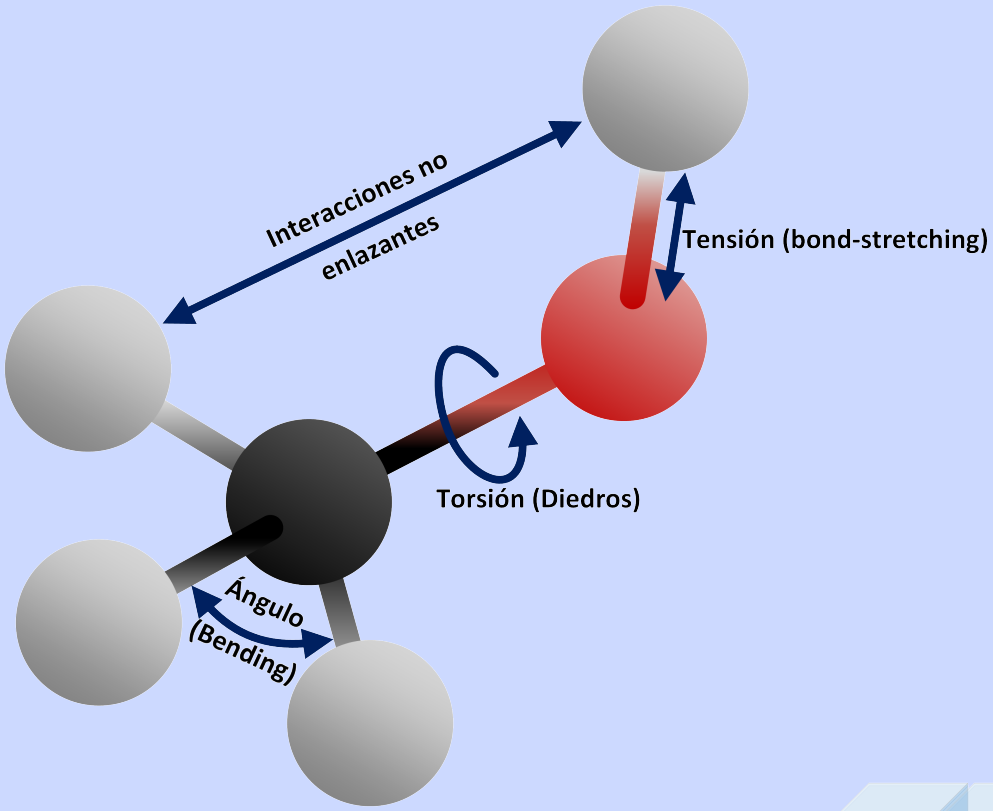
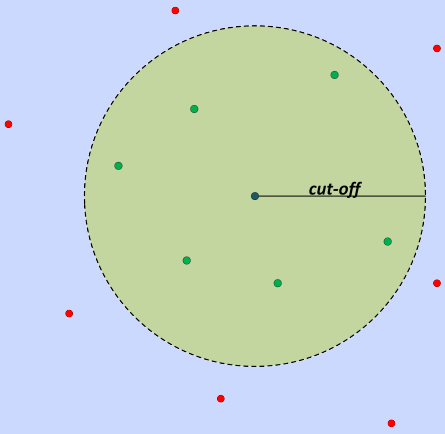
Universidad de la República

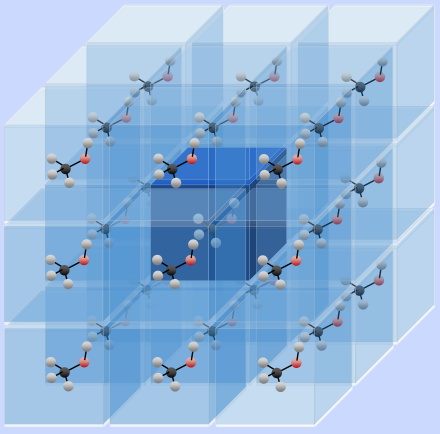
pedeciba

Aceleración de cálculos de dinámica molecular mediante el uso de GPUs





Yamandú González



Universidad de la República

pedeciba

Aceleración de cálculos de dinámica molecular mediante el uso de GPUs

Montevideo, Noviembre de 2014

Tesis de Maestría en Bioinformática

Presentada por: Yamandú González

Director de Tesis: Dra. Margot Paulino

Director de Tesis: Dr. Pablo Ezzatti

Resumen

La simulación de sistemas juega un rol importante en la investigación científica, y en este campo, la dinámica molecular ha surgido como una herramienta de gran relevancia por su capacidad de simular una gran variedad de sistemas. En muchos aspectos la dinámica molecular es similar a los experimentos reales; primero, se prepara una muestra, a continuación, se la equilibra termodinámicamente, y finalmente se realiza la medición. La aplicación de este tipo de herramientas para predecir el comportamiento de biomoléculas *in-silico* de una manera determinista no es la única ventaja de este método; ahorrar tiempo de trabajo en laboratorio, así como optimizar y/o ahorrar recursos en muchos procedimientos son de gran importancia en industrias vinculadas al descubrimiento de fármacos por ejemplo. Varias herramientas de software han sido desarrolladas para realizar simulaciones de dinámica molecular, algunas de las cuales son referencia en la comunidad científica, marcando así la evolución de esta disciplina en lo que al desarrollo de nuevas técnicas refiere. Pero, así cómo evolucionan estas herramientas también crece la demanda sobre ellas, en otras palabras existe una mayor demanda de potencia de cálculo debido al aumento del tamaño de los sistemas, así como el período de tiempo a simular.

Lo anterior configura un escenario propicio para la aplicación de soluciones vinculadas a la computación de alto rendimiento (HPC del inglés High Performance Computing). Una limitante para el uso de técnicas de HPC son los elevados costos económicos asociados a estas técnicas. Sin embargo en los últimos años, se han desarrollado nuevas alternativas económicas en lo que refiere a hardware de HPC, en particular, el uso de plataformas híbridas compuestas por procesadores multinúcleo en combinación con unidades de procesamiento gráfico (GPU del inglés Graphic Processing Unit) para hacer frente a problemas de propósito general se han hecho cada vez más populares.

Este trabajo propone una alternativa al uso de GPUs para dinámica molecular, mediante la aplicación directa de la tercera ley de Newton, tomando como punto de partida los fuentes del paquete NAMD [1]. Junto con el desarrollo propuesto se describe la puesta a punto de la distribución convencional de NAMD, mediante el ajuste de parámetros y pequeños cambios

en el código. Esta versión optimizada de NAMD representa el punto de referencia con respecto al cual el trabajo propuesto es comparado.

Los resultados obtenidos son alentadores, confirmando que la aplicación de la tercera ley de Newton en un entorno orientado a procesamiento paralelo puede ofrecer beneficios. En particular, con la propuesta se logra una mejora en el uso de la GPU, y dependiendo del caso, dicha mejora impacta en el rendimiento general de la solución.

Palabras claves: NAMD, GPU, Mecánica Clásica, Dinámica Molecular

Agradecimientos

Al Programa de Desarrollo de las Ciencias Básicas (PEDECIBA).

A la Comisión Sectorial de Investigación Científica (CSIC), UdelaR por la beca de finalización de maestría otorgada.

A mis tutores, Margot y Pablo, por su paciencia, comprensión y gran oficio en la labor académica.

A la responsable de despertar mi interés en esta maestría y que me llevó a ésta, mi gran compañera de todas las horas, por tu paciencia y tu eterna sonrisa, sencillamente gracias María.

Índice general

1. Introducción **1**

1.1. Objetivos 3

1.2. Estructura de la tesis 3

2. Dinámica molecular **5**

2.1. Mecánica molecular 5

2.2. Campos de fuerza 11

2.3. Condiciones de borde 12

2.4. Condiciones periódicas 13

2.5. Radios *cut-off* 14

2.5.1. Verlet-Lists 16

2.5.2. Cell-Lists 17

2.6. Particle Mesh Ewald 18

3. Procesadores Gráficos **21**

3.1. Aspectos generales 21

3.2. Manejo de memoria 25

3.2.1. Uso de la memoria global 25

3.2.2. Uso de la memoria compartida 26

4. GPUs en Dinámica Molecular **27**

4.1. Introducción 27

4.2. Implementación del radio *cut-off* 28

4.3. Aplicación de la tercera ley de Newton . . . 30
4.4. Uso de la memoria . . . 32
4.5. Otras consideraciones . . . 33
4.6. NAMD con GPUs . . . 34
4.6.1. Arquitectura de NAMD . . . 34
4.6.2. Cómputo de fuerzas en NAMD . . . 35
4.6.3. Integración de GPUs a NAMD . . . 36

5. Propuesta **39**
5.1. Variaciones en el uso de la memoria . . . 39
5.2. Aplicación de la tercera ley de Newton . . . 40
5.2.1. Cambios en la transferencia de memoria . . . 41
5.2.2. Implicancias en la relación comunicación-ejecución . . . 42
5.3. Evaluación y resultados . . . 42
5.3.1. Casos de prueba . . . 42
5.3.2. Entorno de pruebas . . . 44
5.3.3. Evaluación experimental de NAMD . . . 44
5.3.4. Evaluación de la propuesta . . . 47
5.3.5. Impacto de la propuesta en el desempeño general de la herramienta . . . 49
5.4. Evaluación bioinformática . . . 50
5.4.1. Temperatura del sistema . . . 50
5.4.2. APOA1 . . . 51
5.4.3. $\alpha 7$ . . . 59

6. Conclusiones y trabajo futuro **63**
6.1. Discusión . . . 63
6.2. Trabajo Futuro . . . 64

A. Representación en punto flotante **67**

Bibliografía **71**

Índice de figuras

2.1. Desfasajes entre cálculo de distancia y de velocidad en el método *Leap Frog.* 6
2.2. Enlace covalente entre dos átomos. 8
2.3. Ángulo entre dos enlaces covalentes. 8
2.4. Ángulo diedro, compuesto por dos semiplanos formado por dos tríos de átomos respectivamente. 9
2.5. Interacciones no enlazantes entre dos átomos. 10
2.6. Todas las fuerzas consideradas en el cálculo de campo de fuerza (*Force field*). 11
2.7. Condiciones periódicas, celda primaria (azul oscuro) e imágenes (celeste). . . 13
2.8. Esfera de corte de radio *cut-off.* 14
2.9. Potencial de van der Waals con uso de radio *cut-off.* 15
2.10. Potenciales de van der Waals con y sin uso de *Shifting function* [2]. 15
2.11. Potencial de Coulomb con y sin uso *Shifted potentials* [2]. 16
2.12. División del espacio en celdas de longitud r_c (Cell-Lists). 18
2.13. Distribuciones de carga en las sumas de Ewald: (a) de las cargas puntuales y el fondo compensante; (b) de la carga cancelante; (c) distribución de cargas resultante de la suma de Ewald [3]. 20

3.1. Jerarquía de memoria de CUDA. 25

4.1. Curvas de Hilbert, en dos (izquierda) y tres (derecha) dimensiones. 30
4.2. Estrategia de AMBER para evaluar fuerza entre pares [4]. 31
4.3. Descomposición espacial de NAMD. 35

5.1. Estrategia de NAMD para evaluar fuerza entre pares. 40

5.2. Estrategia de AMBER aplicada a NAMD para evaluar fuerza entre pares. . . 41
5.3. Pentámero $\alpha 7$ en membrana lipídica, imagen tomada del software VMD [5]. . 43
5.4. Diferentes versiones de NAMD tratadas en este trabajo. 45
5.5. RMSD de simulación completa con NAMD optimizado y propuesta sobre complejo APOA1. 51
5.6. RMSD de primer nanosegundo de simulación con NAMD optimizado y propuesta sobre complejo APOA1. 53
5.7. RMSD de simulación durante fase de calentamiento con NAMD optimizado y propuesta sobre complejo APOA1. 53
5.8. RMSD de simulación entre 1 y 4.5 nanosegundos con NAMD optimizado y propuesta sobre complejo APOA1. 54
5.9. Muestreo de la energía total del sistema en primer nanosegundo simulado con NAMD optimizado y propuesta sobre complejo APOA1. 55
5.10. Muestreo de la energía total del sistema durante fase de calentamiento con NAMD optimizado y propuesta sobre complejo APOA1. 56
5.11. Muestreo de la energía total del sistema entre 1 y 4.5 nanosegundos de simulación con NAMD optimizado y propuesta sobre complejo APOA1. 56
5.12. Muestreo de la energía total del sistema en el primer nanosegundo simulado con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD. 57
5.13. Muestreo de la energía total del sistema en fase de calentamiento simulado con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD. 57
5.14. Muestreo de la energía total del sistema entre 1 y 4.5 nanosegundos de simulación con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD. 58
5.15. Estado inicial de APOA1, para simulación con NAMD optimizado y propuesta 58
5.16. Estados finales de APOA1 para NAMD optimizado (izquierda) y propuesta (derecha). 59

5.17. RMSD de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$. 60
5.18. Muestreo de la energía total del sistema de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$. 61
5.19. Muestreo de la energía total del sistema de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$ tomando como referencia el RMSD. . 61
5.20. Estados inicial (izquierda) y final (derecha) de $\alpha 7$, para simulación con NAMD optimizado. 62

Índice de tablas

3.1. Características de los diferentes tipos de memoria de las GPUs. 23

5.1. Rendimiento de NAMD en relación a los registros usados en GPU por cada hilo. 46

5.2. Evaluación de performance en GPU, NAMD versus NAMD optimizado. . . . 47

5.3. Evaluación de performance en GPU, NAMD optimizado versus propuesta. . 47

5.4. Performance total del uso de GPU, NAMD optimizado versus propuesta. . . 48

5.5. Performance total (GPU+*host*), NAMD optimizado versus propuesta. 49

Capítulo 1

Introducción

En términos simples la Bioinformática es la aplicación de la Informática a la gestión y análisis de datos biológicos. La misma comprende a diferentes campos de estudios que incluyen Informática, Matemática aplicada, Estadística, Ciencias de la computación, Inteligencia Artificial, Química y Bioquímica con el fin de solucionar problemas, analizar datos, o simular sistemas o mecanismos, todos ellos de índole biológica, y usualmente (pero no de forma exclusiva) a nivel atómico y molecular. El foco principal de estas técnicas se encuentra en la utilización de recursos computacionales para solucionar o investigar problemas sobre escalas de tal magnitud que sobrepasan el discernimiento humano.

Algunos de los principales esfuerzos de investigación en estos campos incluyen el alineamiento de secuencias, predicción de genes (*motif finding*), montaje del genoma, alineamiento estructural de proteínas, predicción de estructura de proteínas, predicción de la expresión génica, interacciones proteína-proteína, etc.

Una constante en proyectos de Bioinformática es el uso de herramientas matemáticas para extraer información útil de datos generados por técnicas biológicas de alta productividad, como por ejemplo la llamada secuenciación masiva. En particular, el montaje o ensamblado de secuencias genómicas de alta calidad desde fragmentos obtenidos tras la secuenciación del ADN a gran escala es un área de alto interés.

Las problemáticas antes mencionadas, gracias al avance en las técnicas de extracción de información, se han vuelto cada vez más demandantes en recursos informáticos. Esto ha

provocado que un simple computador de escritorio a menudo no sea capaz de procesar esta información en un tiempo razonable, por lo que se ha recurrido a otras técnicas complementarias para acelerar el procesamiento de esta información. Dichas técnicas generalmente englobadas en el área de computación de alto desempeño (HPC, del inglés High Performance Computing) usualmente implican el uso de hardware específicamente concebidos para este propósito como clusters de computadores, FPGAs (del inglés Field Programmable Gate Array), procesadores construidos específicamente para Bioinformática y GPUs (del inglés Graphic Processing Unit) entre otros.

Las GPUs fueron originalmente diseñadas para aligerar la carga de trabajo del procesador central (CPU) en aplicaciones como los videojuegos y/o aplicaciones 3D interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos. Algunas de las características principales de las GPUs son el bajo precio en relación a su potencia de cálculo, la gran capacidad de procesamiento y la optimización para cálculos en aritmética de punto flotante. Estas características las posicionan como un hardware atractivo para ser explotado en aplicaciones no relacionadas directamente con los gráficos por computadora, como lo son, en el ámbito científico, la simulación del comportamiento de sistemas biomoleculares.

Diversos esfuerzos han incluido el uso de GPUs para acelerar aplicaciones en el campo de la Bioinformática. Estas aplicaciones van desde el alineamiento de secuencias [6, 7, 8, 9], al acoplamiento molecular o *docking* [10, 11, 12], plegamiento molecular o *folding* [13, 14] y dinámica molecular [15, 16, 17].

Actualmente diversas áreas dentro de la Bioinformática evidencian una fuerte dependencia tanto de las técnicas como del hardware considerado de HPC, en particular la Bioinformática Estructural, estando referida en términos generales esta última al análisis de estructuras biomoleculares y sus interacciones con otras estructuras. En particular, cuando los estudios estructurales se realizan en función del tiempo, los mismos se refieren al uso de campos de fuerza y la dinámica molecular.

Existen programas capaces de llevar a cabo simulaciones de dinámica molecular que son ampliamente aceptados en la comunidad, entre los más reconocidos: NAMD [1], AMBER [18] y GROMACS [19].

NAMD integra varios aspectos de HPC, entre estos incluye el uso de GPUs para realizar parte del cálculo implicado en una simulación. Como se verá en este trabajo, el cálculo de las fuerzas no enlazantes entre átomos (calculados de a pares), en cada paso de una simulación de dinámica molecular, implica un costo computacional de orden cuadrático con respecto al número de átomos del sistema. En NAMD, así como en otras soluciones de dinámica molecular, las GPUs son principalmente usadas para resolver dicho cálculo en forma eficiente.

1.1. Objetivos

Este trabajo propone estudiar la solución ya existente de NAMD que integra el uso de GPUs para acelerar cálculos de dinámica molecular en simulaciones de complejos proteicos y diseñar e implementar una alternativa. Este objetivo general incluye los siguientes objetivos específicos:

i) Revisar el estado del arte en cuanto a uso de GPUs para la aceleración de dinámica molecular.

ii) Detectar aspectos no considerados o mejoras posibles en las soluciones existentes en la comunidad que utilizan GPUs, tomando como referencia el software de uso general NAMD.

iii) Desarrollar al menos un prototipo, partiendo de la distribución original de NAMD, que aborde los aspectos y mejoras que surjan del punto anterior.

iv) Aplicar el software desarrollado a modelos de complejos proteína-ligando-membrana, utilizando como sujetos de pruebas a receptores nicotínicos.

1.2. Estructura de la tesis

En el Capítulo 2, se hace un repaso de los conceptos básicos de la dinámica molecular, revisando conceptos de mecánica newtoniana hasta estrategias de resolución mediante métodos numéricos.

El Capítulo 3 ofrece una introducción al uso de las GPUs para la resolución de problemas de propósito general. Se describe la arquitectura de las GPUs modernas y se profundiza en las características de CUDA [20].

En el Capítulo 4 se revisa el estado del arte en el uso de GPUs para acelerar herramientas de dinámica molecular, discutiendo los principales aspectos abordados y las diferentes soluciones propuestas. Sobre el final de este capítulo se describe en profundidad el uso de GPUs en NAMD, explicando las principales características de dicha solución.

En base a las ideas recogidas en los trabajos abordados en el Capítulo 4, en el Capítulo 5 se propone una alternativa de uso de GPUs para acelerar la simulación de dinámica molecular, implementada sobre NAMD [1]. A la vez, se propone abordar la versión original de NAMD y afinarla lo mejor posible sin modificar su lógica de ejecución, para así tener un sujeto de comparación de exigencia. Luego, se evalúa la solución propuesta con respecto a la versión optimizada de NAMD, dando lugar a la discusión de los resultados experimentales.

Finalmente, en el Capítulo 6, se presentan las conclusiones obtenidas y las líneas de trabajo a futuro identificadas durante el desarrollo de esta tesis.

Capítulo 2

Dinámica molecular

La Dinámica Molecular (MD, del inglés Molecular Dynamics) es un método de simulación que permite estimar el comportamiento de uno o varios complejos moleculares a lo largo del tiempo. En términos simples, las técnicas de MD se concentran en calcular las interacciones que rigen a dichos complejos para luego, con los valores obtenidos, guiar la transición al estado inmediato en cada paso de la simulación.

2.1. Mecánica molecular

La aplicación directa de la mecánica clásica o newtoniana recibe el nombre de mecánica molecular, por lo que, los complejos moleculares son modelados tomando a los átomos (núcleos y electrones en conjunto) como puntos en el espacio con propiedades físicas y fisicoquímicas asociadas, como pueden ser masa, carga, etc. Las propiedades también incluyen las coordenadas espaciales que dichos átomos tendrán, en el espacio cartesiano por ejemplo, y eventualmente podrán o no tener velocidades asignadas al inicio de una simulación, esto último relacionado intrínsecamente a la temperatura del sistema. Otros métodos, como los *ab initio* (basados en la mecánica cuántica), consideran a todos los electrones de la "esfera" electrónica que rodea al núcleo del átomo, implicando cálculos y consecuentemente métodos numéricos más complejos y costosos computacionalmente que, si bien aseguran una mayor convergencia al resultado, su aplicación se restringe a sistemas de dimensiones acotadas.

Como se mencionó anteriormente, las bases físicas de la MD son la aplicación de las

leyes de Newton, implicando principalmente la integración de la segunda ley de Newton (Ecuación 2.1), mediante un método numérico, comúnmente llamado integrador, para generar sucesivas configuraciones de un sistema. Notar que estos algoritmos son responsables de llevar el hilo principal de la simulación.

$$\frac{d^2x_i}{dt^2} = \frac{Fx_i}{m} \quad ó \quad \vec{F} = m.\vec{a} \tag{2.1}$$

A continuación, se resumen algunos de los algoritmos de integración más conocidos.

