, en la investigación farmacéutica para el diseño y descubrimiento de fármacos o la cura de enfermedades.

Al estudiar la causa de una enfermedad desde el punto de vista de la expresión de los genes puede requerir de demasiado trabajo y recursos si la experimentación biológica se utiliza desde el principio. Una aproximación desde la bioinformática permite disminuir el rango de variables y obtener las referencias más próximas y así reducir el desperdicio por ensayo y error del proceso experimental.

Particularmente, como los problemas en bioinformática están relacionados con cómputo y datos masivos, es necesario tener buenos algoritmos y computación de alto desempeño [30]. Sin duda, la bioinformática puede ser considerada como una de las fuerzas que están cambiando la forma en la cual las ciencias son conducidas en la actualidad [31].

3.3.1 Dinámica molecular computacional

En la actualidad se conoce la estructura tridimensional de sólo un pequeño conjunto de proteínas dado que para encontrarlas, se necesita experimentación en laboratorio, la cual resulta costosa. Determinar dicha estructura en forma computacional es un problema importante pues acelera la investigación y reduce el tiempo de análisis del experto.

La simulación por **mecánica y dinámica molecular** es el nombre de la técnica en la cual las trayectorias de un sistema de partículas interactuantes se calculan resolviendo numéricamente sus ecuaciones de movimiento.

Ha emergido como una herramienta viable y una poderosa opción de la bioquímica para la comprensión de las interacciones y particularmente para el diseño asistido por computador de fármacos basada en la estructura molecular.

La dinámica molecular permite estudiar la dinámica de macromoléculas grandes, incluyendo los sistemas biológicos. Los eventos dinámicos juegan un papel importante en controlar procesos que afectan las propiedades funcionales de las biomoléculas. El diseño de fármacos es comúnmente usado en la industria farmacéutica para probar las propiedades de una molécula en el computador sin necesidad de sintetizarla (lo cual es mucho más costoso).

Diseñar una simulación de dinámica molecular puede frecuentemente encontrar límites en el poder computacional.

3.3.2 Modelado Molecular

La modelización molecular es un conjunto de métodos que nos ayudan a imitar el comportamiento de las moléculas o de los sistemas moleculares, se basa en aplicaciones informáticas que permiten la construcción de modelos moleculares tridimensionales.

En el modelado molecular se calcula la estructura y energía de moléculas basado en movimientos nucleares, anteriormente se entendía por estructura molecular de un compuesto como la unión de los átomos entre sí, sin embargo con los estudios de espectroscopia vibracional y la difracción de rayos X, dieron un significado mas profundo a este término, en el cual además de considerar las conectividades de los átomos, se consideran también las longitudes de enlace, los ángulos de enlace y los ángulos de torsión de una molécula, desde entonces la determinación de la estructura molecular ha tenido un papel decisivo, tanto en las propiedades físicas de un compuesto como de su reactividad química.

Los **métodos de mecánica molecular** utilizan un conjunto de ecuaciones obtenidas empíricamente y variables según los autores, para reproducir la superficie de energía potencial [Engler, Altona, Burkert, Berg]. A este conjunto de ecuaciones, cuya forma matemática deriva de la mecánica clásica, se denomina campo de fuerzas, y contiene un conjunto de parámetros ajustables, optimizados de forma que se obtenga la mejor concordancia posible entre las propiedades moleculares, tales como geometrías, energías conformacionales relativas, calores de formación, etc., experimentales y calculadas.

La filosofía básica de la mecánica molecular reside en la consideración del campo de fuerzas como un modelo computacional para describir la superficie de energía potencial en función de los grados de libertad de una molécula.

En sentido estricto cada molécula posee su campo de fuerzas, sin embargo la interpolación y a veces la extrapolación de los datos es una metodología válida. Por esta razón se asume que el conjunto de ecuaciones y parámetros se puede transferir de una molécula a otra, es decir la parametrización se efectúa para un grupo de moléculas. La parametrización del campo de fuerzas es uno de los puntos fundamentales, dado que el grado de fiabilidad del método está limitado por la precisión de los mismos.

3.3.3 Mecánica molecular

“Conjunto de técnicas informáticas basadas en los métodos de química teórica y en los datos experimentales que pueden ser usados para analizar las moléculas o sistemas moleculares y/o para predecir las propiedades moleculares y biológicas”. **N. Claude Cohen**

El objetivo es la caracterización completa de compuestos, obteniéndose los valores de sus propiedades dependientes de su estructura 3D, que resulten de interés para el estudio que se esté realizando, de manera objetiva, rápida y adecuada.

Los cálculos de mecánica molecular requieren tiempos considerablemente menores que los mecánico-cuánticos, se pueden aplicar a compuestos de mayor magnitud y complejidad y reproducen los valores experimentales referentes a geometrías y energías con buena precisión.

La idea central de la mecánica molecular es que los enlaces poseen unos valores de longitudes y ángulos **ideales o naturales**, los cuales van a condicionar la geometría molecular y utilizan un conjunto de ecuaciones para reproducir la superficie de energía potencial. A este conjunto de ecuaciones, cuya forma matemática deriva de la mecánica clásica, se denomina campo de fuerzas y contiene un conjunto de parámetros ajustables, optimizados de forma que se obtenga la mejor concordancia posible entre las propiedades moleculares.

Puesto que la mecánica molecular sólo considera el movimiento nuclear, prescindiendo de los electrones, una molécula se puede representar mediante un modelo mecánico como un conjunto de masas (los átomos) unidas (a través de los enlaces) por fuerzas armónicas o elásticas. Estas funciones se pueden

describir como funciones de energía potencial de los parámetros estructurales tales como longitudes, ángulos de enlace, ángulos diedros, interacciones no enlazantes, etc. La combinación de las funciones de **energía potencial** es el **campo de fuerzas** y la energía **E**, de la molécula surge como resultado de las desviaciones de estos parámetros de los valores ideales. De acuerdo con esta suposición, la energía del sistema se puede desglosar en una suma de términos energéticos que representarán todas las interacciones posibles: deformaciones de los enlaces por elongación, plegamiento o rotación, interacciones entre masas, como se muestra en la Figura 11.

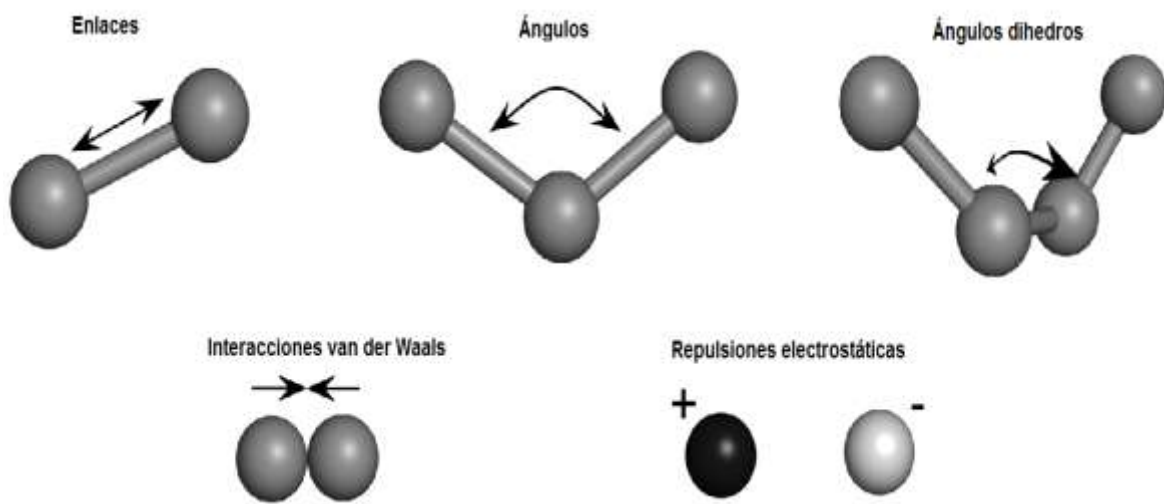


Figura 11: campo de fuerzas

Quedando una ecuación de la siguiente manera:

$$\mathbf{E}_{\text{total}} = \mathbf{E}_{\text{enlaces}} + \mathbf{E}_{\text{ángulos}} + \mathbf{E}_{\text{diedros}} + \mathbf{E}_{\text{vderW}} + \mathbf{E}_{\text{elec}}$$

Cada uno de los términos de la ecuación anterior tiene unas posiciones de equilibrio preferentes, asociadas con unos valores naturales o ideales de los parámetros estructurales y unas constantes de fuerza, conocidas experimentalmente o estimadas teóricamente, que permite evaluar las contribuciones energéticas de cada tipo de desviaciones presentes en una estructura determinada.

3.3.4 Conformación y propiedades moleculares

Las propiedades físicas, y en gran medida las propiedades químicas, de una molécula dependen de su geometría, esto es, de la posición de sus átomos en el espacio. Para muchas moléculas esta posición no permanece constante, sino que puede cambiar, con mayor o menor facilidad, en función de varios factores.

El análisis conformacional se encarga de estudiar entre otras cosas, los factores que hacen que una conformación (cualquier geometría que pueda adoptar una molécula por simple rotación de enlaces sencillos).

Se puede deducir que una molécula pequeña, al tener pocos átomos, no cuenta con una gran libertad de movimiento, por lo que el número de confórmeros (conformación o geometría, que corresponde a un mínimo en la superficie de la energía potencial de la proteína) posibles será pequeño. Lo mismo se puede decir de sistemas bastante rígidos que a pesar de tener un número relativamente grande de átomos, presentan un "confórmero" único debido a la rigidez de su estructura.

Sin embargo, a medida que una molécula tiene un mayor número de átomos generalmente se vuelve más flexible y por lo tanto el número de confórmeros posibles también aumenta. En estos casos es más difícil determinar experimentalmente cuántos y cuáles son los confórmeros más importantes que contribuyen a sus propiedades, por lo que el análisis conformacional de moléculas muy flexibles es bastante complicado.

Las consideraciones anteriores no sólo se aplican a los casos en los cuales obtenemos diferentes geometrías de una molécula por rotación de enlaces sencillos. Podemos imaginar por ejemplo, que un sistema formado por un grupo de varios átomos o moléculas que interactúan entre sí también puede adoptar diferentes arreglos. Dichos arreglos son dependientes de la posición de cada uno de los componentes del sistema, con respecto a los demás. Algunos de estos arreglos serán equivalentes a lo que anteriormente denominamos confórmeros es decir, son mínimos en su superficie de energía potencial. Entre mayor sea el número de componentes, el número de arreglos correspondientes a mínimos será mayor. Luego entonces un estudio equivalente al análisis conformacional en un sistema de este tipo representaría un problema aún más complicado. Generalmente la intensidad de las fuerzas presentes en interacciones intermoleculares es relativamente pequeña, por lo que la interconversión entre los diferentes mínimos posibles es extremadamente

rápida. Por lo tanto, se requiere de un trabajo más especializado para tratar de determinar experimentalmente los "confórmeros" de estos sistemas.

3.3.5 Interacciones moleculares

En un sistema molecular se promueve la unión de sus miembros a través de diversos tipos de interacciones, las cuales están definidas por las fuerzas de atracción y repulsión entre los diferentes miembros del sistema. Dichas fuerzas son responsables de la presencia, en mayor o menor grado, de geometrías espaciales específicas del sistema en cuestión, por ejemplo, de un confórmero. Las interacciones dentro de un sistema molecular están gobernadas principalmente por fuerzas atractivas y repulsivas que involucran electrones y núcleos.

Las interacciones intra e intermoleculares se pueden clasificar en dos tipos: Interacciones de enlace e interacciones no enlazantes. Dicha clasificación se explica brevemente a continuación.

