Universidad Zaragoza
1542

Proyecto Fin de Carrera

Implementación e integración en GROMACS de un algoritmo eficiente y preciso para imponer ligaduras en simulaciones de dinámica molecular

Autora:

María Astón Serrano Gracia

Directores:

Pablo García Risueño

Institut für Physik und IRIS Adlershof
Humboldt Universität zu Berlin

Jesús Alastruey Benedé

Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Ingeniería informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2012/2013

Implementación e integración en GROMACS de un algoritmo eficiente y preciso para imponer ligaduras en simulaciones de Dinámica Molecular

RESUMEN

Las simulaciones de dinámica molecular describen la evolución en el tiempo de un sistema de partículas. Herramientas computacionales para realizar dichas simulaciones son fundamentales para los investigadores de campos tan diversos como la medicina (búsqueda de tratamientos para enfermedades como el Alzheimer, la fibrosis quística o el cáncer; diseño computacional de medicamentos), la química (diseño de catalizadores) o la ingeniería de materiales. GROMACS es un extendido y versátil paquete de dinámica molecular escrito en el lenguaje de programación C. Originalmente desarrollado en la Universidad de Groningen (Países Bajos), actualmente se mantiene por desarrolladores de universidades y centros de investigación de todo el mundo, y cuenta con miles de usuarios.

Para realizar simulaciones eficientes, una opción común es imponer ligaduras, es decir, restricciones a la longitud de los enlaces atómicos o al valor de los distintos ángulos entre átomos. Esto permite incrementar el paso temporal (*time step*) de las simulaciones y con ello lograr realizar simulaciones más duraderas, con lo que se incrementa el poder predictivo y el realismo de la simulación. Los dos algoritmos más utilizados para la implementación de ligaduras, los denominados SHAKE y LINCS (*LINear Constraint Solver*), presentan limitaciones a la hora de imponer ligaduras en los ángulos. Además, están basados en aproximaciones, lo cual afecta negativamente a su exactitud, eficiencia y estabilidad numérica.

El presente proyecto fin de carrera se basa en la reciente propuesta teórica del Dr. Pablo García Risueño, la cual propone mejorar la imposición de ligaduras en las simulaciones de dinámica molecular mediante el uso de cálculos analíticos. Por lo tanto, el objetivo de este proyecto es implementar el algoritmo denominado ILVES-S, propuesta alternativa al algoritmo SHAKE, e integrarlo en el paquete de dinámica molecular GROMACS.

AGRADECIMIENTOS

Con este proyecto se cierra una importante etapa y muchas son las personas que han pasado por ella, quisiera que todas se sintieran agradecidas pero algunas se merecen una especial mención.

Primero, quiero agradecer el apoyo y la inspiración que he recibido de las personas con las que he trabajado en este proyecto y con las que tanto he aprendido: el Dr. Pablo García Risueño, el Dr. Jesús Alastruey Benedé y el Dr. Carl Christian Kjelgaard Mikkelsen.

Gracias a todos mis amigos por los buenos momentos inolvidables.

Víctor, gracias por estar ahí siempre y ser mi gran apoyo.

Finalmente, gracias a mi familia por todo el cariño que recibo de todos ellos pero muy especialmente gracias a mis padres por su lucha y sacrificio continuo para que mis hermanos y yo tengamos un futuro mejor.

Índice general

Capítulo 1

INTRODUCCIÓN

1.1 Contexto del proyecto y motivación

La dinámica molecular es una técnica de simulación por ordenador en la que átomos y moléculas interactúan durante un período de tiempo, dando lugar a una descripción visual y detallada del movimiento de las partículas. Estas simulaciones se basan en aproximaciones de la física conocida y se utilizan con frecuencia en el estudio de muchas moléculas bioquímicas como proteínas, lípidos y ácidos nucleicos. La dinámica molecular permite a los científicos escudriñar en el movimiento de los átomos de un modo que no es viable en experimentos de laboratorio: es posible tomar instantáneas de las estructuras permitiendo el acceso a todas las escalas de tiempo y movimiento con resolución atómica.

Debido a la complejidad de los sistemas biológicos, estas simulaciones conllevan un alto consumo de recursos, tanto de memoria como de capacidad de procesamiento. Es por tanto habitual la utilización de ordenadores de gran rendimiento y con grandes capacidades para la computación paralela o distribuida. Gracias a los avances tecnológicos esta es una ciencia en constante cambio y sobre la que se buscan mejoras en el rendimiento y la precisión de los resultados de las simulaciones.

Una técnica muy común que permite realizar simulaciones más eficientes es aumentar el paso temporal (*time step*) cuyos valores típicos son del orden de un femtosegundo (10^{-15} s). Este valor puede ser ampliado si se usan algoritmos como SHAKE [1], que permiten imponer ligaduras para fijar las vibraciones de los átomos más rápidos. El presente proyecto pretende conseguir mejorar las simulaciones de dinámica molecular mediante el estudio del algoritmo SHAKE y la posterior implementación de un algoritmo alternativo, denominado ILVES-S, más eficiente y estable.

Este proyecto surge como fruto de la colaboración entre el físico Doctor Pablo García Risueño (Humboldt Universität zu Berlin, Alemania) y el grupo de Arquitectura de Computadores de la Universidad de Zaragoza. El trabajo desarrollado parte de la investigación realizada por el Dr. Pablo García Risueño. Además, en el desarrollo del proyecto colabora el investigador Dr. Carl Christian Kjelgaard Mikkelsen (Universidad de Umea, Suecia), matemático

especializado en cálculos analíticos y resolución eficiente de sistemas de ecuaciones lineales de banda.

1.2 Objetivos

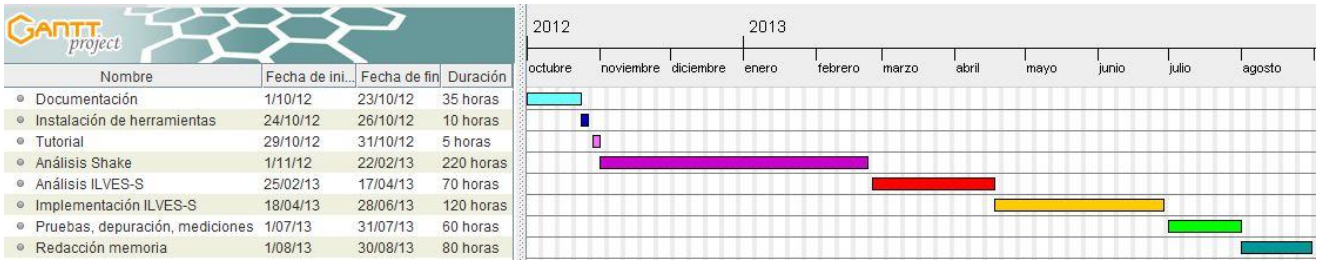
El objetivo principal de este proyecto es mejorar la precisión y estabilidad de SHAKE, un algoritmo utilizado en paquetes software de dinámica molecular, mediante la implementación de una versión alternativa: ILVES-S. Para ello, primero es necesario adquirir ciertos conocimientos básicos de dinámica molecular además de la realización de un análisis exhaustivo de algunas de las funcionalidades que ofrece un paquete específico de dinámica molecular: GROMACS [2].

Por lo tanto las tareas a realizar en este proyecto son las siguientes:

- Revisión bibliográfica: lectura de referencias sobre dinámica molecular, algoritmos de implementación de ligaduras (SHAKE y LINCS), y propuesta a implementar (ILVES-S).
- Análisis de la implementación y la ejecución del algoritmo SHAKE en el paquete de dinámica molecular GROMACS.
- Implementación de ILVES-S.
- Integración de ILVES-S en GROMACS.
- Evaluación de resultados.

1.3 Planificación y trabajo realizado

La siguiente figura muestra el diagrama de Gantt correspondiente a la ejecución del proyecto:



Nombre	Fecha de ini...	Fecha de fin	Duración
Documentación	1/10/12	23/10/12	35 horas
Instalación de herramientas	24/10/12	26/10/12	10 horas
Tutorial	29/10/12	31/10/12	5 horas
Análisis Shake	1/11/12	22/02/13	220 horas
Análisis ILVES-S	25/02/13	17/04/13	70 horas
Implementación ILVES-S	18/04/13	28/06/13	120 horas
Pruebas, depuración, mediciones	1/07/13	31/07/13	60 horas
Redacción memoria	1/08/13	30/08/13	80 horas

Figura 1. Diagrama de Gantt de la ejecución del proyecto

El tiempo total de ejecución del proyecto es de, aproximadamente, 600 horas. Las diferentes tareas desarrolladas que aparecen en el diagrama se describen a continuación.

1. **Documentación**: lectura de varios artículos de divulgación científica con temática de física y química computacional. Las lecturas realizadas son:

   - ✓ Artículo sobre el que se basa el proyecto: "*Constraint implementation based on analytical calculations: a possible way to improve widely used solvers*" [3].

- ✓ Algoritmo de imposición de ligaduras SHAKE: "*Numerical integration of the cartesian equations of motion of a system with constraints; molecular dynamics of n-alkanes*" [1].
- ✓ Algoritmo de imposición de ligaduras LINCS: "*LINCS: A Linear Constraint Solver for Molecular Simulations*" [4].

2. **Instalación de las herramientas y librerías** necesarias para la ejecución de las distintas tareas del proyecto:
   - ✓ GROMACS [2]: Software de dinámica molecular.
   - ✓ FFT (*Fast Fourier Transform*) [5]: librería de subrutinas para calcular la transformada discreta de Fourier, utilizada por GROMACS.
   - ✓ UCSF Chimera [6]: programa que permite visualizar y analizar las estructuras moleculares y datos relacionados con las mimas.
   - ✓ Avogadro [7]: Editor de moléculas
   - ✓ GNU DDD (*Data Display Debugger*) [8]: Interfaz gráfica para usar con depuradores basados en línea de comandos, como por ejemplo, GDB.
   - ✓ LAPACK [9]: librería software que proporciona rutinas para la resolución de sistemas de ecuaciones lineales, entre otros.
   - ✓ DOXYGEN[10]: herramienta para la generación automática de documentación.
3. **Tutorial GROMACS:** Toma de contacto con el software GROMACS y ejecución del primer experimento de dinámica molecular [11].
4. **Análisis SHAKE:** Análisis del código fuente de GROMACS, familiarización con el estilo de codificación de GROMACS e identificación de las ecuaciones teóricas con su implementación.
5. **Análisis ILVES-S:** Análisis de las matrices dispersas (ANEXO III) y planteamiento del nuevo algoritmo.
6. **Implementación ILVES-S:** Implementación en el lenguaje de programación C del nuevo algoritmo de tratamiento de ligaduras así como su integración en el paquete GROMACS.
7. **Pruebas, depuración, mediciones:** Pruebas y depuración de la implementación realizada y obtención de mediciones y resultados para comparar las simulaciones de dinámica molecular usando ambos algoritmos: SHAKE e ILVES-S.
8. **Redacción de la memoria**

1.4 Organización del documento

El contenido de este documento está organizado del siguiente modo:

- En el capítulo 2 se desarrolla el concepto de dinámica molecular, se analiza la base teórica de las simulaciones de dinámica molecular y se expone la propuesta teórica en la que se basa este proyecto.

- En el capítulo 3 se presenta el paquete de software GROMACS y el análisis realizado sobre el código fuente del algoritmo SHAKE.

- El capítulo 4 contiene la explicación sobre el algoritmo implementado, ILVES-S.

- En el capítulo 5 se recogen los principales resultados obtenidos.

- El capítulo 6 contiene las conclusiones.

Además, en los anexos se recoge la siguiente información:

- Anexo I: Extensión del apartado “Campos de fuerza” dentro del capítulo 1.

- Anexo II: Desarrolla la ejecución de una simulación de dinámica molecular

- Anexo III: Tarea de análisis de las matrices dispersas típicas en simulaciones de dinámica molecular.

- Anexo IV: Trabajo desarrollado durante estancia de verano en el *Ireland's High-Performance Computing Centre* (ICHEC) en Dublín, Irlanda.

Capítulo 2

DINÁMICA MOLECULAR

El propósito de este capítulo es explicar en qué consiste una simulación de dinámica molecular, centrándose, al final, en el algoritmo de imposición de ligaduras SHAKE, y la propuesta teórica del algoritmo alternativo ILVES-S.

2.1 Simulaciones de dinámica molecular

Ecuación del movimiento

El objetivo de las simulaciones de dinámica molecular (Molecular Dynamics - MD) es describir la evolución en el tiempo de un sistema de partículas. Para obtener la trayectoria del sistema se resuelve la ecuación clásica de movimiento según la segunda ley de Newton:

$$m_i \frac{\partial^2 \boldsymbol{x}_i(t)}{\partial t^2} = \boldsymbol{F}_i(t)$$

Ecuación 1. Ecuación del movimiento según la segunda ley de Newton

donde $\boldsymbol{x}_i(t)$ es el vector posición (x_x, x_y, x_z) de la partícula i en el instante temporal t y $\boldsymbol{F}_i(t)$ es la fuerza que actúa sobre la partícula i-ésima en el instante temporal t debida a la interacción con el resto de partículas del sistema. Por último, m_i es la masa de la partícula.

Campos de fuerza (*force fields*)

La siguiente fórmula permite calcular la fuerza $\boldsymbol{F}_i$ que actúa sobre cada átomo:

$$\boldsymbol{F}_i = -\frac{\partial V(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)}{\partial \boldsymbol{x}_i}$$

Ecuación 2. Fuerza que actúa sobre un átomo

donde $V(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$ representa la energía potencial por la interacción de los N átomos, como una función de sus posiciones $\boldsymbol{x}_i = (x_x, x_y, x_z)$, y se especifica mediante los denominados campos de fuerza (*force fields*). Un campo de fuerza hace referencia a la forma y a los parámetros de las funciones matemáticas utilizadas para describir la energía potencial de un

sistema de partículas (típicamente moléculas y átomos). Encontrar un potencial realístico que modele adecuadamente la verdadera energía del sistema es un problema que no es trivial, pero permite significativas simplificaciones computacionales. En el Anexo I puede encontrarse más información sobre las ecuaciones usadas para describir el campo de fuerzas típico de las simulaciones de dinámica molecular.

Algoritmos de integración numérica (numerical integrators)

Recordemos que el objetivo de las simulaciones de dinámica molecular es resolver las ecuaciones del movimiento (véase ECUACIÓN 1) para así obtener una trayectoria del sistema en función del tiempo. Por ahora hemos desglosado la parte correspondiente a la fuerza que actúa sobre cada átomo (parte derecha de dicha ecuación). Ahora es necesario saber cómo calcular las posiciones y velocidades de cada átomo en el instante $t + \Delta t$ cuando dichos valores son conocidos en el instante t (Δt es conocido como el paso temporal, *time step*). Para ello, se hace uso de algoritmos numéricos (*numerical integrators*). Los algoritmos más utilizados son el algoritmo del salto de la rana (*leap-frog algorithm*) y el algoritmo de velocidad-Verlet (*velocity Verlet algorithm*).

Leap frog integrator

El algoritmo del salto de la rana [12] usa las posiciones x_i en el instante t y las velocidades v_i en el instante $t - \frac{1}{2}\Delta t$. Se actualizan las posiciones y velocidades usando las fuerzas $\boldsymbol{F}_i(t)$ determinadas por la posición en el instante t:

$$\boldsymbol{v}_i(t + \frac{1}{2}\Delta t) = \boldsymbol{v}_i(t - \frac{1}{2}\Delta t) + \frac{\Delta t}{m_i}\boldsymbol{F}_i(t)$$

$$\boldsymbol{x}_i(t + \Delta t) = \boldsymbol{x}_i(t) + \Delta t\boldsymbol{v}_i(t + \frac{1}{2}\Delta t)$$

Ecuación 3. Ecuaciones del algoritmo del salto de la rana

La FIGURA 2 muestra de forma gráfica como se va alternado el cálculo de las velocidades y las posiciones:

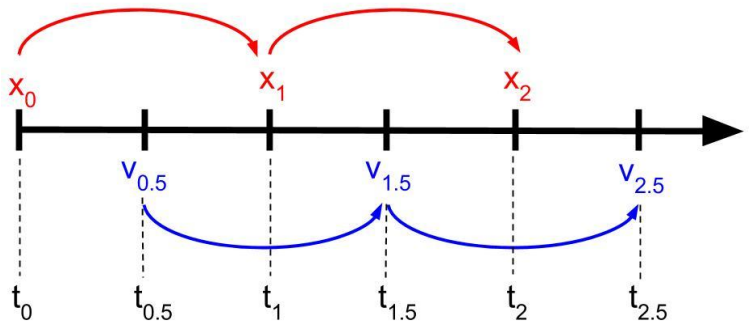


Figura 2. Leap frog integrator

Velocity Verlet integrator

En el algoritmo de velocidad-Verlet [13] las posiciones y velocidades en instante t se usan para integrar las ecuaciones de movimiento del siguiente modo:

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + \Delta t \boldsymbol{v}_i(t) + \frac{\Delta t^2}{2m_i}\boldsymbol{F}_i(t)$$

$$\boldsymbol{v}_i(t+\Delta t) = \boldsymbol{v}_i(t) + \frac{\Delta t}{2m_i}[\boldsymbol{F}_i(t) + \boldsymbol{F}_i(t+\Delta t)]$$

Ecuación 4. Ecuaciones del algoritmo de velocidad-Verlet

Ligaduras

La efectividad de las simulaciones de dinámica molecular está fuertemente limitada por los pequeños pasos temporales (*time steps,* Δt) que deben ser usados: la alta frecuencia de los movimientos internos en las moléculas hace que el paso temporal estándar para una simulación (sin restricciones) sea del rango de 1 fs ($10^{-5}s$). Esto es así porque, para seguir fielmente el movimiento de los átomos, es preciso un paso temporal, al menos, 5 veces menor que el periodo de vibración más pequeño de los átomos que componen el sistema. Muchos de los procesos físicos y químicos que se desean simular se mueven en el rango de los segundos (o más) por lo que el paso temporal limita o incluso imposibilita la simulación de dichos procesos. El paso temporal se puede aumentar si se incorporan restricciones, denominadas ligaduras, a los grados de libertad internos más rápidos. Restringirlos supone congelarlos, hacer que desaparezcan de las ecuaciones, con la esperanza (justificada por la experiencia) de que ésta supresión no afecte al cálculo de las magnitudes de interés del sistema.