1. *Leap Frog*: Método que ofrece como ventaja el cálculo explícito de la velocidad (en otros caso ésta se calcula en función de la posición solamente) [21]. Se usa explícitamente la velocidad para el cálculo de la posición y la aceleración para el cálculo de la velocidad como se muestra a continuación:

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t+\tfrac{1}{2}\Delta t)\Delta t \\ v(t+\tfrac{1}{2}\Delta t) &= v(t-\tfrac{1}{2}\Delta t) + a(t)\Delta t \end{aligned} \tag{2.2}$$

   Si bien el método *Leap Frog* calcula explícitamente las velocidades, lo cual es una ventaja, estas son desfasadas con respecto a las posiciones, de ahí su nombre (que en español significa salto de rana) lo cual representa una desventaja. Por lo tanto, las posiciones se calculan en intervalos enteros y las velocidades en el medio de intervalos como muestra la Figura 2.1.

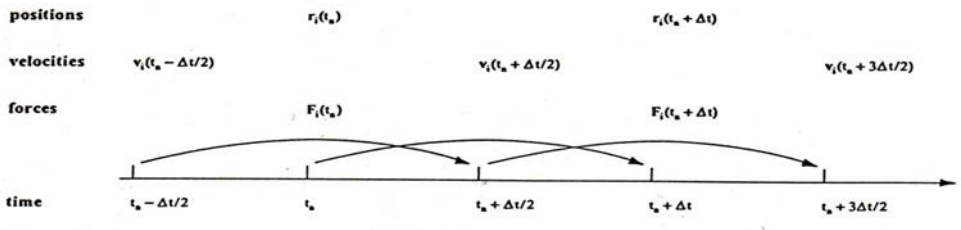


Figura 2.1: Desfasajes entre cálculo de distancia y de velocidad en el método *Leap Frog*.

2. *Velocity Verlet*: Similar a *Leap Frog* pero con la ventaja que calcula posición y velocidad en fase [22]:

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t)\Delta t + \tfrac{1}{2}a(t)\Delta t^2 \\ v(t+\Delta t) &= v(t) + \tfrac{1}{2}\big(a(t) + a(t+\Delta t)\big)\Delta t \end{aligned} \tag{2.3}$$

3. *Beeman*: Similar a *Velocity Verlet* pero ofrece mayor exactitud (más costoso computacionalmente) [23]:

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t)\Delta t + \tfrac{1}{6}\big(4a(t) - a(t-\Delta t)\big)\Delta t^2 \\ v(t+\Delta t) &= v(t) + \tfrac{1}{6}\big(2a(t+\Delta t) + 5a(t) - a(t-\Delta t)\big)\Delta t \end{aligned} \tag{2.4}$$

Si bien los integradores difieren entre sí en algún término, todos comparten el mismo mecanismo; obtener $v(t+\Delta t)$ y $r(t+\Delta t)$ a partir de $v(t)$ y $r(t)$.

Aunque la segunda ley de Newton, a través de un método numérico o integrador, brinda el mecanismo que sustenta las MDs, los átomos que componen al sistema mediante sus propiedades y relaciones con otros átomos definen a posteriori los valores de $v(t)$ y $r(t)$ que la simulación irá tomando. En otras palabras, la energía potencial del sistema será la forma de evaluar como él (molécula y entorno) se comporta en respuesta a la transición cuyo origen radica en el estado anterior. Por definición la fuerza es el valor negativo del gradiente de la energía potencial, quedando de esta manera, integrador y estado del sistema relacionados según la Ecuación 2.5.

$$\overrightarrow{F}(\overrightarrow{r}) = -\nabla U(\overrightarrow{r}) \tag{2.5}$$

La energía potencial del sistema será la suma de las energías potenciales desarrolladas por términos que cubren las interacciones posibles dentro de dicho sistema:

1. Tensión de los enlaces covalentes, se presenta en forma gráfica en la Figura 2.2.

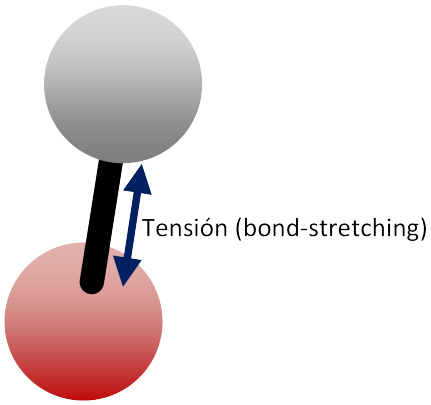


Figura 2.2: Enlace covalente entre dos átomos.

La energía potencial en dichos enlaces será evaluada acorde a la ley de Hooke (energía potencial elástica). Ponderando como distancia la diferencia entre las posiciones de equilibrio del par de átomos a evaluar y las posiciones actuales de dicho par, ver Ecuación 2.6.

$$E_{enlaces} = \sum_{enlaces} k_r(r - r_0)^2 \tag{2.6}$$

2. Ángulos entre dos enlaces covalentes, ver Figura 2.3.

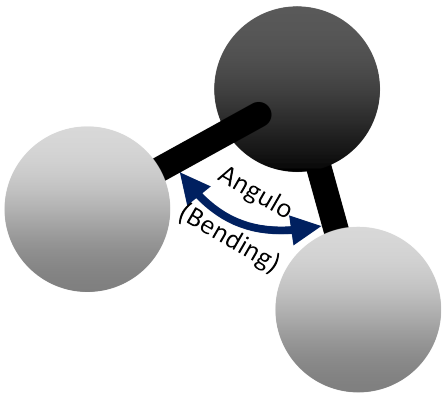


Figura 2.3: Ángulo entre dos enlaces covalentes.

Al igual que en el punto anterior, se utilizará la ley de Hooke para evaluar la tensión generada por la flexión del ángulo formado por tres átomos unidos entre ellos por dos

enlaces covalentes. En este caso se usará la diferencia entre el ángulo de equilibrio con respecto al actual, ver Ecuación 2.7.

$$E_{angulos} = \sum_{angulos} k_0(\theta - \theta_0)^2 \tag{2.7}$$

3. Ángulos diedros, representado en la Figura 2.4.

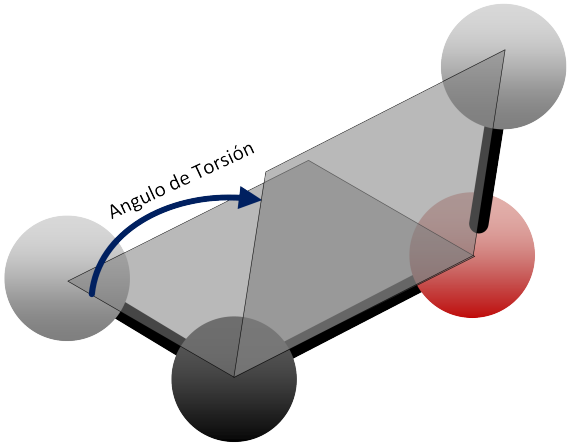


Figura 2.4: Ángulo diedro, compuesto por dos semiplanos formado por dos tríos de átomos respectivamente.

Este término cubre los ángulos formados por dos semiplanos, los cuáles se desprenden de una cadena formada por cuatro átomos y tres enlaces que se alternan entre ellos. Surgen de la necesidad de representar restricciones en la rotación de los enlaces y átomos de los extremos con respecto al enlace intermedio. El potencial de torsión será representado mediante una función de naturaleza trigonométrica, ver Ecuación 2.8, siendo n la periodicidad, $\frac{V_n}{2}$ la barrera torsional y γ la fase [24].

$$E_{diedros} = \sum_{diedros} \sum_{n} \frac{V_n}{2}\left[1 + \cos(n\theta - \gamma)\right] \tag{2.8}$$

4. Interacciones no enlazantes, ver Figura 2.5.

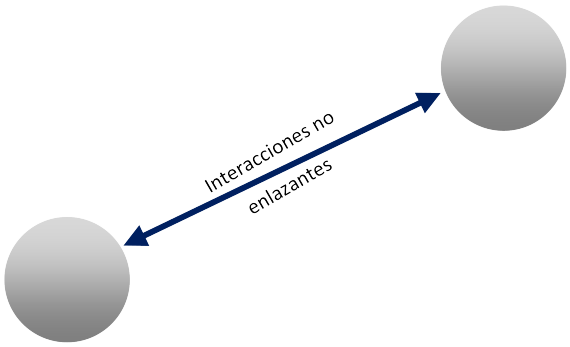


Figura 2.5: Interacciones no enlazantes entre dos átomos.

Los términos anteriores cubren el potencial generado entre átomos unidos de forma covalente, ya sea entre ellos o a través de un tercer átomo. Sin embargo es muy importante tomar en cuenta también el potencial generado por la interacción entre dos átomos que no están considerados en los puntos anteriores. Este tipo de interacciones, de carácter no enlazante son de gran importancia y por lo tanto deben ser tomados en cuenta, a grandes rasgos se distinguen dos tipos de interacciones: fuerzas de van der Waals e interacciones de tipo electrostáticas (ley de Coulomb), cuya suma representa el potencial de estas interacciones (ver Ecuación 2.9):

$$E_{no-enlazante} = E_{vdW} + E_{Coulomb} \tag{2.9}$$

a) Fuerzas de van der Waals: Esto tipo de fuerzas son de corto alcance y toma gran importancia cuando las esferas electrónicas de dos átomos cualesquiera están muy cercanas, ya que una "colisión" de este tipo no es viable en términos físicos. Este término puede ser representado por distintas aproximaciones, siendo una de las más difundidas el potencial de Lennard Jones, ver Ecuación 2.10.

$$E_{vdW} = \sum_{i} \sum_{j>i} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \tag{2.10}$$

b) Interacciones electrostáticas: Este término comprende las interacciones de largo alcance, mediante la aplicación directa de la ley de Coulomb (Ecuación 2.11).

$$E_{Coulomb} = \sum_{i} \sum_{j>i} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \tag{2.11}$$

Conocidos los términos se establece que la energía potencial total del sistema está dada por la suma de los términos antes descriptos según la Ecuación 2.12 (representado gráficamente en la Figura 2.6).

$$E_{total} = E_{enlaces} + E_{angulos} + E_{diedros} + E_{no-enlazante} \tag{2.12}$$

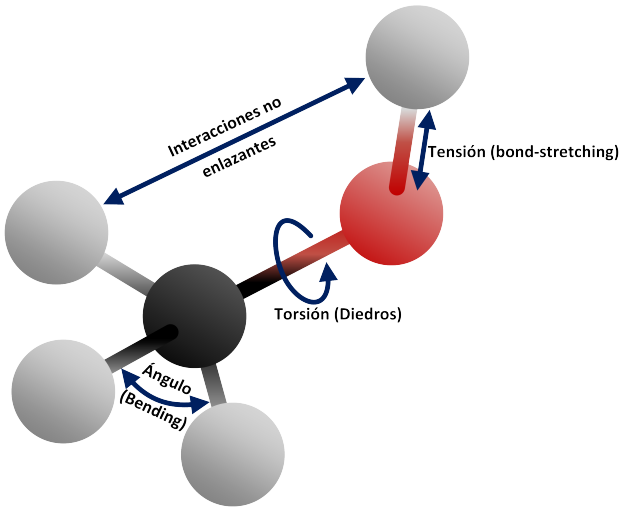


Figura 2.6: Todas las fuerzas consideradas en el cálculo de campo de fuerza (*Force field*).

2.2. Campos de fuerza

Los términos que componen la ecuación de energía potencial no necesariamente deben ser modelados por las ecuaciones definidas anteriormente, estas simplemente esbozan la formulación general, al igual que las constantes usadas, que también pueden variar. El conjunto de constantes, ecuaciones y valores usados para representar a los átomos según su naturaleza se definen como Campo de Fuerza (del inglés *Force field*). Los datos que componen a un campo de fuerza son definidos/ajustados buscando la representación más exacta posible de la realidad. Dentro del espectro de Campos de Fuerza disponibles se destacan CHARMM [25], AMBER [26], GROMOS [27]. Cabe mencionar que, dependiendo del campo de fuerza,

pueden aparecer términos adicionales o sustitutivos, que en algunos casos pueden tener como objetivo corregir alguna desviación de la realidad. Por ejemplo los diedros impropios, cuyo propósito es restringir el ángulo implicado en dicho diedro para que no tome valores que en términos químicos nunca suceden. También se puede distinguir a los llamados términos cruzados, que tienen la finalidad de establecer una dependencia entre un término y otro, por ejemplo la longitud de un enlace covalente en relación a un ángulo determinado.

Los diferentes campos de fuerzas, no solo varían en sus términos o constantes, también pueden variar en cómo estos modelan a ciertas moléculas, por ejemplo H_2O (agua), como tratan a los enlaces de hidrógeno o incluso en como representan al entorno en el cuál se lleva a cabo la simulación. En tiempos en que el poder de cómputo disponible era menor, fue de vital importancia reducir la cantidad de cálculos implicados en las simulaciones, por lo cual modelar por ejemplo una molécula de agua como un único "átomo" con propiedades que emularan correctamente su comportamiento era de gran ayuda.

Otro aspecto en el que los campos de fuerzas pueden variar son las opciones que se ofrecen para modelar el medio en el que está el complejo molecular a simular, es decir, el solvente. Este detalle establece un criterio interesante para categorizar en los campos de fuerzas entre aquellos que soportan solvente explícito y aquellos que soportan solvente implícito. Solvente explícito es aquél medio que es modelado tomando en cuenta todas las moleć ulas que lo componen, por ejemplo, moléculas de H_2O con moléculas de $NaCl$ integradas (sal) formando así un medio salino. Por otra parte el solvente implícito es aquel cuyo efecto es modelado usando una expresión matemática empírica, o sea, en lugar de representar las moléculas que componen al solvente, solo se modela el efecto de dicho solvente sobre el complejo mediante dicho término, disminuyendo sensiblemente el volumen de cálculo.

2.3. Condiciones de borde

En muchas simulaciones, probablemente la mayoría, es necesario trabajar con un ambiente en el cuál los límites del mismo no están claramente definidos, en otras palabras, determinar el límite de un medio salino (solvente) que es varios ordenes de magnitud más grande que el complejo a simular puede no ser trivial. Incluso, aunque el límite este clara-

mente definido, el número total de partículas que componen al sistema podría ser de una magnitud cuyo resultado podría ser una simulación computacionalmente muy costosa o directamente inviable. Una de las alternativas para encarar este problema es el uso de condiciones periódicas, este trabajo hace foco en esta técnica.

2.4. Condiciones periódicas

PBC (del inglés Periodic Boundary Conditions) posibilita simulaciones con un número de átomos bajo pero el efecto es similar a si se encontraran rodeado de un medio cualquiera (solvente). El sistema se restringe típicamente a un cubo, dejando en éste el complejo a estudiar y parte del medio que lo contiene. Dicho cubo es replicado en todas las direcciones (imágenes) de forma de emular una red infinita, bajo la suposición que el movimiento de las partículas del cubo concéntrico es idéntico al de las partículas contenidas en las imágenes de éste, y así simular un medio continuo. Por lo tanto, dichas imágenes (copias) que rodean a la celda original contienen los mismos átomos que esta última y, durante una simulación, cada uno de los átomos de los cubos adyacentes se mueve de la misma forma que los átomos del cubo concéntrico. Si un átomo de cualquier cubo (centro o imagen) abandona éste por una de sus caras, el mismo ingresará por la cara opuesta con la misma velocidad y dirección, por lo que no existen superficies limitantes del sistema [28]. Es importante notar que, si bien se manejó el cubo como la forma que contiene al sistema existen otras opciones que consideran formas como dodecaedros por ejemplo.

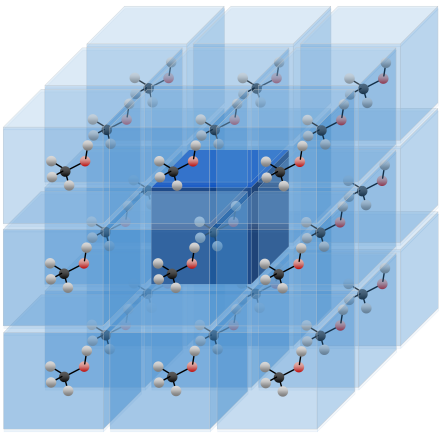


Figura 2.7: Condiciones periódicas, celda primaria (azul oscuro) e imágenes (celeste).

A pesar que se está modelando un sistema periódico a partir de varias réplicas, sólo es necesario almacenar los datos de los átomos de la celda primaria, ya que las coordenadas de las imágenes pueden deducirse fácilmente a partir de las dimensiones de la caja (cubo).

2.5. Radios *cut-off*

Uno de los aspectos a tener en cuenta al momento de preparar una simulación de dinámica molecular es el manejo de las fuerzas no enlazantes. Cualquier átomo en un complejo biomolecular dado presentará un número de enlaces covalentes menor a 10, por lo que, la cantidad de cómputo requerida para los términos enlazantes (enlace, ángulos y diedros) de la ecuación de potencial será $O(N)$ siendo N la cantidad de átomos presentes en el sistema. En cambio, un átomo cualquiera evidencia interacciones no enlazantes (sobre todo del tipo electrostáticas) con todos los átomos del sistema, por lo que, el costo computacional del cálculo de los términos no enlazantes alcanza $O(N^2)$. Una estrategia para reducir la cantidad de interacciones a evaluar, consiste en establecer, para cada átomo, una esfera de corte de radio *cut-off* que define hasta que distancia se toman en cuentas las interacciones no enlazantes. Aquellos átomos comprendidos en dicha esfera serán tomados en cuenta para calcular las interacciones, los átomos no comprendidos serán ignorados como se ilustra en la Figura 2.8.

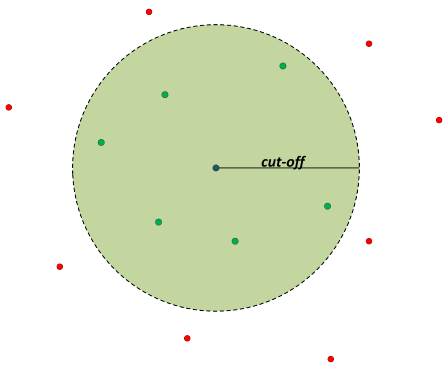


Figura 2.8: Esfera de corte de radio *cut-off*.

Dicha esfera será definida acorde a un radio de corte, o radio *cut-off*, valor que debería ser elegido cuidadosamente, en función de lo que el usuario defina como interacciones despreciables, en otras palabras un radio pequeño podría obviar interacciones importantes, un radio exagerado podría agregar interacciones de poco aporte en perjuicio de un alto costo

computacional. La elección del *cut-off*no solo se regirá por el criterio del usuario sino también por la naturaleza de la interacción, al tener las fuerzas de van der Waals un rango de alcance significativamente menor que las fuerzas electrostáticas dos radios diferentes deben ser definidos.

El uso de radios *cut-off* también tiene sus limitantes, el corte abrupto producido por la esfera de cortes puede conducir a "saltos" en la función de energía. En la función de potencial de van der Waals en la Figura 2.9 se puede observar que la curva es cortada abruptamente en $x =$*cut-off*, por lo que un infinitésimo más allá de x el potencial salta abruptamente de cero.

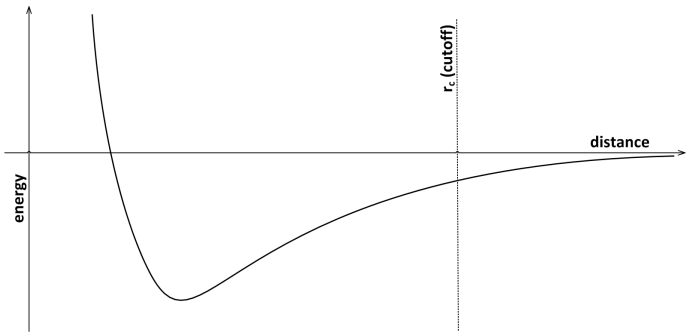


Figura 2.9: Potencial de van der Waals con uso de radio *cut-off*.

Existen estrategias para corregir estas discontinuidades, por ejemplo el uso de *Switching functions* introduce una alteración en la curva de manera que la misma converja a cero en $x =$*cut-off* de forma continua, ver Figura 2.10.

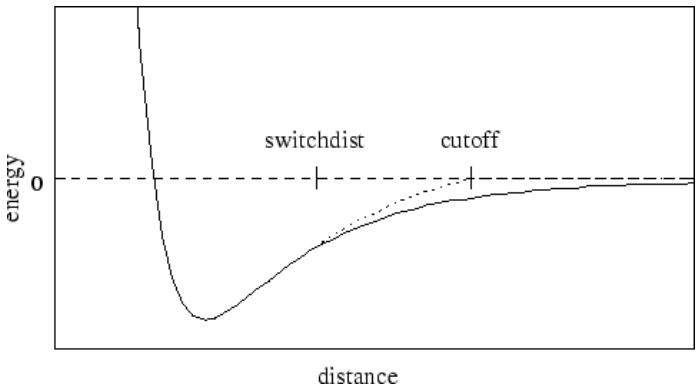


Figura 2.10: Potenciales de van der Waals con y sin uso de *Shifting function* [2].

Otra estrategia (Figura 2.11), consiste en el uso de *Shifted potentials*, la cual en lugar de

alterar la pendiente de la curva desplaza la misma en torno al eje y. Lo anterior corresponde a la reformulación de la ecuación de potencial con la expresión $U'(x) = U(x) - U(\textit{cut-off})$.

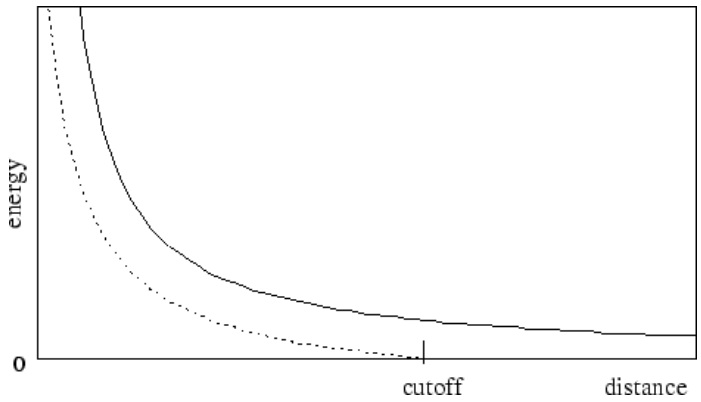


Figura 2.11: Potencial de Coulomb con y sin uso *Shifted potentials* [2].