3.3.5.1 Interacciones de enlace.

Dentro de esta clasificación, se encuentran las dos interacciones más fuertes: Interacciones iónicas y covalentes, y se describen brevemente en las siguientes líneas.

Interacción iónica: En los términos más simples, una interacción iónica es aquella atracción electrostática que se genera entre dos átomos cuando uno de ellos transfiere un electrón al otro. Comparadas con otro tipo de interacciones, las iónicas son las que se presentan más intensamente a través de la distancia espacial.

Interacción covalente: Esta interacción tiene su origen en la compartición de un par de electrones entre dos átomos. Esta compartición puede entenderse mejor en términos de orbitales atómicos y orbitales moleculares. El par de electrones que se comparte se localiza en un orbital molecular formado por la combinación de orbitales atómicos de los átomos participantes. Aunque el enlace covalente por lo general es bastante fuerte, esta intensidad depende una vez más y en gran medida, de los átomos que estén compartiendo los electrones. Por ejemplo, los enlaces covalentes entre átomos pequeños son generalmente más fuertes que los enlaces entre átomos más grandes. Esto se debe a que los orbitales de los átomos más pequeños pueden presentar un mayor traslape, produciendo orbitales moleculares más estables.

3.3.5.2 Interacciones no enlazantes

Las interacciones de van der Waals: Cuando dos átomos que únicamente pueden interactuar entre sí por interacciones no enlazantes, se encuentran a una distancia más corta de la correspondiente a la óptima para que ocurra la interacción, comienzan a operar fuerzas repulsivas. Estas fuerzas pueden ser lo suficientemente fuertes como para producir deformaciones estructurales de distancias y ángulos de enlace en los enlaces covalentes del resto de la molécula.

3.3.6 Energía potencial

Como se mencionó anteriormente, la posición relativa de los átomos de una molécula, o de un grupo de moléculas, determina la energía del sistema, cualquier variación en la posición de uno o más átomos modifica dicha energía.

Si fuera posible representar gráficamente el cambio en la energía potencial en función de cada uno de los cambios en las geometrías posibles para el sistema molecular, se obtendría como resultado la superficie de energía potencial total correspondiente. Puede hacerse entonces una analogía entre la superficie de energía potencial (SEP) y una malla extendida horizontalmente, la cual presenta deformaciones semejantes a montes y valles. La posición más baja de los valles correspondería a geometrías que son mínimos en la SEP, mientras que los puntos más altos de los montes corresponderían a geometrías que son máximos estados de transición en la misma superficie.

A pesar de que esta descripción parece ser muy simple, el desarrollo y análisis de una SEP pueden ser bastante complicados, como se describe a continuación.

La SEP es dependiente de los grados de libertad de un sistema llevados a un espacio tridimensional: Una molécula o un sistema de varias moléculas con un número N de átomos, tiene en total $3N$ grados de libertad. Cada grado de libertad corresponde a los movimientos posibles de cada uno de los átomos en relación a un sistema de ejes coordenados tridimensional, ya que cualquier movimiento se puede definir en función de un cambio en las coordenadas (x , y , z) de cada átomo. De los $3N$ grados de libertad, 6 de ellos corresponden a traslaciones y rotaciones del sistema como un todo, es decir a movimientos de todos los átomos en la misma dirección y al mismo tiempo.

Si se deseara trazar una gráfica de la SEP completa de cualquiera de estos sistemas necesitaríamos de cuatro o cinco dimensiones, respectivamente. El

problema es obvio y se vuelve mucho más complejo a medida que aumenta el número de átomos que componen el sistema bajo estudio.

La determinación de todos los mínimos presentes en la SEP de una molécula o grupo de moléculas, es una tarea complicada. Finalmente, no es posible conocer en forma absoluta ni el número total de mínimos, ni las geometrías correspondientes a cada uno de ellos por simple inspección del sistema, sobre todo para sistemas que presentan una gran flexibilidad.

En muchos de los casos, no es posible ni siquiera determinar cual es la geometría correspondiente al mínimo global el cual, como ya se ha mencionado, es el que determina en mayor grado las propiedades de la molécula.

3.3.7 Potenciales y campos de fuerza

El término campo de fuerza —force field— se utiliza para describir al conjunto de ecuaciones —potenciales— y parámetros que tratan de representar los diversos tipos de interacciones existentes en un sistema. A la aplicación y estudio de estos potenciales o campos de fuerza para el estudio de un sistema molecular se le denomina en general mecánica molecular. Los métodos de mecánica molecular pueden aplicarse entonces al estudio de la SEP de un sistema, teniendo en cuenta que la SEP obtenida por mecánica molecular será tan parecida a la real en la medida en que los potenciales sean capaces de representar de manera fidedigna a las interacciones presentes en el sistema.

3.3.8 Desarrollo de los campos de fuerza

Todos los métodos de mecánica molecular difieren en el número y el tipo de las funciones de energía potencial que componen el campo de fuerzas y no es posible un tratamiento general

El proceso de desarrollo de un campo de fuerzas consta de las siguientes etapas:

1. Suposición de un conjunto de ecuaciones para expresar las funciones de energía potencial
2. Parametrización de esas ecuaciones

3. Retoque y reparametrización de esas ecuaciones hasta llegar a una concordancia aceptable entre los valores calculados y los datos experimentales.

En general no existen reglas estrictas sobre el número y tipo de funciones de energía potencial que debe contener un campo de fuerzas.

3.3.9 Proceso de optimización

Al proceso de obtención de la geometría correspondiente a un mínimo también se le conoce como optimización geométrica.

Una vez conocidas las geometrías de menor energía de las moléculas, se puede realizar un estudio más profundo de las propiedades del sistema, por lo que en la última década se ha aprovechado el incremento de poder de procesamiento que han tenido las computadoras, para realizar simulaciones de sistemas químicos empleando funciones matemáticas simples, ya que incrementa dramáticamente la aceleración en el proceso.

Dado que la energía de un sistema molecular tiene una relación directa con su geometría, es muy conveniente expresarla por medio de las ecuaciones que nos definen un campo de fuerza y las estructuras de mayor interés son aquellas que nos representan un mínimo en la superficie de la energía potencial del sistema.

Es conveniente señalar que cada una de las ecuaciones de un campo de fuerzas nos define una parte de la energía potencial del sistema.

3.4 Operadores lineales

Un operador lineal, también denominado transformación o función lineal, es aquella aplicación cuyo dominio y contradominio son espacios vectoriales.

Consideremos dos espacios vectoriales V y W , entonces una transformación lineal T desde el espacio V en W , es una función que convierte vectores del espacio V a vectores en el espacio W , esas asignaciones son únicas y deben cumplir propiedades para que sea una transformación.

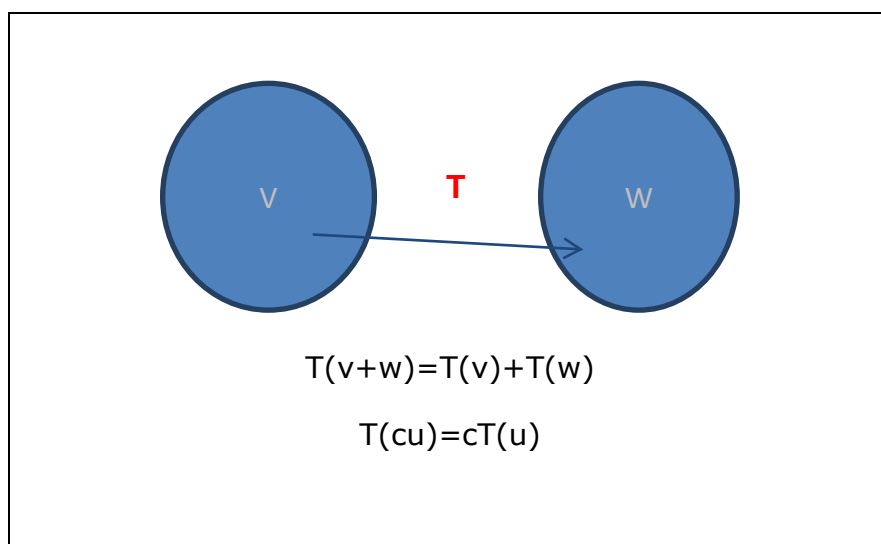


Figura 12: Reglas de un operador lineal

Para la minimización de energía los operadores lineales que se consideraron, son las transformaciones geométricas, debido a que nos permiten ajustar la estructura molecular de manera eficiente, para comprender un poco más la forma en que se emplearán, se describen a continuación las transformaciones que se emplearán.

3.4.1 Transformaciones geométricas

Existen varias transformaciones geométricas, pero sólo se revisarán el *escalamiento*, *la traslación* y *la rotación*.

3.4.1.1 Escalamiento

El escalamiento permite cambiar el tamaño de un objeto expandiéndolo o contrayéndolo en sus dimensiones.

Extendiendo la idea a 3D, el escalamiento implica el cambio de tamaño de un poliedro, donde cada punto $p = (x_1, x_2, x_3)$ es transformado por la multiplicación de tres factores de escalamiento: s_1 , s_2 , y s_3 a lo largo de los ejes X_1 , X_2 y X_3 respectivamente, de esta forma, las coordenadas del nuevo punto $p' = (x_1', x_2', x_3')$ se obtiene como:

$$x_1' = x_1 \cdot s_1$$

$$x_2' = x_2 \cdot s_2$$

$$x_3' = x_3 \cdot s_3$$

Sea $s = (s_1', s_2', s_3')$ el vector de factores de escalamiento y $S(s)$ la matriz de escalamiento, en coordenadas homogéneas el escalamiento de un punto p en 3D se puede expresar como el producto matricial $p' = p \cdot S(s)$, es decir:

$$[x_1' \quad x_2' \quad x_3' \quad 1] = [x_1 \quad x_2 \quad x_3 \quad 1] \cdot \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ecuación 1: Expresión matricial para el escalamiento 3D

La Figura 13 muestra el efecto de escalamiento de una figura con $s_1=2$, $s_2=2.5$, y $s_3=1.5$

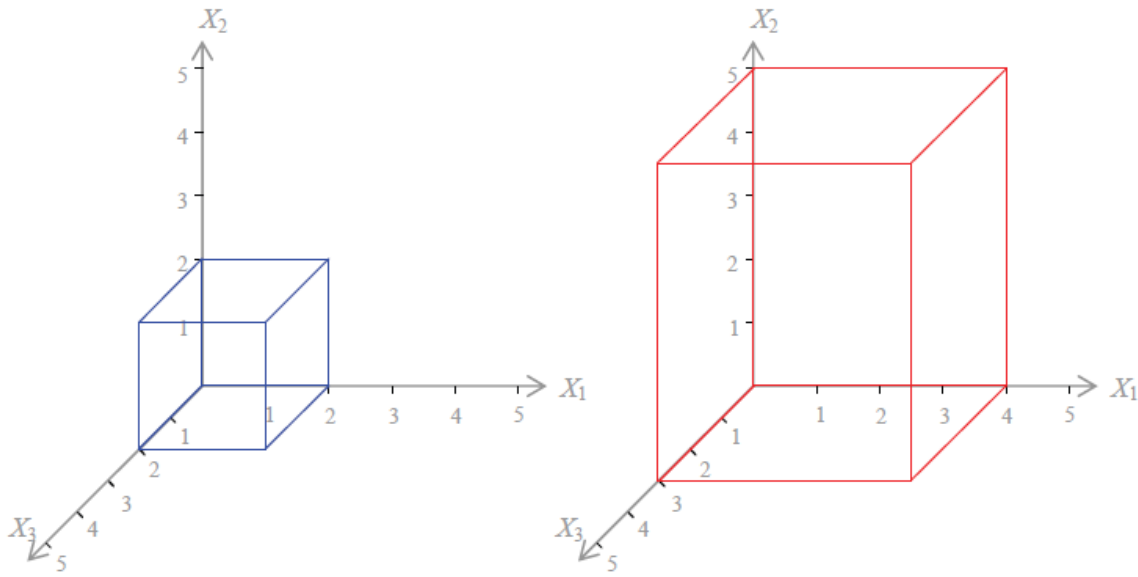


Figura 13: Ejemplo de escalamiento 3D

3.4.1.2 Traslación

La traslación permite desplazar un objeto a lo largo de sus dimensiones, como resultado se obtiene un cambio de posición.