La imposición de ligaduras implica que nuevas fuerzas aparezcan en el sistema, las denominadas *fuerzas de ligadura*. Esto quiere decir que con la imposición de ligaduras aparecen nuevas ecuaciones en el conjunto de las ecuaciones clásicas del movimiento de Newton. Por lo tanto, el nuevo sistema de ecuaciones diferenciales es:

$$m_i \frac{\partial^2 \boldsymbol{x}_i(t)}{\partial t^2} = \boldsymbol{F}_i(t) - \sum_{k=1}^{N_c} \lambda_k \frac{\partial \boldsymbol{\sigma}_k}{\partial \boldsymbol{x}_i}(t)$$

Ecuación 5. Ecuaciones del movimiento modificadas

donde el significado de las variables es:

N	Número de átomos ($i = 1..N$)
N_c	Número de ligaduras impuestas al sistema
m_i	Masa del átomo i
$\boldsymbol{x}_i(t)$	Posición del átomo i en el instante t
$\boldsymbol{F}_i(t)$	Fuerza sobre el átomo i en el instante t

$-\sum_{k=1}^{N_c} \lambda_k \frac{\partial \boldsymbol{\sigma}_k}{\partial \mathbf{x}_i}(t)$	Fuerza de ligadura sobre el átomo i en el instante t
λ_k	Multiplicador de Lagrange asociado a la ligadura k
$\boldsymbol{\sigma}_k$	Ligadura k

La existencia de las N_c ligaduras convierte un sistema de $3N$ ecuaciones diferenciales con $3N$ incógnitas en un sistema de $3N + N_c$ ecuaciones diferenciales con $3N + N_c$ incógnitas. Estas ecuaciones se resuelven en dos fases: en primer lugar se calculan las nuevas posiciones de los átomos sin tener en cuenta las ligaduras impuestas al sistema y en segundo lugar se calculan las fuerzas de ligadura y se corrigen las posiciones. Por lo tanto, una vez resueltas las ecuaciones clásicas del movimiento de Newton, las ecuaciones por las ligaduras son las nuevas ecuaciones que se deben resolver y, los multiplicadores de Lagrange, son las nuevas incógnitas cuyos valores han de ser encontrados para resolver el sistema.

Las ligaduras en los enlaces o en los ángulos de enlace pueden ser expresadas del siguiente modo:

$$\boldsymbol{\sigma}_{k(i,j)}(x) \coloneqq (\boldsymbol{x}_i - \boldsymbol{x}_j)^2 - (a_{i,j})^2 = 0$$

Ecuación 6. Ecuación que define una ligadura

donde $a_{i,\mathrm{j}}$ es una constante (distancia entre los átomos i y j) y $\sigma_{k(i,\mathrm{j})}$ representa la ligadura k que implica a los átomos i y j.

Un algoritmo clásico para el tratamiento de las ligaduras es el algoritmo SHAKE.

Algoritmo SHAKE

El algoritmo SHAKE [1] corrige el conjunto de coordenadas calculadas sin aplicar las ligaduras, $\boldsymbol{x}_i'(t + \Delta t)$, por un conjunto de coordenadas $\boldsymbol{x}_i(t + \Delta t)$ que cumplen con la lista de ligaduras, usando las coordenadas $\boldsymbol{x}_i(t)$ como referencia.

En el algoritmo SHAKE se calcula una aproximación γ_k de los multiplicadores de Lagrange λ_k, resolviendo la siguiente ecuación para cada una de las ligaduras (desde 1 hasta N_c):

$$\begin{aligned} -2\Delta t^2 \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)\left(\sum_{k=1}^{N_c} \gamma_k \left(\frac{\nabla_i \boldsymbol{\sigma}_k}{m_i}(t) - \frac{\nabla_j \boldsymbol{\sigma}_k}{m_j}(t)\right)\right) \\ + \Delta t^4 \sum_{k=1}^{N_c}\sum_{k'=1}^{N_c} \gamma_k \gamma_{k'} \left(\frac{\nabla_i \boldsymbol{\sigma}_k}{m_i}(t) - \frac{\nabla_j \boldsymbol{\sigma}_k}{m_j}(t)\right)\left(\frac{\nabla_i \boldsymbol{\sigma}_{k'}}{m_i}(t) - \frac{\nabla_j \boldsymbol{\sigma}_{k'}}{m_j}(t)\right) \\ = (a_{i,j})^2 - \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)^2 \end{aligned}$$

Ecuación 7. Sistema de N_c ecuaciones para obtener los multiplicadores de Lagrange

Todos los términos excepto γ_k son conocidos y:

$$\nabla_i \boldsymbol{\sigma}_{k(l,m)} = 2(\boldsymbol{x}_l - \boldsymbol{x}_m)(\delta_{i,l} - \delta_{i,m})$$

donde $\delta_{i,l}$ y $\delta_{i,m}$ representan la delta de Kronecker, función matemática de dos variables, que vale 1 si son iguales, y 0 si son diferentes:

$$\delta_{\alpha,\beta} = \begin{cases} 1, & \alpha = \beta \\ 0, & \alpha \neq \beta \end{cases}$$

Ecuación 8. Delta de Kronecker

Se tiene un sistema de N_c ecuaciones cuadráticas en γ_k que se puede resolver de forma iterativa. En cada iteración it se resuelve, por lo tanto, el siguiente sistema lineal:

$$A\gamma^{(it)} = b,$$

$$A_{kk'} := -2\Delta t^2 \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)\left(\frac{\nabla_i \boldsymbol{\sigma}_{k'}}{m_i}(t) - \frac{\nabla_j \boldsymbol{\sigma}_{k'}}{m_j}(t)\right),$$

$$b := \left(a_{i,j}\right)^2 - \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)^2$$

Ecuación 9. Sistema lineal para obtener de forma iterativa λ_k.

La resolución de este sistema lineal $A\gamma = b$ se lleva a cabo en SHAKE de forma aproximada, resolviendo en primer lugar la primera ecuación, después la segunda y así sucesivamente. La resolución de la ecuación k-ésima garantiza la satisfacción de la ligadura k-ésima una vez refrescadas las posiciones; pero asimismo, rompe parcialmente la satisfacción de las ligaduras con índices menores a k. Se obtiene primero $\gamma_0^{(0)}$, considerando que $\gamma_1^{(0)}$ … $\gamma_k^{(0)}$ son iguales a cero en la primera iteración. En las sucesivas iteraciones se usan los γ_k obtenidos en la iteración anterior.

De esta forma, una vez obtenidas las aproximaciones de los multiplicadores de Lagrange, se sustituyen en la ECUACIÓN 10 para conseguir las nuevas coordenadas corregidas. La posición $\boldsymbol{x}_i(t+\Delta t)$ del átomo i en el instante $t+\Delta t$ se obtiene a partir de la posición $\boldsymbol{x}_i'(t+\Delta t)$ del siguiente modo:

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i'(t+\Delta t) - \frac{\Delta t^2}{m_i}\sum_{k=1}^{N_c} \gamma_k(t) \nabla_i \boldsymbol{\sigma}_k\left(\boldsymbol{x}_i(t)\right)$$

Ecuación 10. Ecuación para obtener las posiciones corregidas del los átomos

Si las nuevas posiciones corregidas no cumplen las restricciones dentro de una tolerancia relativa, `shake_tol`, se realiza una nueva iteración para obtener nuevos multiplicadores de Lagrange.

El objetivo de este proyecto es mejorar este algoritmo resolviendo el proceso de obtención de las aproximaciones de los multiplicadores de Lagrange de forma más precisa. Es decir, se usan técnicas de resolución de sistemas lineales con matrices banda para resolver el sistema lineal $A\gamma = b$.

Esquema del algoritmo completo de dinámica molecular

A continuación se muestra un esquema general del algoritmo de dinámica molecular:

ALGORITMO GLOBAL DE DINÁMICA MOLECULAR

1. Input: Condiciones iniciales

Energía potencial $V(\mathrm{x})$ del sistema,
Posiciones x_i de todos los átomos del sistema,
Velocidades v_i de todos los átomos del sistema

⇩

---

repetir los pasos 2, 3, 4, 5, 6 para el número de pasos (*time steps*) requeridos

---

2. Calcular fuerzas

Para cada átomo se calcula $\boldsymbol{F}_i(t) = -\frac{\partial V(\boldsymbol{x}_i,\ldots,\boldsymbol{x}_N)}{\partial \boldsymbol{x}_i}$ sin tener en cuenta las restricciones

⇩

3. Actualizar configuración: algoritmo de integración numérica

3.1 Calcular velocidades $\boldsymbol{v}_i(t+\frac{1}{2}\Delta t) = \boldsymbol{v}_i(t-\frac{1}{2}\Delta t) + \frac{\Delta t}{m_i}\boldsymbol{F}_i(t)$

3.2 Calcular posiciones $\boldsymbol{x}_i'(t+\Delta t) = \boldsymbol{x}_i(t) + \Delta t\boldsymbol{v}_i(t+\frac{1}{2}\Delta t)$

⇩

4. Algoritmo Shake

---

repetir 4.1 y 4.2 hasta que todas $\boldsymbol{x}_i(t+\Delta t)$ cumplan las restricciones dentro de una tolerancia

---

4.1 Resolver el sistema lineal $A\gamma = b$ (A y b dependen de las posiciones actualizadas $\boldsymbol{x}_i'(t+\Delta t)$) para obtener los multiplicadores de Lagrange γ

4.2 Calcular posiciones corregidas $\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i'(t+\Delta t) - \frac{\Delta t^2}{m_i}\sum_{k=1}^{N_c}\gamma_k(t)\nabla_i\boldsymbol{\sigma}_k\big(\boldsymbol{x}(t)\big)$

⇩

5. Corregir velocidades de acuerdo a las nuevas posiciones

⇩

6. Output

Escribir las coordenadas, energía… en los ficheros que recogen la trayectoria

2.2 Propuesta teórica: ILVES-S

El Dr. Pablo García Risueño propone un algoritmo alternativo al algoritmo SHAKE, denominado ILVES-S [3]. La motivación para el desarrollo de este nuevo algoritmo es que SHAKE puede ser fuente de inestabilidades y tener problemas de convergencia. Estos problemas son especialmente importantes si se imponen ligaduras a los ángulos entre enlaces (*bond angles*). El proceso para resolver el sistema lineal puede ser más preciso si se usan técnicas de resolución de matrices banda, ya que la matriz A (véase ECUACIÓN 9) es una matriz dispersa para las moléculas biológicas y puede convertirse en una matriz banda reordenando las ligaduras apropiadamente. Además, el formalismo ofrece la posibilidad de imponer ligaduras en ángulos diedros, algo que actualmente no está contemplado con el algoritmo SHAKE.

Por lo tanto, se esperan alcanzar mejoras en la eficiencia, exactitud y estabilidad de las simulaciones de dinámica molecular usando el algoritmo ILVES-S para resolver de forma analítica el sistema lineal de la ECUACIÓN 9.

Capítulo 3

GROMACS

En este capítulo se introduce el paquete software de dinámica molecular GROMACS y se expone el análisis realizado del algoritmo SHAKE. Como ejemplo para el análisis se presenta la glicina, el aminoácido más pequeño.

3.1 Introducción

GROMACS (*GROningen MAchine for Chemical Simulations*) [2] es un versátil paquete de dinámica molecular. Es utilizado, entre otras cosas, para simular las ecuaciones newtonianas (*F = ma*) de movimiento para sistemas con cientos de millones de partículas.

GROMACS fue originalmente desarrollado en la Universidad de Groningen (Países Bajos) pero en la actualidad se mantiene por desarrolladores de diversas universidades y centros de investigación (Suecia, Alemania, EEUU, Australia) [14]. Las rutinas de dinámica molecular están escritas en el lenguaje de programación C y basadas en el programa *GROMOS*, desarrollado por el mismo grupo de la Universidad de Groningen.

GROMACS es uno de los paquetes de software más rápido y popular disponibles y se puede ejecutar tanto en CPUs como en GPUs. Es de código abierto y libre, publicado bajo una Licencia Pública General de GNU (*General Public License*). GROMACS está incluido en SPEC CPU 2006 [18], un conjunto de programas intensivos en cálculo que se utilizan para medir y comparar el rendimiento sistemas informáticos.

Además es ampliamente utilizado en los denominados proyectos de computación distribuida como *Folding@home* [15], de la Universidad de Stanford, o *EvoGrid* [16]. También es utilizado como una de las aplicaciones que forman parte del *ScalaLife Competence Center* [17]. Proyectos como el *Folding@home* contribuyen en la investigación de enfermedades como el alzhéimer, la fibrosis quística o el cáncer. Para ello, se realizan simulaciones del plegamiento de proteínas[1]. La función biológica de una proteína depende de su correcto plegamiento. Si una proteína no se pliega correctamente no será funcional y, por lo tanto, no será capaz de cumplir su función biológica. Las simulaciones permiten determinar las causas de un mal plegamiento.

[1] Proceso por el que una proteína alcanza su estructura tridimensional final.

En menor medida, estos proyectos también son capaces de determinar la estructura final de una proteína o cómo éstas interactúan con otras moléculas, lo que tiene aplicaciones útiles en el diseño computacional de medicamentos.

3.2 Análisis del algoritmo SHAKE en GROMACS.

La implementación del algoritmo ILVES-S se desarrolla en el paquete software GROMACS debido a su extendido uso y la importante repercusión que puede tener. Para ello es necesario realizar un análisis de una simulación de dinámica molecular primero, y del código del actual algoritmo que implementa la imposición de ligaduras (SHAKE) después. La ejecución detallada de una simulación de dinámica molecular puede encontrarse en el Anexo II.

Para realizar este análisis es necesario recompilar GROMACS y modificar su fichero de configuración (`configure`) para cambiar algunos flags de compilación. Se elimina el flag `-O3` y se añaden los flags `-O0` y `-g`. Esto permite depurar el código usando el depurador GNU DDD [8]. Además también es necesario eliminar el flag `-fomit-frame-pointer` puesto que es incompatible con el flag `-g`.

A continuación se detalla el análisis realizado del código de GROMACS que implementa el algoritmo SHAKE. Este análisis es importante para entender como se ha llevado a cabo la implementación del algoritmo que se va a sustituir y también para familiarizarse con el estilo de codificación y las variables de GROMACS.

Datos de entrada

Dado que en GROMACS se implementan dos algoritmos de tratamiento de ligaduras (SHAKE y LINCS), es posible elegir uno de ellos mediante un parámetro en el fichero de entrada donde se establecen los parámetros de la simulación: el fichero `.mdp` (Ver Anexo II):

`constraint_algorithm = SHAKE` ó `LINCS`

Otros parámetros interesantes que también se incluyen en este fichero son:

`integrator`	Algoritmo de integración numérica
`nsteps`	Número máximo de pasos temporales (*time steps*) de la simulación
`dt`	Valor del paso temporal en picosegundos
`shake_tol`	Tolerancia para el algoritmo SHAKE (por defecto su valor es 0,0001)
`constraints`	Tipo de ligaduras que se implementan

Este último parámetro es muy importante ya que indica que tipo de ligaduras se desean imponer. Existen cinco posibles valores para este parámetro:

`none`	No se imponen ligaduras
`hbonds`	Se imponen ligaduras sólo a los enlaces que involucran átomos de Hidrógeno
`all-bonds`	Se imponen ligaduras a todos los enlaces

`h-angles`	Se imponen ligaduras a todos los enlaces y además a los ángulos que involucran átomos de Hidrógeno
`all-angles`	Se imponen ligaduras a todos los enlaces y a todos los ángulos

La glicina

Para el análisis de la implementación del algoritmo SHAKE en GROMACS se realizan simulaciones de la glicina (el aminoácido más sencillo) en su estado zwitterion. Es un residuo con 10 átomos y 9 enlaces. La FIGURA 3 muestra la composición y una representación del compuesto escogido.

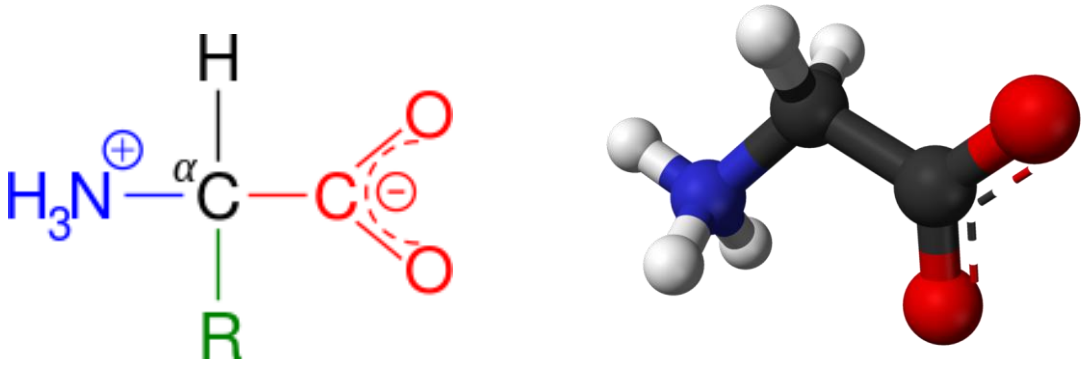


Figura 3. Representación de la Glicina en su estado zwitterion

La composición de la glicina es la siguiente:

- Átomo azul: Nitrógeno
- Átomos blancos: Hidrógeno
- Átomos negros: Carbono
- Átomos rojos: Oxígeno

La FIGURA 4 muestra la glicina tal y como se almacena en GROMACS: los números de átomos y enlaces se utilizan en el acceso a los vectores que contienen su descripción (masas, posiciones, velocidad, tipo…) y el tipo de enlace determina la ligadura asociada al mismo.