El uso de radios *cut-off* implica otros cuidados, es necesario el uso de estructura de datos adicionales para representar las esferas de influencia de un átomo. No es óptimo establecer a cada paso de la simulación los átomos pertenecientes a todas y cada una de las N esferas de influencia en el sistema siendo N el número de átomos. Aunque comúnmente un *cut-off* de $8 - 10\mathring{A}$ debería ser suficiente, aplicar esto implica un buen entendimiento de MD y del sistema biomolecular a simular, no usar parámetros correctamente ajustados respecto al período de actualización de las listas puede introducir errores numéricos en la simulación. A continuación se explicarán dos tipos de enfoques ampliamente aplicados para afrontar esta problemática.

2.5.1. Verlet-Lists

Este enfoque plantea mantener una lista de vecinos por cada partícula del sistema, todas las partículas vecinas en un radio $R > r_c$ (siendo r_c el radio *cut-off*) serán incluidas y durante k pasos de la simulación se usará la misma lista hasta que ésta deba ser actualizada en el paso k [29]. Por lo tanto, el problema se centra en cuál es el radio R adecuado, durante la simulación las partículas están en constante movimiento, por lo tanto es altamente posible un constante ingreso y egreso de partículas en las diferentes esferas. Por esto, si d es la mayor distancia que un átomo se mueve en cada paso una lista deberá comprender a todos los vecinos en un radio $R = r_c + 2kd$. La explicación es sencilla, kd es la distancia máxima

que se desplaza cualquier átomo en k pasos, por lo tanto dos átomos que se mueven en direcciones opuestas se alejaran o acercaran como máximo $2kd$ en k pasos. Si bien se ha manejado R en función de k, también puede plantearse de forma recíproca, o sea, establecer un $R = r_c + margin$ ($margin$ no necesariamente coincide con $2kd$) y actualizar la lista cuando cualquier átomo de la esfera se haya desplazado una distancia igual o mayor a $\frac{1}{2}margin$ independientemente de la cantidad de pasos de simulación que hayan transcurrido. Más allá de cuál valor sea el constante y cuál se establezca en función del otro se desprende que a mayor k menor serán la cantidad de actualizaciones de lista pero mayor deberá ser la esfera y mayor la cantidad de pares a considerar en los cálculos. En el otro sentido, a menor k más actualizaciones de listas serán requeridas pero menor será la cantidad de pares a considerar en el cálculo.

2.5.2. Cell-Lists

Esta estrategia consta de dividir la caja de simulación en cuadrantes llamados celdas, cuyas aristas serán al menos de longitud r_c (recordar r_c como el radio *cut-off*), de manera que todos los átomos pertenecen a una celda y solo a una [30]. Durante la simulación, al momento de calcular las interacciones electrostáticas para un átomo en particular, se tomarán en cuenta los restantes átomos presentes en la celda que contiene al primero, y también a los átomos en las celdas adyacentes. Al tratarse de celdas de longitud r_c, para cualquier átomo, el peor caso (por ejemplo que esté alojado en el borde de una celda) los vecinos más lejanos a éste, o sea a una distancia r_c, estarán alojados en la celdas adyacentes a la celda que lo contiene.

Este enfoque por un lado evita la carga de mantener una lista para cada átomo del sistema pero por otra parte obliga a mantener actualizada la información de cada celda; los átomos estarán en constante movimiento y por ende muchos estarán migrando de una celda a otra. Por lo tanto, las actualizaciones cada cierta cantidad de pasos sigue siendo necesaria, dependerá de la "movilidad" del sistema la cantidad de pasos adecuada entre actualizaciones consecutivas. No obstante la "carga" generada por las estructuras adicionales disminuye con este enfoque, dado que es sensiblemente menor el número de celdas requeridas con respecto al número de listas requeridas en las *Verlet-Lists*. Por otra parte, como se verá más adelante, la

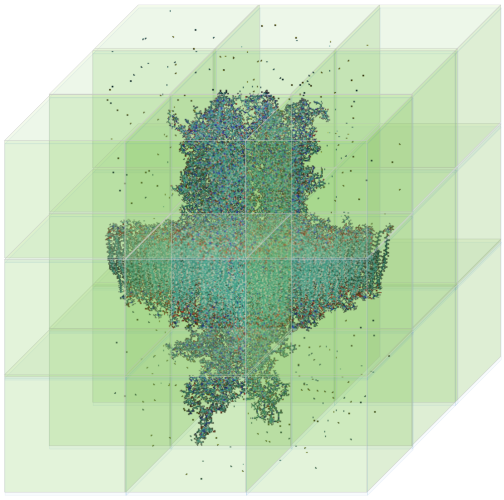


Figura 2.12: División del espacio en celdas de longitud r_c (Cell-Lists).

elección de que estructura usar no solo depende de lo que se intuya más apropiado en términos de fiabilidad o de eficiencia en memoria. Otros factores como pueden ser el hardware usado y sus particularidades (ventajas y desventajas) también deberán ser tenidos en cuenta.

2.6. Particle Mesh Ewald

Las esferas de corte son una herramienta sumamente útil cuando se desea reducir el costo computacional y aunque las fluctuaciones en la energía que produce dicha frontera pueden ser atenuadas mediante el uso de funciones de *switching*, este método no es capaz de atenuar situaciones generadas por ejemplo por la presencia de iones en el sistema [24]. La influencia de las interacciones no enlazantes de largo alcance (Coulomb) son de relevancia incluso más allá del radio *cut-off* y deben ser tomadas en cuenta para una mayor fiabilidad de la simulación. Ahora bien, si se incluye el cálculo de dichas interacciones a todas las imágenes en todas las direcciones se obtiene la siguiente ecuación:

$$E_{Coulomb} = \frac{1}{2} \sum_{v} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{q_i q_j}{|r_{ij}+v|}, \tag{2.13}$$

$$\text{donde } v = v_1 a + v_2 b + v_3 c.$$

En la Ecuación 2.13, r_{ij} representa la distancia entre los átomos i y j de la celda origen,

por lo tanto $\mid r_{ij} + v \mid$ representa la distancia entre el átomo i de la celda origen y el átomo j de la imagen en la posición $\{v_1, v_2, v_3\}$, correspondiendo el origen $\{0, 0, 0\}$ a la celda central [3]. Esta serie es condicionalmente convergente [31], dependiendo del orden en el que se sumen los términos [32, 33]. Además si se suman los términos en orden comenzando por las imágenes más cercanas al origen, tomando la distancia r como referencia, se infiere que el potencial disminuye a razón de $\frac{1}{r}$ mientras que la cantidad de iones aumenta a razón de r^2. Las sumas de Ewald proponen una forma más apropiada de evaluar las interacciones electrostáticas que aceleran y aseguran la convergencia de dicho cálculo. A grandes rasgos, este método plantea el cálculo de dichas interacciones como una suma de dos términos. Un primer término representa las contribuciones de corto alcance, planteado en el espacio real o directo, un segundo término representa las contribuciones de largo alcance, planteado en el espacio de Fourier o recíproco y un tercer término con fines correctivos, ver Ecuación 2.14.

$$E_{Ewald} = E_{directo} + E_{recíproco} + E_0 \tag{2.14}$$

El término en el espacio directo (distribución apantallante) utiliza una distribución gaussiana y el término en el espacio recíproco (distribución compensatoria) utiliza una trasformada de Fourier. La suma neta de todos los términos será el potencial neto, mediante la cancelación de la distribución apantallante con la distribución compensatoria, asegurando una rápida convergencia, ver Figura 2.13.

De esta manera se considera el cubo real y sus imágenes (condiciones de periodicidad). Si bien este método es más apropiado para tratar las interacciones de largo alcance el mismo implica un costo computacional $O(N^2)$.

PME (del Inglés Particel Mesh Ewald) es un método de grilla que surge como alternativa para calcular la suma de Ewald con un bajo nivel de error y con un costo computacional $O(N\log N)$ [34]. Esta técnica propone usar transformadas rápidas de Fourier (FFT del inglés Fast Fourier Transform) para calcular el término planteado en el espacio recíproco, para esto es necesario discretizar el espacio, por lo que se divide el espacio en cuadrantes (notar que no se relaciona con el concepto de *Cell-Lists*) donde cada uno de estos tendrá un potencial asociado acorde a las partículas que en estos residan. Para calcular el potencial ejercido

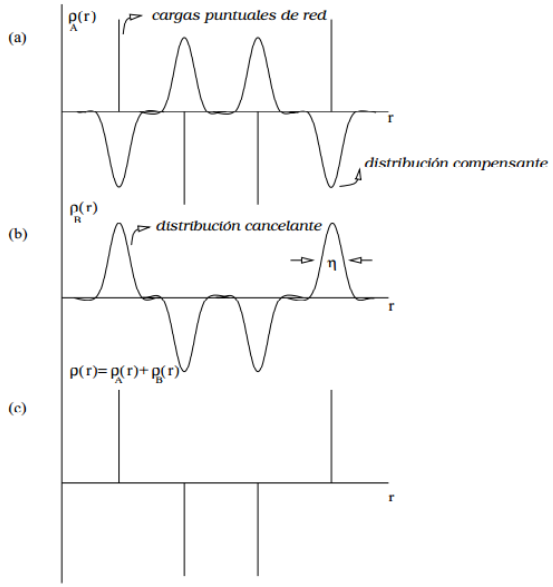


Figura 2.13: Distribuciones de carga en las sumas de Ewald: (a) de las cargas puntuales y el fondo compensante; (b) de la carga cancelante; (c) distribución de cargas resultante de la suma de Ewald [3].

sobre una partícula primero se determina a que cuadrante pertenece ésta, luego se obtiene el potencial ejercido por los cuadrantes vecinos sobre éste y, finalmente, se refina el resultado en función de la partícula para la cual se desea calcular el potencial.

Capítulo 3

Procesadores Gráficos

En este capítulo se presentan los principales aspectos que componen a un procesador gráfico o GPU, haciendo énfasis en la arquitectura CUDA. En breves palabras CUDA representa la infraestructura de base de las GPUs de NVIDIA para la resolución de problemas de propósito general. Específicamente, esta tecnología permite el uso de las GPUs como verdaderos procesadores de HPC extendiendo su campo de aplicación por encima de la tarea para la cual fueron originalmente concebidas; desplegar imágenes en pantalla.

3.1. Aspectos generales

Antes de introducir los detalles de las GPUs se hará un breve repaso del funcionamiento de un procesador tradicional (CPU, del inglés Central Processing Unit), para así tener un mejor entendimiento de los aspectos que componen una GPU en comparación con el primero. El CPU está diseñado para la ejecución de código de manera secuencial, es decir una instrucción de computadora a la vez. Durante la ejecución de un programa, el CPU recoge las instrucciones y los datos asociados a éstas desde la memoria RAM (del inglés Random Access Memory), interpreta dicha instrucción, la ejecuta y finalmente escribe el resultado en la memoria RAM. Este tipo de mecanismo se clasifica como SISD (del inglés Single Instruction, Simple Data) según la taxonomía de Flynn [35], es decir, solo una instrucción se ejecuta sobre una sola unidad de información a la vez.

Por otra parte las GPUs son capaces de ejecutar la mismas instrucciones sobre un con-

junto de datos a la vez, esto mediante estructuras lógicas, llamadas bloques, que agrupan a varios hilos de ejecución concurrentes. Este tipo de arquitectura se categoriza como SIMD (del inglés Single Instruction, Multiple Data).

Tomando en cuenta las ventajas provistas por la arquitectura CUDA [20] para la programación de GPUs, estas pueden ser vistas como un conjunto de procesadores multicore trabajando sobre una única memoria compartida por todos estos. De hecho las GPUs comúnmente son concebidas como procesadores many-cores (procesadores cuyos números de cores exceden ampliamente los de un procesador multi-core) por la gran cantidad de pequeños cores que contienen. Formalmente una GPU se apega al paradigma SPMT (del inglés Single Program, Multiple Thread) de programación paralalela, debido a que muchos cores ejecutan el mismo programa sobre diferentes unidades de dato mediante el uso de diferentes hilos, pero no necesariamente todo los hilos estarán ejecutando la misma intrucción en un mismo instante de tiempo [36]. Actualmente, cientos de hilos se pueden ejecutar en una GPU de forma paralela, y se espera que este número siga creciendo rápidamente, lo cual coloca a este tipo de dispositivos como una herramienta poderosa; y atractiva para la implementación de algoritmos paralelos a un bajo precio en comparación con soluciones basadas en clusters, etc.

A grandes rasgos CUDA [20] comprende un conjunto de capas que incluyen: diversas bibliotecas y herramientas en la capa superior, un lenguaje basado en C como interfaz de programación y un driver CUDA responsable de la transferencia de información entre la GPU y la CPU. CUDA está disponible a partir de GPUs de la serie GeForce 8 y superiores de la firma NVIDIA.

En cuanto a hardware la arquitectura CUDA está basada en un arreglo de multiprocesadores donde cada uno de estos consta de 32 procesadores escalares. El número de procesadores escalares varía dependiendo de la familia de GPU; por ejemplo en Tesla es 8, en Kepler 192 y en este trabajo se describe Fermi, que como ya se mencionó son 32 los procesadores escalares. Junto con estos multiprocesadores también se incluyen unidades adicionales, que están dedicadas a tareas como multithreading o manejo de memoria *on-chip*. Cuando se identifica una porción de código de una aplicación que puede ser ejecutada varias veces en diferentes porciones de datos, esta puede ser aislada en una función especial denominada *kernel*. Dicho kernel es asignado al dispositivo (GPU) para su ejecución mediante varios hilos (threads),

por lo que es necesario que esta función (kernel) sea compilada previamente para luego ser transferida al dispositivo para su posterior ejecución.

Cuando un kernel es invocado un importante número de threads es generado en la GPU. El grupo que comprende a todos los hilos creados en una ejecución de kernel se denomina grilla, la cual se divide en varios subconjuntos de hilos llamados bloques. Cada bloque será asignado, en un instante de tiempo dado, a un multiprocesador, es decir, todos los hilos de un mismo bloque estarán en el mismo multiprocesador.

No hay un orden de ejecución determinado para todos los bloques, si hay suficientes multiprocesadores[1] todos los bloques serán ejecutados en paralelo, sino una estrategia de tiempo compartido es aplicada para que dichos bloques se distribuyan sobre el conjunto de multiprocesadores disponibles.

Durante la ejecución del kernel los hilos podrán acceder a varios espacios de memoria, hoy en día se distinguen seis tipos de memoria diferente en las GPUs: registros, memoria local, memoria compartida, memoria global, memoria constante y memoria de textura. En la Tabla 3.1 se muestran las principales características de estos tipos de memoria que serán comentados mas adelante.

Memoria	**Alcance**	**Tiempo de vida**	**Tamaño**	**Velocidad**
Registros	Thread	Kernel	Muy pequeño	Muy rápida
Local	Thread	Kernel	Pequeño	Lenta
Compartida	Block	Kernel	Muy pequeño	Muy rápida
Global	Grid	Application	Grande	Lenta
Constante	Grid	Application	Muy pequeño	Rápida
Textura	Grid	Application	Muy pequeño	Rápida

Tabla 3.1: Características de los diferentes tipos de memoria de las GPUs.

1. Registros: Los registros constituyen el tipo más rápido de memoria y se encuentra alojado *on-chip*, en otras palabras en el propio multiprocesador. Cada hilo cuenta con sus propios registros, siendo estos accesibles sólo por el propio hilo. Si bien este tipo de memoria sería la idónea para muchas tareas es escasa de por sí, y más aún con respecto

[1]Cada multiprocesador puede procesar varios bloques al mismo tiempo, la cantidad depende de la arquitectura, la cantidad de registros que utilice el kernel, etc.

a los demás tipos de memoria. La cantidad de memoria de este tipo que será asignada a cada hilo es gestionada por el compilador, es decir se determina al momento de construcción del kernel.

2. Memoria Local: Adicionalmente, los hilos también disponen de otro tipo de memoria, también de uso individual por los threads, pero es una de las memorias más lentas del dispositivo, es *off-chip* (esta fuera del multiprocesador) además de no contar con caché. Al iguál que en el caso anterior la cantidad de memoria de este tipo será gestionada por el compilador.

3. Memoria Compartida: Cada bloque de hilos tiene a disposición un espacio de memoria compartida, la cuál es casi tan rápida como los registros. Este espacio de memoria será compartido por todos los hilos del bloque y sólo por éstos. Al igual que los registros esta memoria es *on-chip* y tiene un ciclo de vida igual al de bloque.

4. Memoria Global: El mayor volumen de memoria disponible en una GPU es de este tipo. Todos los hilos, independientemente del bloque al que pertenezcan, tienen acceso a todo el espacio de memoria global. Sin embargo esta memoria, a pesar de contar con caché para la familia Fermi, es una de las más lentas.

5. Memoria Constante: Este tipo de memoria rápida es de solo lectura (*read-only*) y es *off-chip* aunque dispone de caché.

6. Memoria de textura: Este caso cuenta con las mismas características que la memoria constante. Tanto este tipo de memoria como el anterior suelen ser utilizados para almacenar datos cuyos valores no varían a lo largo de la vida del kernel.

La Figura 3.1 presenta un diagrama de la jerarquía de memoria en la arquitectura CUDA incluyendo los seis tipos de espacios de memoria antes descriptos. Aunque el diagrama muestra a la memoria local cercana a los hilos (por su carácter de uso individual por parte de los hilos), está realmente se localiza en la memoria del dispositivo al igual que la memoria global.

Respecto a la transferencia de datos entre host (CPU) y dispositivo (GPU) la misma puede ser realizada de forma sincrónica como asincrónica. De hecho, es posible realizar la

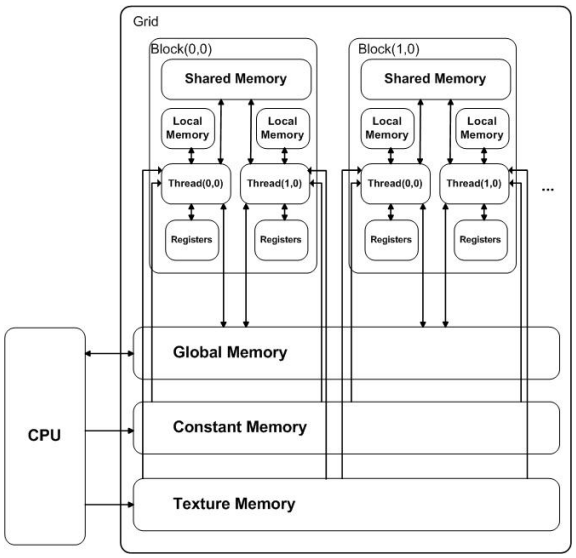


Figura 3.1: Jerarquía de memoria de CUDA.

transferencia de información a la vez que se está ejecutando una rutina en el dispositivo, para la cuál es necesario el uso de memoria no paginable. En otras palabras, se bloquea un espacio de memoria RAM para que la GPU puede trabajar sobre ésta sin requerir asistencia del host, esta característica asegura una transferencia de memoria más rápida gracias a la ausencia de *Paging* [37].

3.2. Manejo de memoria

Determinar cómo debe ser distribuida la información sobre la jerarquía de memoria es relevante. Además del espacio necesario, que en muchos casos determina como utilizar la memoria, conocer como dicha información está organizada o como es accedida durante una rutina ejecutada en GPU es otro aspecto a considerar.

3.2.1. Uso de la memoria global

Por más que se intente minimizar su uso, es inevitable el uso de memoria global en lo que respecta a MD. Si se considera un sistema con unos pocos miles de átomos la información asociada a dicha sistema solo podrá ser almacenada en este medio. El acceso a esta memoria es costoso, pero lo es aún más si no se utiliza correctamente. Las lecturas en memoria global

se realizan en bloques, es decir, si se requiere la lectura de una "palabra" de 4 bytes, un bloque de 32 bytes de memoria global será transferido. El tamaño del bloque podrá variar entre 32, 64 y 128 bytes, dependiendo del tamaño de la lectura original (palabra). La lectura de la memoria global en bloques por parte de la GPU obedece al criterio de que hilos contiguos leen secciones contiguas de memoria. En el ejemplo anterior si 8 hilos requieren una palabra de 4 bytes cada uno, dispuestos de forma contigua en memoria, una única lectura en bloque de 32 bytes es suficiente para cumplir con el requisito de todos estos hilos. Cuando la memoria es accedida acorde al patrón descripto anteriormente, se dice que el acceso a memoria es *coalesced*. Cabe mencionar que en las tarjetas gráficas actuales existen otros escenarios, un poco menos restrictivos que el anterior, que también traen aparejado un acceso *coalesced* a la memoria.

3.2.2. Uso de la memoria compartida

La memoria compartida constituye una de las memorias más rápidas que provee una GPU. Sin embargo, también hay buenas prácticas asociadas a la utilización de dicho recurso. En este caso no importa el patrón de acceso a los datos siempre y cuando no se provoque un conflicto de bancos al acceder a la memoria. Cuando más de un hilo en un mismo bloque requiere acceder a la misma dirección de memoria compartida con fines de escritura se genera un conflicto de acceso. Esto provoca la serialización de los accesos, impactando en el rendimiento del kernel. En otras palabras, las lecturas a memoria, sin importar el patrón de acceso no afecta el rendimiento, en cambio el patrón de acceso para escritura determina la presencia o no de conflictos y por ende, el rendimiento.

Capítulo 4

GPUs en Dinámica Molecular

En este capítulo se discute el estado del arte respecto a la implementación de software de Dinámica Molecular que integra el uso de GPUs. Se describen los principales trabajos del área, las problemáticas que se presentan y los diferentes enfoques para abordarlas.

Finalmente, se profundiza en NAMD como principal caso de estudio, dando una breve explicación de su arquitectura para luego profundizar en como esta herramienta explota las GPUs.