La traslación 3D implica el desplazamiento de un poliedro, en el que para cada punto $p = (x_1, x_2, x_3)$ es trasladado d_1 unidades en el eje X_1 , d_2 unidades en el eje X_2 y d_3 unidades en el eje X_3 , de esta forma las coordenadas del nuevo punto $p' = (x_1', x_2', x_3')$ se obtiene como:

$$x_1' = x_1 + d_1$$

$$x_2' = x_2 + d_2$$

$$x_3' = x_3 + d_3$$

Sea $d = (d_1', d_2', d_3')$ el vector de distancias y $T(d)$ la matriz de traslación, en coordenadas homogéneas la traslación de un punto p en 3D se puede expresar como el producto matricial $p' = p \cdot T(d)$, es decir:

$$[x_1' \quad x_2' \quad x_3' \quad 1] = [x_1 \quad x_2 \quad x_3 \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_1 & d_2 & d_3 & 1 \end{bmatrix}$$

Ecuación 2: Expresión matricial para la traslación 3D

La Figura 14 muestra el efecto de traslación de una figura con $d_1=2$, $d_2=0$, y $d_3=2$

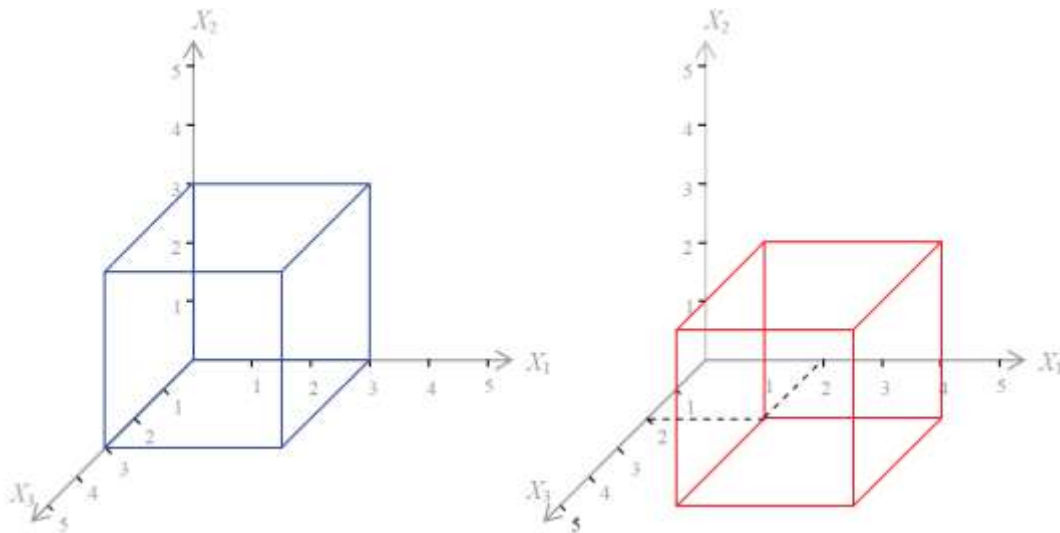


Figura 14: Ejemplo de traslación 3D

3.4.1.3 Rotación

La rotación permite girar un objeto sobre un eje de rotación, dado un valor de ángulo de rotación y su dirección.

En 3D para hacer rotar un objeto se necesitan dos puntos no coincidentes que determinan un segmento de recta, cuya línea de soporte define un eje lineal de rotación.

Las *rotaciones principales* 3D, son aquellas cuando el eje de rotación se encuentra sobre alguno de los tres ejes principales: X_1 , X_2 o X_3 , las rotaciones sobre cualquier otro eje arbitrario son llamadas *rotaciones generales* 3D. Se recuerda que inicialmente, se analizan las rotaciones principales.

Por convención, los ángulos de rotación positivos producen rotaciones en contra de las manecillas del reloj sobre el eje de rotación, esto es si se observa el giro desde la parte positiva del eje hacia el origen.

Otra forma de determinar la dirección de un giro positivo mediante la **regla de la mano derecha** como se muestra en la Figura 15 es de la siguiente manera:

"Si se coloca el dedo pulgar de la mano derecha sobre el eje de rotación apuntando a la parte positiva de dicho eje, el giro natural del resto de los dedos indica la dirección positiva del giro".

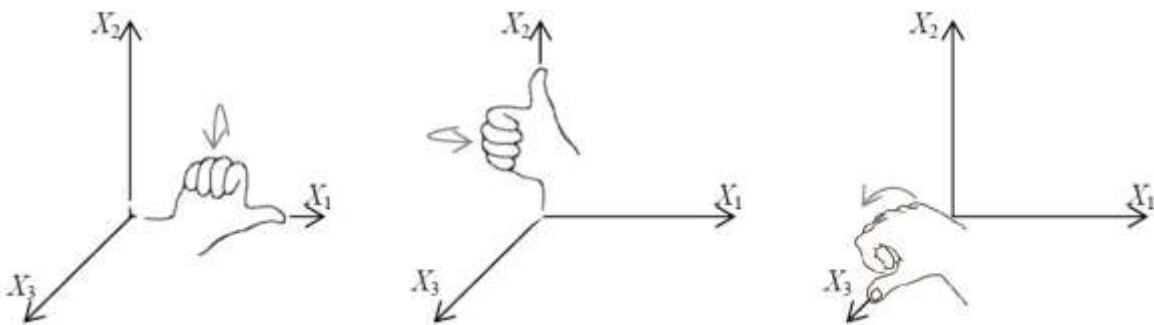


Figura 15: Regla de la mano derecha para obtener la dirección de un giro positivo en 3D

Es necesario entender el concepto de rotación en 3D como una extensión de la rotación 2D y recordar que la rotación 2D es el giro sobre el eje de rotación, que es perpendicular al plano X_1X_2 , el cual en 3D corresponde al eje X_3 , entonces se tiene la primera de las rotaciones principales.

De esta forma, por cada punto $p = (x_1, x_2, x_3)$ dado un ángulo ϑ , puede ser rotado sobre el eje X_3 en sentido contrario a las manecillas del reloj, obteniendo las coordenadas del nuevo punto $p' = (x_1', x_2', x_3')$ de la misma

forma en como se analizó en el espacio 2D, quedando la coordenada X_3 sin cambio entonces se tiene:

$$\begin{cases} x_1' = x_1 \cos \theta - x_2 \sin \theta \\ x_2' = x_1 \sin \theta + x_2 \cos \theta \\ x_3' = x_3 \end{cases}$$

Ecuación 3: Fórmulas para la rotación 3D alrededor del eje X_3

Sea $R_3(\theta)$, es decir:

$$\begin{bmatrix} x_1' & x_2' & x_3' & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ecuación 4: Expresión matricial para la rotación 3D alrededor del eje X_3

La FIGURA 16 muestra el efecto de rotación sobre el eje X_3 de una figura con $\theta=20^\circ$.

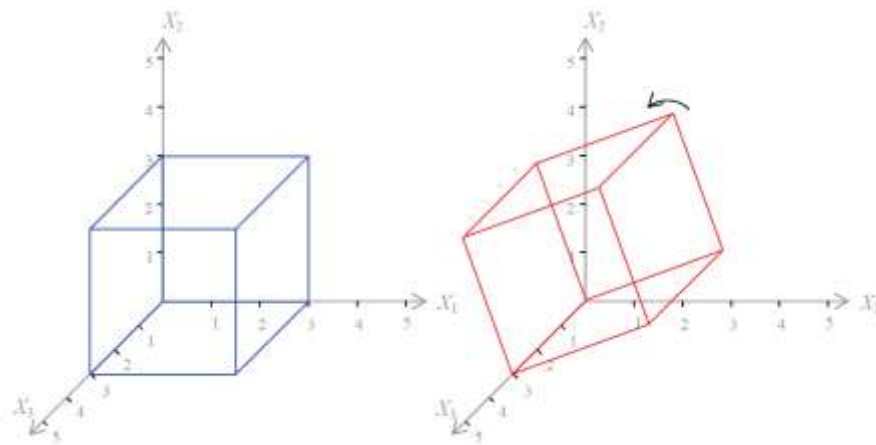


Figura 16: Ejemplo de rotación 3D sobre el eje X_3

Las ecuaciones para las rotaciones sobre el eje X_1 y eje X_2 , pueden ser obtenidas mediante las permutaciones cíclicas de los parámetros x_1, x_2, x_3 :

$$X_1 \rightarrow X_2' \rightarrow X_3'' \rightarrow X_1$$

como se muestra en la siguiente Figura

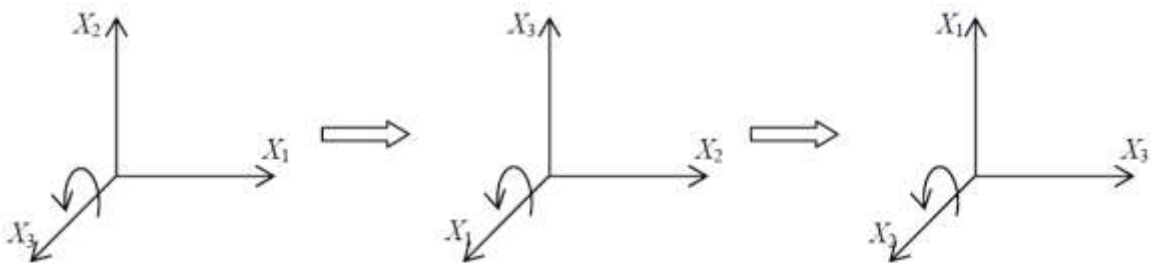


Figura 17: Permutaciones cíclicas de los ejes coordenados

Entonces, aplicando estas sustituciones cíclicas en la ECUACIÓN 3, se obtienen las ecuaciones para la rotación alrededor del eje X_1 dado un ϑ .

$$\begin{cases} x_2' = x_2 \cos \theta - x_3 \sin \theta \\ x_3' = x_2 \sin \theta + x_3 \cos \theta \\ x_1' = x_1 \end{cases} \Rightarrow \begin{cases} x_1' = x_1 \\ x_2' = x_2 \cos \theta - x_3 \sin \theta \\ x_3' = x_2 \sin \theta + x_3 \cos \theta \end{cases}$$

Ecuación 5: Fórmulas para la rotación 3D alrededor del eje X_1 .

Sea $R_1(\vartheta)$ la matriz de rotación alrededor del eje X_1 , en coordenadas homogéneas la rotación de un punto p alrededor de dicho eje, se puede expresar como el producto matricial $p' = p \cdot R_1(\vartheta)$, es decir:

$$\begin{bmatrix} x_1' & x_2' & x_3' & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ecuación 6: Expresión matricial para la rotación 3D alrededor del eje X_1 .

Aplicando nuevamente las sustituciones cíclicas en la Ecuación 5, se obtienen las siguientes fórmulas para la rotación alrededor del eje X_2 dado un ángulo ϑ .

$$\begin{cases} x_3' = x_3 \cos \theta - x_1 \sin \theta \\ x_1' = x_3 \sin \theta + x_1 \cos \theta \\ x_2' = x_2 \end{cases} \implies \begin{cases} x_1' = x_1 \cos \theta + x_3 \sin \theta \\ x_2' = x_2 \\ x_3' = -x_1 \sin \theta + x_3 \cos \theta \end{cases}$$

Ecuación 7: Fórmulas para la rotación 3D alrededor del eje X_2 .