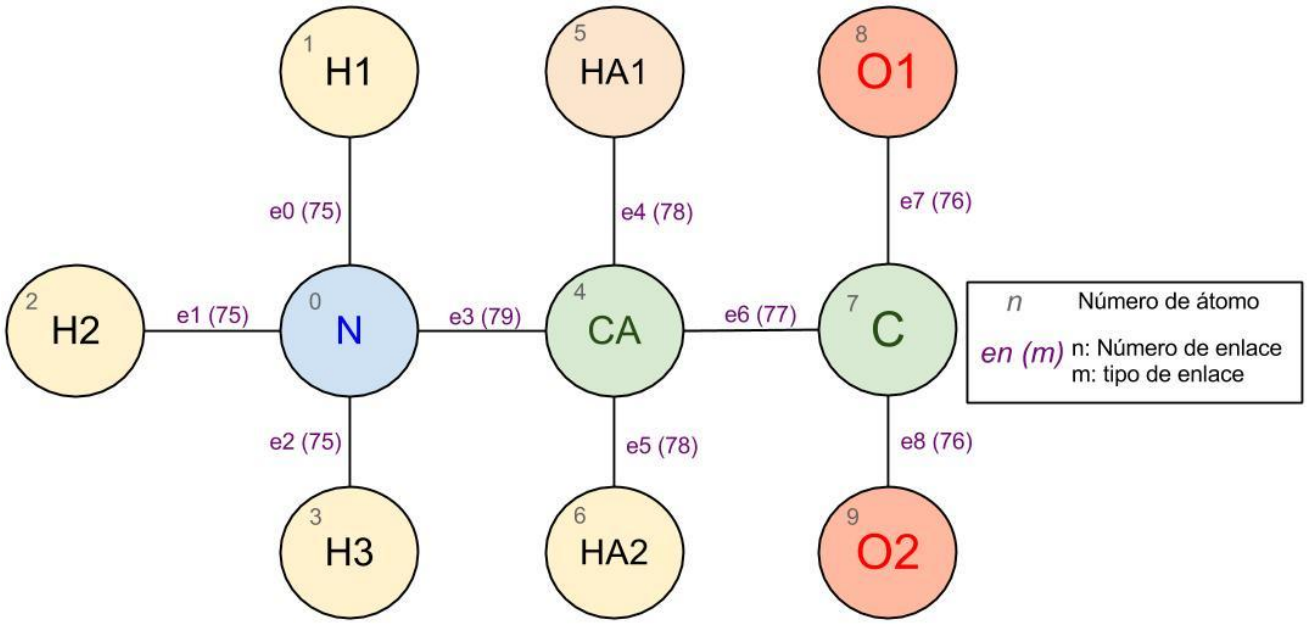


Figura 4. Representación de la Glicina en GROMACS

Estructuras de datos y variables

En esta sección se presentan las estructuras de datos que se usan en GROMCAS para la implementación del algoritmo SHAKE. Para describirlas y mostrar sus valores (si son constantes a lo largo de toda la simulación) se usa como ejemplo la glicina y se imponen ligaduras a todos sus enlaces, es decir, el parámetro `constraints`, descrito al comienzo de este capítulo es igual a `all-bonds`.

Primero se enumeran las variables y después se muestra una imagen descriptiva de las mismas:

- **`int ncons`**

Constante que representa el número de ligaduras impuestas al sistema (number *of constraints*). En el ejemplo de la glicina su valor es 9 (una ligadura por cada enlace).

- **`int natoms`**

Número total de átomos (incluyendo los átomos del disolvente, agua por ejemplo). Para el ejemplo que nos ocupa, su valor es 718.

- **`real invdt`**

Inversa de `dt`. `dt` es un parámetro de entrada en el fichero *.mdp definido como el paso temporal (*time step*) en picosegundos. Para el ejemplo, `dt` tiene el valor de 0.002 ps, por lo tanto, `invdt` es 500.

- **real shake_tol**

Valor usado para calcular la tolerancia relativa bajo la cual todas las ligaduras han de satisfacerse. Para este ejemplo se toma su valor por defecto: 0,0001.

- **t_iatom *iatoms (t_iatom = real)**

Vector que describe las ligaduras mediante grupos de tres elementos. El primero, describe el tipo de ligadura y el segundo y tercer elementos son los dos átomos involucrados en la ligadura. El tamaño del vector es, por lo tanto, tres veces el número de ligaduras (`3*ncons`). Si `k` es el número de la ligadura de tipo `t` que involucra a los átomos `i` y `j` entonces:

```
iatoms[3*k] = t;    iatoms[3*k+1] = i;    iatoms[3*k+2] = j
```

- **real invmass[]**

Vector que contiene la inversa de la masa de los átomos. En el caso de la glicina el tamaño del vector es 10 y sus valores pueden verse en la FIGURA 5.

- **real lagr**

Vector que contiene los multiplicadores de Lagrange. Su tamaño coincide con el número de ligaduras. En el caso de la glicina, su tamaño es 9

- **real dist2[]**

Vector que contiene el valor de las ligaduras al cuadrado, es decir, $dist2[k] = (\mathrm{a_{i,j}})^2$ (véase ECUACIÓN 6). El tamaño del vector es igual al número de ligaduras; para la glicina es 9.

- **real x[]**

Vector que contiene las posiciones (x,y,z) de los átomos al inicio del paso temporal, es decir, contiene los valores $x_i(t)$. El tamaño del vector es 3 veces el número de átomos. En el caso de la glicina son 10 átomos, por lo tanto el tamaño del vector es 30.

- **real xp[]**

Vector que contiene las posiciones (x,y,z) de los átomos al final del paso temporal. Antes de ejecutar el algoritmo SHAKE contiene las posiciones antes de ser corregidas, es decir, $x_i'(t + \Delta t)$ y después de ejecutar SHAKE contiene las posiciones corregidas, es decir, $x_i(t + \Delta t)$. Al igual que el vector `x[]`, su tamaño es 30.

- **real xij[]**

Vector que contiene la distancia (x,y,z) entre los átomos ligados en el paso temporal anterior, es decir $xij[k] = x_i(t) - x_j(t)$ cuando existe una ligadura k que relaciona los átomos i y j. El tamaño de este vector para el ejemplo de la glicina es 3 veces el número de ligaduras del sistema: 3*9 = 27.

ncons	int	9
natoms	int	718
invdt	real	500
shake_tol	real	0.0001

iatoms	
0	75
1	0
2	1
3	75
4	0
5	2
6	75
7	0
8	3
9	79
10	0
11	4
12	78
13	4
14	5
15	78
16	4
17	6
18	77
19	4
20	7
21	76
22	7
23	8
24	76
25	7
26	9

invmass	
0	0.0713944063
1	0.992063463
2	0.992063463
3	0.992063463
4	0.0832570195
5	0.992063463
6	0.992063463
7	0.0832570195
8	0.0625023469
9	0.0625023469

lagr	
0	
1	
2	
3	
4	
5	
6	
7	
8	

dist2	
0	0.0102010006
1	0.0102010006
2	0.0102010006
3	0.0216384102
4	0.0118809994
5	0.0118809994
6	0.0231648404
7	0.015625
8	0.015625

x				
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

xp				
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

xij				
0				
1				
2				
3				
4				
5				
6				
7				
8				

Figura 5. Representación de las principales estructuras de datos para una molécula de Glicina en GROMACS

Capítulo 4

IMPLEMENTACIÓN DEL ALGORITMO ILVES-S

En este capítulo se desarrolla la implementación, paso a paso, del algoritmo ILVES-S. Para ejemplificar algunos de los pasos, se usa la glicina y las estructuras de datos que la definen.

4.1 Introducción

Recordemos que el algoritmo ILVES-S consiste en resolver un sistema lineal $A\lambda = b$ de forma analítica para obtener los multiplicadores de Lagrange, λ, y usarlos para corregir las posiciones de los átomos. Las componentes de la matriz A y el vector b, partiendo de la ECUACIÓN 9, se definen como:

$$A_{kk'} = -2\Delta t^2 \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right) 2\left(\boldsymbol{x}_l(t) - \boldsymbol{x}_m(t)\right)\left(\frac{\delta(i,l)}{m_i} - \frac{\delta(i,m)}{m_i} - \frac{\delta(j,l)}{m_j} + \frac{\delta(j,m)}{m_j}\right)$$

$$b_k = \left(a_{i,j}\right)^2 - \left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)^2$$

Ecuación 11. Componentes de la matriz A y el vector b.

Y la ECUACIÓN 12 se utiliza para actualizar las posiciones.

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i'(t+\Delta t) - \frac{2\Delta t^2}{m_i}\sum_{k'=1}^{N_c} \lambda_{k'} 2\left(\boldsymbol{x}_l(t) - \boldsymbol{x}_m(t)\right)\left(\frac{\delta(i,l)}{m_i} - \frac{\delta(i,m)}{m_i}\right)$$

Ecuación 12. Corrección de las posiciones de los átomos

El término $-2\Delta t^2$ se puede omitir en la matriz A, puesto que luego aparece en la corrección de posiciones. En colaboración con el matemático Dr. Carl Christian Kjelgaard Mikkelsen (Universidad de Umeå) se desarrolla el algoritmo que se resume en la siguiente página:

ILVES-S

1. **Inicializar estructuras de datos necesarias**

1.1 Generar el grafo de adyacencia para las ligaduras
1.2 Reordenar ligaduras para reducir el ancho de banda de la matriz A
1.3 Generar tabla con información constante de la matriz A

⇩

repetir 2, 3, 4 y 5 hasta que todas las posiciones cumplan las restricciones dentro de una tolerancia

2. **Generar la matriz A**

⇩

3. **Generar el vector b**

⇩

4. **Resolver sistema lineal $A\boldsymbol{\lambda} = \boldsymbol{b}$**

⇩

5. **Corregir posiciones de los átomos implicados en las ligaduras**

El algoritmo se divide en dos partes: una fase de inicialización y el propio algoritmo que se ejecuta, al igual que SHAKE, cada paso temporal (*time step)*. A continuación se detalla cada paso, comenzando con la fase de inicialización.

4.2 Inicialización

Grafo de adyacencia

Es necesario generar el grafo de adyacencia, el cuál es útil para la construcción automática de la matriz A. El grafo de adyacencia representa qué ligaduras comparten átomos. Volviendo al ejemplo de la glicina (con todos los enlaces ligados), a continuación se muestra su grafo de adyacencia:

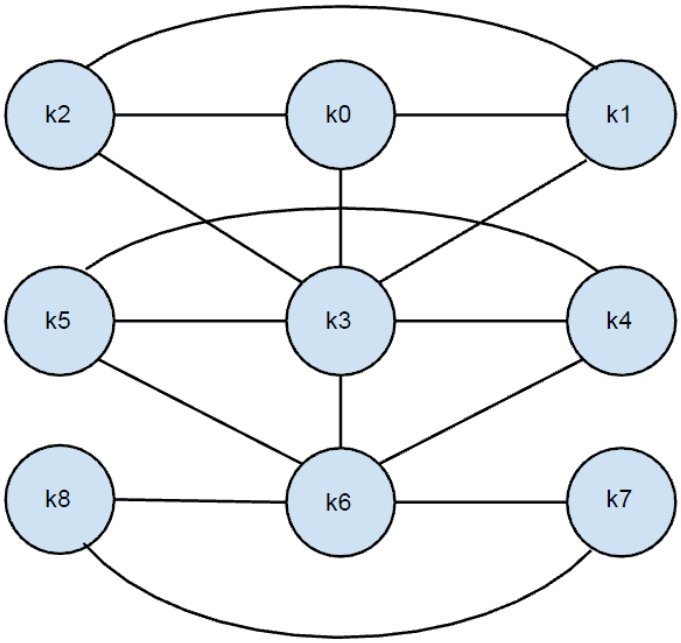


Figura 6. Grafo de adyacencia para la glicina cuando se imponen ligaduras a los enlaces.

En el grafo de adyacencia hay un nodo por cada una de las ligaduras, que en el caso de la glicina, coincide con los enlaces (véase FIGURA 4). Por ejemplo, el nodo $k0$ representa la ligadura impuesta al enlace 0 ($e0$). En el grafo, existe un enlace entre dos nodos si comparten algún átomo. Por ejemplo, el nodo correspondiente a la ligadura $k0$: $\sigma_{0(0,1)}$ y el correspondiente a la ligadura $k2$: $\sigma_{0(0,3)}$ están unidos por un eje porque comparten el átomo 0. En la siguiente figura se muestra la estructura de datos que almacena el anterior grafo de adyacencia:

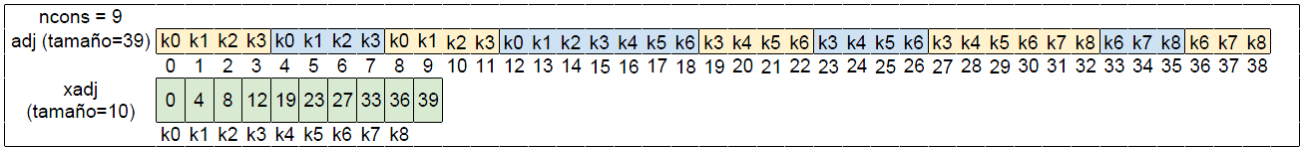


Figura 7. Representación vectorial del grafo de adyacencia para la glicina cuando se imponen ligaduras a los enlaces

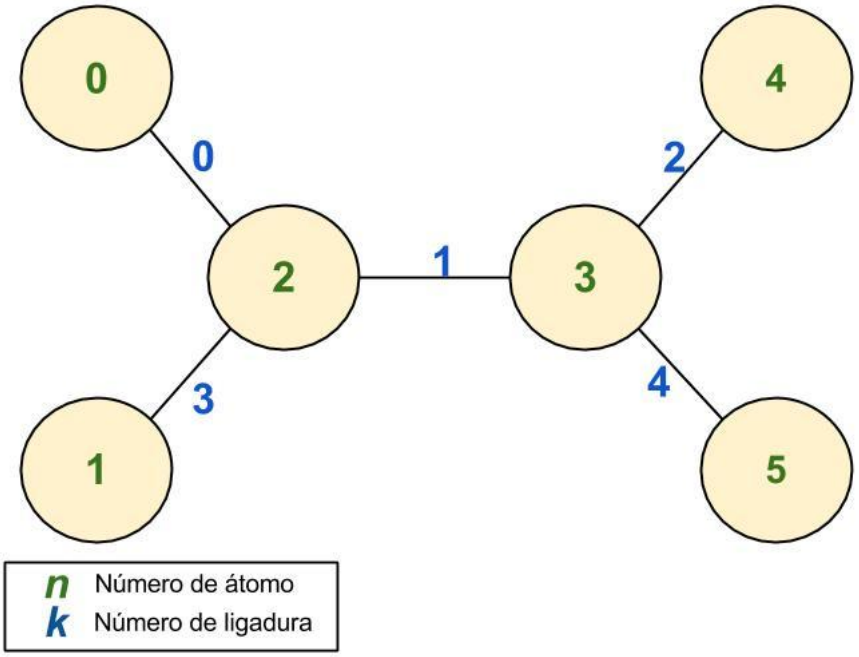
`ncons` es el número de ligaduras impuestas al sistema (número de nodos del grafo), `xadj` es el vector que apunta, para cada nodo, al comienzo de su lista de adyacencia (`adj`). Esta lista representa los ejes del grafo (nótese que cada nodo está conectado consigo mismo, aunque no se representa en la FIGURA 6, puesto que cada ligadura comparte consigo misma ambos átomos).

El grafo de adyacencia será útil para calcular de forma más eficiente la matriz A en cada paso temporal (*time step)*.

Reordenación de las ligaduras

Este paso es importante en la implementación del nuevo algoritmo puesto que una correcta ordenación de las ligaduras reduce el ancho de banda de la matriz asociada al sistema de ecuaciones lineales, lo que permite resolver el sistema de forma más eficiente.

Para ilustrar en qué consiste este paso del algoritmo mostramos el siguiente ejemplo: en él se puede observar una molécula con una "mala numeración" de las ligaduras puesto que la matriz no es banda:



Ligaduras:

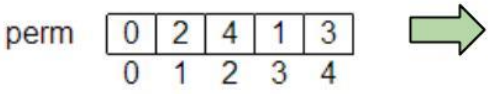
	Átomo 1	Átomo 2
k=0	0	2
k=1	2	3
k=2	3	4
k=3	1	2
k=4	3	5

Matriz A:

k\k'	0	1	2	3	4
0	x	x	0	x	0
1	x	x	x	x	x
2	0	x	x	0	x
3	x	x	0	x	0
4	0	x	x	0	x

Figura 8. Ejemplo de molécula antes de reordenar sus ligaduras.