4.1. Introducción

Dentro de los aspectos que fueron mencionados en los trabajos analizados hay uno en particular en el que todos los autores hacen énfasis: las interacciones no enlazantes. Las interacciones no enlazantes implican la mayor parte del cálculo en la ecuación de potencial [17], alcanzando incluso el 90 % del total de costo computacional [38]. Notar que, para cada átomo se calculan unos pocos enlaces covalentes, ángulos y diedros en lo que refiere a las fuerzas enlazantes, en cambio deben calcularse k interacciones no enlazantes con k dependiendo del radio *cut-off*. Por esto, la gran mayoría de las soluciones se enfocan principalmente en la resolución del término no enlazante de la ecuación de potencial [4, 17], o incluso otros se concentran sólo en la resolución de dicho término [15, 38, 39, 40].

Cuando solo la CPU es explotada en una dinámica molecular la evolución del costo computacional en relación a la cantidad de átomos es $O(N^2)$ [38], lo cuál da una clara pauta

que las interacciones no enlazantes representan la mayor parte del cálculo implicado [41]. En cambio en una GPU el costo computacional escala de forma sub-lineal, lineal o sub-cuadrática dependiendo de la relación *número de átomos*↔*cantidad de cores* [38, 41].

4.2. Implementación del radio *cut-off*

Respecto de este punto hay un significativo debate entre dos enfoques para manejar las interacciones no enlazantes con uso de radios *cut-off*; el uso de *Verlet-Lists* (2.5.1) o *Cell-Lists* (2.5.2).

En términos conceptuales, se podría argumentar que el uso de *Verlet-Lists* sería más efectivo, debido a que este tipo de listas contienen a los átomos comprendidos en un radio $R = cut\text{-}off + margin$. De esta manera, solo se incluyen los átomos a una distancia menor o igual a *cut-off* y aquellos átomos que son plausibles de entrar en el radio *cut-off* en los próximos pasos aunque con un costo de actualización $O(N^2)$ [39]. En cambio *Cell-Lists* representa una forma sencilla de administrar el *cut-off*, con un costo de actualización $O(N)$ [39], pero con un mayor número de pares de átomos a evaluar, producto que en lugar de usar una esfera (*Verlet-Lists*) se usa un cubo con aristas de longitud *cut-off*. Por otra parte no se debe perder de vista que el objetivo en todos los casos es sacar provecho de la potencia de la GPU. Por esto, es muy importante tomar reparo en algunos puntos al momento de implementar la solución basada en GPU; la gestión de acceso a memoria suele ser un importante cuello de botella (ver punto 3.2). *Cell-Lists* representa una estrategia de mayor afinidad con respecto a la memoria de la GPU, donde establecer una correspondencia átomo↔hilo y celda↔bloque ha sido una elección dominante [15, 17, 39, 40]. El Algoritmo 4.1 generaliza la estrategia utilizada para calcular las interacciones no enlazantes en una GPU para los trabajos que explotan las *Cell-Lists*.

Del Algoritmo 4.1 también se puede observar una estrategia aplicada en varios casos por las herramientas de MD que integran GPU. Cuando se aplica el uso de celdas al espacio del sistema cada bloque de hilos representa a una celda. Dicha celda suele ser cargada en memoria compartida desde la memoria global y dicha carga es realizada por todo el bloque. Los hilos del bloque acceden a direcciones contiguas de memoria, donde cada dirección de

```
i = blockId*ThreadsPerBlock+threadId   // Se determina el átomo i a ser cargado
                                       // por el hilo threadid del bloque blockId
a_i = loadAtomIntoRegistry(i)          // Accesso coalesced, hilos contiguos
                                       // toman átomos contiguos
F_sum = 0
for all 27 cells C surrounding a_i do
  for all atoms b_j in C
    b_j = loadAtomFromCell(j, C)
    d = calcDistance(a_i, b_i)
    if ( d ≤ cut-off ) then
      F_ab = calcPairForce(a_i, b_j)
      F_sum = F_sum + F_ab
    end if
  end for
end for
```

Algoritmo 4.1: Cálculo de fuerzas enlazantes usando *Cell-Lists* con un bloque por celda y un hilo por átomo.

memoria contiene un átomo de la celda. Esta es una forma de garantizar el acceso *coalesced* mencionado en el punto 3.2.2.

Por otra parte Anderson et al. [16] dividen el espacio en celdas, pero usan *Verlet-Lists* mediante una matriz NBL tal que $NBL_{i,j}$ contiene el j-ésimo vecino del átomo i. El trabajo propone una estrategia para ordenar los vecinos en dicha lista de forma que átomos cercanos accedan a vecinos cercanos entre sí para optimizar el acceso a memoria y favorecer el acceso *coalesced* a ésta. Trott et al. [40] con su aporte para LAMMPS también proponen el uso de *Verlet-Lists* en su solución. Para esto, proponen el uso de curvas de Hilbert [42] en el espacio de átomos, como se ve en la Figura 4.1 este tipo de construcciones en base a fractales cubre el espacio mediante una sola línea. Una de las propiedades de estas curvas es, que en general, si dos puntos de dicha curva están cerca, los puntos en el espacio también lo estarán.

Salomon-Ferrer et al. [43] también hace uso de las curvas de Hilbert para ordenar los vecinos en su propuesta de solución de PME sobre GPUs para AMBER para modelar solventes explícitos.

Otros trabajos se vuelcan por la idea de calcular las interacciones para todos los pares de átomos en el sistema, [4, 41]. Esto se debe a que ambos trabajos usan solvente implícito, aplicar truncaturas mediante radios *cut-off* puede llevar a inconsistencias debido a que las

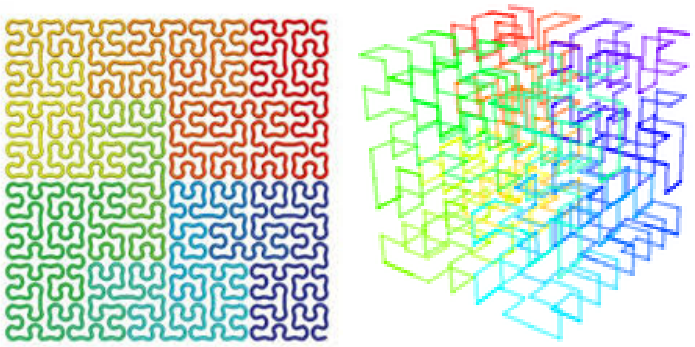


Figura 4.1: Curvas de Hilbert, en dos (izquierda) y tres (derecha) dimensiones.

ecuaciones que emulan el efecto del solvente no consideran dichos recortes. Por otra parte, usar solvente implícito disminuye significativamente la cantidad de átomos en el sistema, reduciendo así considerablemente la cantidad de pares sobre los cuáles calcular el potencial.

4.3. Aplicación de la tercera ley de Newton

La tercera ley de Newton representa un concepto importante para las aplicaciones de Dinámica Molecular. Cuando estas aplicaciones se restringen a soluciones para arquitectura de computadores clásicas basadas en CPU y memoria RAM, este concepto representa una reducción importante (en teoría 50 %) en el cómputo a realizar. Dado un par $i \leftrightarrow j$ de átomos el potencial ejercido por j sobre i será de igual magnitud y sentido opuesto al potencial ejercido por i sobre j. Si se visualizan las relaciones entre todos los pares de átomos del sistema como una matriz M de dimensiones $N \times N$, aplicar la tercera ley implica solo calcular una de las dos matrices triangulares que componen a M, la matriz triangular complemento se infiere de la primera ya que $F(i,j) = -F(j,i)$. Sin embargo, la aplicación de esta estrategia a las herramientas que usan GPU no es trivial [16, 17, 38]. Los potenciales generados por los diferentes pares usualmente son acumulados en un registro reservado para dicho átomo como se muestra en el Algoritmo 4.1. Este algoritmo muestra a grandes rasgos el cálculo del potencial cuando se opta por el uso de *Cell-Lists*, donde un bloque representa una celda y cada hilo representa un átomo en dicha celda. Como se observa, el hilo carga la información del átomo i en un registro (la memoria más rápida de la GPU) y recorre los demás átomos j acumulando los potenciales en otro registro para aquellos pares $i \leftrightarrow j$ cuya distancia $d(i,j) \leq$

cut-off. Sin embargo, acumular $-F(j,i)$ puede ser costoso en términos de accesos a memoria; son muchos los hilos que estarán accediendo a la misma dirección de memoria. Mientras el hilo i acumula el potencial sobre j en la variable F_j el hilo $i+1$ también necesita acumular el potencial en F_j. Esto sucedería con varios hilos a la vez, generando conflictos por acceso a memoria, y por ende, un detrimento en el rendimiento al serializarse accesos a memoria que quizás pudieran ser paralelizados. El escenario anterior se debe a que estas soluciones lanzan varios hilos de ejecución que, a pesar que cada uno tiene un átomo diferente, comienzan evaluando el potencial contra el mismo átomo j, luego $j+1$ y así sucesivamente.

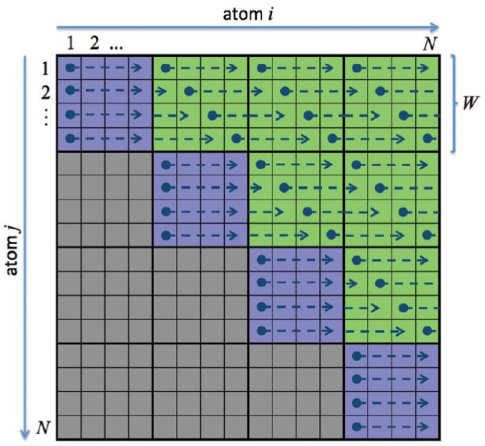


Figura 4.2: Estrategia de AMBER para evaluar fuerza entre pares [4].

En este sentido Götz et al. [4] proponen que cada hilo tenga una posición inicial única como muestra la Figura 4.2. En este caso se modelan las interacciones mediante una matriz M de dimensiones $N \times N$, la cuál es subdivida en cuadrantes. Los cuadrantes de la diagonal se calculan de igual manera que en los casos anteriores, o sea, con todos los hilos comenzando en la misma posición, ya que en la diagonal de la matriz estarán los mismos átomos en ambos ejes. En cambio, para los cuadrantes por encima de la diagonal los átomos en un eje no son los mismos que en el otro, por lo que es útil calcular el recíproco. Cada hilo comenzará en la posición (i,i) de la matriz, o sea en la diagonal, de manera que cuando el hilo i calcula $F(i,j)$ el hilo $i+1$ calcula $F(i+1,j+1)$. En otras palabras el hilo i acumulará el recíproco en F_j mientras que el hilo $i+1$ acumulará el recíproco en F_{j+1}. Como se verá más adelante, esta estrategia es la que el presente trabajo de tesis intenta aplicar sobre NAMD para sacar provecho de la tercera ley de Newton.

4.4. Uso de la memoria

En las diferentes soluciones se pueden detectar ciertos criterios en común a la hora de cómo sacar provecho de la jerarquía de memoria que brinda una GPU. Todos los trabajos discutidos, con alguna variante, pero sin excepción, proponen el siguiente esquema de uso de memoria en lo que refiere a los tres primeros hitos.

- Registro por átomo: Acorde a lo visto anteriormente la idea general es que cada hilo tome un átomo, cargue su información en un registro (la memoria más rápida) y recorra sus vecinos acumulando el potencial sobre dicho átomo en otro registro.

- Memoria compartida para los vecinos: Como se menciona en el item anterior, generalmente un hilo recorrerá todos los vecinos de un átomo calculando el potencial. Para esto es que se apuesta a cargar los vecinos de un átomo dado de a "tandas" en la memoria compartida, reduciendo el acceso a memoria global.

- Memoria Global: Por ser la memoria más abundante y de mayor ciclo de vida es la que al principio y al final de las diferentes ejecuciones tendrá casi todo la información relativa al complejo. Aunque esta memoria no se use comúnmente al momento del cálculo las posiciones de los átomos, sus propiedades, sus vecinos, entre otros deberán ser recogidos de dicha memoria.

- Memoria de Textura: Esta memoria de solo lectura puede ser cargada previo a la ejecución de un kernel, por lo que, los datos alojado en esta deben ser de naturaleza constante. Anderson et al. [16] proponen usar esta memoria para alojar listas de exclusión. Estas listas definen que átomos cercanos no deben ser tomados en cuenta para el cálculo de interacciones no enlazantes. Por ejemplo, si el átomo i está unido covalentemente al átomo j no tiene sentido calcular interacciónes no enlazantes sobre estos. Sin embargo, Stone et al. [15], proponen otro uso para esta memoria. La memoria de textura permite ejecutar interpolaciones entre los datos de esta, por lo cual NAMD usa dicha memoria para interpolar valores para el cálculo de factores aplicados tanto para fuerzas de van der Waals como electrostáticas.

- Memoria Constante: Esta memoria, de características similares a la de textura, pero sin funcionalidades de interpolación, es utilizada por NAMD para alojar las listas de exclusión [15].

4.5. Otras consideraciones

Hay otros aspectos en la aplicación de GPU en Dinámica Molecular, que si bien no tienen el rol preponderante como el de los mencionados en los puntos anteriores, merecen ser mencionados.

Usar un hilo por átomo y un bloque de hilos por celda ha sido la opción de mayor consenso respecto a cómo explotar la gran cantidad de hilos que una GPU provee. Sin embargo, algunos trabajos proponen otros esquemas, con mayor o menor éxito, como por ejemplo Liu et al. [38] que proponen usar un átomo por hilo pero sin dividir el espacio de simulación en celdas. Esto causa que dos hilos de un mismo bloque no necesariamente representen átomos cercanos, valiéndose del uso de *Verlet-Lists* para representar los vecinos. Por otra parte, Trott et al. [40] implementan dos esquemas que son intercambiados a lo largo de una simulación acorde la aplicación considera conveniente. Uno de estos esquemas es el ya tratado *átomo* $\leftrightarrow$ *hilo* tomando partido de la división en celdas, el otro esquema consta de usar un bloque de hilos por átomos, que propone distribuir las k interacciones posibles entre los hilos del bloque. Es decir, si un átomo tiene k vecinos y el bloque de hilos es de tamaño $\frac{k}{2}$, cada hilo debe procesar dos interacciones para dicho átomo. Este enfoque implica definir una grilla de mayor volumen, ya que cada átomo implica un bloque de hilos.

Otro aspecto importante es la distribución de trabajo entre GPU y *host* (CPU), donde algunos autores intentan asignar todo el trabajo, cálculo de fuerzas e integración, a la GPU [4, 16, 17]. Sin embargo, por otra parte están las soluciones híbridas (CPU+GPU), como por ejemplo Friedrichs et al. [41] que propone solo calcular los términos de la ecuación de potencial en GPU dejando la integración en la CPU. También Stone et al. [15] y Trott et al. [40] constituyen soluciones híbridas, en este caso concentrándose solo en resolver el término no enlazante en GPU, dejando el resto del cálculo en CPU.

4.6. NAMD con GPUs

NAMD es un software de computación paralela para Dinámica Molecular, diseñado con el fin de optimizar la simulación de grandes sistema biomoleculares. Un detalle no menor es su capacidad para adaptarse tanto a computadores personales (máquinas de escritorio) como a plataformas de computación paralela, compuestos por cientos de procesadores [1]. Basado en Charmm++, un sistema de programación paralela [44], e implementado en C++ NAMD soporta tanto los campos de fuerzas CHARMM (nativo) y AMBER así como los formatos de archivos asociados a estos. NAMD ofrece diversas opciones, como puede ser algoritmos optimizados para la evaluación de fuerzas electrostáticas, como por ejemplo PME, control de presión y temperatura, *Steered MD* así como cálculos de energía libre (FEP del inglés Free Energy Perturbations) [45]. Como se mencionó anteriormente, NAMD ofrece soluciones para aumentar la capacidad de cómputo tomando partido de la evolución en las GPUs. En esta sección se hace un breve repaso de la arquitectura general de NAMD y la integración del uso de GPUs a dicha arquitectura.

4.6.1. Arquitectura de NAMD

Como ya se mencionó, NAMD usa Charm++ [44], un sistema de programación paralela orientado a ocultar la latencia en la comunicación en sistemas distribuidos. Charmm++ descompone el trabajo en objetos que interactúan entre sí mediante el envío de mensajes, independientemente de la localización de los objetos, residan en la misma máquina o no. Estos mensajes son de naturaleza asíncrona, es decir, enviados desde un objeto a otro sin una negociación previa y tiene como objetivo disparar la ejecución de algún método del objeto destino.

NAMD fue una de los primeras herramientas en aplicar descomposición espacial a la vez que aplica descomposición en el cálculo de la energía potencial. NAMD descompone el espacio del sistema en celdas (ver Figura 4.3) denominadas *patch*, que refiere a dicha celda como una unidad de trabajo. Las dimensiones de estas celdas (*patches*) se determinan aplicando un criterio similar a *Cell-Lists*, las celdas serán de longitud $d_{min} = r_c + margin$

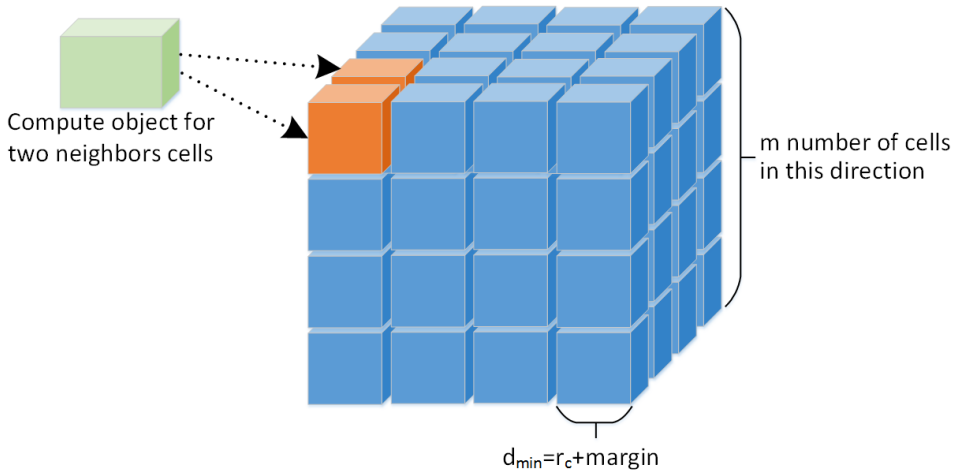


Figura 4.3: Descomposición espacial de NAMD.

en todas las direcciones. No obstante, el anterior es el caso más simple, las dimensiones de la celda pueden ser mayor que d_{min}, dependiendo esto del tamaño de la caja de simulación. El número de celdas en cada dirección es un valor entero, por lo cual d_{min} es ajustado a un valor mayor de manera de cumplir con la especificación. Esto puede generar que d_{min} no sea el mismo en (x, y, z), siendo posible contar con celdas de dimension $d_{minx}, d_{miny}, d_{minz}$ siendo estos valores diferentes entre sí.

4.6.2. Cómputo de fuerzas en NAMD

Por cada par de celdas (C_1, C_2) para las cuáles exista una interacción se le asigna un *compute-object* que será responsable de computar las interacciones entre todos los pares de átomos (i, j) con $i \in C_1$ y $j \in C_2$, pudiendo ser $C_1 = C_2$ inclusive.

Cada celda cuenta con 26 celdas vecinas, por lo que el total de pares de celdas a considerar (contando cada par una sola vez y el par que forma una celda consigo misma) es $14 \times p$, que corresponde a la cantidad de *compute-objects* que serán creados, siendo p la cantidad de *patches* (celdas) en el sistema. Estos *compute-objects* son distribuidos entre los procesadores disponibles obedeciendo a una estrategia de balance de carga, provista por la arquitectura de Charmm++, que evalúa la carga de trabajo en cada procesador. La estrategia anterior es parte de la descomposición de fuerzas mencionada anteriormente, donde a grandes rasgos

la Ecuación 4.1 expone el criterio usado por NAMD.

$$E_{total} = E_{enlazantes} + E_{no-enlazantes} \tag{4.1}$$

Como ya se mencionó en el capítulo anterior, NAMD implementa el uso de radios *cut-off*, ofreciendo incluso la opción de usar funciones de *switching* para las funciónes de *Coulomb* y *van der Waals* forzando la aproximación "suave" a 0 en el punto de truncamiento. Más allá del ahorro de cálculo implicado, en algunos casos estas funciones pueden considerarse no representativas de la realidad, como ya se trató en la Sección 2.6, aunque el potencial decrece con la distancia, en algunos casos despreciable, este no es nulo. NAMD también provee una implementación de PME para las interacciones electrostáticas, permitiendo obtener un cálculo cualitativamente mejor en tiempo $O(N \log(N))$.

4.6.3. Integración de GPUs a NAMD

La arquitectura original de NAMD, distribuye los *patches* y *compute-objects* sobre todos los procesadores disponibles, con la premisa que los *compute-objects* estén alojados cerca de los *patches* implicados en el cómputo. Un importante aspecto a tener en cuenta al momento de integrar el uso de GPUs, es que originalmente NAMD invoca un *compute-object* para cada uno de todos los pares de *patches* implicados en el cálculo del término no enlazante. Si se aplicara dicho criterio de forma directa sobre la GPU, implicaría una llamada al kernel por cada par de *patches* considerados en el cálculo [46]. Es deseable evitar transferencias de memoria entre GPU y *host* dado que reduce el rendimiento de la solución. Por esto es que NAMD propone un nuevo tipo de *compute-object*, que comprende a todos los pares de *patches* implicados en el término no enlazante en lugar de comprender uno sólo. De esta manera se reduce sensiblemente el número de llamadas a kernels, causando de forma indirecta la reducción del volumen de datos transferidos entre *host* y dispositivo.