Sea $R_2(\vartheta)$ la matriz de rotación alrededor del eje X_2 , en coordenadas homogéneas la notación del punto p alrededor de dicho eje, se puede expresar como el producto matricial.

$p' = p \cdot R_2(\vartheta)$, es decir:

$$\begin{bmatrix} x_1' & x_2' & x_3' & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ecuación 8: Expresión matricial para la rotación 3D alrededor del eje X_2 .

3.4.1.4 Transformaciones compuestas

Las transformaciones compuestas son útiles para obtener el efecto combinado de una serie de transformaciones geométricas.

Con la representación matricial manejada hasta el momento, se puede aplicar una secuencia de transformaciones, calculando simplemente la multiplicación matricial de cada una de las matrices de transformación.

Dado que se está manejando una representación de la posición de un punto p como vector renglón, se puede aplicar una transformación compuesta a un punto p , multiplicando las matrices de izquierda a derecha, comenzando con el punto p .

Por ejemplo, si se desea aplicar una traslación, seguida de un escalamiento a un punto p , la posición final del punto p' se calcula de la siguiente manera:

$$p' = T(d) S(s) p$$

O bien una traslación, seguida de una rotación:

$$p' = T(d) R_{a,b}(\vartheta) p$$

Las matrices de transformaciones compuestas para el caso que sean del mismo tipo, se comportan de la siguiente manera:

- Las traslaciones sucesivas son aditivas: $T(d_a) \cdot T(d_b) = T(d_a + d_b)$
- Los escalamientos sucesivos son multiplicativos: $S(s_a) \cdot S(s_b) = S(s_a \cdot s_b)$
- Las rotaciones sucesivas son aditivas: $R_{a,b}(\theta) \cdot R_{a,b}(\theta) = R_{a,b}(\theta + \theta)$

El fin de obtener esta secuencia de transformaciones es obtener una transformación geométrica total, que nos ajuste la energía del sistema.

3.5 Opengl

3.5.1 ¿Qué es OpenGL?

OpenGL es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada por SGI (Silicon Graphics Inc.) la cual nos da la posibilidad de dibujar primitivas de forma sencilla y rápida aprovechando las capacidades de la tarjeta gráfica instalada en el sistema.

La librería se ejecuta a la par con nuestro programa, independientemente de la capacidad gráfica de la máquina que usamos. Eso significa que la ejecución se dará por software a no ser que contemos con hardware gráfico específico en nuestra máquina. Si contamos con tarjetas aceleradoras de video por supuesto que gozaremos de una ejecución muchísimo más rápida en tiempo real.

Entre las principales acciones que podemos realizar mediante comandos OpenGL se incluyen la especificación de los objetos y operaciones necesarias para producir aplicaciones interactivas en 3D.

Como se mencionó anteriormente al ser multiplataforma no hay comandos para ejecutar tareas dependientes del hardware disponible como la gestión de ventanas o la lectura de datos de entrada del usuario. OpenGL tampoco provee de comandos de alto nivel para describir modelos en 3D.

Esta biblioteca puede usarse usando una gran variedad de lenguajes de programación.

3.5.2 OpenGL como una máquina de estados

OpenGL tiene un comportamiento similar al de una máquina de estados, lo cual significa que se tienen que activar y desactivar opciones, además de realizar ciertas acciones que tendrán como resultado una representación en pantalla o no, de una serie de datos dependiendo del estado en que nos encontremos.

De forma muy abstracta, para dibujar en OpenGL se deben seguir estos pasos:

- 1.- Activar todas las opciones que van a ser persistentes en la escena
- 2.- Activar las opciones que establecen el estado de un objeto específico

- 3.- Dibujar el objeto
- 4.- Desactivar las opciones propias de ese objeto
- 5.- Volver al punto 2, hasta haber dibujado todos los objetos

3.5.3 Bibliotecas relacionadas con OpenGL

OpenGL dispone de un conjunto de comandos potente pero simple, que puede utilizarse para crear comandos de dibujo de más alto nivel a través de distintas bibliotecas que aumentan la potencia de OpenGL:

OpenGL Utility Library (GLU): contiene bastantes rutinas que usan ogl a bajo nivel para realizar tareas como transformaciones de matrices, para tener una orientación específica, subdivisión de polígonos, etc.

OpenGL Utility Toolkit (GLUT): es un sistema de ventanas, escrito por Mark Kilgard, que es independiente del sistema de ventanas usado, dándonos funciones tipo `abrir_ventana()`.

3.5.4 Transformaciones geométricas con OpenGL

Las funciones disponibles en OpenGL para especificar transformaciones en 3D son: traslación, rotación y escalado. Estas rutinas actúan sobre la matriz de Modelo/Vista, por lo que serán de utilidad tanto para aplicar transformaciones a los distintos objetos de la escena, como para definir las transformaciones asociadas al posicionamiento de dicha escena en el volumen de la vista.

Las tres rutinas que proporciona OpenGL para definir transformaciones son **`glTranslate*`, `glRotate*` y `glScale*`.**

La rutina de traslación `void glTranslate{fd} (TYPE x, TYPE y, TYPE z);` multiplica la matriz actual por una matriz de traslación con desplazamientos indicados por x, y, z .

La rutina de giro `void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);` multiplica la matriz que gira el objeto en el sentido contrario a las agujas del reloj alrededor del eje que va desde el origen al punto indicado por x, y, z . Lo más habitual es girar sobre los ejes de coordenadas, por lo que los valores de los parámetros serán:

Giro sobre X: $x = 1.0, y = 0.0, z = 0.0$

Giro sobre Y: $x = 0.0, y = 1.0, z = 0.0$

Giro sobre Z: $x = 0.0$, $y = 0.0$, $z = 1.0$

Por último la rutina de escalado *void glScale{fd} (TYPE x, TYPE y, TYPE z);* multiplica la matriz actual por una matriz que amplía o reduce el objeto con respecto a los ejes coordenados. Cada coordenada (x , y , z) del objeto se multiplica por el argumento correspondiente x,y,z . Valores de estos parámetros inferiores a 1.0 reducen el objeto, mientras que valores superiores a 1.0 lo amplían.

3.5.5 Aplicaciones

Existen múltiples aplicaciones para los que OpenGL ha sido de gran ayuda dentro de las que encontramos las siguientes:

- Videojuegos
- Simulaciones
- Ciencia y análisis

Capítulo 4

Metodología

Primero que nada hay que recordar que el objetivo del trabajo es hacer uso de las GPU para acelerar el proceso de evaluación y minimización de la energía potencial de las proteínas, ya que es uno de los procesos más importantes en las aplicaciones de dinámica molecular y para el estudio de las estructuras moleculares, el cual además es apto para ser manejado de tal forma que se aproveche la arquitectura de las GPU por la gran cantidad de datos que se requiere procesar.

Vale la pena recordar que la caracterización completa de un compuesto, con la que se pueda obtener los valores de sus propiedades, es dependiente de su estructura 3D.

Para esta caracterización es necesario primeramente llevar a cabo la construcción de modelos moleculares tridimensionales y hacer un análisis de sus posibles conformaciones asociadas con sus energías, las cuales pueden ser calculadas por métodos de mecánica molecular.

Los métodos de mecánica molecular realizan los cálculos aplicando la mecánica clásica o Newtoniana, por tal motivo al aplicar las leyes de la mecánica clásica:

- Se tratan a las moléculas como conjuntos de esferas (átomos), unidos por muelles (enlaces) con constante K (asimilable a fuerza de enlace).
- No tratan explícitamente los electrones.
- Basan los estados energéticos en interacciones entre núcleos.
- Describen perfectamente los enlaces que se forman y no los que se rompen.
- Permiten obtener funciones de energía potencial sencillas.
- Aplican la aproximación de Born-Oppenheimer, según la cual el estado de la molécula depende de los núcleos.

Entonces la **hipótesis** de partida que tenemos es que los enlaces tienen longitudes y ángulos naturales o de equilibrio a los que las moléculas se ajustan, por lo que para que las proteínas cumplan su función es necesario que estos valores estén dentro de ciertos límites.

Ahora hay que considerar que los núcleos están sometidos a un **campo de fuerzas**, que de acuerdo a lo explicado en el apartado de mecánica molecular del capítulo 3 (Fundamentos teóricos), este campo incluirá los términos que corresponden a las las tensiones de los enlaces, de sus ángulos, torsiones y las interacciones de V der W., de tal forma que lo que tenemos es una ecuación de la siguiente manera.

$$E_{total} = E_{enlaces} + E_{ángulos} + E_{diedros} + E_{VdW}$$

Ecuación 9: campo de fuerzas

Donde la aproximación de este campo de fuerzas permite:

- Describir las moléculas empleando una descripción de términos descriptores tales como (distancias de enlace, ángulos, torsiones, diedros, etc).
- Describir la energía potencial debida a interacciones entre átomos enlazados y entre átomos no enlazados.

Como se puede apreciar en la Figura 18.

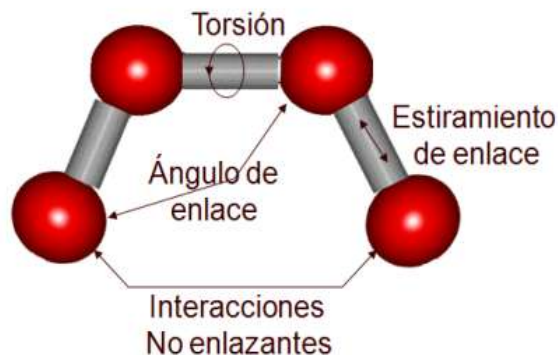


Figura 18: Fuerzas de interacción entre partículas

Una vez considerando esto, es necesario tener cierta información para llevar a cabo los cálculos tal como:

- Lista de átomos
- Nomenclatura para la escritura
- Parámetros para los términos funcionales

Todo esto nos ayuda a la prevención de errores, por último con respecto al campo de fuerzas, cada uno de los términos se desglosa en ecuaciones de mecánica clásica, las cuales nos describen la superficie de la energía y las propiedades físicas de la molécula.

$$E_{total} = \sum E_{enlaces} + \sum E_{ángulos} + \sum E_{diedros} + \sum E_{vdW}$$

$$\begin{aligned} E_{potencial} &= E_{total} \\ &= \sum_{enlaces} k_b(b_0 - b)^2 + \sum_{ángulos} k_\theta(\theta_0 - \theta)^2 + \\ &\quad \sum_{diedros} k_\phi(1 - \cos(n\phi - \delta)) + \\ &\quad \sum_{vdW} \left(\frac{A}{r_{ij}} + \frac{B}{r_{ij}^2} + \frac{Q_i Q_j}{r_{ij}} \right) \end{aligned}$$

Ecuación 10: Modelo matemático de la energía potencial

Una vez planteado el modelo matemático de la Ecuación 10, es necesario modelar la estructura molecular de la proteína a estudiar y posteriormente realizar el análisis.

Aunque pareciera tarea fácil no lo es, debido a que sólo contamos con la descripción de los átomos que componen la molécula y con la cual construiremos su estructura tridimensional y su topología, que es la relación que se tiene para cada uno de los átomos y la cual nos permita visualizar la molécula como se aprecia en la Figura 19, para así proceder con su análisis.