En este paso del algoritmo se reordenan las ligaduras, es decir, se numeran de nuevo para que la matriz sea una matriz banda. Se halla el denominado vector permutación que indica para cada ligadura la numeración que tenía antes (en los índices del vector) y después (en las componentes del vector):

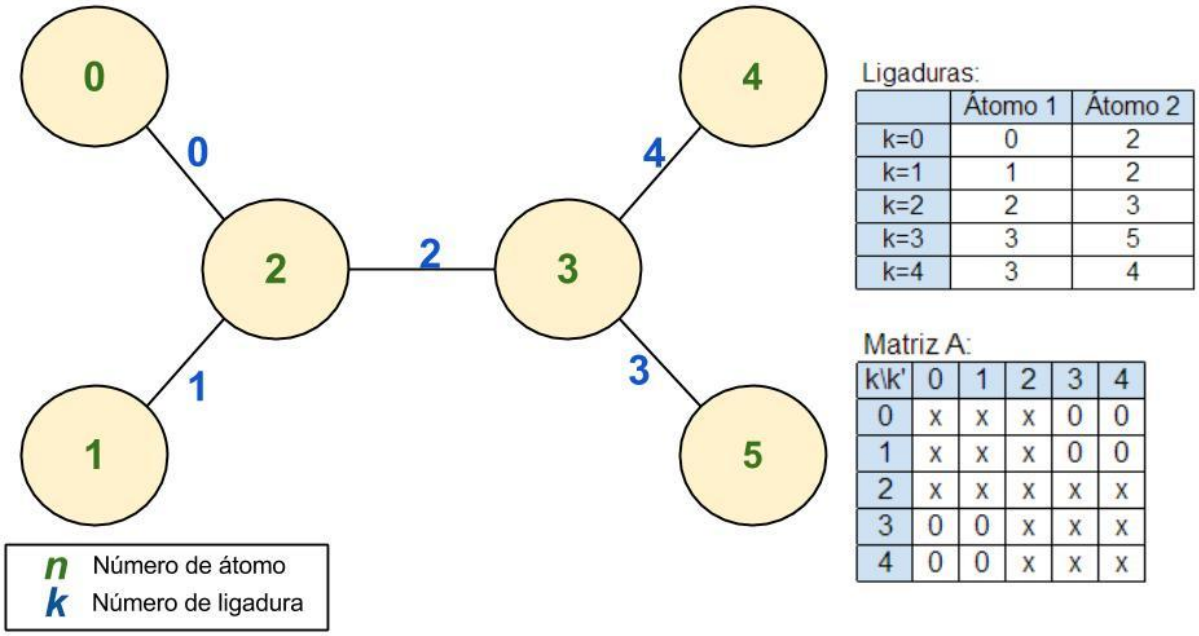


Ligaduras:

Numeración		Átomo 1	Átomo 2
Antes	Después		
k=0	k=0	0	2
k=1	k=2	2	3
k=2	k=4	3	4
k=3	k=1	1	2
k=4	k=3	3	5

Figura 9. Resultado de la reordenación.

De este modo, la nueva representación de la molécula y la nueva matriz A, ahora si, matriz banda, serían:



Ligaduras:

	Átomo 1	Átomo 2
k=0	0	2
k=1	1	2
k=2	2	3
k=3	3	5
k=4	3	4

Matriz A:

k\k'	0	1	2	3	4
0	x	x	x	0	0
1	x	x	x	0	0
2	x	x	x	x	x
3	0	0	x	x	x
4	0	0	x	x	x

Figura 10. Ejemplo de molécula después de reordenar sus ligaduras

Para la glicina el vector `perm[]` es el siguiente:

perm	0	2	1	3	5	4	6	8	7
	0	1	2	3	4	5	6	7	8

Figura 11. Vector que contiene la reordenación de las ligaduras para la Glicina.

Generación de la tabla con información de la matriz A

Si se observa la ECUACIÓN 11 es fácil ver que el término $2\left(\frac{\delta(i,l)}{m_i}-\frac{\delta(i,m)}{m_i}-\frac{\delta(j,l)}{m_j}+\frac{\delta(j,m)}{m_j}\right)$ es constante a lo largo de todos los pasos temporales. Además, este término también determina cuando un elemento $A_{kk'}$ es cero o no. Por lo tanto, definimos una tabla con estos elementos constantes y no nulos de la matriz A para no repetir este cálculo cada paso temporal.

La tabla contiene la siguiente información:

- `nnz`: Número de elementos no nulos de la matriz A (Igual al número de registros de la tabla).
- `ii[]`: Vector de tamaño `nnz` con los valores de k (filas de la matriz).
- `jj[]`: Vector de tamaño `nnz` con los valores de k' (columnas de la matriz).
- `weight[]`: Vector constante de tamaño `nnz` con los valores constantes.

Para el ejemplo de la glicina, el valor de `nnz` es 39. Esto implica que su matriz $A(9x9)$ tenga 39 entradas no nulas cuyos índices y términos constantes se muestran en la tabla de la FIGURA 12.

ii -> k(i,j)	jj -> k'(l,m)	weight
0 (0,1)	0 (0,1)	$2 * (m_0^{-1} + m_1^{-1})$
0 (0,1)	1 (0,3)	$2 * (m_0^{-1})$
0 (0,1)	2 (0,2)	$2 * (m_0^{-1})$
0 (0,1)	3 (0,4)	$2 * (m_0^{-1})$
1 (0,3)	0 (0,1)	$2 * (m_0^{-1})$
1 (0,3)	1 (0,3)	$2 * (m_0^{-1} + m_3^{-1})$
1 (0,3)	2 (0,2)	$2 * (m_0^{-1})$
1 (0,3)	3 (0,4)	$2 * (m_0^{-1})$
2 (0,2)	0 (0,1)	$2 * (m_0^{-1})$
2 (0,2)	1 (0,3)	$2 * (m_0^{-1})$
2 (0,2)	2 (0,2)	$2 * (m_0^{-1} + m_2^{-1})$
2 (0,2)	3 (0,4)	$2 * (m_0^{-1})$
3 (0,4)	0 (0,1)	$2 * (m_0^{-1})$
3 (0,4)	1 (0,3)	$2 * (m_0^{-1})$
3 (0,4)	2 (0,2)	$2 * (m_0^{-1})$
3 (0,4)	3 (0,4)	$2 * (m_0^{-1} + m_4^{-1})$
3 (0,4)	4 (4,6)	$2 * (-m_4^{-1})$
3 (0,4)	5 (4,5)	$2 * (-m_4^{-1})$
3 (0,4)	6 (4,7)	$2 * (-m_4^{-1})$
4 (4,6)	3 (0,4)	$2 * (-m_4^{-1})$
4 (4,6)	4 (4,6)	$2 * (m_4^{-1} + m_6^{-1})$
4 (4,6)	5 (4,5)	$2 * (m_4^{-1})$
4 (4,6)	6 (4,7)	$2 * (m_4^{-1})$
5 (4,5)	3 (0,4)	$2 * (-m_4^{-1})$
5 (4,5)	4 (4,6)	$2 * (m_4^{-1})$
5 (4,5)	5 (4,5)	$2 * (m_4^{-1} + m_5^{-1})$
5 (4,5)	6 (4,7)	$2 * (m_4^{-1})$
6 (4,7)	3 (0,4)	$2 * (-m_4^{-1})$
6 (4,7)	4 (4,6)	$2 * (m_4^{-1})$
6 (4,7)	5 (4,5)	$2 * (m_4^{-1})$
6 (4,7)	6 (4,7)	$2 * (m_4^{-1} + m_7^{-1})$
6 (4,7)	7 (7,9)	$2 * (-m_7^{-1})$
6 (4,7)	8 (7,8)	$2 * (-m_7^{-1})$
7 (7,9)	6 (4,7)	$2 * (-m_7^{-1})$
7 (7,9)	7 (7,9)	$2 * (m_7^{-1} + m_9^{-1})$
7 (7,9)	8 (7,8)	$2 * (m_7^{-1})$
8 (7,8)	6 (4,7)	$2 * (-m_7^{-1})$
8 (7,8)	7 (7,9)	$2 * (m_7^{-1})$
8 (7,8)	8 (7,8)	$2 * (m_7^{-1} + m_8^{-1})$

Figura 12. Elementos no nulos de la matriz A para el ejemplo de la glicina.

4.3 Proceso iterativo

Generación de la matriz A

Para resolver el sistema lineal $A\lambda = b$ se hace uso de una librería externa denominada LAPACK (*Linear Algebra PACKage*) [9] que proporciona rutinas para, entre otras cosas, resolver sistemas de ecuaciones lineales.

Una vez generada la tabla presentada anteriormente, es muy sencillo construir la matriz A: simplemente hay que añadir a cada entrada el término $\left(\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right)\left(\boldsymbol{x}_l(t) - \boldsymbol{x}_m(t)\right)$. El mayor problema en este paso es almacenar la matriz en el formato de entrada para las funciones LAPACK. En la siguiente figura se puede observar a la derecha la matriz A para la glicina, y a la izquierda, la misma matriz en formato LAPACK:

A (all-bonds)		k' (l,m)								
		0 (0,1)	1 (0,3)	2 (0,2)	3 (0,4)	4 (4,6)	5 (4,5)	6 (4,7)	7 (7,9)	8 (7,8)
k (i,j)	0 (0,1)	A(0,0)	A(0,1)	A(0,2)	A(0,3)	0	0	0	0	0
	1 (0,3)	A(1,0)	A(1,1)	A(1,2)	A(1,3)	0	0	0	0	0
	2 (0,2)	A(2,0)	A(2,1)	A(2,2)	A(2,3)	0	0	0	0	0
	3 (0,4)	A(3,0)	A(3,1)	A(3,2)	A(3,3)	A(3,4)	A(3,5)	A(3,6)	0	0
	4 (4,6)	0	0	0	A(4,3)	A(4,4)	A(4,5)	A(4,6)	0	0
	5 (4,5)	0	0	0	A(5,3)	A(5,4)	A(5,5)	A(5,6)	0	0
	6 (4,7)	0	0	0	A(6,3)	A(6,4)	A(6,5)	A(6,6)	A(6,7)	A(6,8)
	7 (7,9)	0	0	0	0	0	0	A(7,6)	A(7,7)	A(7,8)
	8 (7,8)	0	0	0	0	0	0	A(8,6)	A(8,7)	A(8,8)

A	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	A(0,3)	0	0	A(3,6)	0	0
4	0	0	A(0,2)	A(1,3)	0	A(3,5)	A(4,6)	0	A(6,8)
5	0	A(0,1)	A(1,2)	A(2,3)	A(3,4)	A(4,5)	A(5,6)	A(6,7)	A(7,8)
6	A(0,0)	A(1,1)	A(2,2)	A(3,3)	A(4,4)	A(5,5)	A(6,6)	A(7,7)	A(8,8)
7	A(1,0)	A(2,1)	A(3,2)	A(4,3)	A(5,4)	A(6,5)	A(7,6)	A(8,7)	0
8	A(2,0)	A(3,1)	0	A(5,3)	A(6,4)	0	A(8,6)	0	0
9	A(3,0)	0	0	A(6,3)	0	0	0	0	0

Figura 13. A la derecha, matriz A para la glicina. A la izquierda, la misma matriz en formato LAPACK.

Generación del vector b

El vector b se obtiene a partir de los vectores `dist2[]` y `xp[]` del siguiente modo:

$$b_k = \left(a_{i,j}\right)^2 - \left|\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\right|^2$$

Ecuación 13. Vector b del sistema lineal $A\lambda = b$

Resolución del sistema lineal

La resolución del sistema lineal se lleva a cabo mediante la llamada a dos funciones de la librería externa LAPACK. En concreto, para precisión doble se usan las funciones `dgbtrf()` y `dgbtrs()`. La primera rutina calcula la factorización LU de la matriz A, mientras que la

segunda resuelve el sistema de ecuaciones utilizando la factorización LU realizada por la primera función.

Corrección de posiciones

Por último, la corrección de posiciones se realiza aplicando la siguiente ecuación a cada átomo implicado en las ligaduras:

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i'(t+\Delta t) + \frac{1}{m_i}\sum_{k'=1}^{N_c} \lambda_{k'} 2\big(\boldsymbol{x}_l(t) - \boldsymbol{x}_m(t)\big)\left(\frac{\delta(i,l)}{m_i} - \frac{\delta(i,m)}{m_i}\right)$$

Ecuación 14. Corrección de posiciones

Condición de parada

Una vez aplicadas todas las operaciones hay que comprobar si se aceptan las nuevas posiciones, es decir si las ligaduras impuestas al sistema se satisfacen dentro de una tolerancia dada. Si no es así, será necesario realizar una nueva iteración. Para ello se hace uso de la misma tolerancia usada por el algoritmo SHAKE: `shake_tol`, que por defecto vale 0.0001. A partir de esta variable, que se puede cambiar en el fichero de entrada (`.mdp`), se define $iconvf_k$ para cada ligadura:

$$iconvf_k = |b_k| * \frac{1}{\big(a_{i,j}\big)^2 * 2 * shake\_tol} = \frac{\Big|\big(a_{i,j}\big)^2 - \big|\boldsymbol{x}_i'(t+\Delta t) - \boldsymbol{x}_j'(t+\Delta t)\big|^2\Big|}{\big(a_{i,j}\big)^2 * 2 * shake\_tol}$$

Ecuación 15. Condición de parada para el algoritmo ILVES-S

Si $iconvf_k$es mayor que 1 para alguna de las ligaduras, entonces es necesario realizar una nueva iteración.

Capítulo 5

RESULTADOS

Este capítulo presenta el entorno experimental (máquina en la que se ejecutan las simulaciones, molécula simulada) y los principales resultados obtenidos.

5.1 Entorno experimental

Maquina de trabajo

La máquina de trabajo en la que se ejecutan los experimentos tiene las siguientes características:

- Sistema operativo: CentOS *release 6.3*
- Kernel: Linux 2.6.32-220.2.1.el6.x86_64
- Procesador: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz (microarquitectura *Sandy Bridge*): 4 núcleos físicos con *hyperthreading* (8 núcleos lógicos).
- 3 niveles de cache integrados en el chip:
  - L1I + L1D: 32 kB + 32 kB (por núcleo)
  - L2: 256 kB (por núcleo)
  - L3: 8 MB (compartida)
- Compilador: gcc versión 4.4.6 20120305 (*Red Hat 4.4.6-4*) (GCC)

Simulaciones de dinámica molecular

La verificación del correcto funcionamiento del algoritmo se ha realizado mediante sencillos experimentos. Primero con la simulación de la glicina, el aminoácido más pequeño, y después con cadenas de tres aminoácidos.

Para evaluar el resultado de la implementación de la primera versión del algoritmo ILVES-S y cuantificar su rendimiento y precisión, se realizan simulaciones de dinámica molecular de la arginina, uno de los aminoácidos esenciales que forma parte de las proteínas. Esta molécula tiene 27 átomos y 26 enlaces que dan lugar a 26 ligaduras.

5.2 Verificación de los resultados

Para realizar la verificación del correcto funcionamiento de ILVES-S, además de realizar sencillos test en los que se comparan los multiplicadores de Lagrange obtenidos con el algoritmo SHAKE y los que se obtienen con ILVES-S, es importante comprobar que las posiciones corregidas cumplen las restricciones dentro de una cierta tolerancia. Así, el error relativo se calcula del siguiente modo:

$$error_{i,j}(t+\Delta t) = \frac{\left|\left(a_{i,j}\right) - \left|\boldsymbol{x}_i(t+\Delta t) - \boldsymbol{x}_j(t+\Delta t)\right|\right|}{\left(a_{i,j}\right)}$$

Ecuación 16. Error absoluto cometido en el cálculo de las nuevas posiciones

Además, el código implementado e integrado en GROMACS y el código GROMACS original se han modificado para volcar en un fichero los datos necesarios que permiten: (a) comparar el algoritmo original de GROMACS (SHAKE) y el nuevo algoritmo implementado (ILVES-S), y (b) comprobar el correcto funcionamiento de este último. También se han programado scripts que obtienen de forma automatizada todas estas medidas con diferentes entradas y configuraciones del algoritmo.

5.3 Análisis de los resultados

Precisión

Para comparar las precisiones de ambos algoritmos hemos utilizado el error relativo máximo y el error relativo medio. En cada paso temporal de simulación, se ha registrado el error relativo máximo y el error relativo medio de todas las ligaduras (valores absolutos, ver ECUACIÓN 16). Posteriormente se han promediado los errores registrados en cada paso de simulación. La FIGURA 14 muestra el error relativo máximo y el error relativo medio de cada algoritmo para distintos valores de tolerancia relativa, `shake_tol`, especificada en los parámetros de simulación.

En primer lugar, se observa que ILVES-S resuelve correctamente la imposición de ligaduras en simulaciones de dinámica molecular. Además, para todos los valores de tolerancia con los que se ha experimentado, ILVES es más preciso que SHAKE, tanto en error relativo máximo como en error relativo medio. Reseñar que para una tolerancia de 1e-4 (el valor de `shake_tol` por defecto en GROMACS), el error cometido por ILVES-S es del orden de tres órdenes de magnitud menor que el de SHAKE. Por lo tanto, las simulaciones con ILVES-S son más fieles a la realidad física que modelan.