Otro aspecto importante a mencionar es como se aplica la división de cálculo de fuerzas con respecto a la GPU. Específicamente, NAMD asigna solo las interacciones no enlazantes de corto alcance, es decir, solo fuerzas de van der Waals e interacciones electrostáticas de corto alcance se asignarán a la GPU. Interacciones enlazantes, o no enlazantes de largo al-

cance serán calculadas en CPU. En detalle, una vez transferidos todos los pares de *patches* implicados a la GPU, se asignará un bloque de hilos por cada par de *patches*, considerando también que un *patch* constituye un par consigo mismo, ya que las interacciones no enlazantes dentro de la misma celda también deben ser calculadas. A raíz de esto, es importante mencionar, que cuando un par de *patches* está compuesto por dos *patches* distintos, se debe considerar el par y su recíproco. Por ejemplo, si un sistema constara de dos *patches* A y B, se asignaría a GPU los pares $A \rightarrow A$, $A \rightarrow B$, $B \rightarrow A$, $B \rightarrow B$ para su procesamiento. Para cada par de celdas asignadas a GPU solo se debe calcular el potencial para el primer elemento del par, esto explica por qué se asignan tanto $A \rightarrow B$ como $B \rightarrow A$.

Cuando se lanza un kernel, cada hilo de un bloque copia un átomo del primer *patch* del par en un registro, mientras que los átomos del segundo elemento del par son copiados a memoria compartida. De esta manera, cada hilo recorrerá los átomos alojados en memoria compartida calculando las fuerzas correspondientes y acumulando dicho valor en otro registro. Esta tarea es realizada por todos los hilos al mismo tiempo, finalmente, una vez calculada y acumulada tanto la fuerza como la energía, se escribirán dichos valores en memoria global, la cual estará mapeada a la memoria RAM del *host*. Esto, junto a lo expuesto en la sección anterior, denotan que NAMD trata de explotar los diferentes tipos de memoria en GPU en función del tipo de información que se trate.

La tarea de integrar GPU a NAMD no solo se enfoca en obtener una rutina lo más afinada posible en función de los recursos que esta provee, también el cómo invocar al kernel desde el *host* implica un trabajo adicional. Una solución que serialice el trabajo en CPU y GPU, además de oponerse a las "buenas prácticas" sugeridas por CUDA, genera un camino crítico innecesario en la aplicación. Esto se opone a la concepción original de NAMD como un software de computación paralela, por lo tanto dos estrategias son usadas para atacar este problema [46]:

- En NAMD el trabajo se descompone en objetos que interactúan entre sí mediante el envío de mensajes, los cuáles son despachados y manejados acorde un criterio de prioridad. Este criterio es modificado para permitir el solapamiento en la ejecución de CPU y GPU.

- Otra buena práctica sugerida por CUDA es, en lo posible, solapar el procesamiento en GPU con respecto a la comunicación entre GPU y *host*. Dada la naturaleza distribuida de NAMD, los distintos pares de *patches* asignados a una GPU pueden estar compuestos por un primer elemento que reside en el mismo *host* (local), o con un primer elemento que reside en otro *host* (remoto). Si se tratan todas estas unidades de cómputo en el mismo lote aquellos procesos remotos que esperan por el resultado de una interacción entre pares son tratados de igual manera que los procesos locales independientemente de su lejanía. Se debe tener presente que en el caso de un *patch* remoto el tiempo en obtener un resultado será, ademas del tiempo de cómputo de GPU, el tiempo de transferencia desde el *host* local al remoto. Por esta razón, NAMD distingue entre aquellos pares cuyo primer elemento es un *patch* local de aquellos cuyo primer elemento es un *patch* remoto, agendando a estos últimos en una primera invocación del kernel. Una vez finalizado dicho kernel se transfiere el resultado al *host* correspondiente a la vez que se lanza la ejecución de un segundo kernel que solo contiene a los pares compuesto por *patches* locales. Con este solapamiento se logra que la recepción del resultado para los *patches* remotos suceda en un instante de tiempo cercano al del despacho del resultado para los *patches* locales.

Además de explicar cómo las diferentes características de una GPU son explotadas por NAMD, es de orden mencionar que hay varias opciones respecto a cómo lanzar el kernel de cálculo para interacciones no enlazantes. Por ejemplo, es posible que los diferentes hilos de ejecución que corren en diferentes núcleos de un CPU compartan un mismo GPU. También es posible que un único hilo maestro administre la GPU y recepcione el trabajo que los restantes hilos asignen para ejecución sobre GPU. En un escenario que conste de un único nodo con múltiples cores (núcleos) y múltiples GPUs se puede aplicar un criterio de asignación en cuál los hilos 0 y 1 usan la GPU 0, luego los hilos 2 y 3 usan el GPU 1. Este criterio, así como cualquier otro criterio, es definido en los parámetros de arranque del NAMD en línea de comandos.

Capítulo 5

Propuesta

En este capítulo se presenta y evalúa una alternativa de la implementación de NAMD sobre GPU. Esta solución aplica la tercera ley de Newton, de forma similar a la propuesta de Götz et al. [4], mediante la modificación de la distribución original de NAMD. Adicionalmente, se aplican algunos cambios a la carga de ciertos datos para sacar provecho de la jerarquía de memoria de CUDA.

Reconociendo la naturaleza distribuida de la arquitectura de NAMD, este desarrollo se concentra en mejorar y evaluar el desempeño sobre un único nodo. Recordando que NAMD distribuye el trabajo mediante una partición del espacio (*patch*), el rendimiento de un conjunto de terminales depende del rendimiento de los elementos que lo componen.

5.1. Variaciones en el uso de la memoria

En el Capítulo 4 se establece que un kernel de NAMD ejecuta un bloque de hilos por cada par de *patches* $A \rightarrow B$. Cada hilo del bloque cargará de memoria global el átomo $a \in A$ en un registro y evaluará el potencial iterando sobre todos los átomos $b \in B$, evaluando y acumulando la fuerza sobre a. Las coordenadas y parámetros que corresponden a b, son cargados en memoria compartida. La sola carga de las coordenadas para un átomo a por si solo implica tres accesos a memoria, uno por cada eje cartesiano. Dado que los registros constituyen un tipo de memoria más rápida que la memoria compartida, se propone cargar b desde memoria compartida cuando sea sujeto de evaluación con respecto al átomo a. En

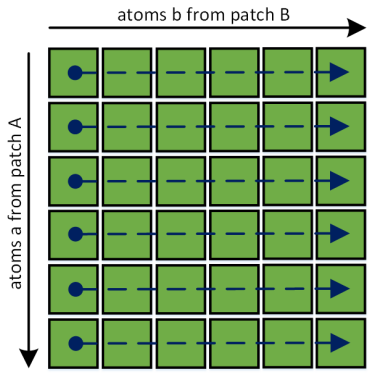


Figura 5.1: Estrategia de NAMD para evaluar fuerza entre pares.

cada ciclo del *loop* que recorre los átomos $b \in B$ cada hilo carga el átomo b en un registro, para luego utilizar este registro para todos los cálculos correspondientes, en el siguiente ciclo otro átomo b será cargado en dicho registro.

5.2. Aplicación de la tercera ley de Newton

La arquitectura original de NAMD para GPU intenta explotar los beneficios de la arquitectura CUDA, sacando provecho del alto grado de paralelismo implícito en ésta. Cuando un hilo carga el átomo $a \in A$, recorre los átomos $b \in B$ comenzando siempre por el átomo b_0, para luego tomar el átomo b_1 y así sucesivamente. Es decir, si se usa por ejemplo un bloque de 128 hilos, estos 128 hilos comenzarán evaluando la fuerza con respecto al mismo átomo b_0, ver Figura 5.1).

La idea anterior representa conceptualmente la estrategia más simple para recorrer los átomos $b \in B$, pero trae aparejado que no sea posible acumular el recíproco de la fuerza entre a y b debido a la cantidad de conflictos de memoria implicados. Por ejemplo si 32 hilos evalúan la fuerza entre a_i y b_0, la acumulación de $F(a_i, b_0)$ se realiza sin inconvenientes, pero acumular $-F(b_0, a_i)$ no es trivial. En el caso anterior cada hilo calcula parte del aporte sobre el recíproco, causando conflictos de escritura ya que todos los hilos del bloque intentarán acceder a la dirección de memoria donde se aloja la fuerza para b_0 al mismo tiempo.

Como alternativa se aplica la propuesta planteada por Götz et al. [4] para la solución de AMBER sobre GPU para resolver las fuerzas no enlazantes. En dicho trabajo cada hilo del

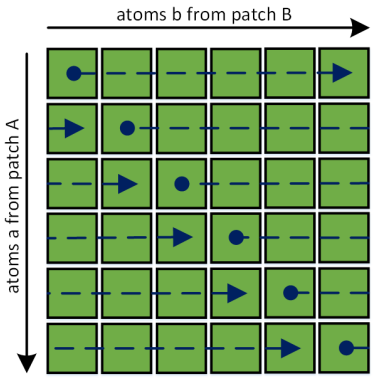


Figura 5.2: Estrategia de AMBER aplicada a NAMD para evaluar fuerza entre pares.

bloque comienza en una dirección contigua al anterior, generando un recorrido "escalonado" de la matriz de cálculo. Más formalmente el hilo i comenzará por el átomo a_i, por lo que el hilo 0 toma el átomo a_0, el hilo 1 el átomo a_1, etc.

La Figura 5.2 muestra el patrón de recorrido detallado anteriormente. Esta idea garantiza que para cada átomo b_i solo un hilo estará evaluando sobre este a cada vez, permitiendo obtener el recíproco y acumularlo en una dirección común de memoria sin generar conflictos. Con esta estrategia, se pueden procesar los pares de *patches* $A \rightarrow B$ y $B \rightarrow A$ en la misma ejecución, por lo que en lugar de agendar ambos pares para su cálculo ahora solo es necesario agendar $A \leftrightarrow B$.

5.2.1. Cambios en la transferencia de memoria

En la sección anterior queda establecido que la cantidad de pares de *patches* agendados para evaluación en GPU es reducido. Sea p el número de *patches* que componen una caja de simulación, NAMD originalmente agenda $pp = p{\times}27$ pares de *patches*. La cifra 27 corresponde a las 26 celdas que rodean a la celda (*patch*) en cuestión más la propia celda, ya que las interacciones entre átomos de una misma celda deben ser consideradas.

En cambio, la rutina propuesta solo requiere agendar $pp = p{\times}14$ pares de *patches* que será la propia celda más 13 celdas que representan la mitad de los *patches* necesarios debido a la aplicación de la tercera ley de Newton. Esta reducción impacta directamente en la ejecución del kernel así como en la trasferencia de datos implicada. Antes de la invocación de un kernel, la información referente a todos los pares de *patches* es copiada desde el host a la GPU, este

volumen de información en esta propuesta se reduce por un factor de $\frac{p\times 14}{p\times 27} = \frac{14}{27} \cong \frac{1}{2}\times$.

5.2.2. Implicancias en la relación comunicación-ejecución

En cada paso de una simulación MD, dos invocaciones al kernel de NAMD son disparadas. La primera invocación tiene como objeto resolver primero las evaluaciones para aquellos *patches* que no son locales (pertenecen a otro nodo). En cambio la segunda invocación corresponde a los *patches* locales (L), cuya evaluación es realizada a la vez que se transfieren los resultados de las evaluaciones sobre los *patches* remotos (R). Si bien esta estrategia se sigue respetando en la nueva propuesta, el número de *patches* cambia drásticamente. Con respecto a esto se puede clasificar los pares de *patches* en 4 tipos: $R \leftrightarrow R$, $R \leftrightarrow L$, $L \leftrightarrow R$ y $L \leftrightarrow L$. En la versión original de NAMD, cada par, excepto aquellos que representan interacciones de una celda consigo misma, es agendado dos veces, tanto un par como otro serán agendados en uno u otro kernel dependiendo de si el primer elemento del par es local o remoto. Por ejemplo, un par del tipo $R \leftrightarrow L$, implica agendar $R \rightarrow L$ en la primera invocación y $L \rightarrow R$ en la segunda. En cambio, cuando se aplica la solución propuesta los pares $R \leftrightarrow R$, $R \leftrightarrow L$, $L \leftrightarrow R$ serán agendados para la primera invocación, y solo los pares $L \leftrightarrow L$ serán agendados en la segunda invocación. Cabe mencionar que entre una invocación y otra los valores calculados permanecen en memoria global, por lo que queda a disposición de la segunda invocación (*patches* locales), aquellos valores "recíprocos" para el cálculo total de las fuerzas.

5.3. Evaluación y resultados

En esta sección se trata el criterio usado para evaluar la solución propuesta anteriormente, los sujetos de pruebas, criterios comparación, métricas, etc.

5.3.1. Casos de prueba

Se usaron dos sujetos de prueba para evaluar la solución propuesta, los cuáles son descriptos a continuación.

- APOA1: Apolipoproteína A-I constituye el primer caso de prueba, además de ser usado durante la construcción de la propuesta en etapas previas de desarrollo. Esta proteína contiene y transporta lípidos tanto en el sistema sanguíneo como el linfático, particularmente en este caso se usa una variante de apolipoproteína humana, identificada como 1GW3. Este complejo es utilizado ampliamente en la comunidad como sujeto de pruebas para evaluar herramientas de MD, siendo referencia en dichas evaluaciones. Este complejo junto con su entorno de simulación (solvente) comprende 92224 átomos.

- $\alpha 7$: Este sujeto de prueba es un modelo de un subtipo de receptor nicotínico de acetilcolina (nAChR), preparado mediante método de modelado por homología tomando como punto de partida un neuroreceptor de *Aplysia Californica* (Figura 5.3). El receptor $\alpha 7$ se localiza en el sistema nervioso central y junto con otro subtipo, $\alpha 4\beta 2$, son considerados fundamentales en el estudio de trastornos neurodegenerativos [47], como pueden ser Parkinson, Alzheimer, etc. El ambiente de simulación para este caso consta del receptor, una membrana lipídica que lo contiene, medio salino e iones, implicando un total de 319632 átomos.

Figura 5.3: Pentámero $\alpha 7$ en membrana lipídica, imagen tomada del software VMD [5].

5.3.2. Entorno de pruebas

La evaluación fue realizada en un computador equipado con un procesador Intel Quad-Core i5-2400 CPU de 3.10GHz, con una GPU modelo GTX480. El equipo utiliza un sistema operativo Linux Fedora 19 y se dispone de los compiladores *GCC* versión 4.8.2 y *NVCC* versión 4.1 para C/C++ y CUDA respectivamente. NAMD fue compilado usando las opciones disponibles por defecto, salvo con respecto a los kernels, esto fueron compilados con un máximo permitido de 50 y 63 registros por hilos, además del tope por defecto (32). En la Sección 5.3.3 se dará mayor detalle respecto a las opciones de registro mencionadas anteriormente.

Ambos modelos fueron evaluados usando condiciones periódicas, además de PME. En ambos casos el paso de simulación representa un femtosegundo (1×10^{-15} segundos), tomando muestras periódicas cada 10 picosegundos (1×10^{-12} segundos) simulados (10000 pasos).

5.3.3. Evaluación experimental de NAMD

Para realizar una comparación justa de la propuesta se considera mandatorio ajustar lo mejor posible, es decir optimizar su desempeño, la versión original de NAMD, para luego sí llevar a cabo la evaluación y comparación. Para esto, se aplicaran dos pequeñas variantes a la distribución original de NAMD. Por una parte se ajusta el tope permitido de registros por hilo, evaluación mediante, hasta encontrar un valor óptimo. Luego, de manera adicional, se hicieron pequeños cambios en el código, sin alterar su algoritmo, reemplazando en algunas partes el uso de memoria compartida por registros, logrando también una mejora en este punto. En este apartado se evaluará y discutirá estrictamente el desempeño de la GPU con respecto a las modificaciones hechas, en el siguiente apartado se tratará el impacto no solo en GPU sino también la aplicación con el mejor candidato obtenido en esta parte versus la solución de NAMD propuesta en este trabajo.

La Figura 5.4 muestra las diferentes versiones de NAMD manejadas durante este trabajo. Inicialmente contamos con la distribución normal de NAMD, *versión original*, sobre la cual, como se verá a continuación, se hizo un trabajo de "tuning", obteniendo una *versión optimizada en registro*. Sobre esta última versión se hacen cambios leves en el código

$$\left\{\begin{array}{l} \textit{versión original} \\ \left.\begin{array}{l} \textit{versión optimizada en registro} \\ \textit{versión mejorada en codigo} \end{array}\right\} \textit{versión optimizada} \\ \textit{versión propuesta} \end{array}\right.$$

Figura 5.4: Diferentes versiones de NAMD tratadas en este trabajo.

fuente, obteniendo así la *versión mejorada en código*. Finalmente, se encuentra la *versión propuesta*, que incluye la aplicación directa de la tercera ley de Newton, como se discutió en la Sección 5.2.

Ajuste en tiempo de compilación: registros

Como se trató en el Capítulo 3 los registros constituyen el tipo de memoria más rápida en una GPU pero son un recurso escaso. Cada hilo cuenta con su propio conjunto de registros, que, si bien es pequeño puede ser ajustado en tiempo de compilación, es decir, antes de generar el ejecutable. Se debe tener presente que alterar este valor repercute "de forma positiva en la disponibilidad de memoria del hilo" pero repercute "de forma negativa en la cantidad de hilos ejecutándose efectivamente". Los registros son un recurso físico de cada multiprocesador, por lo que si se aumenta la cantidad de registros por hilo necesariamente debe bajar la cantidad efectiva de hilos ejecutándose en éste en un instante de tiempo dado. Más allá de las heurísticas que se pueden sugerir para determinar cuál es el equilibrio entre los dos aspectos mencionados la evaluación experimental ofrece un valor concreto y es el camino elegido en este trabajo.

La distribución de NAMD disponible en línea establece por defecto un máximo de 32 registros por hilo en GPU. Para esta parte del trabajo se evalúa el desempeño en GPU para tres valores, usando 32 (valor por defecto), 50 y 63 registros (número máximo). Estas evaluaciones fueron llevadas a cabo sobre ambos casos de pruebas (α7 y APOA1), y para ambos casos se diferencia la evaluación de *patches* remotos de la evaluación de *patches* locales. Si bien en la distribución corriente de NAMD la cantidad de *patches* locales y remotos es la misma, esta diferenciación es de importancia en etapas posteriores, donde la cantidad de *patches* locales y remotos son significativamente diferentes.

A continuación, la Tabla 5.1 presenta los tiempos de ejecución (en milisegundos) de los kernels de NAMD en GPU en función del número de registros.

Caso de prueba	**Cantidad de registros**	**Tiempo en GPU (milisegundos)**	**Patches evaluados**	**Localización del patch**
APOA1	32	46,506	1944	Remote
APOA1	32	45,954	1944	Local
APOA1	50	43,407	1944	Remoto
APOA1	50	42,945	1944	Local
APOA1	63	47,161	1944	Remoto
APOA1	63	46,575	1944	Local
$\alpha 7$	32	124,415	10584	Remoto
$\alpha 7$	32	123,228	10584	Local
$\alpha 7$	50	111,829	10584	Remoto
$\alpha 7$	50	110,869	10584	Local
$\alpha 7$	63	123,522	10584	Remoto
$\alpha 7$	63	122,724	10584	Local

Tabla 5.1: Rendimiento de NAMD en relación a los registros usados en GPU por cada hilo.

En la Tabla 5.1 se puede apreciar que el mejor desempeño de la versión original de NAMD en GPU se alcanza usando 50 registros. En particular se puede notar una mejora cercana al 6 % con el caso de estudio APOA1, mientras que para $\alpha 7$ la mejora alcanza un 11 %.

Modificación leve sobre NAMD

Luego de determinar experimentalmente la mejor cantidad de registros por hilo a usar se aplicaron pequeños cambios en el código fuente original del kernel de GPU para NAMD sin alterar en nada el diseño original de la solución. En detalle, se aplican solo los cambios propuestos en la Sección 5.1, o sea, se sustituye el uso de memoria compartida por el uso de registros. Otros aspectos como pueden ser la estructura principal del kernel, estructuras de control, parámetros de entrada al kernel, transferencias de memoria al kernel, etc. no fueron alteradas.

Para esta prueba también se usaron ambos casos de estudio, pero solo para 50 registros, valor óptimo encontrado a la etapa anterior (se realizaron experimentos no formalizados con otras cantidades de registros pero no se obtuvieron mejores rendimientos). La Tabla 5.2 ilustra los tiempos de ejecución obtenidos para esta alternativa.

Caso de prueba	Kernel utilizado	Tiempo de GPU (milisegundos)	Patches evaluados	Localización del Patch
APOA1	NAMD	43,407	1944	Remoto
APOA1	Optimizado	41,639	1944	Remoto
APOA1	NAMD	42,945	1944	Local
APOA1	Optimizado	41,194	1944	Local
$\alpha 7$	NAMD	111,829	10584	Remoto
$\alpha 7$	Optimizado	108,450	10584	Remoto
$\alpha 7$	NAMD	110,869	10584	Local
$\alpha 7$	Optimizado	107,666	10584	Local

Tabla 5.2: Evaluación de performance en GPU, NAMD versus NAMD optimizado.

La Tabla 5.2 muestra que las pequeñas modificaciones introducidas impactan en el tiempo de ejecución del kernel. Tomando como punto de partida la versión ya optimizada de NAMD (en cuanto a registros) se obtuvo una mejora del 3 % y 4 % para $\alpha 7$ y APOA1, respectivamente.

Una vez obtenido una versión optimizada del NAMD original, a partir de ahora NAMD optimizado, se procede a la evaluación de la propuesta descripta en este trabajo.

5.3.4. Evaluación de la propuesta

En la Tabla 5.3 se detallan los tiempos de ejecución de la propuesta para ambos casos de prueba.

Caso de prueba	Kernel utilizado	Tiempo de GPU (milisegundos)	Patches evaluados	Localización del Patch
APOA1	Optimizado	41,639	1944	Remoto
APOA1	Propuesta	52,115	1512	Remoto
APOA1	Optimizado	41,194	1944	Local
APOA1	Propuesta	19,574	504	Local
$\alpha 7$	Optimizado	108,450	10584	Remoto
$\alpha 7$	Propuesta	140,768	8176	Remoto
$\alpha 7$	Optimizado	107,666	10584	Local
$\alpha 7$	Propuesta	52,110	2800	Local

Tabla 5.3: Evaluación de performance en GPU, NAMD optimizado versus propuesta.