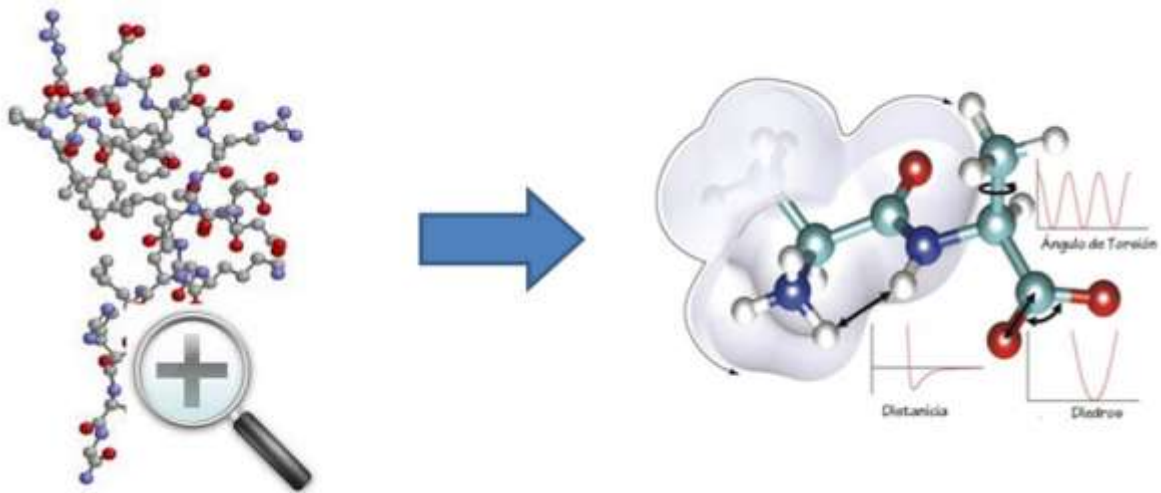


Figura 19: Estructura tridimensional de una proteína

El algoritmo que realiza todos estos procesos se muestra en la Figura 20 y posteriormente se presenta una descripción de cada proceso.

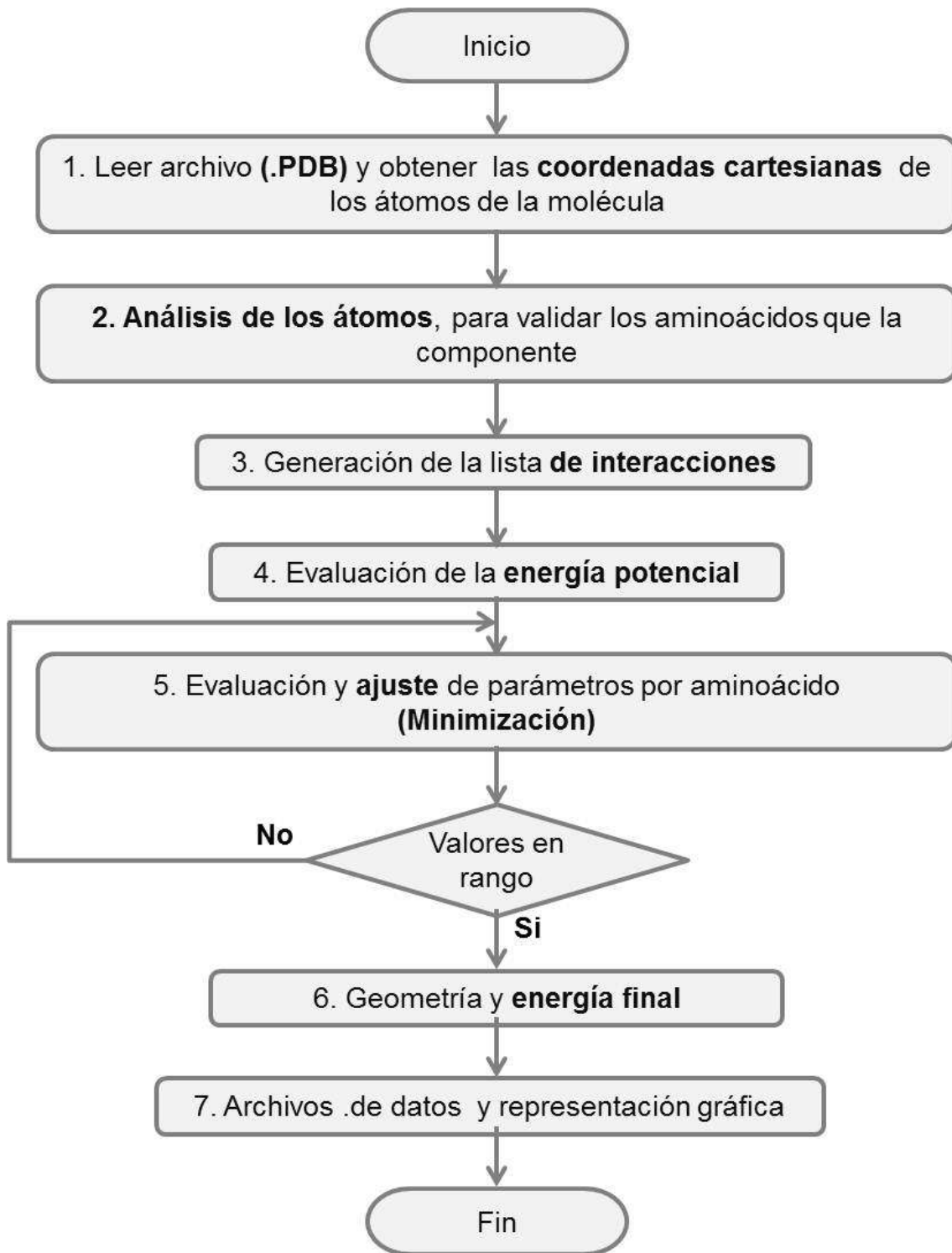


Figura 20: Algoritmo general de evaluación y minimización de energía

Proceso 1

En este proceso se analizará el archivo con extensión (.pdb), el cual puede ser extraído de la una de las diferentes bases de datos de proteínas o bien de datos de laboratorio, el único requisito es que cumpla con la estructura del formato establecido en los archivos (.pdb).

Una vez teniendo este archivo, se comenzará el análisis de la estructura primaria de la molécula. Para ello lo primero que se tiene que hacer es la extracción de la información que se tiene de cada átomo, la parte que se refiere a la composición química servirá para validar que es una molécula válida para su análisis y la parte de las coordenadas cartesianas para realizar el análisis estructural, como se puede apreciar en la Figura 21.

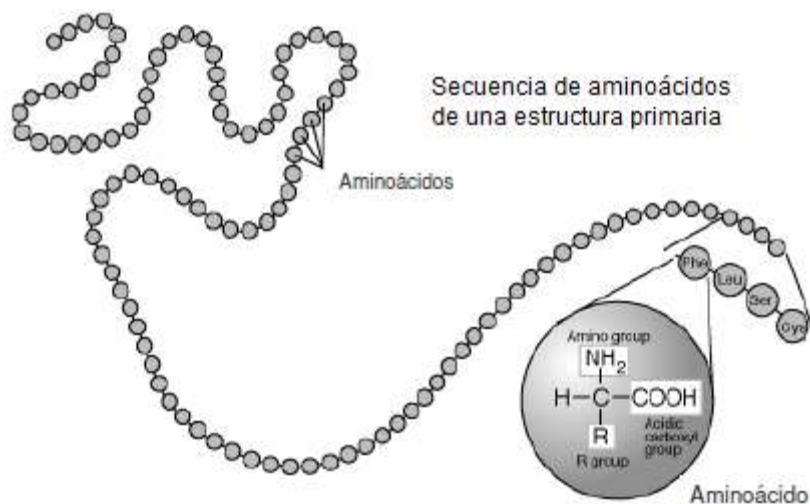


Figura 21: Estructura primaria de la proteína

Proceso 2

Una vez validado el archivo (.pdb), se procede a separar los átomos por aminoácido, para armar la información de los aminoácidos que conforman la estructura primaria, de alguna manera desde aquí, se comienza con el proceso de la paralelización, ya que al separar los datos por grupos la información se comienza a trabajar de manera independiente, para que esta sea manejada de manera simple en el algoritmo que procesara los datos.

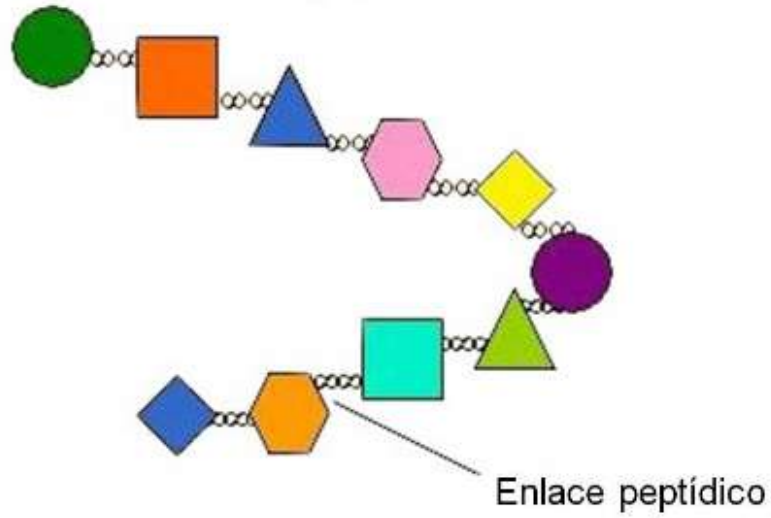


Figura 22: Representación de tipos de aminoácidos

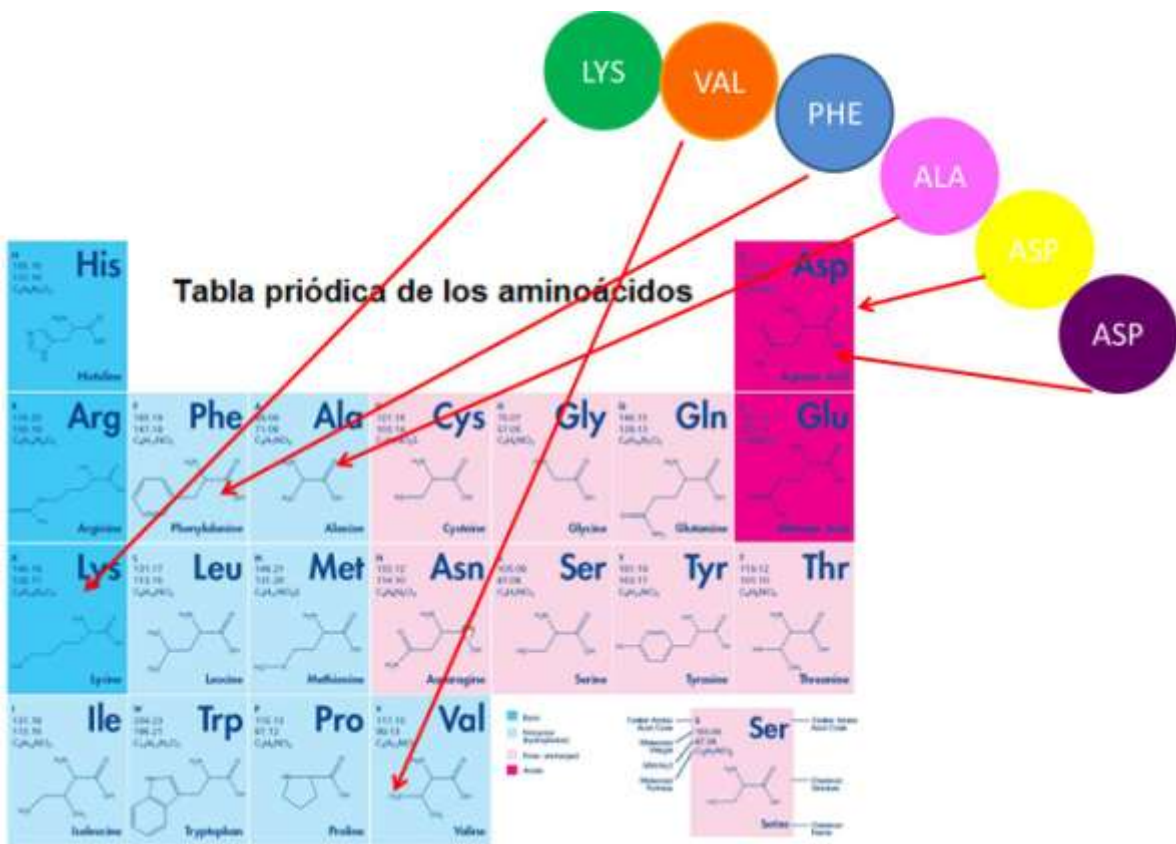


Figura 23: estructura química de los aminoácidos

Proceso 3

De acuerdo a la Ecuación 10 nos interesan las interacciones que tienen entre átomos con respecto a los enlaces, ángulos y diedros que se forman entre ellos, pues son los que determinan la estructura de la molécula y de la que depende su función.