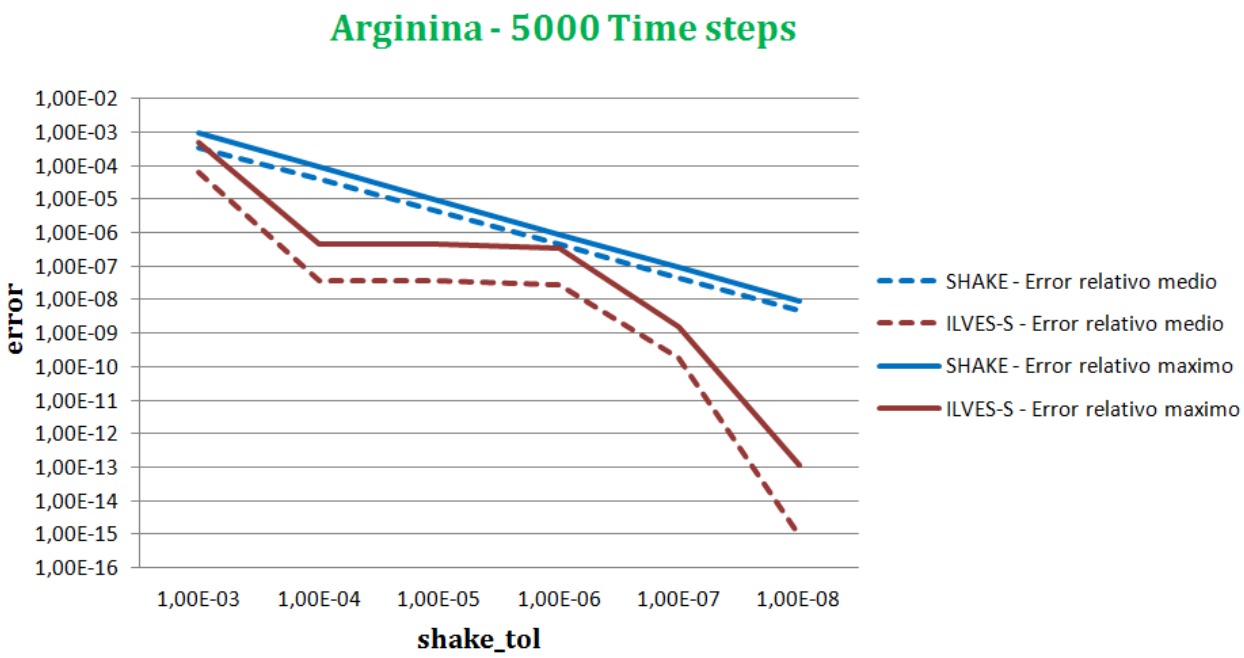


Figura 14. Error relativo máximo y error relativo medio para distintos valores de tolerancia relativa (shake_tol). Observar que la escala del eje y es logarítmica.

Las variaciones de precisión observadas en la anterior gráfica pueden explicarse analizando el número medio de iteraciones ejecutado por cada algoritmo. En cada paso temporal de simulación, se ha registrado el número de iteraciones ejecutadas por ambos algoritmos de cálculo (SHAKE e ILVES-S). Posteriormente se han promediado los valores registrados en cada paso de simulación.

En la FIGURA 15 se puede observar el número medio de iteraciones de cada algoritmo para distintos valores de tolerancia relativa `shake_tol`. Para el caso de tolerancias menores o iguales a 1e-6, ILVES-S es capaz de cumplir las ligaduras ejecutando una media de en torno a dos iteraciones, mientras que las tolerancias mayores (1e-7 y 1e-8) requieren la ejecución de tres iteraciones. El comportamiento de SHAKE es distinto, ya que el número de iteraciones necesario para cumplir las ligaduras crece al aumentar la precisión requerida. Para una tolerancia de 1e-6, SHAKE ejecuta una media de casi 11 iteraciones frente a las 2 que ejecuta ILVES-S. Es de esperar que en moléculas con muchas ligaduras esto se traduzca en un menor tiempo de ejecución por parte de ILVES-S.

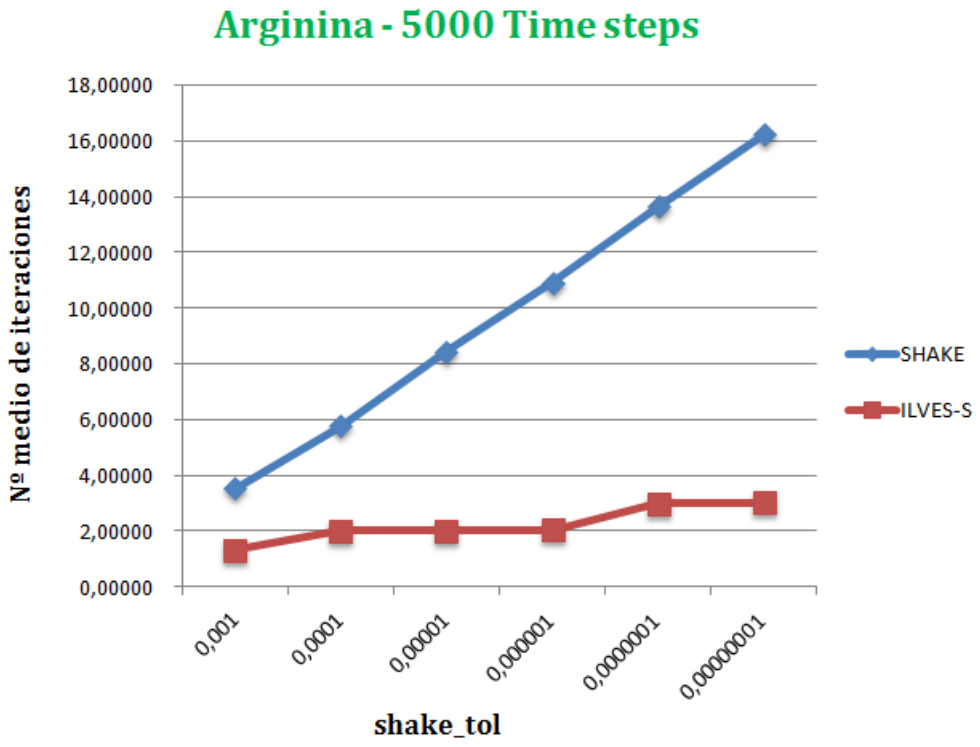


Figura 15. Número medio de iteraciones efectuadas por los algoritmos de cálculo para distintos valores de tolerancia relativa (shake_tol).

Tiempo

Durante cada simulación, GROMACS registra el tiempo (medido en ciclos de CPU) que dedica a distintas tareas. Una de ellas es el cumplimiento de las ligaduras. La FIGURA 16 muestra el número de ciclos de CPU dedicados a los cálculos relacionados con las ligaduras. Para todas las tolerancias, SHAKE es más rápido, con diferencias que oscilan entre el 4.8% y el 7.3%. Estas diferencias son debidas a varios factores:

- La implementación de SHAKE en GROMACS está muy optimizada, mientras que la versión evaluada de ILVES-S es una primera implementación centrada en analizar su potencial y no en optimizar su rendimiento.

- La versión implementada del algoritmo ILVES-S utiliza funciones de la librería LAPACK para resolver el sistema lineal de ecuaciones. Actualmente, el equipo de investigación (en concreto, el matemático Carl Christian Kjelgaard Mikkelsen) está trabajando en una función para resolver sistemas de ecuaciones lineales banda, lo que permitirá reducir de forma significativa el tiempo de ejecución de ILVES-S.

Así pues, se espera que próximamente ILVES-S resuelva las ligaduras más rápidamente que SHAKE. Además, a medio plazo se pretende implementar una versión paralela de ILVES-S, lo cual permitirá mejorar todavía más sus prestaciones (la implementación de SHAKE en GROMACS es secuencial).

Por otra parte, puede observarse que el tiempo de ejecución no escala con el número de iteraciones. Esto puede ser debido al reducido número de ligaduras. Próximamente, esperamos realizar experimentos con moléculas más grandes y con más ligaduras para validar esta hipótesis.

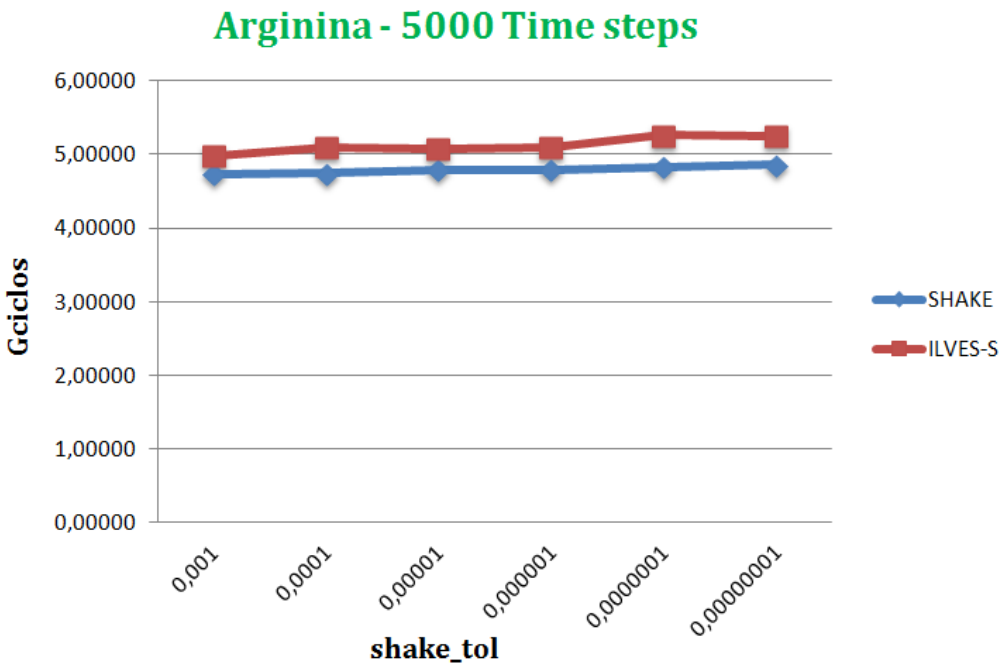


Figura 16. Número de ciclos de CPU (en miles de millones) invertidos en cálculos relacionados con las ligaduras.

Capítulo 6

CONCLUSIONES

Este último capítulo recoge las conclusiones del proyecto fin de carrera, el trabajo futuro y la valoración personal.

6.1 Conclusiones

La dinámica molecular es una técnica usada en todo el mundo en numerosas áreas relacionadas con la física, la química y la biología, donde es necesario comprender, estudiar y analizar el comportamiento de los átomos y las moléculas. Es un campo multidisciplinario que requiere el conocimiento de expertos en diversas áreas como las matemáticas, la física, la química y las ciencias computacionales. Ejemplo de ello es el proyecto del que forma parte este proyecto fin de carrera y en el que participa el grupo de Arquitectura de Computadores de la Universidad de Zaragoza, el Dr. Pablo García Risueño (Universidad Humboldt de Berlín, Alemania) experto en física teórica y el Dr. Carl Christian Kjelgaard Mikkelsen (Universidad de Umea, Suecia), matemático especializado en cálculos analíticos y resolución de grandes sistemas de ecuaciones lineales.

Se ha integrado en GROMACS la primera versión de un algoritmo, ILVES-S, que permite imponer ligaduras en simulaciones de dinámica molecular con ligaduras. Para una tolerancia dada, ILVES-S es más preciso que el algoritmo original, SHAKE. La solución que se obtiene ahora se aproxima más a la realidad física. En cuanto al tiempo de ejecución, ILVES-S es ligeramente más lento que SHAKE (entre un 4.8% y un 7.3% según las distintas tolerancias). Esta diferencia es debida en gran parte al uso de rutinas de resolución de sistemas lineales LAPACK que no explotan eficientemente el hecho de que son sistemas banda.

6.2 Trabajo futuro

Actualmente hay dos líneas abiertas de trabajo en el algoritmo ILVES-S:

- Sustituir las funciones de resolución de sistemas de ecuaciones lineales de la librería LAPACK por funciones desarrolladas específicamente resolver los sistema de ecuaciones que surgen

en el contexto de la imposición de ligaduras en sistema bioquímicos. De esta manera, se espera reducir significativamente el tiempo de ejecución.

- Paralelizar el algoritmo ILVES-S, lo que permitirá aumentar el rendimiento del algoritmo en los sistemas actuales (predominantemente multinúcleo). Esto supondría una importante ventaja respecto SHAKE, ya que no existe versión paralela del mismo.

6.3 Valoración personal

Este proyecto me ha dado la oportunidad de conocer la dinámica molecular, un campo de gran interés donde expertos e investigadores de diversas áreas como la física, la química o la biología colaboran y comparten conocimientos. Además, en la actualidad la computación de altas prestaciones en este campo ofrece un futuro prometedor.

Al ser un proyecto de investigación, durante su desarrollo han surgido dudas y retos que afortunadamente han podido solventarse. Otra dificultad que me he encontrado es que era un tema completamente nuevo y ha habido que comenzar desde cero. Valoro de forma muy positiva los conocimientos adquiridos, lo que, junto con los resultados satisfactorios obtenidos, permiten que el proyecto tenga continuidad.

Además, los conocimientos de dinámica molecular adquiridos durante el desarrollo de este proyecto me han ayudado a conseguir una beca competitiva convocada por PRACE (*Partnership for Advanced Computing in Europe*) [19], para realizar una estancia durante el verano en el *Ireland's High-Performance Computing Centre* (ICHEC) [20] en Dublín, Irlanda. El resultado del proyecto desarrollado denominado "*Profiling and optimization of th the hybrid Molecular Dynamics code, DL_POLY, on heterogeneous clusters*", puede consultarse en el Anexo IV. Las simulaciones de dinámica molecular implican un alto consumo de recursos por lo que es oportuno aplicar técnicas de supercomputación como por ejemplo el uso de unidades de procesamiento gráfico (Graphics Processing Units - GPUs). En esto se basa el proyecto desarrollado durante mi estancia en Dublín: la utilización de GPUs para agilizar las simulaciones de dinámica molecular.

ANEXO I

Campos de fuerza

Un campo de fuerza hace referencia a la forma y a los parámetros de las funciones matemáticas utilizadas para describir la energía potencial de un sistema de partículas (típicamente moléculas y átomos). Un campo de fuerzas típico usado en las simulaciones de sistemas biológicos puede tomar la siguiente forma:

$$V(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \sum_{bonds} \frac{a_{ij}}{2}\left(l_{ij} - l_{ij0}\right)^2 + \sum_{angles} \frac{b_{ijk}}{2}\left(\theta_{ijk} - \theta_{ijk0}\right)^2 + \sum_{dihedrals} \frac{c_{ijkl}}{2}\left[1 + cos\left(n\omega_{ijkl} - \gamma_{ijkl}\right)\right] + \sum_{improper} \frac{d_{ijkl}}{2}\left(\varphi_{ijkl} - \varphi_{ijkl0}\right)^2 + \sum_{atom\ pairs} 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{\boldsymbol{x}_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{\boldsymbol{x}_{ij}}\right)^{6}\right] + \sum_{atom\ pairs} k\frac{q_i q_j}{\boldsymbol{x}_{ij}}$$

Ecuación 17. Ecuación del campo de fuerza sobre in sistema biológico

Los parámetros que aparecen en la ecuación anterior varían en función del campo de fuerza concreto, que ha de ser elegido en función del sistema a analizar. La ECUACIÓN 17 consta de seis términos sumatorios y se puede dividir en dos partes. La primera (cuatro primeros términos sumatorios*)* corresponde a las interacciones físicas debidas a los enlaces covalentes (*bonded interactions*), es decir, modelan la energía potencial en el sistema debida a la interacción de los átomos unidos por un enlace covalente. La segunda parte (dos últimos términos sumatorios) modela la energía potencial debida a la interacción entre átomos no enlazados (*non-bonded interactions*). A continuación se describen estos términos.

Bonded interactions:

- $V_{bonds} = \sum_{bonds} \frac{a_{ij}}{2}\left(l_{ij} - l_{ij0}\right)^2$: describe el componente de la energía potencial del sistema que surge debido a la deformación de la longitud de los enlaces covalentes entre pares de átomos (*bond lengths*) l_{ij}, con respecto a su valor de equilibrio l_{ij0} (véase FIGURA 17.A).

- $V_{angles} = \sum_{\text{angles}} \frac{\text{b}_{\text{ijk}}}{2} (\theta_{ijk} - \theta_{ijk0})^2$: describe el componente de la energía potencial del sistema que surge debido al cambio de los ángulos entre enlaces (*bond angles*) θ_{ijkl}, con respecto a su valor de equilibrio θ_{ijkl0} (véase FIGURA 17.B).
- $V_{dihedrals} = \sum_{dihedrals} \frac{c_{ijkl}}{2} [1 + cos(n\omega_{ijkl} - \gamma_{ijkl})]$: este término es conocido como la energía de torsión. Se caracteriza a través de un ángulo diedro (*dihedral angles o proper dihedral angles*) formado por cada cuatro átomos conectados a través de enlaces covalentes. Como se muestra en la FIGURA 17.C, el ángulo diedro es el que forma el plano que contiene los átomos i, j, k con el plano que forman los átomos j, k, l.
- $V_{improper} = \sum_{improper} \frac{\text{d}_{ijkl}}{2} (\varphi_{ijkl} - \varphi_{ijkl0})^2$: describe los llamados ángulos diedros impropios (*improper dihedral angles*) formado por cada cuatro átomos de los cuales, sólo uno de ellos está conectado mediante un enlace covalente a todos los demás En la FIGURA 17.D, el ángulo dihedro impropio es el que forma el plano que contiene los átomos i, j, k con el plano que forman los átomos j, k, l.

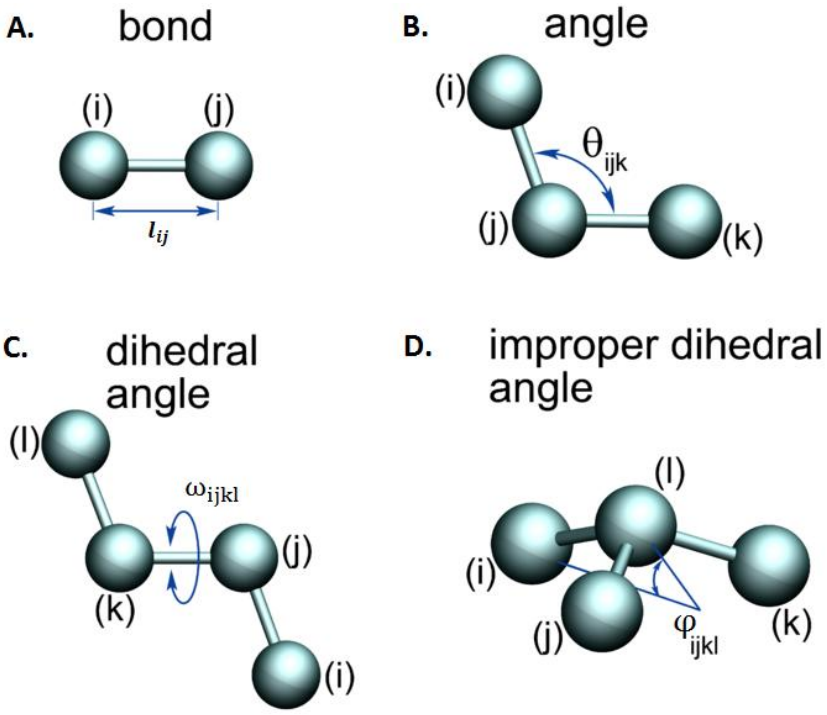


Figura 17. Bonded interactions: l_{ij} es la longitud del enlace entre los átomos i y j, θ_{ijk}es el ángulo formado por los enlaces ij y jk, ω_{ijkl} es el ángulo formado por los planos ijk y jkl y φ_{ijkl}es el ángulo formado por los planos ijk y jkl.