Dos aspectos relevantes se pueden observar en la Tabla 5.3: mientras que en el NAMD optimizado la ejecución de ambos kernels (*patches* remotos y locales) insumen aproximadamente el mismo tiempo la propuesta presenta tiempos de ejecución muy distintos para

ambas invocaciones del kernel. Respecto a esto último, el mayor tiempo insumido en la primera invocación (*patches* remotos) se puede explicar por el hecho que la versión propuesta ejecuta una mayor cantidad de accesos a memoria global para acumular el valor recíproco de las fuerzas. Otro factor a tomar en cuenta, es que la implementación de cálculo del recíproco implica el uso de instrucciones de sincronización. Dichas instrucciones son necesarias para la escritura de los valores recíprocos desde el buffer (memoria compartida) a memoria global, causando parte del *overhead* observado. También, es de orden recordar que en el NAMD original cada átomo para el cuál se calcula la fuerza es leído por todos los hilos al mismo tiempo. En cambio la idea principal de la propuesta radica en que cada hilo lee un átomo diferente, generando quizás un *overhead* por estar solicitando diferentes direcciones de memoria, aunque consecutivas.

Por otra parte, la segunda invocación al kernel implica un tiempo considerablemente menor al del NAMD optimizado. Esto se explica directamente por el hecho que esta invocación implica una cantidad de pares de *patches* que equivale aproximadamente a la tercera parte de los pares implicados en la primera invocación. Como ya se mencionó, la segunda invocación solo comprende aquellos pares conformados por *patches* locales, además de la reducción implícita de pares por el uso del recíproco.

Caso de prueba	**Kernel utilizado**	**Tiempo total de GPU (milisegundos)**	**Patches evaluados en total**
APOA1	Optimizado	82,832	3888
APOA1	Propuesta	71,689	2016
$\alpha 7$	Optimizado	216,116	21168
$\alpha 7$	Propuesta	192,877	10976

Tabla 5.4: Performance total del uso de GPU, NAMD optimizado versus propuesta.

Si se considera el desempeño computacional como la suma del rendimiento de ambas invocaciones (Tabla 5.4), el rendimiento de ambas ejecuciones para el NAMD optimizado para APOA1 es aproximadamente de 82,832 milisegundos, mientras que en la propuesta esta cifra es de 71,689 milisegundos, obteniendo así una mejora de aproximadamente 16 %. Para $\alpha 7$ la performance del NAMD optimizado alcanza los 216,116 milisegundos, mientras que la propuesta insume un rendimiento de 192,877 milisegundos, representando una mejora del 12 %.

5.3.5. Impacto de la propuesta en el desempeño general de la herramienta

Hasta el momento solo se han evaluado los rendimientos de los kernels, es decir, de aquellas porciones de código que sólo se ejecutan en GPU. Corresponde también hacer una valoración del impacto que estos cambios tienen sobre el rendimiento de NAMD en su conjunto.

Caso de prueba	**Kernel utilizado**	**Tiempo total de GPU por paso (milisegundos)**	**Tiempo total de CPU en simulación entera (segundos)**	**Tiempo simulado (femtosegundos)**
APOA1	Optimizado	82,832	94245	1×10^6
APOA1	Propuesta	71,689	82701	1×10^6
$\alpha7$	Optimizado	216,116	2921	1×10^4
$\alpha7$	Propuesta	192,877	2916	1×10^4

Tabla 5.5: Performance total (GPU+*host*), NAMD optimizado versus propuesta.

En la Tabla 5.5 se presenta el desempeño total de NAMD en relación la mejora lograda en el kernel. Para el caso de pruebas APOA1 se aprecia que la propuesta logra una mejora del 14 % con respecto al NAMD ajustado, un porcentaje levemente menor con respecto a la mejora en rendimiento en GPU, pero justificable al haber transferencias de memoria de por medio, tareas de tiempo fijo en CPU, etc. En cambio para el caso de pruebas $\alpha7$ el rendimiento de ambas aplicaciones es casi el mismo. Esto se puede explicar por el hecho que al aumentar el tamaño del sistema (con respecto a APOA1), aumenta la cantidad de cálculos, tanto para GPU como para CPU. Como ya se comentó en el Capítulo 4 la GPU es capaz de escalar de forma lineal con respecto a N, incluso de forma sub-lineal. Como se menciona en el punto 5.3.2, los escenarios de pruebas incluyen PME, lo cual es evaluado enteramente en CPU, y si bien esta técnica insume un tiempo $O(N \log N)$ el costo computacional depende de N. Considerando lo anterior, se sostiene que existe un $N_{APOA1} < N_0 < N_{\alpha7}$ que representa un punto de quiebre respecto a que componente es el "cuello de botella"; el CPU o el GPU. La GPU es un dispositivo diseñado con fines de trabajo de largo aliento, es decir, poca transferencia y mucho cálculo, por esto es que se explica que para un N chico (APOA1) la GPU representa el cuello de botella. Sin embargo para un N más grande el CPU debe afrontar cálculos que escalan con dicha magnitud, generando que este dispositivo sea el cuello

de botella. Determinar cuál es el valor de N_0 excede el alcance de esta tesis, no obstante implica un posible trabajo a futuro que se desprende de la misma.

5.4. Evaluación bioinformática

Hasta el momento se han ponderado solo los aspectos computacionales, pero para demostrar la validez de esta propuesta como solución bioinformática es necesario evaluar otros indicadores para medir la calidad de la solución. Se toman ambos casos de prueba para los cuáles se evalúan indicadores como RMSD y energía total del sistema, junto con esto se presentara un análisis visual de los estados finales para ambos casos. En lo que respecta a APOA1 estos indicadores se evalúan sobre dos simulaciones de 5,5 nanosegundos de duración ($5,5 \times 10^{-9}$ segundos), con un paso de simulación de un femtosegundo (1×10^{-15} segundos), tanto para el NAMD optimizado como para la propuesta de este trabajo. En cambio para $\alpha 7$ los mismos indicadores serán evaluados en una simulación de 2 nanosegundos solo para NAMD optimizado, en la propuesta se presentaron inconvenientes que generaban una simulación incorrecta por lo que no fue tenida en cuenta. No se pudo identificar la causa del error en la propuesta por lo que no se puede brindar mayor información al respecto. Durante la simulación se toma un muestreo de coordenadas del sistema cada 10 picosegundos de simulación (10×10^{-12} segundos), otros indicadores como energía, temperatura, etc. son tomados en todos los pasos.

5.4.1. Temperatura del sistema

Cuando se estudia el comportamiento de un sistema es imperante considerar la temperatura del mismo, ya que en gran medida la temperatura determina el comportamiento de otras variables, entre otros la energía del sistema así como la conformación que dicho sistema toma en el espacio, la cual es evaluada, como se verá en el siguiente punto, mediante el uso de RMSD. Sin embargo esta magnitud es útil cuando otros parámetros son estudiados en función del primero, por lo que no se presentará un estudio "individual" de la temperatura, sino que esta será presentada junto con las magnitudes a continuación.

5.4.2. APOA1

Para APOA1 la simulación está compuesta por dos tramos. El primer tramo corresponde a una simulación de un nanosegundo a bajar energía con una temperatura constante de 187° K. El segundo tramo consiste en someter el sistema a un baño térmico de 298° K, de manera que la temperatura crece desde los 187° K iniciales a la temperatura "ambiente" de 298° K.

RMSD

RMSD (del inglés Root Mean Square Deviation) consiste en estudiar cuanto se diferencia un sistema de coordenadas con respecto a otro mediante mínimos cuadrados. En este caso se toma el conjunto de coordenadas cartesianas para todos los átomos del sistema en el punto inicial de la simulación, los restantes juegos de coordenadas obtenidos durante la simulación son comparados contra este punto de partida.

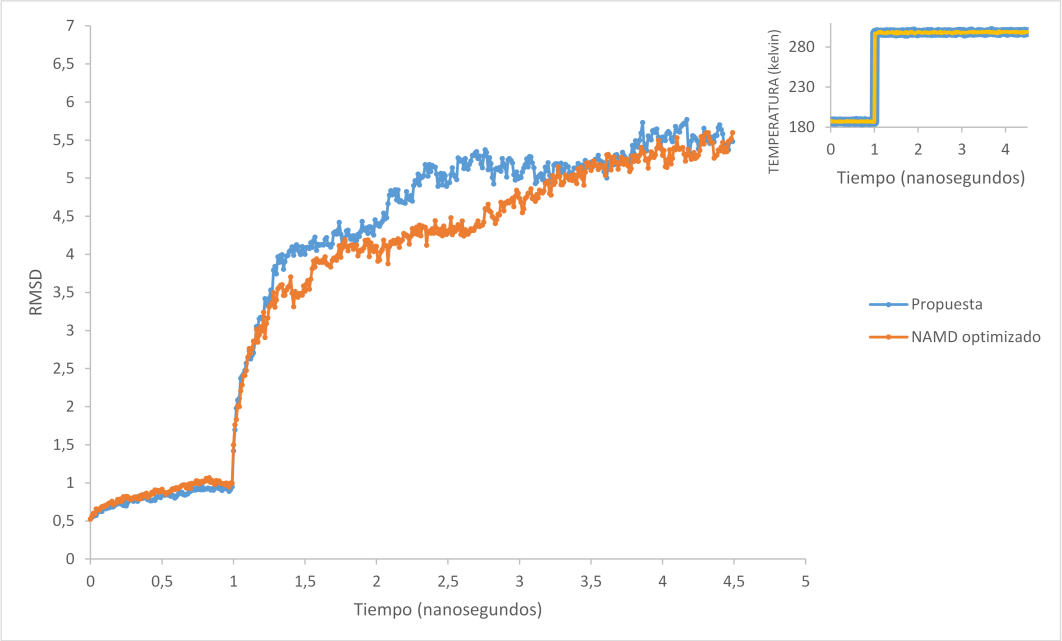


Figura 5.5: RMSD de simulación completa con NAMD optimizado y propuesta sobre complejo APOA1.

En la Figura 5.5 se muestra gráficamente el análisis de RMSD para ambas aplicaciones.

Como se aprecia en la figura, en términos generales, ambas curvas conservan una tendencia creciente, con una pendiente que tiende a disminuir en la primera etapa. En la segunda etapa en cambio se observa un crecimiento abrupto, lo cual se explica porque en este lapso se produce el aumento de temperatura mencionado anteriormente (ver gráfica de temperatura como referencia en la parte superior). Sobre la etapa final la pendiente vuelve a disminuir, producto que el sistema va adquiriendo un equilibrio con respecto a la nueva temperatura. Las Figuras 5.6, 5.7 y 5.8 muestran la gráfica de RMSD para la primer, segunda y tercer etapa identificadas anteriormente.

Sin embargo, el lector podrá notar que luego del primer nanosegundo de simulación y hasta poco después del tercero la curva correspondiente al NAMD optimizado (naranja) el RMSD es menor, con cierta significancia, al de la propuesta (azul) en gran parte del tramo. Esta diferencia numérica puede explicarse por limitaciones en la representación numérica de las computadoras, que son de naturaleza finita.

La representación numérica de números reales trae aparejado limitaciones que pueden manifestarse en situaciones donde hay un alto grado de operaciones aritméticas entre números de diferente orden. En los computadores los números son representados acorde al estándar IEEE 754 [48] para aritmética de punto flotante, para mayor detalle ver Anexo A. Esto explica la variación en el RMSD, ya que la fuerza impacta en las coordenadas, y esta variación en buena parte de los átomos del sistema, causa que el RMSD difiera. No obstante cabe mencionar que esta situación no implica que una estrategía sea correcta y otra no, ya que esta limitante está presente siempre, y no se podría distinguir a priori, cual estrategia sería la más exacta.

A continuación se detallan de manera separada las tres etapas identificadas en esta simulación. El primer nanosegundo de simulación (Figura 5.6) es realizado a bajas temperaturas, a unos 187° K. Como se aprecia el RMSD muestra una leve tendencia creciente al comienzo para luego mantenerse estable, al igual que la temperatura.

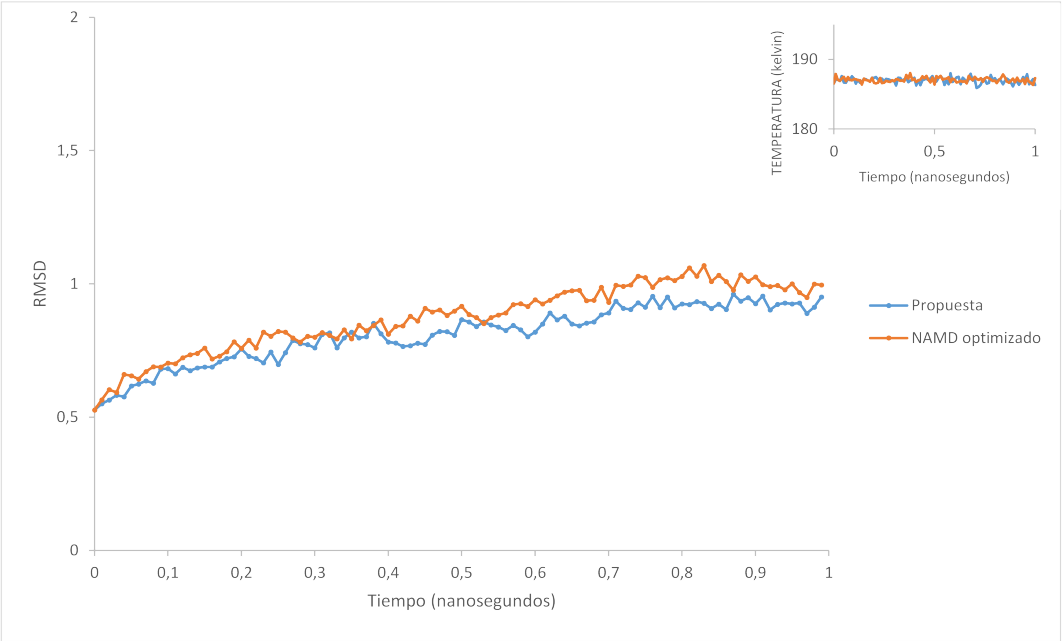


Figura 5.6: RMSD de primer nanosegundo de simulación con NAMD optimizado y propuesta sobre complejo APOA1.

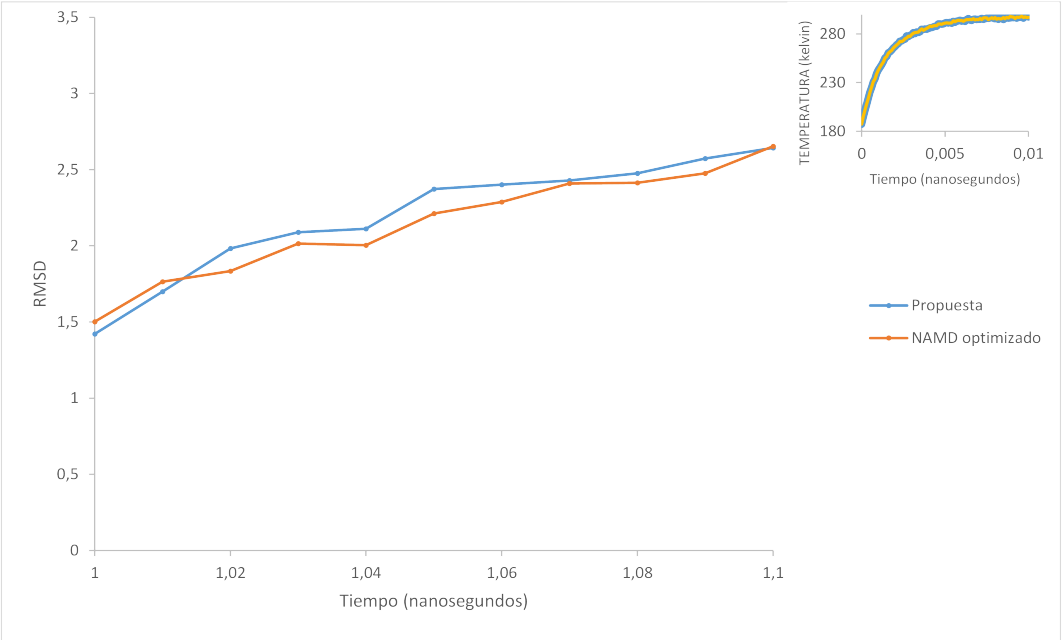


Figura 5.7: RMSD de simulación durante fase de calentamiento con NAMD optimizado y propuesta sobre complejo APOA1.

El segundo tramo (Figura 5.7) consiste en someter el sistema a un baño térmico de 298° K, de manera que la temperatura crece desde los 187° K iniciales a la temperatura "ambiente"

de 298° K. Si bien no se fuerza el aumento abrupto de la temperatura ésta aumenta en tan solo 0,01 nanosegundos, esto explica el segmento vertical que se observa en la gráfica de referencia en la Figura 5.5. En la Figura 5.7 se observa un crecimiento casi lineal del RMSD mientras que la temperatura crece de manera logarítmica en el mismo período. El crecimiento del RMSD se explica debido a que el aumento de la temperatura se traduce en un aumento en las velocidades de los átomos, los cuáles al moverse más rápido alteraran con mayor facilidad las coordenadas cartesianas de los mismos.

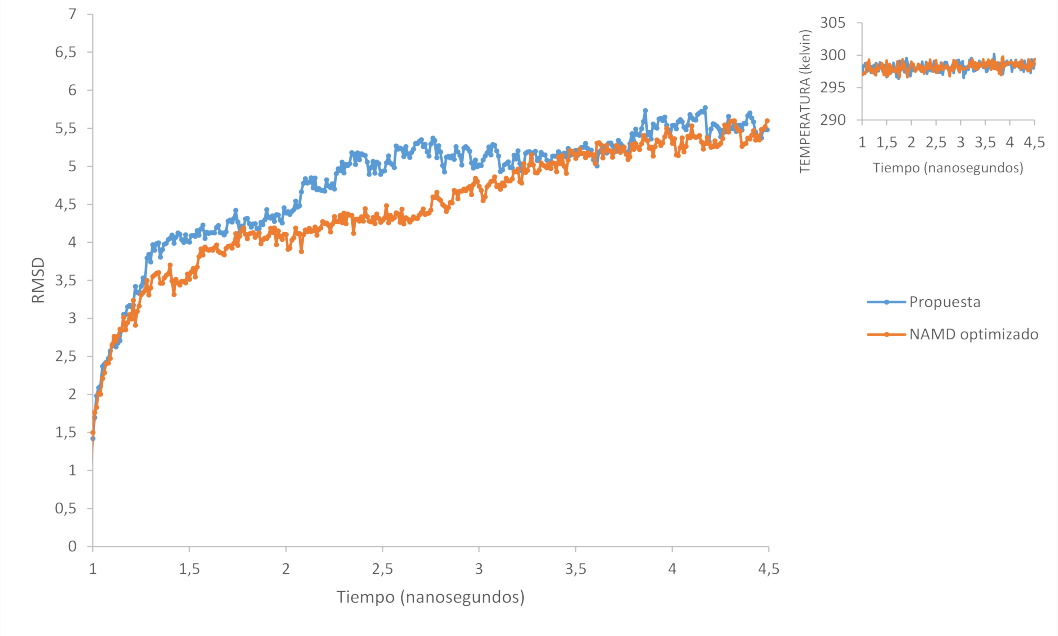


Figura 5.8: RMSD de simulación entre 1 y 4.5 nanosegundos con NAMD optimizado y propuesta sobre complejo APOA1.

En la Figura 5.8 se muestra la última etapa de simulación, en la cual ya se alcanzó la temperatura de 298° K aprox., en este caso la temperatura se mantiene constante mientras que el RMSD crece hasta cierto punto, después del cuál no presenta prácticamente pendiente. Al igual que en la figura anterior el crecimiento del RMSD se explica por la nueva temperatura a lo que se somete el sistema, que si bien ya no aumenta, todavía provoca alteración en la conformación espacial del sistema hasta logrado un equilibrio.

Estudio de la energía total del sistema

Otro parámetro a tomar en cuenta en la validación de la nueva propuesta es la energía total del sistema.

La Figura 5.9 muestra como la energía total del sistema oscila en una franja acotada en el intervalo $(-934940KJ/mol, -935940KJ/mol)$, de manera similar a la temperatura (ver gráfica de referencia). Dado que la temperatura puede ser tomada como un indicativo de la energía de un sistema, ya que una depende de la otra, es razonable ver que ésta oscile de igual manera. Esta razón también sustenta lo que se observa en la Figura 5.10, un cambio abrupto en la energía durante la fase de *acople térmico*, por lo tanto los primeros 10 picosegundos de simulación de la segunda etapa son muestreados aparte al igual que como se hizo con el RMSD.

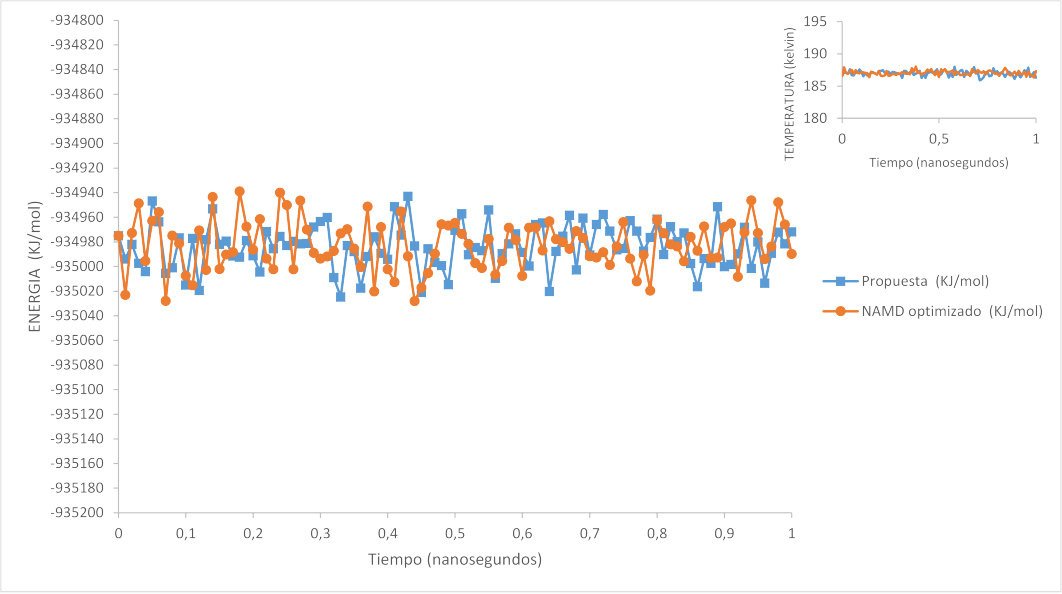


Figura 5.9: Muestreo de la energía total del sistema en primer nanosegundo simulado con NAMD optimizado y propuesta sobre complejo APOA1.