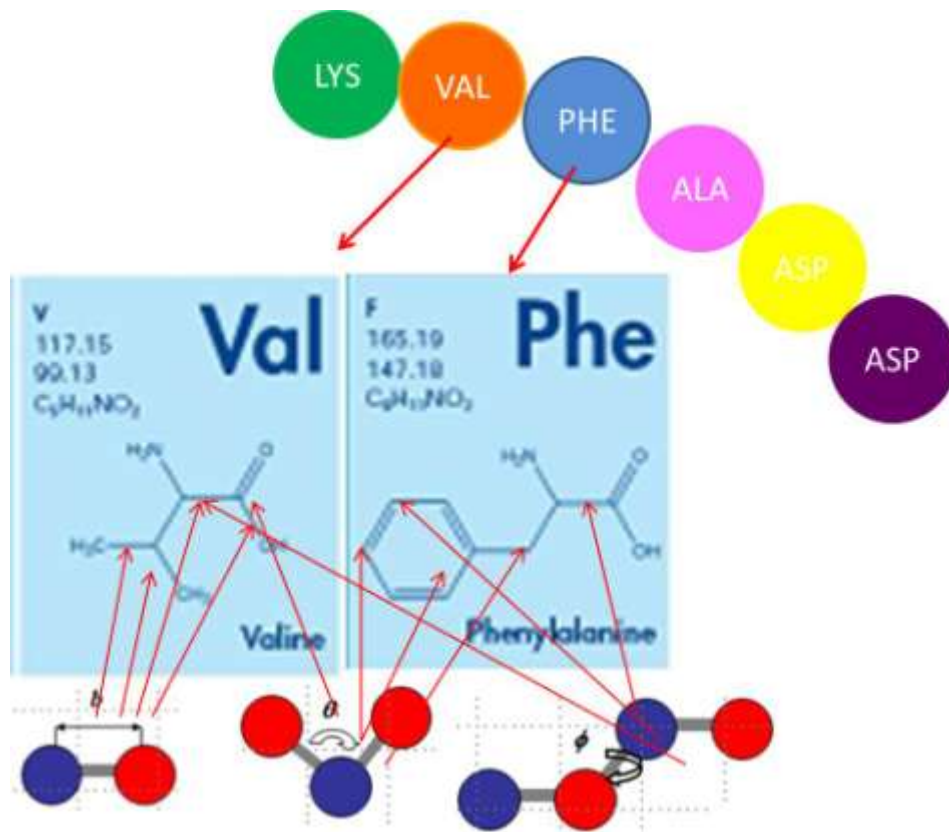



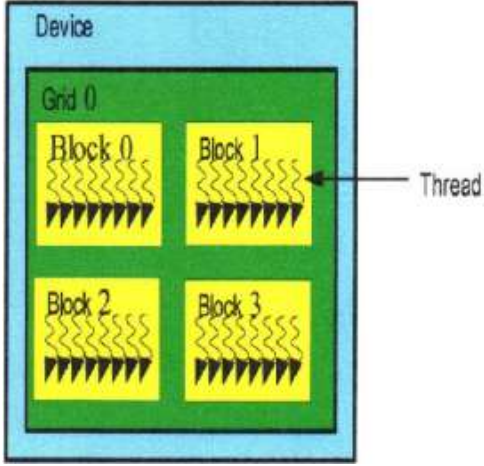
Figura 24: Campo de fuerzas por cada átomo de la molécula

Al ver la Figura 24 podrá quedar un poco más claro que la gran cantidad de relaciones que se tienen que generar y finalmente procesar. Es conveniente aclarar que todos estos cálculos son para obtener sólo una de las configuraciones estructurales de la superficie de energía potencial, tal y como se explico en el apartado de 3.3.6 Energía potencial.

Una vez que se tiene identificados los Id. de los átomos que intervienen en alguna de las interacciones, se realiza las tres listas, la primera corresponderá a los todos los enlaces, la segunda a los ángulos y la tercera a los diedros. Esta lista se genera por niveles, el primero es por aminoácido y el segundo es a nivel molécula.

Proceso 4

Para la evaluación de la energía potencial con un algoritmo en paralelo, este se debe de dividir en dos partes, la parte que se procesará en forma secuencial o lineal y la otra en paralelo. Con respecto al algoritmo de la Figura 20 los procesos (1-3) se procesan por medio de la CPU, ya que únicamente se encargan de realizar la relación entre datos.

<ol style="list-style-type: none"> 1. Leer archivo (.PDB) y obtener las coordenadas cartesianas de los átomos de la molécula 2. Análisis de los átomos, para validar los aminoácidos que la componente 3. Generación de la lista de interacciones 	
<p>Al tener las interacciones separados por enlaces, ángulos y diedros, se procesan en paralelo de la siguiente manera</p> <p style="text-align: center;"><i>Kernel enlaces <<<>>></i></p> <p style="text-align: center;"><i>Kernel ángulos <<<>>></i></p> <p style="text-align: center;"><i>Kernel diedros <<<>>></i></p>	

La lista de interacciones que se genera depende de la molécula o proteína que se está trabajando, por lo que entre más grande sea el número de átomos que tenga la molécula, mayor será el número de interacciones. Por lo general son cantidades realmente grandes las cuales se pueden dividir en grupos más pequeños pero que al final siguen siendo enormes y las cuales necesitan ejecutar la misma operación.

En el proceso de la evaluación de la energía lo más complicado es tener la topología de la molécula, ya que esta es la que no determinará las interacciones que tienen los átomos, pero como en este punto ya la tenemos, lo que resta es evaluar cada una de las interacciones de acuerdo a la Ecuación 10, por lo que si se trata de ejecutar el mismo código N veces lo más fácil es llamar funciones *kernel*, o sea funciones que se ejecuten en la GPU y se ejecuten las operaciones del mismo tipo a la vez, por último para obtener el cálculo de la energía potencial, hay que hacer uso de ciertos parámetros ya definidos.

El resultado que obtendremos de la energía, es la forma en que se está comportando la molécula, pero como anteriormente se mencionó lo que se busca es que la proteína cumpla su función y por lo tanto sus valores deben de estar dentro de los parámetros naturales.

Proceso 5

Para saber si esta estructura molecular está dentro de los parámetros naturales, es necesario realizar un análisis a detalle por átomo, auxiliándonos de tablas que nos definen los valores dentro del rango que deben de cumplir.

H--H	74	436	H--C	109	413	C--C	154	348	C=C	134	614
C--C	154	348	H--N	101	391	C--N	147	308	C≡C	120	839
N--N	145	170	H--O	96	366	C--O	143	360			
O--O	148	145	H--F	92	568	C--S	182	272	O--O	148	145
F--F	142	158	H--Cl	127	432	C--F	135	488	O=O	121	498
Cl-Cl	199	243	H--Br	141	366	C--Cl	177	330			
Br-Br	228	193	H--I	161	298	C--Br	194	288	N--N	145	170
I--I	267	151				C--I	214	216	N=N	110	945

Figura 25: Longitudes y energías de enlace

En caso de no estar dentro del rango, es necesario hacer un ajuste de parámetros, mediante el siguiente algoritmo:

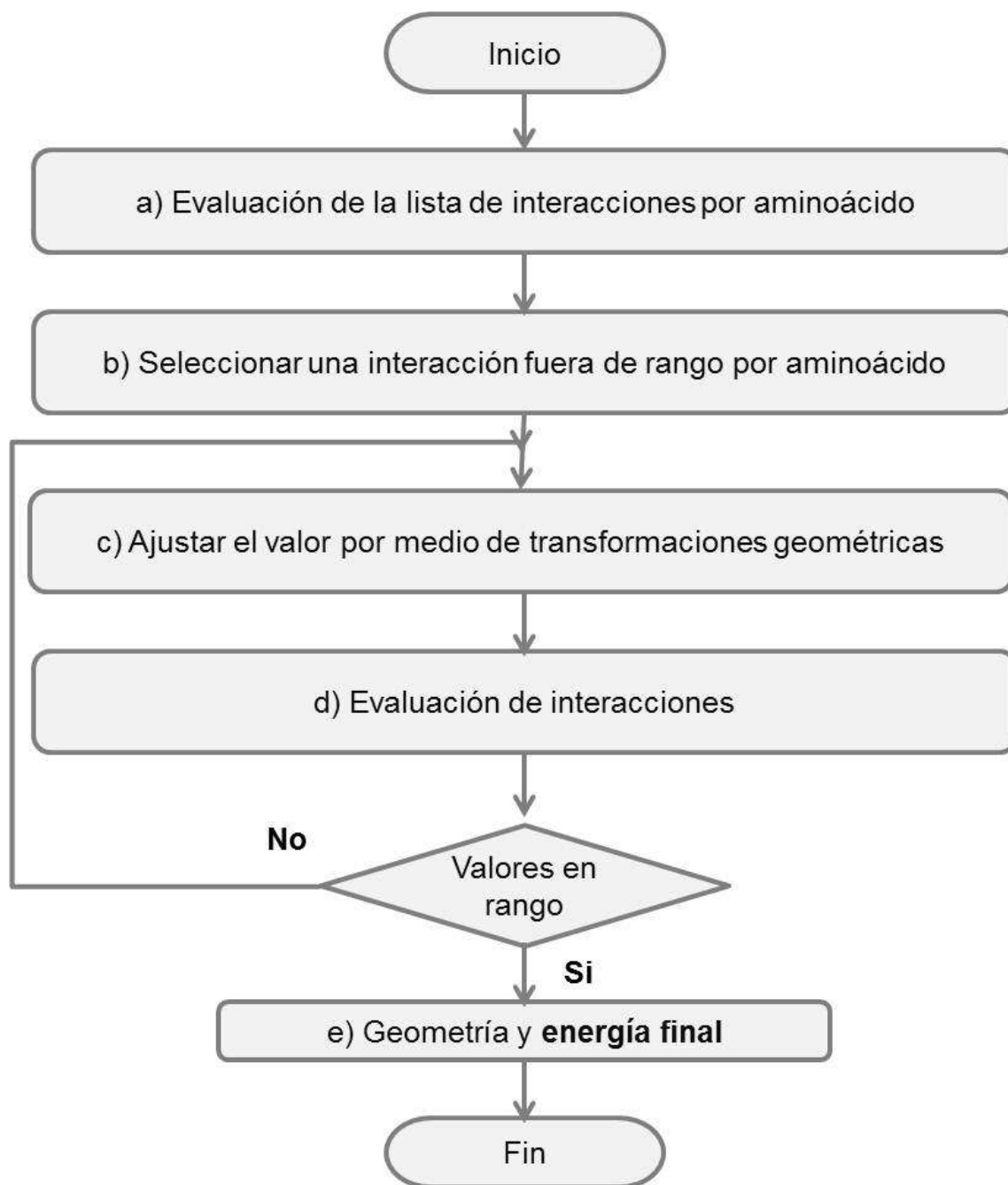
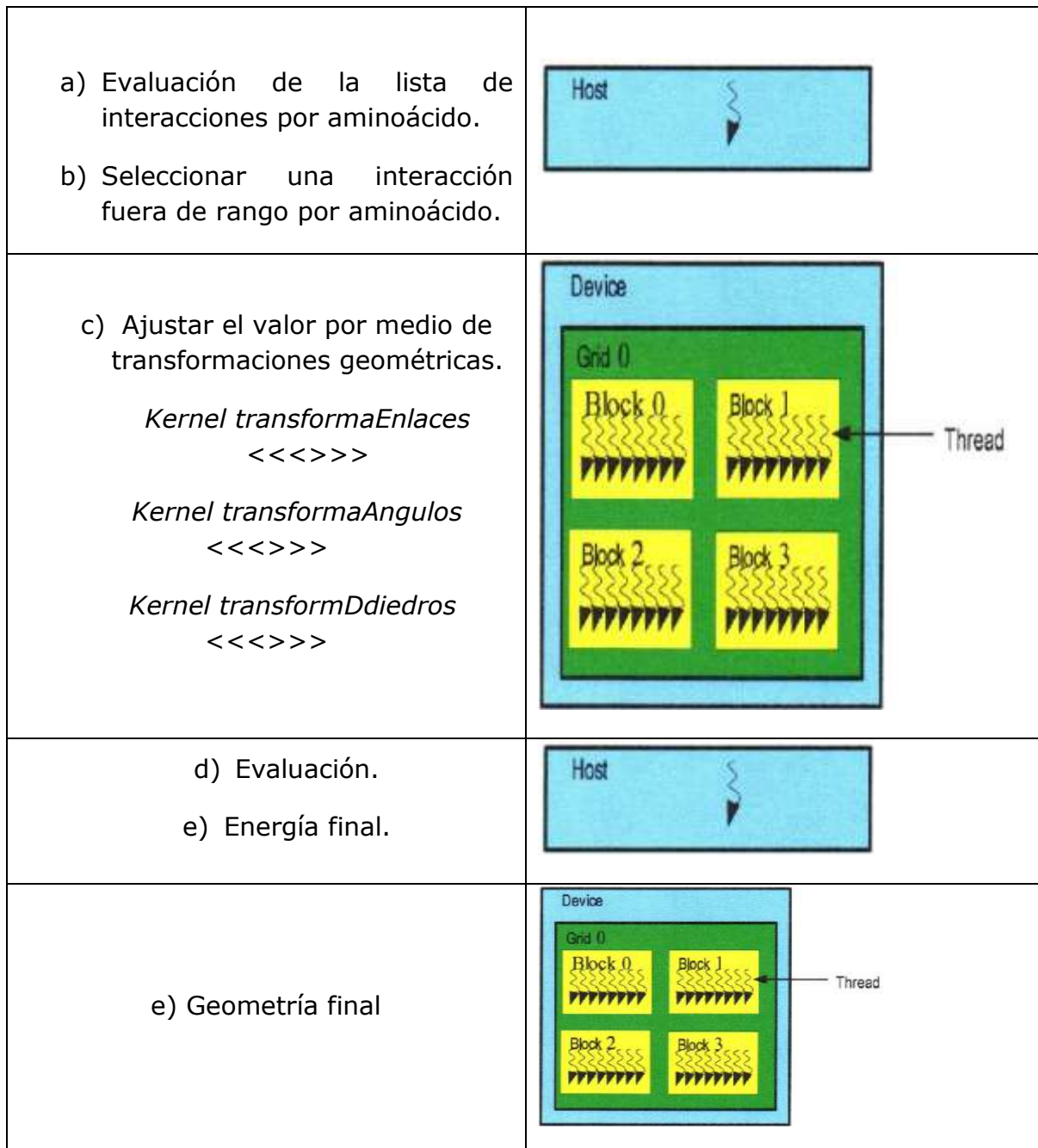


Figura 26: Algoritmo de minimización de energía

La forma en que se procesara este algoritmo esta representado con la Figura 26 y a continuación la forma en que serán procesados los datos.



Proceso 6

Finalmente tendremos una molécula, que esta dentro de los valores naturales como la de la FIGURA 27 y por lo tanto cumplirá con su función.

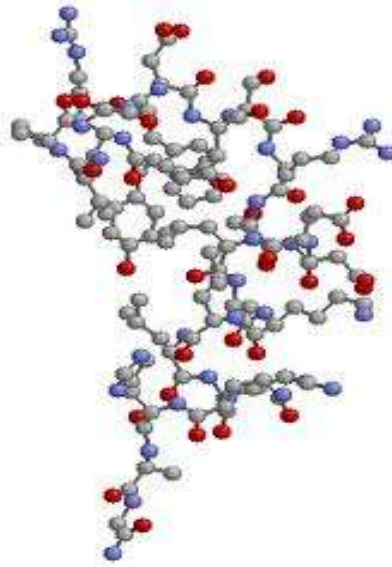


Figura 27: Molécula en equilibrio

Capítulo 5

Resultados

Como primer resultado se puede mencionar que se cumplió el objetivo del trabajo, dado que se pudo realizar la evaluación y minimización de la energía potencial, cumpliéndose con lo establecido en la hipótesis y haciendo uso de las unidades gráficas de procesamiento (GPU), por ende se obtuvo una mayor aceleración.

La afirmación en cuanto a la aceleración esta hecha con base a Hun Joo Myung [26], en el artículo donde se analiza la aceleración de simulaciones de dinámica molecular y en donde se demuestra que el tiempo que tarda una GPU es mucho menor en comparación a que tardaría en procesarlos la CPU, tal y como se muestra en la siguiente gráfica.

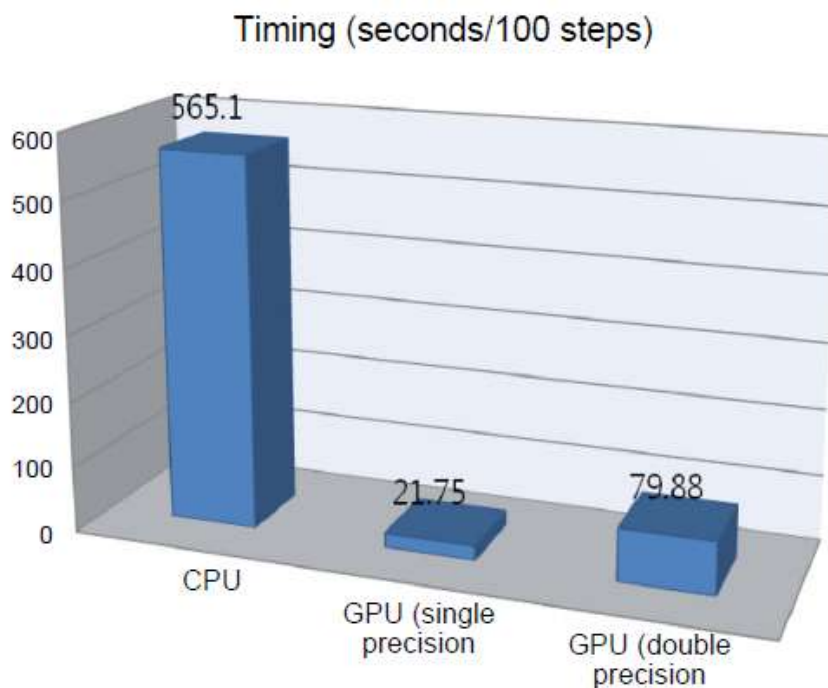


Figura 28: Resultados del rendimiento obtenido en 100 iteraciones realizadas para simulaciones de dinámica molecular de la CPU y GPU

Los resultados generados durante el proceso pueden ser analizados a nivel de datos por medio de los archivos generados en cada uno de los pasos de este proceso o bien por medio de gráficos, los cuales pueden representarse de forma muy simple como la Figura 29 o un poco más a detalle como en la Figura 30.



Figura 28: Representación simple de la molécula

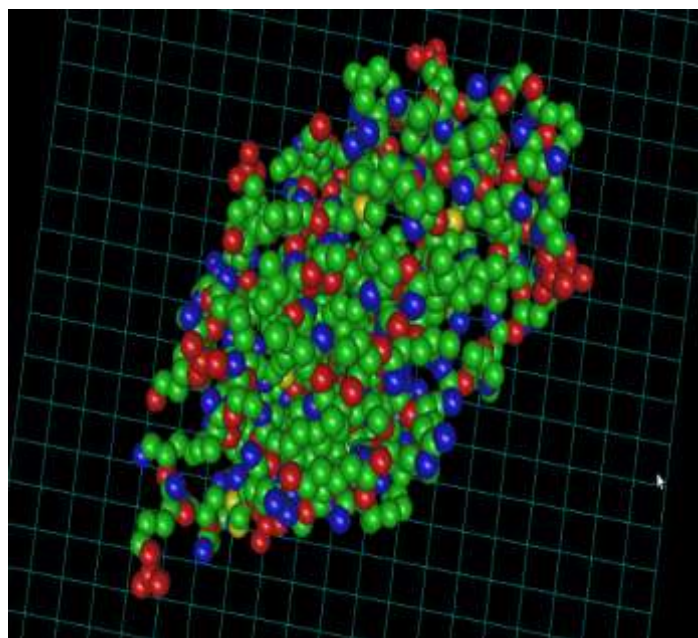


Figura 28: Representación de los átomos como esferas

En cuanto a los archivos de datos generados, una persona que conoce del área puede analizar por medio de estos la geometría de las posibles conformaciones.

Dado que para este trabajo se propuso un método y un algoritmo diferente para llevar a cabo la minimización, se observa que el algoritmo es eficiente en cuanto a la cantidad de modificaciones geométricas que se tienen que realizar y por tanto al tiempo que estas emplean al realizarse en paralelo, por lo que al no haber alguna aplicación que utilice la misma técnica solo se puede decir que se reportan mejorías al tiempo invertido en la optimización.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

- CUDA es una arquitectura que facilita y hace posible cómputo de propósito general y de alto desempeño en las unidades de procesamiento gráfico.
- La programación en CUDA puede resultar fácil de aprender, lo complejo es saber optimizar las operaciones que se realizan para obtener una verdadera aceleración, de lo contrario puede tardar más que lo que se tara un programa secuencial.
- Debido a que el fabricante de las GPU no divulga los detalles de la arquitectura, se complica el uso para aprovechar al máximo su potencia.
- Las simulaciones son de gran ayuda para estudiantes e investigadores bioquímicos, para realizar más rápido y eficientes sus investigaciones.
- Este tipo de aplicaciones se pueden utilizar como laboratorios virtuales, efectivos y de bajo costo.
- La evaluación y minimización de la energía potencial fue realizada con éxito empleando una técnica diferente para la minimización obteniendo buenos resultados en tiempos muy considerados.
- Los resultados gráficos que se obtuvieron para este trabajo se limitan a presentar la molécula por medio de diferentes representaciones y a utilizar el mouse y el teclado para ver la conformación completa de la molécula, pero este visualizador puede ser tan simple o tan complejo como se desee, pero se puede mostrar que hay una buena interoperabilidad que hace atractiva la aplicación, ya que por lo general ninguna incluye este módulo en las aplicaciones de dinámica molecular.

6.2 Trabajos futuros

- Incorporar un visualizador que nos permita visualizar la molécula y sus propiedades a detalle y con valores confiables.