Non-bonded interactions:

- $V_{vanDerWaals} = \sum_{atom\ pairs} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{x_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{x_{ij}} \right)^{6} \right]$: describe las fuerzas de Van der Waals, fuerzas atractivas o repulsivas, entre pares de átomos no enlazados, debidas a polarizaciones de nubes electrónicas. Este término está modelado mediante el llamado potencial de Lennard-Jones [21].

- $V_{Coulomb} = \sum_{atom\ pairs} k\frac{q_i q_j}{x_{ij}}$: es el potencial de Coulomb y describe la interacción entre pares de átomos debido a sus cargas eléctricas. Aunque en general los átomos son eléctricamente neutros, la irregular distribución de carga hace que tengan unas cargas parciales que interactúan entre sí.

ANEXO II

Instalación y ejecución de GROMACS

1. Instalación GROMACS v.4.5.5

A continuación se detallan los pasos para la instalación de GROMACS [22]. Por defecto se instala haciendo posible la ejecución con varios hilos (*threading*). Una alternativa es instalar la versión de GROMACS que incluye paralelismo con múltiples procesadores con MPI [23]. Además, GROMACS requiere la instalación previa de FFTW v.3.3.2 [5], una biblioteca de subrutinas para el cálculo de la transformada discreta de Fourier (DFT).

- FFFTW v.3.3.2 (Precisión simple)

```
./configure --prefix=/home/maston/fftw --enable-float --enable-shared
make
make install
```

Tras la instalación de esta librería ha de hacerse "visible" su directorio destino de instalación (si se ha especificado una ruta distinta de la convencional). Para ello se modifican las variables de entorno *CPPFLAGS* y *LDFLAGS:*

```
export CPPFLAGS="-I/home/maston/fftw/include"
export LDFLAGS="-L/home/maston/fftw/lib"
```

- GROMACS v.4.5.5 (Precisión simple)

```
./configure --prefix=/home/maston/gromacs --enable-shared
make
make install
```

- FFFTW v.3.3.2 (Doble precisión):

```
./configure --prefix=/home/maston/fftw-double --enable-shared
make
make install

export CPPFLAGS="-I/home/maston/fftw-double/include"
export LDFLAGS="-L/home/maston/fftw-double/lib"
```

- GROMACS v.4.5.5 (Doble precisión):

```
./configure --prefix=/home/maston/gromacs --disable-float --enable-shared
make
make install
```

Los flags incluidos en la configuración de las instalaciones son:

`--prefix` para cambiar el directorio destino de instalación.
`--enable-float` para compilar con precisión simple.
`--disable-float` para compilar con doble precisión.
`--enable-shared` librería dinámica [24].

2. Ejecución

En esta sección se detalla en qué consisten los distintos pasos para llevar a cabo, en GROMACS, una simulación de dinámica molecular partiendo del denominado fichero PDB (*Protein Data Bank* [25]). Este fichero recoge la estructura de la molécula que se desea simular. Como ejemplo de ejecución se usa una lisozima, una encima presente en los huevos de gallina que tiene 1079 átomos y cuyo código PDB es 1AKI. Su estructura, recogida en el fichero `.pdb`, es la que se observa en la FIGURA 18.

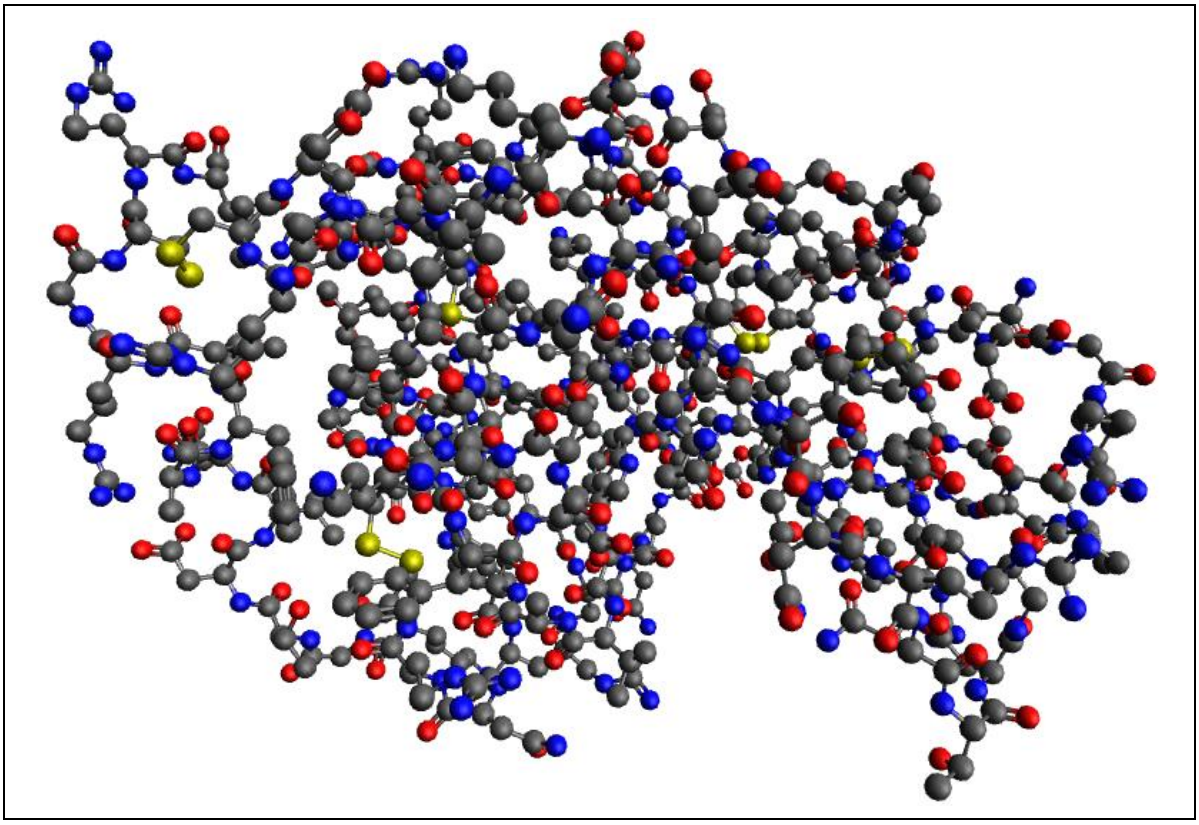


Figura 18. Estructura de la lysozima

PASO 1. Preparar la topología

En primer lugar, partiendo del fichero `.pdb`, se generan algunos ficheros necesarios para la simulación con la herramienta `pdb2gmx`:

```
pdb2gmx -f 1aki.pdb -o 1aki _processed.gro -water spce
```

Se selecciona el *force field* número 14, que corresponde a `OPLS-AA/L all-atom force field (2001 aminoacid dihedrals)`. Los ficheros obtenidos son:

- Topología de la molécula (fichero `topol.top`): contiene toda la información necesaria para definir la molécula en una simulación. Incluye información no relacionada con los enlaces (tipos de átomos y sus cargas) así como parámetros relacionados con los enlaces (enlaces covalentes, ángulos de enlace y ángulos diedros).

- Fichero con la definición de las ligaduras (fichero `posre.itp`)

- Estructura post-procesada (fichero `1aki_processed.gro`)

Además, se obtienen algunos datos importantes en la salida estándar:

```
...
Read 'LYSOZYME', 1079 atoms
Analyzing pdb file
Splitting chemical chains based on TER records or chain id changing.
There are 1 chains and 1 blocks of water and 207 residues with 1079 atoms

  chain  #res #atoms
  1 'A'   129   1001
  2 ' '    78     78  (only water)
…
Now there are 129 residues with 1960 atoms
Making bonds...
Number of bonds was 1984, now 1984
Generating angles, dihedrals and pairs...
Before cleaning: 5142 pairs
Before cleaning: 5187 dihedrals
Keeping all generated dihedrals
Making cmap torsions...There are 5187 dihedrals, 426 impropers,
3547 angles, 5106 pairs, 1984 bonds and 0 virtual sites
Total mass 14313.197 a.m.u.
Total charge 8.000 e
…
Processing chain 2 (78 atoms, 78 residues)
…
Now there are 78 residues with 234 atoms
Making bonds...
Number of bonds was 156, now 156
Generating angles, dihedrals and pairs...
Making cmap torsions...There are 0 dihedrals, 0 impropers, 78 angles,
0 pairs, 156 bonds and 0 virtual sites
Total mass 1405.201 a.m.u.
Total charge 0.000 e
Including chain 1 in system: 1960 atoms 129 residues
Including chain 2 in system: 234 atoms 78 residues
```

```
Now there are 2194 atoms and 207 residues
Total mass in system 15718.398 a.m.u.
Total charge in system 8.000 e
…
```

PASO 2. Definición del recipiente.

A continuación se define el recipiente que contendrá la lisozima (en este caso un sencillo recipiente con forma cúbica), usando para ello la herramienta `editconf`:

```
editconf -f 1aki_processed.gro -o 1aki_nexbox.gro
-c -d 1.0 -bt cubic
```

Se obtiene un nuevo fichero `*.gro`, en este caso `1aki_nexbox.gro`, con la nueva estructura y la siguiente información:

```
...
Read 1960 atoms
Volume: 123.376 nm^3, corresponds to roughly 55500 electrons
No velocities found
    system size :  3.817  4.234  3.454 (nm)
    diameter    :  5.010               (nm)
    center      :  2.781  2.488  0.017 (nm)
    box vectors :  5.906  6.845  3.052 (nm)
    box angles  :  90.00  90.00  90.00 (degrees)
    box volume  : 123.38               (nm^3)
    shift       :  0.724  1.017  3.488 (nm)
new center      :  3.505  3.505  3.505 (nm)
new box vectors :  7.010  7.010  7.010 (nm)
new box angles  :  90.00  90.00  90.00 (degrees)
new box volume  : 344.48               (nm^3)
...
```

PASO 3. Definición del disolvente

En este paso es necesario "añadir" al recipiente creado en el paso anterior, el disolvente deseado, en este caso agua. Para ello se usa la herramienta `genbox`:

```
genbox -cp 1aki_nexbox.gro -cs spc216.gro -o 1aki_solv.gro -p
topol.top
```

Se obtiene otra vez un nuevo fichero con la estructura de la molécula, `1aki_solv.gro`, y se actualiza el fichero que contiene la topología, `topol.top`. Como salida obtenemos la siguiente información relevante:

```
...
Reading solvent configuration
"216H2O,WATJP01,SPC216,SPC-MODEL,300K,BOX(M)=1.86206NM,WFVG,MAR. 1984"
solvent configuration contains 648 atoms in 216 residues

Initialising van der waals distances...
Will generate new solvent configuration of 4x4x4 boxes
Generating configuration
Sorting configuration
```

```
Found 1 molecule type:
    SOL (   3 atoms): 13824 residues
Calculating Overlap...
box_margin = 0.315
Removed 2736 atoms that were outside the box
Neighborsearching with a cut-off of 0.48
Table routines are used for coulomb: FALSE
Table routines are used for vdw:     FALSE
Cut-off's:   NS: 0.48   Coulomb: 0.48   LJ: 0.48
System total charge: 0.000
Grid: 16 x 16 x 16 cells
Successfully made neighbourlist
nri = 61846, nrj = 1388188
Checking Protein-Solvent overlap: tested 36470 pairs, removed 1968 atoms.
Checking Solvent-Solvent overlap: tested 390293 pairs,
removed 4272 atoms.
Added 10832 molecules
Generated solvent containing 32496 atoms in 10832 residues
Writing generated configuration to 1AKI_solv.gro
LYSOZYME

Output configuration contains 34456 atoms in 10961 residues
Volume                 :     344.484 (nm^3)
Density                :     1173.99 (g/l)
Number of SOL molecules:  10832
...
```

PASO 4. Añadir iones

En este paso se añaden iones al sistema, para ello se usa la herramienta `genion` pero antes es necesario ensamblar (`grompp`) los parámetros especificados en un nuevo fichero de entrada (`ions.mdp`) con la topología y las coordenadas del sistema para obtener un fichero que contiene todos los parámetros para todos los átomos del sistema (`ions.tpr`).

Para este ejemplo, el fichero de entrada usado es:

```
; ions.mdp - used as input into grompp to generate ions.tpr
; Parameters describing what to do, when to stop and what to save
Integrator      = steep      ; Algorithm (steep = steepest descent minimization)
emtol           = 1000.0     ; Stop minimization when the maximum force < 1000.0 kJ/mol/nm
emstep          = 0.01       ; Energy step size
nsteps          = 50000      ; Maximum number of (minimization) steps to perform

; Parameters describing how to find the neighbors of each atom and how to calcúlate
; the interactions
nstlist         = 1          ; Frequency to update the neighbor list and long range forces
ns_type         = grid       ; Method to determine neighbor list (simple, grid)
rlist           = 1.0        ; Cut-off for making neighbor list (short range forces)
coulombtype     = PME        ; Treatment of long range electrostatic interactions
rcoulomb        = 1.0        ; Short-range electrostatic cut-off
rvdw            = 1.0        ; Short-range Van der Waals cut-off
pbc             = xyz        ; Periodic Boundary Conditions (yes/no)
```

Ensamblado:

```
grompp -f ions.mdp -c 1aki_solv.gro -p topol.top -o ions.tpr
```

La salida obtenida es:

```
…
Analysing residue names:
There are:   129    Protein residues
There are: 10824      Water residues
There are:     8        Ion residues
Analysing Protein...
Analysing residues not classified as Protein/DNA/RNA/Water and splitting
into groups...
Number of degrees of freedom in T-Coupling group rest is 70845.00
Largest charge group radii for Van der Waals: 0.163, 0.157 nm
Largest charge group radii for Coulomb:       0.164, 0.163 nm
Calculating fourier grid dimensions for X Y Z
Using a fourier grid of 60x60x60, spacing 0.117 0.117 0.117
Estimate for the relative computational load of the PME mesh part: 0.30
…
```

Añadir iones:

genion_d -s ions.tpr -o 1aki_solv_ions.gro -p topol.top -pname NA -nname CL –nn 8

En este caso se selecciona la opción 13 (`Group 13 (SOL) has 32496 elements`). Información relevante en la salida estándar:

```
…
Number of (3-atomic) solvent molecules: 10832

Processing topology
Replacing 8 solute molecules in topology file (topol.top)  by 0 NA and 8
CL ions.

Back Off! I just backed up topol.top to ./#topol.top.2#
Replacing solvent molecule 1270 (atom 5770) with CL
Replacing solvent molecule 8207 (atom 26581) with CL
Replacing solvent molecule 5287 (atom 17821) with CL
Replacing solvent molecule 9164 (atom 29452) with CL
Replacing solvent molecule 3607 (atom 12781) with CL
Replacing solvent molecule 1735 (atom 7165) with CL
Replacing solvent molecule 4182 (atom 14506) with CL
Replacing solvent molecule 9597 (atom 30751) with CL
…
```

PASO 5: Minimización energética

Ahora es necesario asegurarse de que el sistema no tiene la geometría inapropiada para ello se “relaja” la estructura mediante el proceso denominado minimización energética (*Energy Minimization, EM*). El proceso es similar al anterior: primero se realiza un ensamblado de varios ficheros de entrada en un único fichero (`1aki_em.tpr`) y después se realiza el proceso de minimización energética. El fichero de entrada `minim.mdp` es:

```
; minim.mdp - used as input into grompp to generate 1aki_em.tpr
; Parameters describing what to do, when to stop and what to save
integrator       = steep          ; Algorithm (steep = steepest descent minimization)
emtol            = 1000.0         ; Stop minimization when the maximum force < 1000.0
kJ/mol/nm
emstep          = 0.01            ; Energy step size
nsteps           = 50000          ; Maximum number of (minimization) steps to perform

; Parameters describing how to find the neighbors of each atom and how to calculate the
interactions
nstlist          = 1              ; Frequency to update the neighbor list and long range
forces
ns_type          = grid           ; Method to determine neighbor list (simple, grid)
rlist            = 1.0            ; Cut-off for making neighbor list (short range
forces)
coulombtype      = PME            ; Treatment of long range electrostatic interactions
rcoulomb         = 1.0            ; Short-range electrostatic cut-off
rvdw             = 1.0            ; Short-range Van der Waals cut-off
pbc              = xyz            ; Periodic Boundary Conditions (yes/no)
```

Ensamblado:

grompp -f minim.mdp -c 1aki_solv_ions.gro -p topol.top -o 1aki_em.tpr

Minimización energética:

mdrun_d -v -deffnm 1aki_em

La salida estándar es:

```
…
Steepest Descents converged to Fmax < 1000 in 979 steps
Potential Energy  = -6.0590838e+05
Maximum force     =  9.3957776e+02 on atom 736
Norm of force     =  1.9575663e+01
…
```