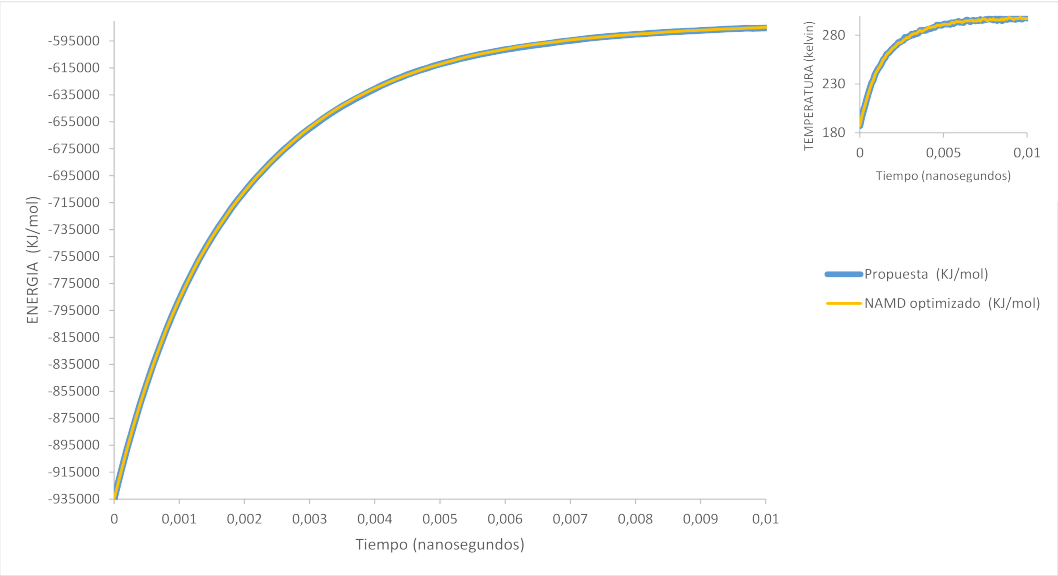


Figura 5.10: Muestreo de la energía total del sistema durante fase de calentamiento con NAMD optimizado y propuesta sobre complejo APOA1.

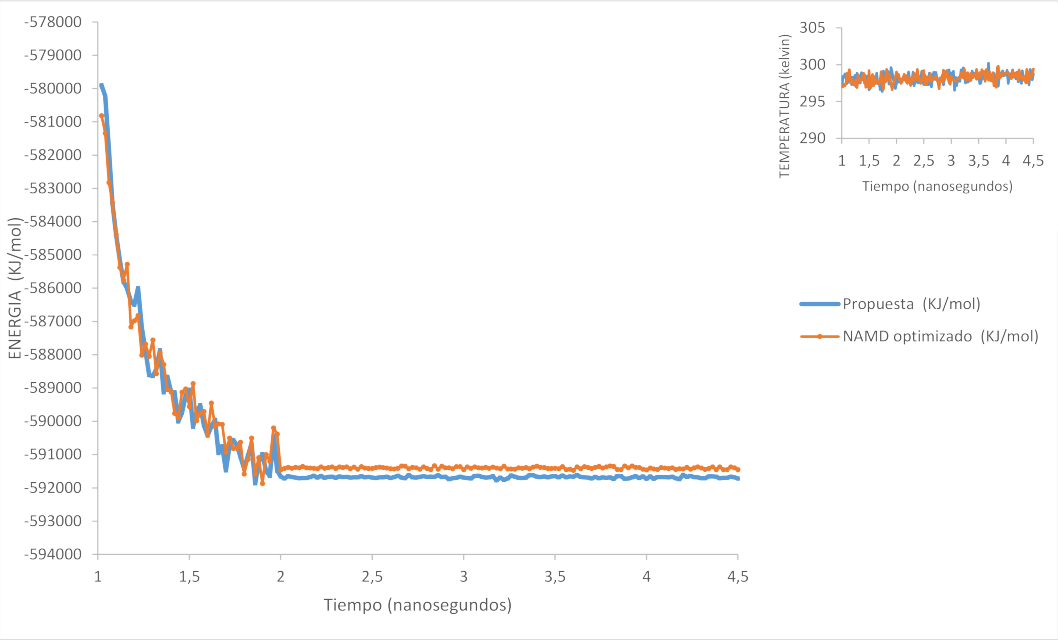


Figura 5.11: Muestreo de la energía total del sistema entre 1 y 4.5 nanosegundos de simulación con NAMD optimizado y propuesta sobre complejo APOA1.

Finalmente, en la Figura 5.11 se observa el resto de la simulación (segunda etapa) donde inicialmente se observa un pico para ambas curvas, el cual se entiende que es producto de la fase de calentamiento. Luego se observa una caída en la energía con una pendiente

leve con una tendencia a estabilizarse sobre el final de la simulación en un valor cercano a $-591500KJ/mol$.

Adicionalmente, a modo ilustrativo también se presentan las Figuras 5.12, 5.13 y 5.14 presentando la energía para las mismas etapas pero en relación al RMSD.

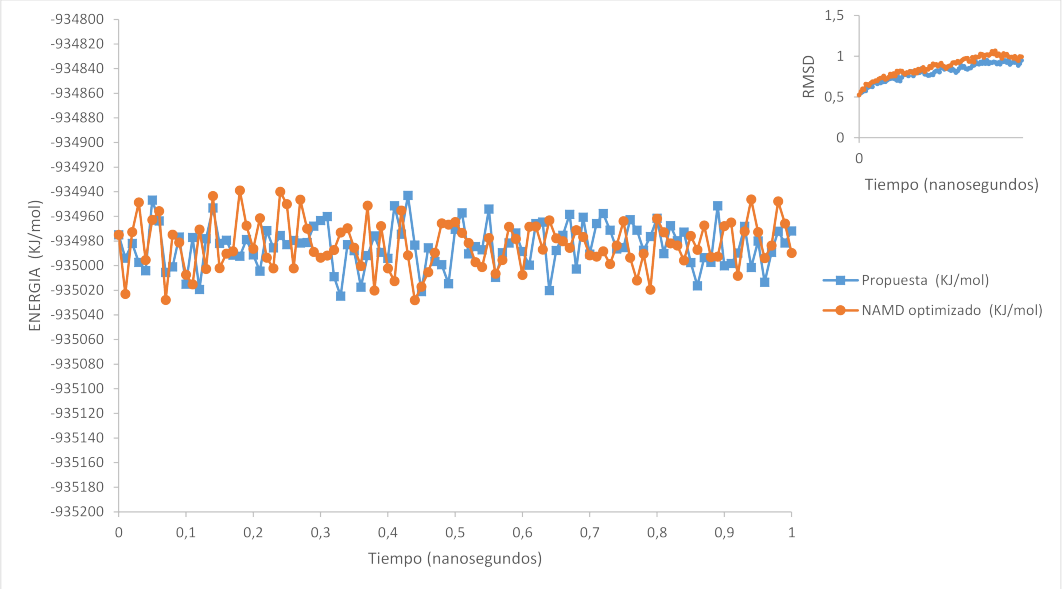


Figura 5.12: Muestreo de la energía total del sistema en el primer nanosegundo simulado con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD.

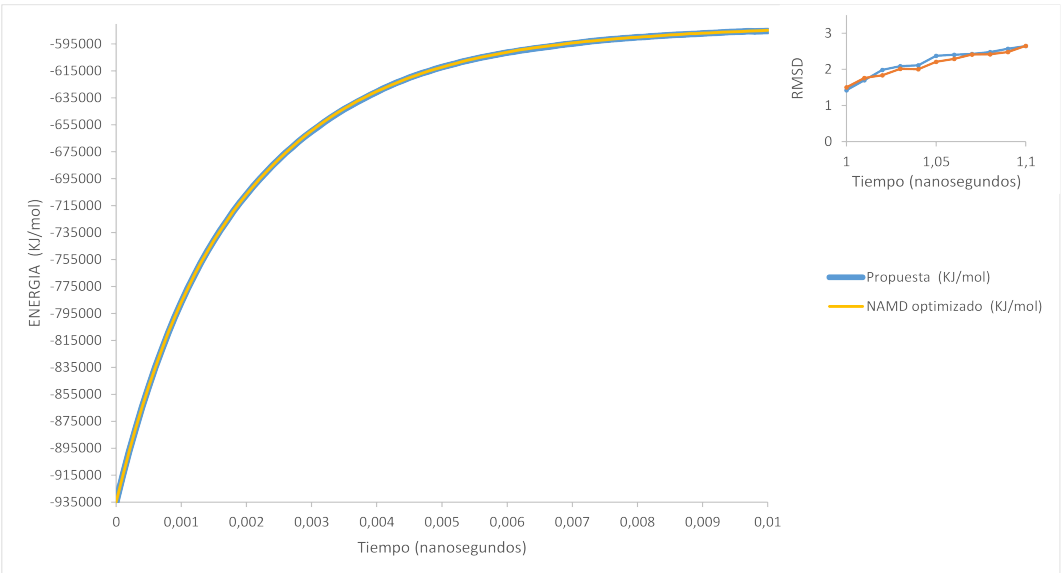


Figura 5.13: Muestreo de la energía total del sistema en fase de calentamiento simulado con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD.

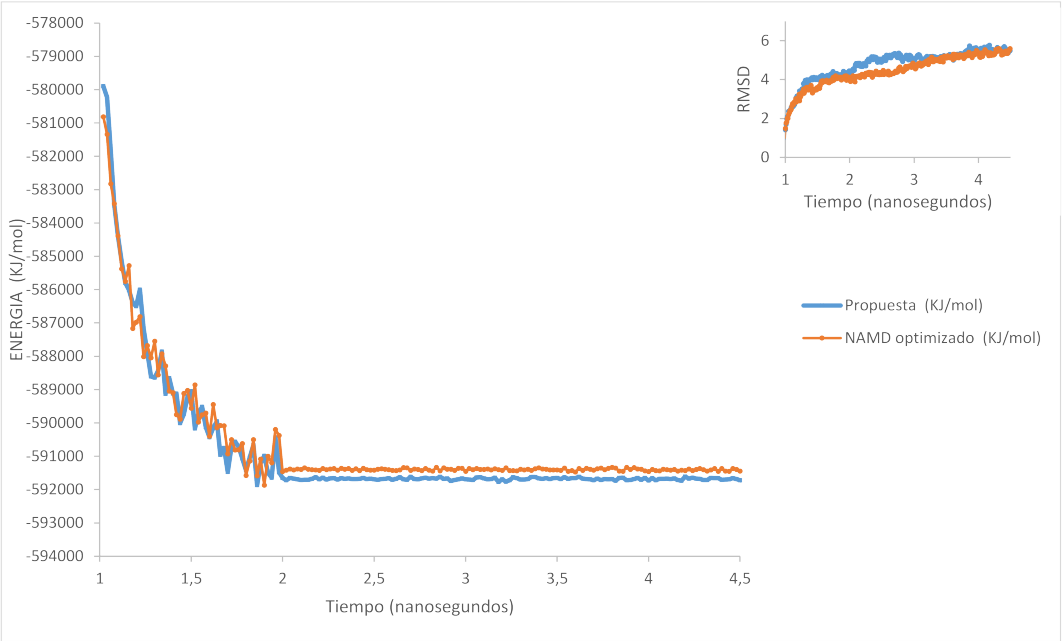


Figura 5.14: Muestreo de la energía total del sistema entre 1 y 4.5 nanosegundos de simulación con NAMD optimizado y propuesta sobre complejo APOA1, tomando como referencia el RMSD.

Análisis gráfico

Como parte final del estudio sobre APOA1 se presenta el estado inicial (Figura 5.15) y los estados finales (Figura 5.16) alcanzados por las simulaciones realizadas tanto para el NAMD optimizado como la propuesta.

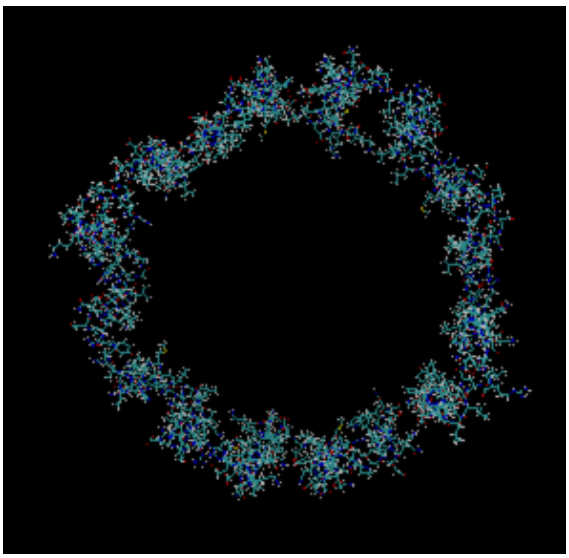


Figura 5.15: Estado inicial de APOA1, para simulación con NAMD optimizado y propuesta

Los estados iniciales no admiten mayor discusión dado que es el mismo en ambos casos, en cambio es de interés analizar el estado final para ambas simulaciones.

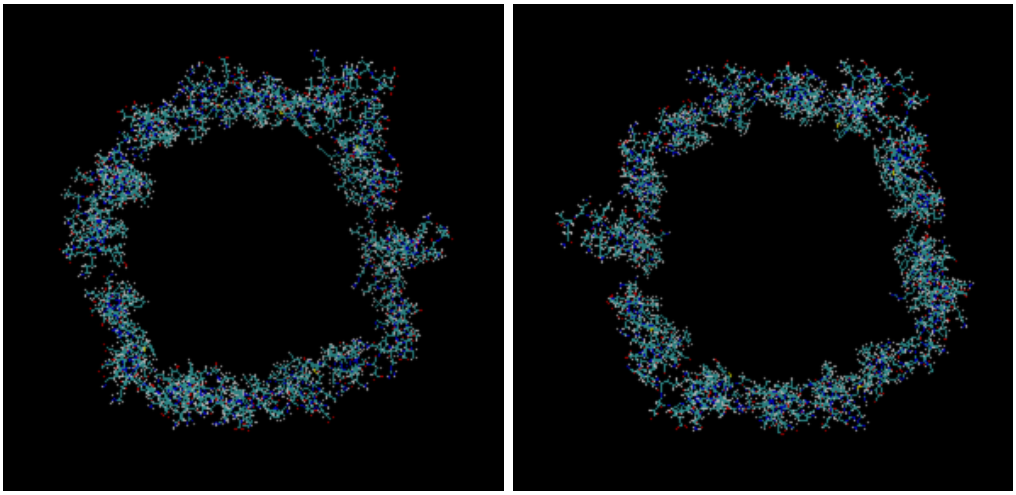


Figura 5.16: Estados finales de APOA1 para NAMD optimizado (izquierda) y propuesta (derecha).

En la Figura 5.16 se observa que los estados finales difieren entre sí, aunque no se puede aseverar que uno es correcto y el otro no. Si se observa en mayor detalle ambas imágenes son similares si se aplica una rotación de 180° grados sobre una de ellas. Esta proteína presenta dos puntos de "unión" entre cadenas, uno de cada lado, por lo que se supone que en un caso una unión se vio modificada y en el otro caso la opuesta es la que se ve alterada. Esta situación se puede explicar por el punto inicial de la simulación, es decir, cuando se arranca una simulación las velocidades y fuerzas iniciales son generadas y asignadas a los átomos de forma aleatoria. Aunque NAMD provee la capacidad de "fijar" el inicio en la práctica esto no sucede, pudiendo causar un recorrido diferente en el espacio de estados del sistema, causando la convergencia a estados diferentes.

5.4.3. α7

Para α7 la simulación está compuesta por dos tramos. El primer tramo corresponde a una simulación de un nanosegundo a baja energía con una temperatura constante de 187° K. El segundo tramo consiste en someter el sistema a un baño térmico de 298° K, como se verá más adelante no se logra, como en el primer caso, un ascenso significativo de la temperatura. Por

esto es que en este caso no es necesario dividir el estudio en etapas ya que la evolución de este sistema no presenta "saltos" en la evolución ya sea de la temperatura, como del RMSD o la energía.

RMSD

A continuación, la Figura 5.17 ilustra la evolución del RMSD para $\alpha 7$, en este caso se observa una tendencia creciente en el crecimiento acompañada por un aumento de la temperatura aunque a menor ritmo.

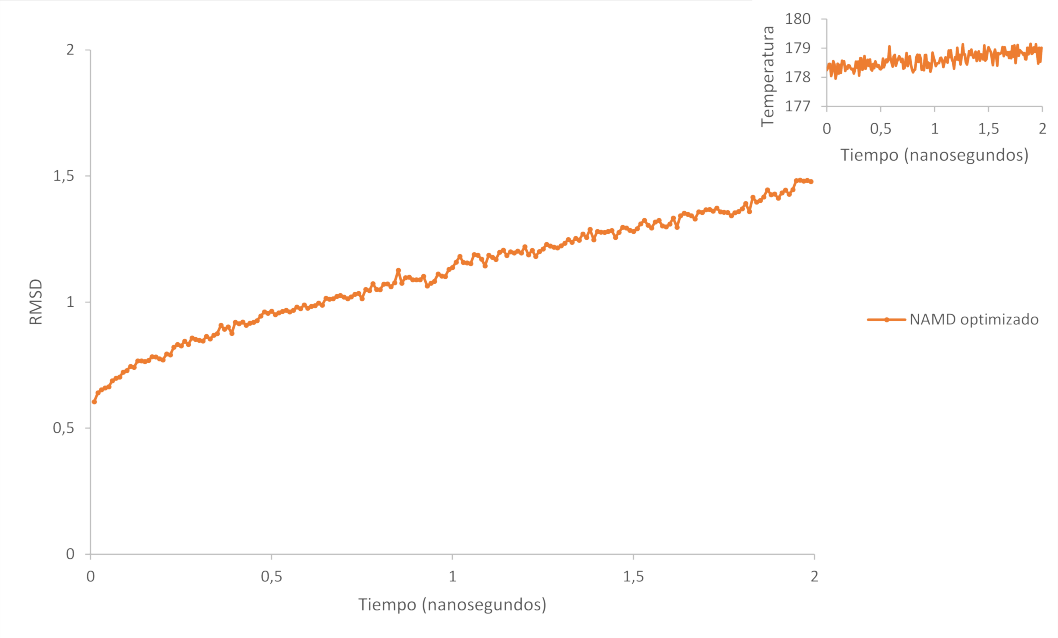


Figura 5.17: RMSD de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$.

Estudio de la energía total del sistema

Como se observa en la Figura 5.18 la energía total se mantiene constante, en el intervalo $(-3621100 KJ/mol, -3621400 KJ/mol)$ mientras la temperatura aumenta levemente.

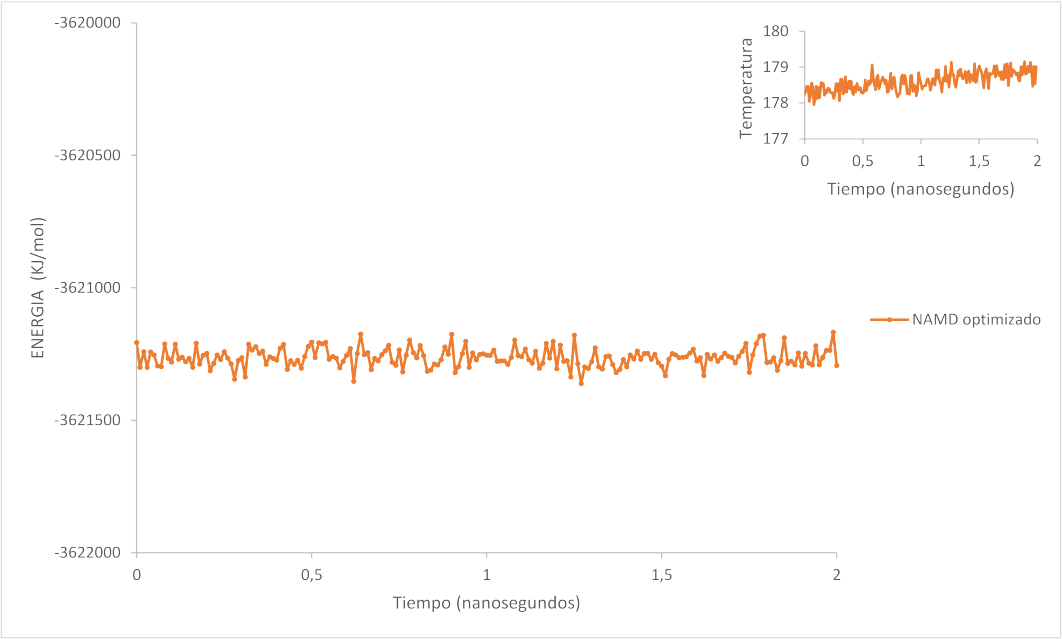


Figura 5.18: Muestreo de la energía total del sistema de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$.

La Figura 5.19 muestra una situación similar a la anterior cuando se presenta la energía poniendo como referencia el RMSD.