- Incluir todos aquellos procesos que para un estudio más complejo de la molécula, como lo es el plegamiento.
- Algo que es muy complejo pero útil, como lo es el análisis conformacional dinámico, el cual se centra en el estudio de aquellas propiedades moleculares que no se pueden explicar en función de las diferencias de estabilidad entre conformeros estáticos y requieren la conformación en equilibrio, es decir se trata de una concepción **dinámica** de una molécula en movimiento.

Apéndice A

Requerimientos de CUDA

Los requerimientos necesarios para poder realizar cómputo paralelo en la arquitectura de NVIDIA CUDA son los siguientes:

1. Un procesador gráfico con arquitectura CUDA
2. El controlador de desarrollo de NVIDIA
3. El conjunto de herramientas de CUDA (CUDA Toolkit)
4. Un compilador de C estándar

A.1. Procesador grafico con arquitectura CUDA

En http://www.nvidia.com.mx/object/geforce_family_la.html se podrán encontrar las especificaciones de todas las unidades gráficas de procesamiento habilitadas para CUDA, la capacidad de procesamiento que tienen, el número de multiprocesadores y la cantidad de núcleos con los que cuentan.

A.1. El controlador de desarrollo de NVIDIA

Nvidia proporciona un controlador o driver de desarrollo mediante el cual son soportadas todas los GPUs que estén basados en la arquitectura CUDA; el driver puede obtenerse del sitio de CUDA

A.3. El conjunto de herramientas de CUDA (CUDA Toolkit)

Una vez que se tiene el hardware (GPU) y el driver de Nvidia, es posible ejecutar código compilado de CUDA C. Esto es, se pueden descargar y ejecutar aplicaciones que exploten la capacidad de paralelismo de la arquitectura CUDA. Sin embargo para poder desarrollar software en CUDA C es necesario software adicional, el cual puede obtenerse de manera gratuita en la página de CUDA.

Dentro de las herramientas que se proporcionan por parte de Nvidia encontramos:

Compilador (nvcc): Debido a que una parte del código que se desarrolla se ejecuta en el GPU, Nvidia proporciona un compilador que genera código ejecutable para el GPU. Este compilador se llama nvcc y está basado en gcc de GNU.

CUDA-GDB: Cuda-GDB es una herramienta de depuración basada en el GDB de GNU Visual Profiler; Es una interfaz gráfica para medir el rendimiento de una aplicación y encontrar posibles optimizaciones para aprovechar al máximo las capacidades del GPU.

Bibliotecas: CUDA proporciona una serie de bibliotecas que contienen funciones optimizadas para realizar tareas comunes.

- **CUBLAS.** Es una implementación en GPU de BLAS (Basic Linear Algebra Subprograms), que es una API para realizar operaciones de álgebra lineal tales como multiplicación de matrices y vectores.

- **CUFFT.** Es la biblioteca de CUDA para la transformada rápida de Fourier. Esta biblioteca cuenta con implementaciones de varios algoritmos para realizar la transformada rápida de Fourier aprovechando el paralelismo de los GPUs. La CUFFT puede ser ampliamente utilizada en aplicaciones de física o tratamiento de señales.

- **CUSPARSE.** La biblioteca CUSPARSE contiene un conjunto de subrutinas de álgebra lineal utilizadas para la manipulación de matrices dispersas diseñadas para ser llamadas desde C o C++.

- **CURAND.** Es la biblioteca de CUDA para el uso y generación de números aleatorios. Posee diferentes algoritmos para la generación de números aleatorios dependiendo de las propiedades estadísticas que se quiere que tengan éstos.

- **Thrust.** Ésta es una de las bibliotecas más recientes introducidas en el CUDA Toolkit y esta basada plantillas basada en la STL de C++.

A.4. Un compilador de C estándar

Como se mencionó anteriormente, Nvidia proporciona un compilador (nvcc) para generar código ejecutable en el GPU. Sin embargo se requiere un compilador que genere la parte que será ejecutada en el CPU. Los compiladores más soportados por Nvidia son gcc de GNU.

Apéndice B

Especificaciones técnicas del ambiente de desarrollo.

B.1 SISTEMA OPERATIVO

El sistema operativo en el que se desarrollará la aplicación es LINUX, ya que es un sistema operativo gratuito y de libre distribución, el cual tiene las siguientes características:

- Admite múltiples usuarios
- Ejecuta múltiples tareas
- Sistema fiable, sólido y potente
- Seguro
- Estable
- Soporte



B.2 API DE GRAFICACIÓN

OpenGL versión 4.10



B.3 ESPECIFICACIONES DE LA GPU UTILIZADA

CARACTERÍSTICA	ESPECIFICACIÓN
Device 0:	"GeForce GT 430"
CUDA Driver Version / Runtime Versio	4.0 / 4.0
CUDA Capability Major/Minor version number:	2.1
Total amount of global memory:	1023 MBytes (1072889856 bytes)
(2) Multiprocessors x (48) CUDA Cores/MP:	96 CUDA Cores
GPU Clock Speed:	1.40 GHz
Memory Clock rate:	800.00 Mhz
Memory Bus Width:	128-bit
L2 Cache Size:	131072 bytes
Max Texture Dimension Size (x,y,z)	1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers	1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block	: 32768

Warp size:	32
Maximum number of threads per block:	1024
Maximum sizes of each dimension of a block:	1024 x 1024 x 64
Maximum sizes of each dimension of a grid:	65535 x 65535 x 65535
Maximum memory pitch:	2147483647 bytes
Texture alignment:	512 bytes
Concurrent copy and execution:	Yes with 2 copy engine(s)
Run time limit on kernels:	Yes
Integrated GPU sharing Host Memory:	No
Support host page-locked memory mapping:	Yes
Concurrent kernel execution:	Yes
Alignment requirement for Surfaces:	Yes
Device has ECC support enabled:	No
Device is using TCC driver mode:	No
Device supports Unified Addressing (UVA):	Yes
Device PCI Bus ID / PCI location ID:	1 / 0

Referencias

- [1] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, *GPU computing*, Proc. IEEE 96 (2008) 879–899.
- [2] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, V. Volkov, *Parallel computing experiences with CUDA*, IEEE Micro 28 (2008) 13–27
- [3] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, *A survey of general-purpose computation on graphics hardware*, Comput. Graph. Forum 26 (2007) 80–113.
- [4] G. Giupponi, M. Harvey, G.D. Fabritiis, *The impact of accelerator processors for high-throughput molecular modeling and simulation*, Drug Discov. Today 13 (2008) 1052–1058.
- [5] J.C. Phillips, J.E. Stone, K. Schulten, *Adapting a message-driven parallel application to GPU-accelerated clusters*, in: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, Piscataway, NJ, USA, 2008.
- [6] J.A. Anderson, C.D. Lorenz, A. Travasset, *General purpose molecular dynamics simulations fully implemented on graphics processing units*, J. Chem. Phys. 227 (2008) 5342–5359.
- [7] J. Yang, Y. Wang, Y. Chen, *GPU accelerated molecular dynamics simulation of thermal conductivities*, J. Chem. Phys. 221 (2007) 799–804.
- [8] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, *Accelerating molecular modeling applications with graphics processors*, J. Comp. Chem. 28 (2007) 2618–2640.
- [9] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, V. Pande, *N-body simulation on GPUs*, in: SC06 Proceedings, IEEE Computer Society, 2006.
- [10] **NVIDIA Corp.**, *CUDA Architecture Introduction & Overview*, NVIDIA Cor., Santa Clara California. Versión 1.1. (2009) 3.
- [11] **Smit, B.; Frenkel, D.** *Understanding Molecular Simulation*; Academic Press: Orlando, 2001.
- [12] K.J. Bowers, et al., *Scalable Algorithms for molecular dynamics simulations*

on commodity clusters, in: Proc. ACM/IEEE Conf. on Supercomputing (SC06), Tampa, FL, 2006.

- [13] **D.E. Shaw, et al.**, **Anton: A special-purpose machine for molecular dynamics simulation**, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, California, 2007.
- [14] **R. Fine, G. Dimmler, C. Levinthal**, **FASTRUN: A special purpose hardwired computer for molecular simulation**, *Proteins 11* (1991) 242–253.
- [15] **T. Narumi, Y. Ohno, N. Okimoto, A. Suenaga, R. Yanai, M. Taiji**, *A high-speed special-purpose computer for molecular dynamics simulations: MDGRAPE-3*, in: NIC Workshop 2006, NIC Series, vol. 34, John von Neumann Institute for Computing, 2006, pp. 29–36.
- [16] **S. Toyoda**, *Development of MD engine: High-speed accelerator with parallel processor design for molecular dynamics simulations*, *Journal Computational Chemistry 20* (1999) 185–199.
- [17] **J. Sanders and E. Kandrot.**, *E. Cuda by Example. An Introduction to General-Purpose GPU Programming*, Addison-Wesley (2010) 2-161
- [18] **NVIDIA Corp.**, *CUDA C Programming Guide.*, NVIDIA Cor., Santa Clara California. Version 3.2. (2010), 2-25.
- [19] **D. B. Kirk and Wen mei.**, *Programming Massively Parallel Processors: A Handson Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, (2010).
- [20] **Rob Farber.**, *CUDA Application Design and Development*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, (2010).
- [21] **NVIDIA Corp.**, *CUDA. CURAND Library*, NVIDIA Cor., Santa Clara California. Version PG-05328-032_V01. (2010), 50
- [22] **P. A. Kollman, R. W. Dixon, W. D. Cornell, T. Fox, C. Chipot, and A. Pohorille**, *The Development/Application of a "Minimalist" Organic/Biomolecular Mechanic Forcefield Using a Combination of Ab Initio Calculations and Experimental Data*, in *Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications*, W. F. van Gunsteren and P. K. Weiner, Eds. Dordrecht, Netherlands: ESCOM, 1997, 83-96.
- [23] **J. MacKerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. ,** *All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins*, *J. Phys. Chem. B*, 102(18):, (1998), 3586- 3616.
- [24] **W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives**, *Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids*, *J. Am. Chem. Soc.*, 118(45), (1996), 11225-11236.

- [25] **C. Madden, P. Bohnenkamp, K. Kazerounian, and H. T. Ilies.** *Residue level three-dimensional workspace maps for conformational trajectory planning of proteins.* The International Journal of Robotics Research, (2009), 450–463.
- [26] Hun Joo Myung, Ryuji Sakamaki, Kwang Jin Oh, Tetsu Narumi, Kenji Yasuoka,†, and Sik Lee. *Accelerating molecular simulation using graphics processing unit*, Bull Korean Chem 31. (2010)
- [27] **Jaume Casabo I Gispert**, Estructura atómica y enlace químico, Editorial Reverté, S. A. (1996), 5-127
- [28] **Grossman, Stanley I.**, *Algebra lineal*. 6a. ed. México: Mc Graw-Hill, 2008.
- [29] **Wright, R. L.**, *OpenGL superbible: comprehensive tutorial and referente OpenGL Series.* (Edition 4. ed.). Addison-Wesley. (2007)
- [30] **Shreiner, D.**, *OpenGL programming guide: the official guide to learning OpenGL* (4 ed., Vol. 2003). Addison-Wesley. (2004).
- [31] <http://www.nvidia.com>
- [32] <http://gpgpu.org/about>
- [33] <http://folding.stanford.edu/English/HomePage>
- [34] <http://www.unitedboinc.com>
- [35] <http://www.ks.uiuc.edu/Research/namd>
- [36] <http://codeblue.umich.edu/hoomd-blue/about.html>
- [37] <http://download.acellera.com/pub/Acemd2010.pdf>
- [38] <http://www.rcsb.org/pdb>
- [39] <http://www.opengl.org>