PASO 6: Equilibrado del disolvente y los iones que rodean a la proteína

En este paso está dividido en dos fases y se equilibra la temperatura y la presión del sistema. Ambos procesos se realizan de forma similar a los anteriores: ensamblado + ejecución. Por lo tanto se necesitan dos nuevos ficheros de entrada `nvt.mdp` para la primera fase y `npt.mdp` para la segunda fase.

```
title            = OPLS Lysozyme NVT equilibration
define           = -DPOSRES       ; position restrain the protein
; Run parameters
integrator       = md             ; leap-frog integrator
nsteps           = 50000          ; 2 * 50000 = 100 ps
dt               = 0.002          ; 2 fs
; Output control
nstxout          = 100            ; save coordinates every 0.2 ps
nstvout          = 100            ; save velocities every 0.2 ps
nstenergy        = 100            ; save energies every 0.2 ps
nstlog           = 100            ; update log file every 0.2 ps
; Bond parameters
continuation     = no             ; first dynamics run
constraint_algorithm = lincs      ; holonomic constraints
constraints      = all-bonds      ; all bonds (even heavy atom-H bonds) constrained
lincs_iter       = 1              ; accuracy of LINCS
lincs_order      = 4              ; also related to accuracy
; Neighborsearching
ns_type          = grid           ; search neighboring grid cells
nstlist          = 5              ; 10 fs
rlist            = 1.0            ; short-range neighborlist cutoff (in nm)
rcoulomb         = 1.0            ; short-range electrostatic cutoff (in nm)
rvdw             = 1.0            ; short-range van der Waals cutoff (in nm)
; Electrostatics
coulombtype      = PME            ; Particle Mesh Ewald for long-range
electrostatics
pme_order        = 4              ; cubic interpolation
fourierspacing   = 0.16           ; grid spacing for FFT
; Temperature coupling is on
tcoupl           = V-rescale      ; modified Berendsen thermostat
tc-grps          = Protein Non-Protein    ; two coupling groups - more accurate
tau_t            = 0.1    0.1     ; time constant, in ps
ref_t            = 300    300     ; reference temperature, one for each group, in K
; Pressure coupling is off
pcoupl           = no             ; no pressure coupling in NVT
; Periodic boundary conditions
pbc              = xyz            ; 3-D PBC
; Dispersion correction
DispCorr         = EnerPres       ; account for cut-off vdW scheme
; Velocity generation
gen_vel          = yes            ; assign velocities from Maxwell distribution
gen_temp         = 300            ; temperature for Maxwell distribution
gen_seed         = -1             ; generate a random seed
```

```
title        = OPLS Lysozyme NPT equilibration
define       = -DPOSRES   ; position restrain the protein
; Run parameters
integrator = md           ; leap-frog integrator
nsteps       = 50000            ; 2 * 50000 = 100 ps
dt           = 0.002            ; 2 fs
; Output control
nstxout      = 100        ; save coordinates every 0.2 ps
nstvout      = 100        ; save velocities every 0.2 ps
nstenergy  = 100          ; save energies every 0.2 ps
nstlog       = 100        ; update log file every 0.2 ps
; Bond parameters
continuation = yes        ; Restarting after NVT
constraint_algorithm = lincs     ; holonomic constraints
constraints= all-bonds    ; all bonds (even heavy atom-H bonds) constrained
lincs_iter = 1            ; accuracy of LINCS
lincs_order= 4            ; also related to accuracy
; Neighborsearching
ns_type      = grid       ; search neighboring grid cells
nstlist      = 5          ; 10 fs
rlist        = 1.0        ; short-range neighborlist cutoff (in nm)
rcoulomb   = 1.0          ; short-range electrostatic cutoff (in nm)
rvdw         = 1.0        ; short-range van der Waals cutoff (in nm)
; Electrostatics
coulombtype= PME          ; Particle Mesh Ewald for long-range electrostatics
pme_order  = 4            ; cubic interpolation
fourierspacing      = 0.16       ; grid spacing for FFT
; Temperature coupling is on
tcoupl       = V-rescale  ; modified Berendsen thermostat
tc-grps      = Protein Non-Protein     ; two coupling groups - more accurate
tau_t        = 0.1  0.1   ; time constant, in ps
ref_t        = 300  300   ; reference temperature, one for each group, in K
; Pressure coupling is on
pcoupl       = Parrinello-Rahman ; Pressure coupling on in NPT
pcoupltype = isotropic    ; uniform scaling of box vectors
tau_p        = 2.0        ; time constant, in ps
ref_p        = 1.0        ; reference pressure, in bar
compressibility = 4.5e-5  ; isothermal compressibility of water, bar^-1
refcoord_scaling = com
; Periodic boundary conditions
pbc          = xyz        ; 3-D PBC
; Dispersion correction
DispCorr   = EnerPres     ; account for cut-off vdW scheme
; Velocity generation
gen_vel      = no         ; Velocity generation is off
```

Ensamblado y equilibrio de la temperatura:

grompp_d -f nvt.mdp -c 1aki_em.gro -p topol.top -o 1aki_nvt.tpr

mdrun_d -deffnm 1aki_nvt

Ensamblado y equilibrio de la presión:

grompp_d -f npt.mdp -c 1aki_nvt.gro -t 1aki_nvt.cpt -p topol.top -o 1aki_npt.tpr

mdrun_d -deffnm 1aki_npt

PASO 5. Simulación de dinámica molecular

El último paso es la simulación de dinámica molecular y, de nuevo, es necesario realizar un ensamblado previo utilizando un fichero *.mdp (`md.mdp`):

```
title            = OPLS Lysozyme MD
; Run parameters
integrator       = md             ; leap-frog integrator
nsteps           = 500000         ; 2 * 500000 = 1000 ps, 1 ns
dt               = 0.002          ; 2 fs
; Output control
nstxout          = 1000           ; save coordinates every 2 ps
nstvout          = 1000           ; save velocities every 2 ps
nstxtcout        = 1000           ; xtc compressed trajectory output every 2 ps
nstenergy        = 1000           ; save energies every 2 ps
nstlog           = 1000           ; update log file every 2 ps
; Bond parameters
continuation     = yes            ; Restarting after NPT
constraint_algorithm = lincs      ; holonomic constraints
constraints      = all-bonds      ; all bonds (even heavy atom-H bonds) constrained
lincs_iter       = 1              ; accuracy of LINCS
lincs_order      = 4              ; also related to accuracy
; Neighborsearching
ns_type          = grid           ; search neighboring grid cells
nstlist          = 5              ; 10 fs
rlist            = 1.0            ; short-range neighborlist cutoff (in nm)
rcoulomb         = 1.0            ; short-range electrostatic cutoff (in nm)
rvdw             = 1.0            ; short-range van der Waals cutoff (in nm)
; Electrostatics
coulombtype      = PME            ; Particle Mesh Ewald for long-range
electrostatics
pme_order        = 4              ; cubic interpolation
fourierspacing   = 0.16           ; grid spacing for FFT
; Temperature coupling is on
tcoupl           = V-rescale      ; modified Berendsen thermostat
tc-grps          = Protein Non-Protein    ; two coupling groups - more accurate
tau_t            = 0.1    0.1     ; time constant, in ps
ref_t            = 300    300     ; reference temperature, one for each group, in K
; Pressure coupling is on
pcoupl           = Parrinello-Rahman      ; Pressure coupling on in NPT
pcoupltype       = isotropic      ; uniform scaling of box vectors
tau_p            = 2.0            ; time constant, in ps
ref_p            = 1.0            ; reference pressure, in bar
compressibility = 4.5e-5  ; isothermal compressibility of water, bar^-1
; Periodic boundary conditions
pbc              = xyz            ; 3-D PBC
; Dispersion correction
DispCorr         = EnerPres       ; account for cut-off vdW scheme
; Velocity generation
gen_vel          = no             ; Velocity generation is off
```

Ensamblado:

```
grompp -f md.mdp -c 1aki_npt.gro -t 1aki_npt.cpt -p topol.top -o
1aki_md.tpr
```

Simulación:

```
mdrun -deffnm 1aki_md
```

La ejecución de `mdrun` genera cuatro ficheros de salida:

- Fichero con todas las variables de estado (en intervalos temporales) del sistema simulado (fichero `1aki_md.cpt`)
- Información acerca de la energía del sistema (fichero `1aki_md.edr`)
- Estructura post-procesada (fichero `1aki_md.gro`)
- Log de la ejecución (fichero `1aki_md.log`)
- Trayectoria de la molécula: coordenadas, velocidades y fuerzas sobre el sistema (fichero `1aki_md.trr`)
- Versión comprimida de la trayectoria que contiene solo información sobre las coordenadas, tiempo y el recipiente (fichero `1aki_md.xtc`)

A continuación se muestra en forma de esquema un resumen de los distintos pasos de ejecución necesarios para llevar a cabo una simulación de dinámica molecular. Los recuadros en gris indican la herramienta que ha de utilizarse y junto a ellos se describen los ficheros de entrada y los de salida.

Set Up Topology

-f *input*.**pdb**
-o *output*.**gro**
-p *output*.**top**
(-heavyh)

Generate Box

-f *input*.**gro**
-o *output*.**gro**
-bt tric/cubic/….
-box *dimensions* (nm)

Multiple Molecules

-cp *input*.**gro**
-ci *input*.**gro**
-p *input/output*.**top**
-o *output*.**gro**
-nmol *number*
-seed *number*

Solvate Molecules

-cp *input*.**gro**
-cs *blank*
-p *input/output*.**top**
-o *output*.**gro**

Generate Genion Input File

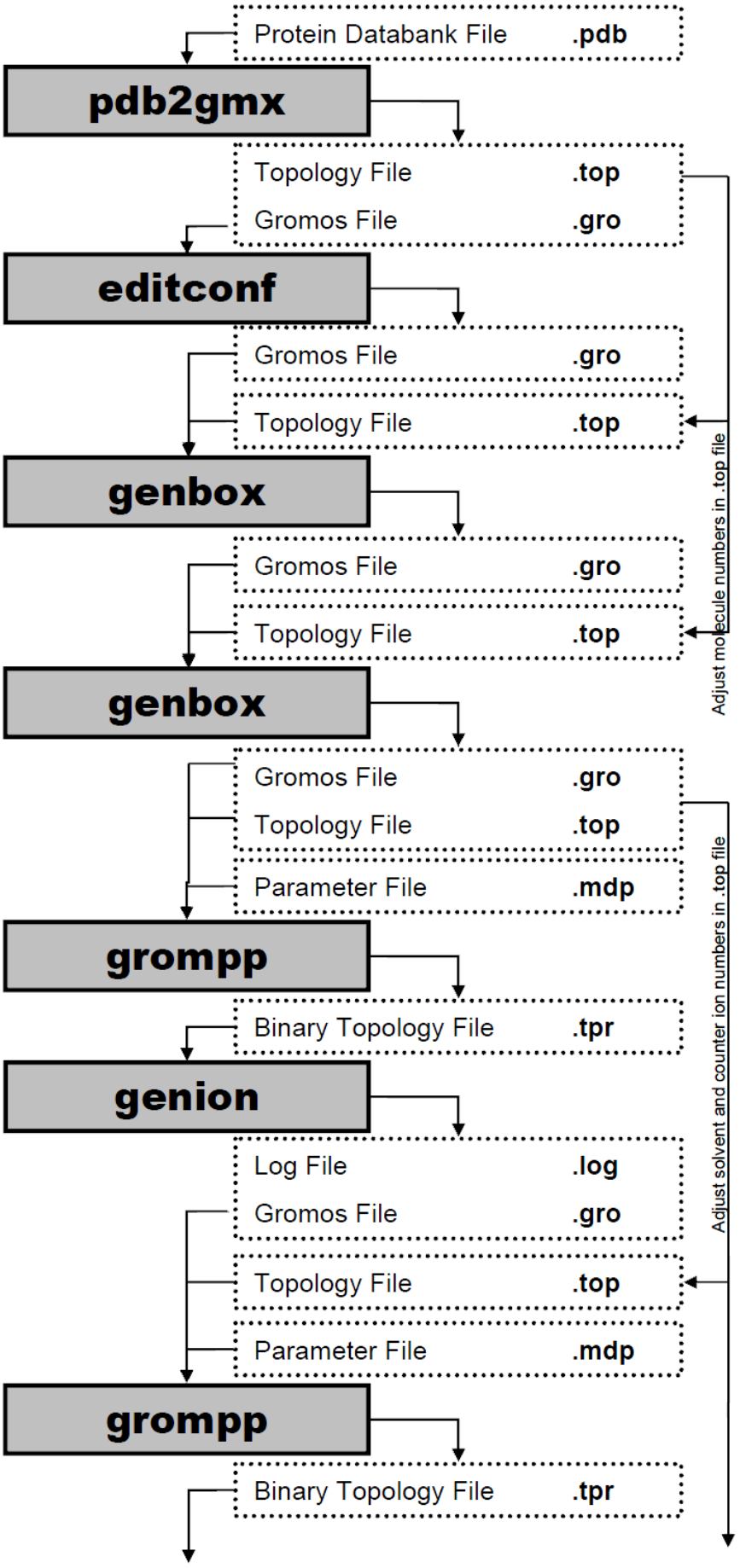
-f *input*.**mdp**
-c *input*.**gro**
-p *input*.**top**
-o *output*.**tpr**

Counter Ions

-s *input*.**tpr**
-o *output*.**gro**
-g *output*.**log**
-np/nn *number*
-pname/nname *name +ve/-ve*
-pq/nq *charge*
-random
-seed *number*

Generate MD Input File

-f *input*.**mdp**
-c *input*.**gro**
-p *input*.**top**
-o *output*.**tpr**



MD for Energy Minimisation

-s *input*.**tpr**
-o *output*.**trr**
-c *output*.**gro**
-e *energy*.**edr**
-g *output*.**log**
-v <*verbose*>

Generate MD Input File

-f *input*.**mdp**
-c *input*.**gro**
-p *input*.**top**
-o *output*.**tpr**

MD PR for Water Equilibration

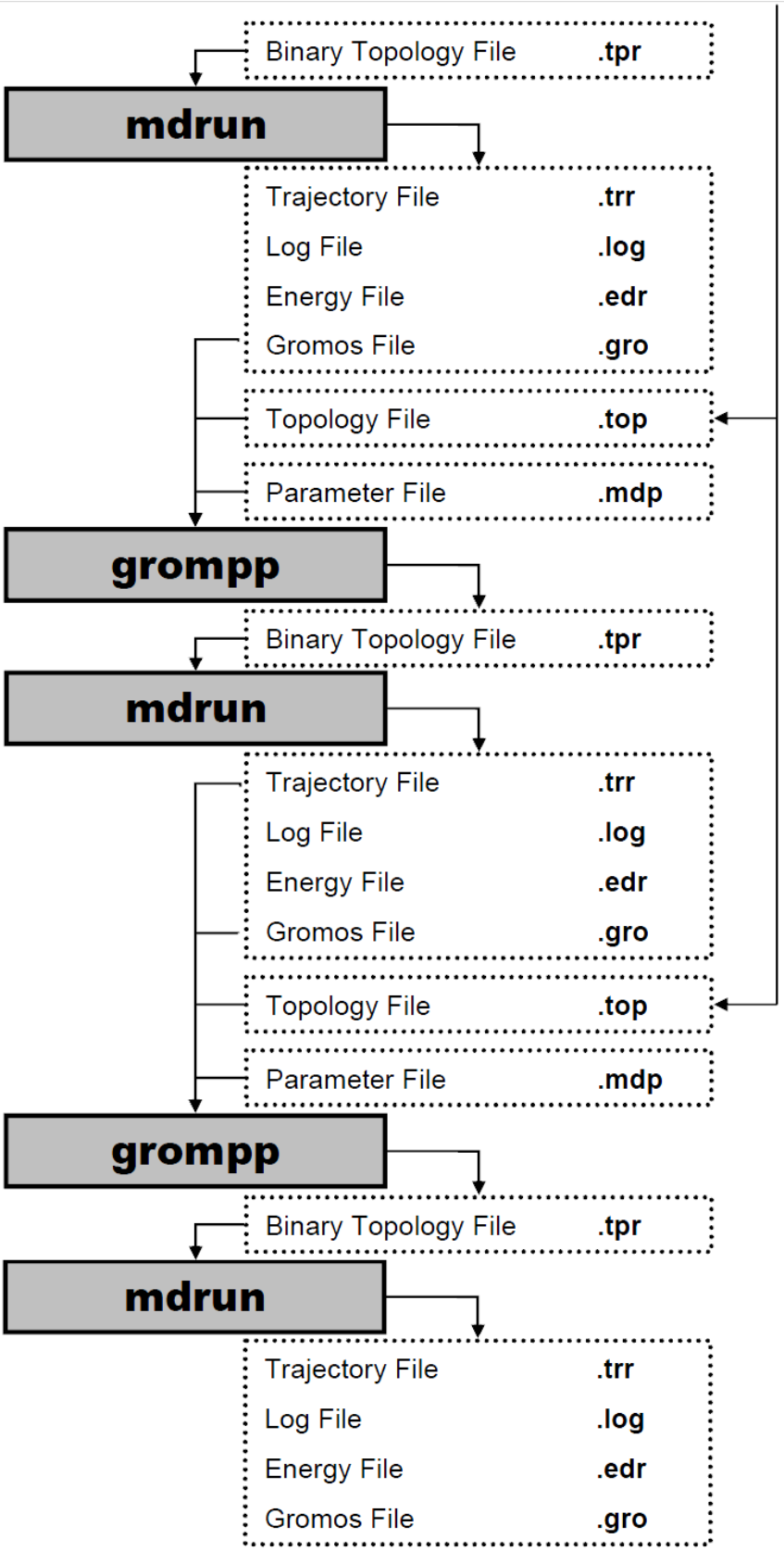
-s *input*.**tpr**
-o *output*.**trr**
-c *output*.**gro**
-e *energy*.**edr**
-g *output*.**log**
-v <*verbose*>

Generate MD Input File

-f *input*.**mdp**
-c *input*.**gro**
-t *input*.**trr**
-p *input*.**top**
-o *output*.**tpr**
-np *number*

MD Simulation

-s *input*.**tpr**
-o *output*.**trr**
-c *output*.**gro**
-e *energy*.**edr**
-g *output*.**log**
-v <*verbose*>



ANEXO III

Análisis de las matrices dispersas

Antes de comenzar el desarrollo del nuevo algoritmo ILVES-S fue necesario realizar un análisis del tipo de matrices que se pueden dar en las moléculas más comunes. Para ello, se modificó el código de GROMACS para, a partir de datos reales, obtener ficheros que contengan la estructura y los valores reales de la matriz A (en un determinado paso temporal) (véase ECUACIÓN 9).