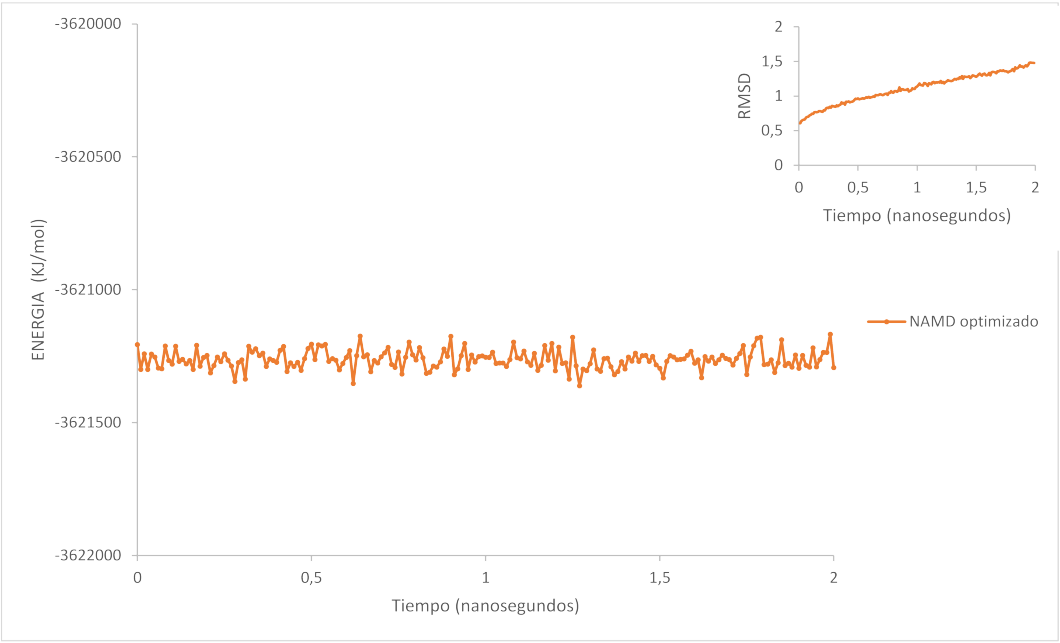


Figura 5.19: Muestreo de la energía total del sistema de simulación de 2 nanosegundos con NAMD optimizado sobre complejo $\alpha 7$ tomando como referencia el RMSD.

Análisis gráfico

Como se mencionó anteriormente para $\alpha 7$ se llevó a cabo la simulación con el NAMD optimizado para validar el modelo. En la Figura 5.20 se presenta el estado inicial y el estado final alcanzado por la simulación realizada.

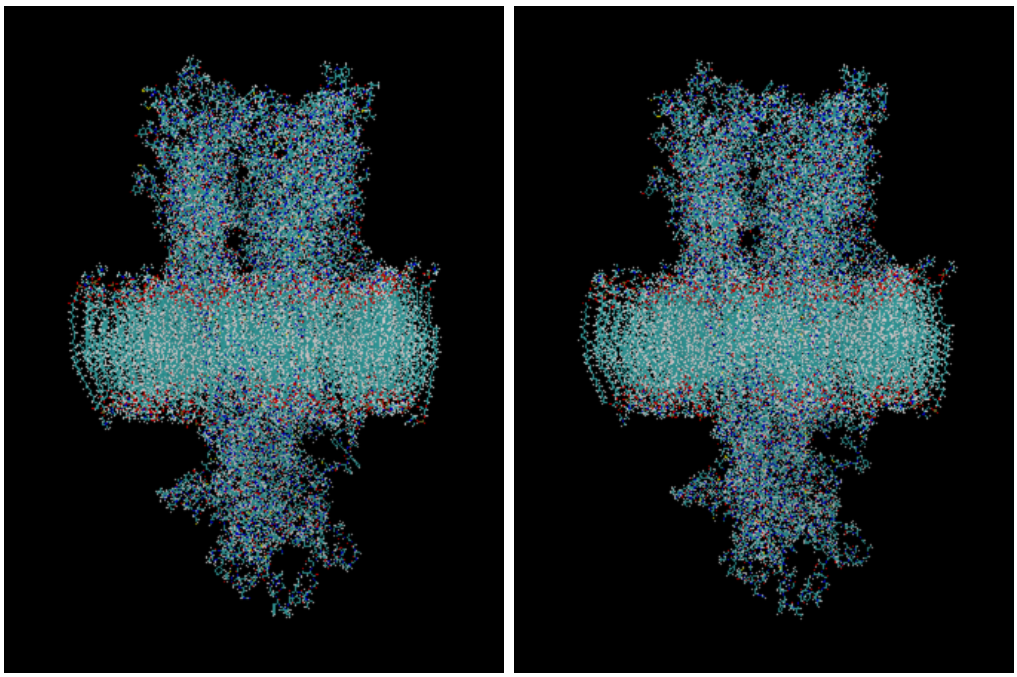


Figura 5.20: Estados inicial (izquierda) y final (derecha) de $\alpha 7$, para simulación con NAMD optimizado.

Como se observa en la Figura 5.20 es difícil percibir los cambios estructurales entre el estado inicial y el final. Si observa de manera rigurosa algunas cadenas en la parte inferior (intracelular) de la proteína difieren levemente, al igual que algunas cadenas en la parte superior (extracelular) de la proteína, cercano a la membrana. En términos generales el análisis gráfico de $\alpha 7$ muestra una gran estabilidad acorde al análisis numérico mencionado anteriormente.

Capítulo 6

Conclusiones y trabajo futuro

Como cierre de este trabajo de tesis se presentan las conclusiones que se desprenden del mismo. En particular, sobre el avance en el conocimiento del uso de GPUs en dinámica molecular y un interesante aporte en lo que respecta a la aplicación de la tercera ley de Newton sobre la versión de GPU de NAMD. También se presentan líneas de trabajo a futuro identificadas durante el desarrollo de este trabajo, donde el ajuste previo al uso de una herramienta de MD se distingue como una de las líneas más interesantes.

6.1. Discusión

La dinámica molecular como disciplina comprende un escenario complejo con muchas variantes. Pretender abarcar o comprender esta temática cabalmente exige mucho conocimiento del tema. Reconociendo esto, es que este trabajo establece una meta muy acotada y conservadora. Se estudian los fundamentos básicos referentes a dinámica molecular para luego evaluar fortalezas y debilidades de las herramientas actuales de dinámica molecular sobre GPU. Todo esto teniendo siempre como referencia central a NAMD, uno de los principales, si no el principal, software de dinámica molecular hoy en día.

Dentro de los aspectos relevados durante el estudio del estado de arte mencionado se identifica la tercera ley de Newton como desafío en este escenario. Por las características inherentes a la GPU, en su arquitectura CUDA, dicha tarea puede ser compleja. El enfoque en programación paralela que brinda esta arquitectura no parece adaptarse sencillamente al

cálculo de la fuerza recíproca. Sin embargo en otros trabajos, como el de Götz et al. [4] se trata una estrategia que concilia ambas cosas; GPU y Tercera ley de Newton. En esta línea, se diseñó e implementó una variante del NAMD y se evaluó con respecto a una versión de NAMD optimizada (con modificaciones leves). Ambas variantes desarrolladas se validaron en forma experimental, obteniendo una mejora de entre un 12 % al 16 % en el rendimiento de la GPU para la propuesta con respecto al NAMD optimizado, también se observa que esta mejora impacta en el rendimiento global de la solución para uno de los casos de estudio con una mejora de un 14 %.

Bajo la premisa que para mejorar una aplicación distribuida una opción es la mejora de sus instancias, este trabajo toma como base experimental un solo computador. Bajo dicho escenario fue posible reducir el tiempo de ejecución de la rutina (kernel) en GPU, y para uno de los dos casos se verificó un impacto en el desempeño del programa en su totalidad. En este aspecto el tamaño del sistema junto con un escenario restringido a un host juega un rol importante. Una GPU, por sus características escala de forma lineal conforme crece N, siendo N el número de átomos, en cambio no una CPU. Particularmente, en NAMD algunos cálculos que escalan de manera supra-lineal con respecto a N se calculan en CPU. Esto explica cómo, para un complejo como APOA1, se logra propagar la mejora hacia CPU, y porqué para $\alpha 7$, un modelo más grande, esto no se logra.

A partir de los resultados obtenidos, se presentó el artículo "Unleashing the Graphic Processing Units-based version of NAMD" a la revista *Transactions on Computational Biology and Bioinformatics* en su número especial *Advanced Parallel Computing Systems*. En este artículo se describe la problemática abordada en este proyecto, y específicamente el trabajo realizado sobre optimización de NAMD y la propuesta anteriormente descrita. Dicho artículo se encuentra en etapa de revisión.

6.2. Trabajo Futuro

Son muchas las variables que fueron limitadas para poder llevar a cabo este trabajo en tiempo y forma. En esto se presentan dos aspectos bien diferenciados: el escenario físico (hardware) y el escenario experimental (bioinformático).

Este trabajo se abordó enteramente sobre una GPU NVIDIA modelo GTX 480, por lo que, evaluar con otras tarjetas gráficas es un punto pendiente. Siendo un poco más ambiciosos, también sería de interés evaluar esta solución en un cluster donde cada nodo esté equipado con (al menos) una GPU y comparar rendimientos.

Respecto al escenario experimental más modelos podrían ser evaluados, variando el número de átomos, uso de PME, condiciones periódicas y radios *cut-off* .

También nuevas líneas de interés fueron encontradas durante este trabajo, como se ve en el desarrollo de la tesis. Por ejemplo alterar algunos valores por defecto (en tiempo de compilación) en NAMD es beneficioso, por lo tanto investigar y desarrollar una herramienta de "tuning" previo al momento de la instalación de NAMD es un aspecto a tener en cuenta.

El estudio llevado a cabo es concluyente para dos sujetos de estudios, APOA1 y $\alpha 7$, pero no para todos, no teniendo argumentos de peso para descartar que la solución provista en NAMD funcione mejor en otros casos. Se debe tomar en cuenta que las GPUs evolucionan muy rápidamente y de dicha evolución puede salir beneficiado uno u otro enfoque. En esta línea, también se propone el estudio de una estrategia de evaluación pre-simulación para elegir uno u otra propuesta según se entienda acorde al modelo y las condiciones de la simulación. Siguiendo la idea anterior y en función de los resultados también es viable explorar la idea de cómo repartir el volumen de cómputo entre GPU y CPU, de manera que en los casos donde la GPU representa el cuello de botella parte del trabajo sea reasignado a CPU.

Anexo A

Representación en punto flotante

En los computadores los números fraccionarios son representados acorde al estándar IEEE 754 [48] para aritmética de punto flotante. Sin entrar en mayor detalle, los números son representados acorde a un signo una mantisa y un exponente, por ejemplo el valor 567,234 será representado como $(+)0,567234 \times 10^4$. Formalmente un número en punto flotante en simple precisión (32 bits)[1] dedica el primer bit al signo, los siguientes 8 al exponente y los restantes 23 bits a la mantisa. Cuando se hace una suma o una resta en esta representación lo primero que se hace es igualar los exponentes, para luego sumar ambos números. Por ejemplo los valores $x = 2560$ e $y = 516000$, corresponden a los valores normalizados $x = 2{,}56 \times 10^3$ e $y = 5,16 \times 10^5$. Si se realiza la suma $x + y$ los exponentes deben ser emparejados, para esto se desplaza el punto hacia a la izquierda del valor de menor exponente hasta que los exponentes se igualen. Retomando el ejemplo $x + y = 2{,}56 \times 10^3 + 5,16 \times 10^5$ será evaluado como $0{,}0256 \times 10^5 + 5,16 \times 10^5$.

Este tipo de notación tiene sus limitantes, representar números fraccionarios en un espacio de memoria acotado puede conducir a errores, en este caso es de relevancia discutir el error denominado *shiftout*. Este error se presenta cuando se suman o se restan números de exponentes con tal grado de desigualdad, que la mantisa del número de menor exponente pierde dígitos al ser desplazada a la derecha. En el ejemplo a continuación se establece un caso de suma donde, para simplificar, la mantisa de un número no tiene más de 9 dígitos, siendo $x = 2565,8762$ e $y = 516000,932$.

[1] Representando los números en binario

$$\begin{array}{rll} & 0,0025658762 \times 10^{6} & x_{norm} \\ + & 0,516000932 \times 10^{6} & y_{norm} \\ \hline & 0,518566808 \times 10^{6} & x+y \end{array}$$

Como se puede ver en el ejemplo anterior, el corrimiento de la mantisa hacia la derecha en x provoca la pérdida del dígito menos significativo, provocando que este error se propague al total. Volviendo al tema principal, la explicación de la variación en el RMSD, se debe tener presente que un ensamble del sistema obtenido en paso n de simulación depende del ensamble en el paso $n-1$. Las coordenadas de los átomos del sistema en el paso n dependen de la fuerza aplicada sobre los mismos en el paso $n-1$, la cual se calcula mediante la evaluación de un átomo con respecto a sus vecinos. Dado esto, se extiende la suma anterior a tres términos, donde cada uno representa la fuerza ejercida por un átomo b_i sobre el átomo a. Para este caso los valores no normalizados son $F(a,b_1) = 2565,8762$, $F(a,b_2) = 516000,932$ y $F(a,b_3) = 3615,3869$.

$$\begin{array}{rll} & 0,0025658762 \times 10^{6} & F(a,b_1)_{norm} \\ + & 0,516000932 \times 10^{6} & F(a,b_2)_{norm} \\ \hline & 0,518566808 \times 10^{6} & \\ + & 0,0036153869 \times 10^{6} & F(a,b_3)_{norm} \\ \hline & 0,522182194 \times 10^{6} & \end{array}$$

Este ejemplo fue elegido para ilustrar explícitamente que, dependiendo del conjunto de números involucrados, pueden existir errores de precisión. Suponiendo que la suma anterior fue realizada en el mismo orden que se realizan en el NAMD original, o sea, comenzando siempre por el mismo átomo b_0, se propone la suma de los mismos términos alterando su orden. Esto tal cuál sucede cuando utilizamos la propuesta basada en NAMD, que comienza a recorrer los átomos desde el átomo b_i.

$$
\begin{array}{lll}
 & 0,36153869 \times 10^4 & F(a,b_3)_{norm} \\
+ & 0,25658762 \times 10^4 & F(a,b_1)_{norm} \\
\hline
 & 0,61812631 &
\end{array}
$$

$$
\begin{array}{lll}
 & 0,006181263\cancel{1} \times 10^6 & \text{(se ajusta la mantisa)} \\
+ & 0,516000932 \times 10^6 & F(a,b_2)_{norm} \\
\hline
 & 0,522182195 \times 10^6 &
\end{array}
$$

Si se compara el resultado obtenido entre una suma y otra se puede apreciar que el dígito menos significativo varía de 4 en primer caso a 5 en el segundo. Esto es un ejemplo sencillo, pero si se extrapola esta suma de tres términos a una suma de cientos de estos, las diferencias propagadas serán mayor.

Bibliografía

[1] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *J. Comp. Chem.*, vol. 26, pp. 1781–802, 2005.

[2] U. o. I. Theoretical Biophysics Group, Beckman Institute. (1999) NAMD User's Guide. [Online]. Available: http://www.ks.uiuc.edu/Research/namd/2.9/ug/

[3] J. M. Menéndez, "Sumas electrostáticas en sistemas bidimensionales: Fundamentos, algoritmos y aplicaciones," Ph.D. dissertation, Universidad de Oviedo, Jun 2002.

[4] A. W. Götz, M. J. Williamson, D. Xu, D. Poole, S. L. Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. generalized born," *J. Chem. Theory and Compu.*, vol. 8, pp. 1542–1555, 2012.

[5] W. Humphrey, A. Dalke, and K. Schulten, "VMD – Visual Molecular Dynamics," *Journal of Molecular Graphics*, vol. 14, pp. 33–38, 1996.

[6] S. A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC bioinformatics*, vol. 9, no. Suppl 2, p. S10, 2008.

[7] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney, "High-throughput sequence alignment using Graphics Processing Units," *BMC bioinformatics*, vol. 8, no. 1, p. 474, 2007.

[8] P. Klus, S. Lam, D. Lyberg, M. S. Cheung, G. Pullan, I. McFarlane, G. S. Yeo, and B. Y. Lam, "BarraCUDA - a fast short read sequence aligner using graphics processing units," *BMC research notes*, vol. 5, no. 1, p. 27, 2012.

[9] P. D. Vouzis and N. V. Sahinidis, "GPU-BLAST: using graphics processors to accelerate protein sequence alignment," *Bioinformatics*, vol. 27, no. 2, pp. 182–188, 2011.

[10] B. Sukhwani and M. C. Herbordt, "GPU acceleration of a production molecular docking code," in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 2009, pp. 19–27.

[11] D. W. Ritchie and V. Venkatraman, "Ultra-fast FFT protein docking on graphics processors," *Bioinformatics*, vol. 26, no. 19, pp. 2398–2405, 2010.

[12] M. Simonsen, C. N. Pedersen, M. H. Christensen, and R. Thomsen, "GPU-accelerated High-accuracy Molecular Docking Using Guided Differential Evolution: Real World Applications," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 1803–1810.

[13] G. Rizk and D. Lavenier, "GPU accelerated RNA folding algorithm," *Computational Science–ICCS 2009*, pp. 1004–1013, 2009.

[14] D.-J. Chang, C. Kimmer, and M. Ouyang, "Accelerating the Nussinov RNA folding algorithm with CUDA/GPU," in *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on*, Dec 2010, pp. 120–125.

[15] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, "Accelerating Molecular Modeling Applications with Graphics Processors," *J. Comp. Chem.*, vol. 28, pp. 2618–2640, 2007.

[16] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," *J. Comp. Phys.*, vol. 227, no. 10, pp. 5342–5359, 2008.

[17] M. J. Harvey, G. Giupponi, and G. D. Fabritiis, "ACEMD: Accelerating bio-molecular dynamics in the microsecond time-scale," *J. Chem. Theory and Compu.*, vol. 5, p. 1632, 2009.

[18] P. K. Weiner and P. A. Kollman, "AMBER: Assisted model building with energy refinement. A general program for modeling molecules and their interactions," *Journal of Computational Chemistry*, vol. 2, no. 3, pp. 287–303, 1981.

[19] "GROMACS: A message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, no. 1-3, pp. 43 – 56, 1995.

[20] D. Kirk and W. Hwu, *Programming Massively Parallel Processors, Second EDITION: A Hands-on Approach.* Morgan Kaufmann, 2012.

[21] R. Hockney, "The potential calculation and some applications." *Methods Comput. Phys.*, vol. 9, pp. 136–211, Jan 1970.

[22] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters," *The Journal of Chemical Physics*, vol. 76, no. 1, 1982.

[23] "Some multistep methods for use in molecular dynamics calculations," *Journal of Computational Physics*, vol. 20, no. 2, pp. 130 – 139, 1976.

[24] C. Chipot, *Métodos numéricos en dinámica molecular*, Nancy Univesité, B.P. 239, 54506 Vandoeuvre-lès-Nancy, France, 2007.

[25] B. R. Brooks, C. L. B. III, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. B. A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, "CHARMM: The Biomolecular simulation Program," *J. Comp. Chem.*, vol. 30, pp. 1545–1615, 2009.

[26] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman, "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules," *J. Am. Chem. Soc.*, vol. 117, pp. 5179–5197, 1995.

[27] W. F. van Gunsteren and H. J. C. Berendsen, *Groningen Molecular Simulation (GROMOS) Library Manual*, BIOMOS b.v., Groningen, The Netherlands, 1987.

[28] C. C. Valcárcel, "Simulación mediante métodos híbridos clásico-cuánticos de la relajación vibracional de moléculas en disolución," Ph.D. dissertation, Universidad de Murcia, Sep 2005.

[29] L. Verlet, "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Phys. Rev.*, vol. 159, pp. 98–103, Jul 1967.

[30] R. Hockney, S. Goel, and J. Eastwood, "Quiet high-resolution computer models of a plasma," *Journal of Computational Physics*, vol. 14, no. 2, pp. 148–158, 1974.

[31] R. Carraro, "Modelado y estudio de complejos de ciclosporina a y compuestos relacionados con una ciclofilina de *Trypanosoma cruzi*," Ph.D. dissertation, Universidad de la República, 2012.

[32] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, 1996.

[33] P. Allen and D. Tildesley, *Computer Simulation of Liquids*. Clarendon, 1987.

[34] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An N.log(N) method for Ewald sums in large systems," *The Journal of Chemical Physics*, vol. 98, no. 12, pp. 10 089–10 092, 1993.

[35] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Trans. Comput.*, vol. 21, no. 9, pp. 948–960, Sep. 1972.

[36] F. Darema, "The SPMD Model: Past, Present and Future," in *Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual*

Machine and Message Passing Interface. London, UK, UK: Springer-Verlag, 2001, p. 1.

[37] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[38] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig, "Molecular Dynamics Simulations on Commodity GPUs with CUDA," in *Proceedings of the 14th International Conference on High Performance Computing*, ser. HiPC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 185–196.

[39] J. van Meel, A. Arnold, D. Frenkel, S. P. Zwart, and R. Belleman, "Harvesting graphics power for MD simulations," *Molecular Simulation*, vol. 34, pp. 259–266, 2008.

[40] C. R. Trott, L. Winterfeld, and P. S. Crozier, "General-purpose molecular dynamics simulations on GPU-based clusters," *Arxiv prEPRINT arXiv:1009.4330*, 2010.

[41] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande, "Accelerating Molecular Dynamic Simulation on Graphics Processing Units," *J. Comp. Chem.*, vol. 30, pp. 864–872, 2009.

[42] D. Hilbert, "Ueber die stetige abbildung einer line auf ein flächenstück," *Mathematische Annalen*, vol. 38, no. 3, pp. 459–460, 1891.

[43] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker, "Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald," *Journal of Chemical Theory and Computation*, vol. 9, no. 9, pp. 3878–3888, 2013.

[44] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," *SIGPLAN Not.*, vol. 28, no. 10, pp. 91–108, Oct. 1993.

[45] C. Chipot and D. A. Pearlman, "Free Energy Calculations. The Long and Winding Gilded Road," *Molecular Simulation*, vol. 28, no. 1-2, pp. 1–12, 2002.

[46] J. C. Phillips, J. E. Stone, and K. Schulten, "Adapting a Message-driven Parallel Application to GPU-accelerated Clusters," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 8:1–8:9.

[47] J. D. Schmitt, "Exploring the nature of molecular recognition in nicotinic acetylcholine receptors." *Current Medicinal Chemistry*, vol. 7, no. 8, pp. 749–800, 2000.

[48] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.