El objetivo es hacerse una idea de cómo son las matrices con las que se va a tratar para poder desarrollar una técnica eficiente de reordenación de ligaduras que proporcione las matrices banda. Se obtienen cuatro matrices por molécula; una por cada uno de los tipos de ligaduras que se pueden imponer al sistema (`constraints = {hbonds, all-bonds, h-angles, all-angles}`). Las moléculas analizadas son:

- Aminoácidos: componente principal de las proteínas. Se obtiene la matriz A para cadenas de 3 aminoácidos iguales.
    - Alanina (ALA)
    - Arginina (ARG)
    - Asparagina (ASN)
    - Aspartato (ASP)
    - Cisteína (CYS)
    - Glutamato (GLU)
    - Glutamina (GLN)
    - Glicina (GLY)
    - Histidina (HIS)
    - Isoleucina (ILE)
    - Leucina (LEU)
    - Lisina (LYS)
    - Metionina (MET)
    - Fenilalanina (PHE)
    - Prolina (PRO)
    - Serina (SER)
    - Treonina (THR)
    - Triptófano (TRP)
    - Tirosina (TYR)
    - Valina (VAL)
- Nucleótidos: componentes del ADN y del ARN, las matrices se analizan para el caso particular de los nucleótidos monofosfatados:
    - Adenosín monofosfato (AMP)
    - Citidina monofosfato (CMP)
    - Ácido guanílico (GMP)
    - Timidilato (TMP)
    - Uridina monofosfatada (UMP)

Como ejemplo, se muestra el resultado del análisis realizado para el aminoácido aspartato (ASP). Su estructura química puede observarse en la FIGURA 19 y su fórmula molecular es:

$$C_4H_7NO_4$$

Como se ha mencionado anteriormente, el análisis se realiza para una cadena de 3, en este caso moléculas de aspartato (véase FIGURA 20), que tiene 39 átomos y 38 enlaces. Su fórmula química es:

$$C_{12}H_{14}N_3O_{10}$$

En la FIGURA 20 los átomos rojos son Oxigeno, los azules Nitrógeno, los blancos Hidrógeno y los grises son átomos de Carbono.

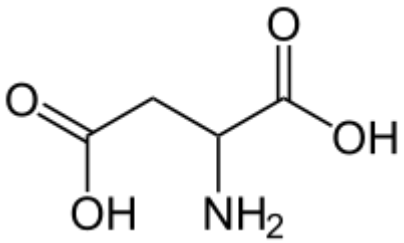


Figura 19. Estructura química del aspartato

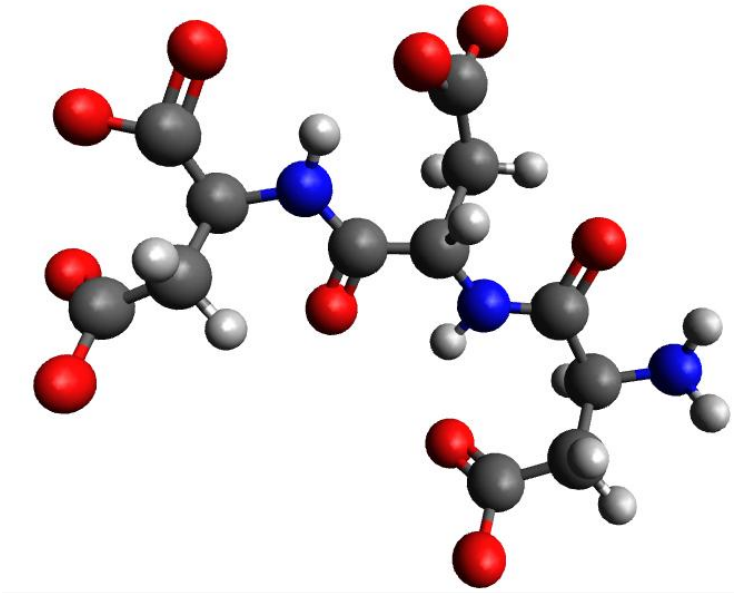


Figura 20. Modelo tridimensional de una cadena de 3 moléculas de aspartato

Para este ejemplo se ha considerado el caso en el que se imponen ligaduras a todos los enlaces (`all-bonds`), por lo tanto, hay tantas ligaduras como enlaces: 39. La tabla de la FIGURA 21 muestra dos columnas para cada residuo o para cada enlace entre dos residuos. La primera columna, denominada "Índice", contiene los valores de k, siendo k el índice de la ligadura que corresponde a la posición (fila o columna) que ocupa dicha ligadura en la matriz A. La segunda columna, denominada "Ligadura", contiene las ligaduras representadas como (i, j), siendo i y j los índices de los átomos ligados. Por ejemplo, el primer elemento de la tabla, índice 0 y ligadura $(0, 1)$ representa $\sigma_{k(i,j)} = \sigma_{0(0,1)}$ (de acuerdo a la ECUACIÓN 6).

La tabla está dividida en 5 partes: tres de ellas corresponden a los tres residuos de aspartato y las otras dos partes corresponden a los enlaces que unen los residuos (líneas azules en la matriz, FIGURA 22):

Residuo 1		Enlace entre los residuos 1-2		Residuo 2		Enlace entre los residuos 2-3		Residuo 3	
Indice	Ligadura	Indice	Ligadura	Indice	Ligadura	Indice	Ligadura	Indice	Ligadura
0	(0, 1)	13	(12, 14)	14	(14, 15)	25	(24, 26)	26	(26, 27)
1	(0, 2)			15	(14, 16)			27	(26, 28)
2	(0, 3)			16	(16, 17)			28	(28, 29)
3	(0, 4)			17	(16, 18)			29	(28, 30)
4	(4, 5)			18	(16, 24)			30	(28, 36)
5	(4, 6)			19	(18, 19)			31	(30, 31)
6	(4, 12)			20	(18, 20)			32	(30, 32)
7	(6, 7)			21	(18, 21)			33	(30, 33)
8	(6, 8)			22	(21, 22)			34	(33, 34)
9	(6, 9)			23	(21, 23)			35	(33, 35)
10	(9, 10)			24	(24, 25)			36	(36, 37)
11	(9, 11)							37	(36, 38)
12	(12, 13)								

Figura 21. Tabla que contiene la descripción de las ligaduras de una cadena de 3 moléculas de aspartato

A continuación se muestra la matriz A obtenida:

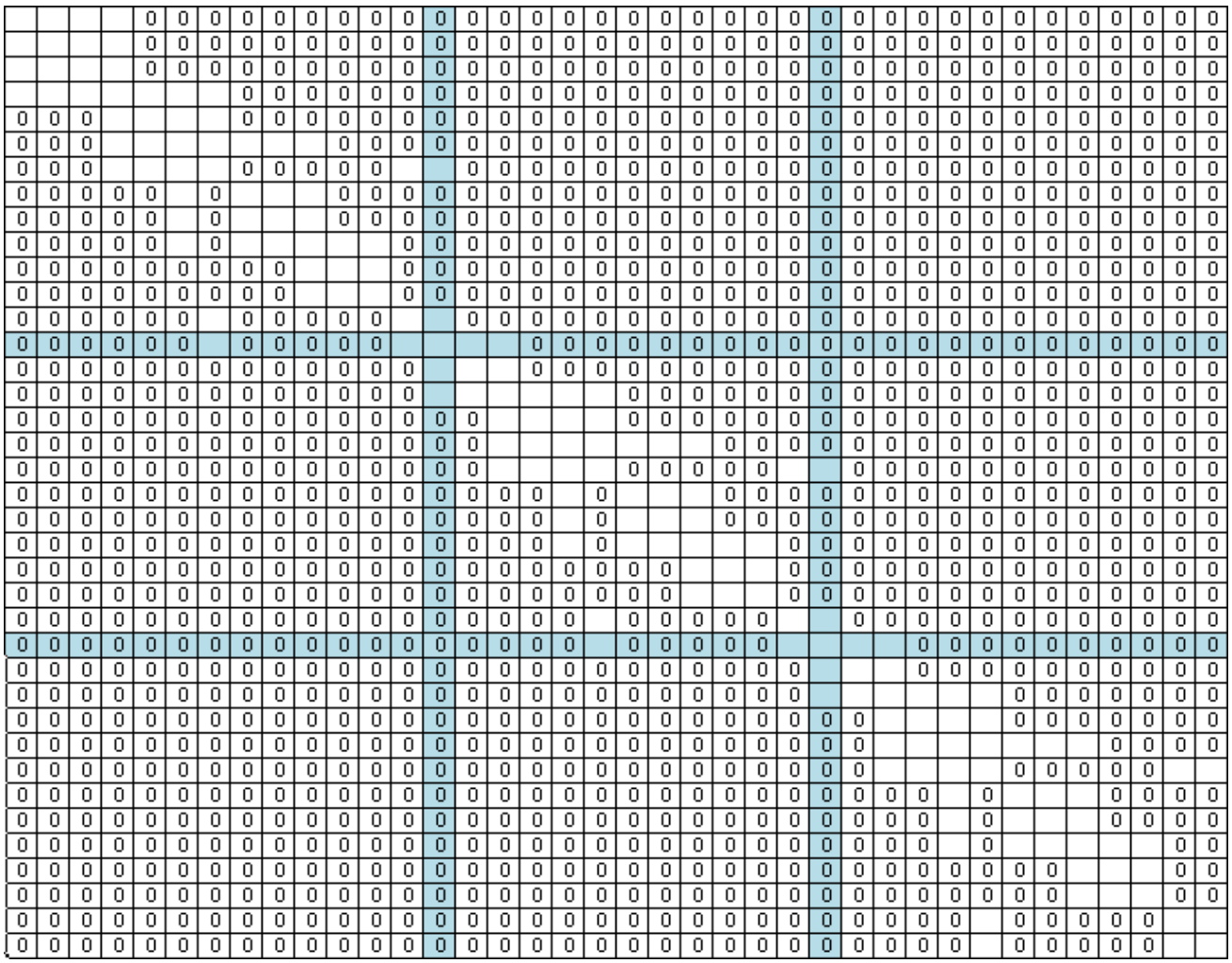


Figura 22. Matriz A correspondiente a una cadena de 3 moléculas de aspartato

Los índices de las filas y columnas de la matriz A son los índices de las ligaduras, es decir, son matrices cuadradas. Además, de la ecuación 9 se deduce que las matrices A representan con un valor distinto de 0 dos ligaduras que tienen algún átomo en común y con un 0, dos ligaduras que no tienen átomos comunes. Por lo tanto, debido a la estructura característica de las moléculas, las matrices A son matrices dispersas que, como se ha mencionado anteriormente, se pueden convertir en matrices banda reordenando las ligaduras.

Un espacio en blanco en la matriz representa un valor distinto de cero. Por ejemplo, si nos fijamos en el elemento $A(0,2)$, es un valor distinto de cero porque las ligaduras $\sigma_{0(0,1)}$ y $\sigma_{2(0,3)}$ tienen en común el átomo 0. Sin embargo, si nos fijamos en el elemento $A(1,4)$, su valor es cero porque las ligaduras $\sigma_{1(0,2)}$ y $\sigma_{4(4,5)}$ no tienen ningún átomo en común.

ANEXO IV

Trabajo desarrollado en el ICHEC (Irlanda)

Los conocimientos de dinámica molecular adquiridos durante el desarrollo de este proyecto fin de carrera, me han ayudado a conseguir una beca competitiva convocada por PRACE (*Partnership for Advanced Computing in Europe*) [19], para realizar una estancia durante el verano en el *Ireland's High-Performance Computing Centre* (ICHEC) [20] en Dublín, Irlanda. El resultado del proyecto desarrollado denominado "*Profiling and optimization of th the hybrid Molecular Dynamics code, DL_POLY, on heterogeneous clusters*", puede consultarse a partir de la siguiente página.

Profiling and optimization of the hybrid Molecular Dynamics code, DL_POLY, on heterogeneous clusters

Profiling and optimization of DL_POLY

Maria Aston Serrano

DL_POLY is a general purpose classical Molecular Dynamics (MD) software application. The aim of this SoHPC project has been to profile this application and to optimize it for the latest NVIDIA Kepler K20 GPU architecture.

ICHEC
Irish Centre for High-End Computing

Molecular dynamics is a computational technique for simulating the interaction between atoms and molecules within a period of time. These simulations involve a high consumption of resources, both memory and CPU cycles. Therefore, these applications are specially suited for HPC (High-Performance Computing) environments.

DL_POLY is developed by the Science and Technology Facilities Council (STFC) in Daresbury Laboratory, UK [1]. DL_POLY is a package of subrutines, programs and data files, designed to facilitate molecular dynamics simulations on a distributed memory parallel computer. It is used world-wide in many areas of physics, chemistry and biology - wherever the classical behaviour of atoms and molecules needs to be understod. It has been applied in a broad range of scientific studies since its public release in 1996. It has over 14.000 user licenses worldwide since 1994 and 1200 registered users to the forum since 2005.

DL_POLY has been parallelized using a Domain Decomposition (DD) strategy. Its name derives from the division of the simulated system into equi-geometrical spatial blocks or domains, each of which is allocated to a specific processor of a parallel computer. I.e. the arrays defining the atomic coordinates, velocities and forces, for all N atoms in the simulated system, are divided in to sub-arrays of approximate size N/P, where P is the number of processors, and allocated to specific processors, employing the Message Passing Interface (MPI) library for inter-process communication.

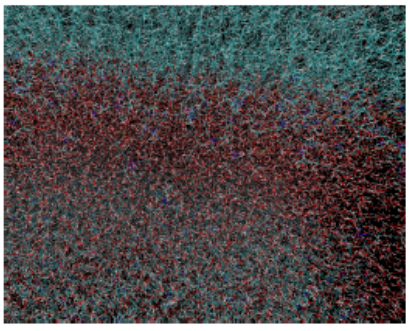


Figure 1: DMPC in water.

In collaboration with the Irish Centre for High-End Computing (ICHEC) [2], a CUDA + OpenMP port of DL_POLY has been developed. So far, the developers at ICHEC have focused on enabling DL_POLY for the NVIDIA Fermi architecture.

The work performed in this Summer of HPC project has aimed to provide a detailed profiling of the hybrid Molecular Dynamics (MD) application, DL_POLY. Based on visualizations of the profiling results, the further aim of this project has been to optimize DL_POLY for the latest NVIDIA Kepler K20 GPU.

Algorithms on GPUs: the kernels

The first part of this project was to gain a deper understanding of general purpose GPU computing (combination of GPU and CPU) and how it works in practice. CUDA is a parallel computing platform and programming model that enables dramatic increases in computing performance by harnessing the

power of the Graphics Processing Unit (GPU).

link_cell_pairs_cuda_k1()		
Parameter		Environment variable
BLOCK Size	x	dlpolycuda_linkcellpairs_k1_block_dims_x
	y	dlpolycuda_linkcellpairs_k1_block_dims_y
	z	dlpolycuda_linkcellpairs_k1_block_dims_z

two_body_forces_cuda_k2()		
Parameter		Environment variable
GRID Size	y	dlpolycuda_twobody_k2_unroll_slices
BLOCK Size	x	dlpolycuda_twobody_k2_block_dims_x

two_body_forces_cuda_k1()		
Parameter		Environment variable
GRID Size	y	dlpolycuda_twobody_k1_grid_dims_y

ewald_spme_forces_cuda_ccarray_k1()		
Parameter		Environment variable
GRID Size	x	dlpolycuda_ewald_spme_forces_ccarray_k1_grid_dims_x
	y	dlpolycuda_ewald_spme_forces_ccarray_k1_grid_dims_y

spme_forces_cuda_k1()		
Parameter		Environment variable
GRID Size	y	dlpolycuda_spme_forces_grid_dim_y

Figure 2: Environment variables to change some kernel parameters.

CUDA C extends C by allowing the programmer to define C functions, called kernels, that, when called, are executed N times in parallel by N different CUDA threads. Threads are grouped into blocks and blocks are organised in a grid so that the total number of threads is equal to the number of threads per block times the number of blocks. Moreover, blocks and grids can have 1, 2 or 3 dimensions. Often a challenging task is to select the optimal combination of these parameters: 3D grid size and 3D block size.

link_cell_pairs_cuda_k1()						
GRID Size			Block Size			Time
x	y	z	x	y	z	
4096	1	1	1	1	1	20.1676s
4096	1	1	1	1	2	13.8833s
4096	1	1	1	1	4	8.98367s
4096	1	1	1	1	8	5.40246s
4096	1	1	1	1	16	3.16902s
4096	1	1	1	1	32	1.91245s
4096	1	1	1	1	64	1.17598s
4096	1	1	1	2	1	15.9982s
4096	1	1	1	2	2	10.5035s
4096	1	1	1	2	4	6.42362s
4096	1	1	1	2	8	3.74302s
4096	1	1	1	2	16	2.18555s
4096	1	1	1	2	32	1.25382s
4096	1	1	1	2	64	0.87194s
4096	1	1	1	4	1	12.6406s
4096	1	1	1	4	2	7.99251s
4096	1	1	1	4	4	4.70627s
4096	1	1	1	4	8	2.71418s
4096	1	1	1	4	16	1.50354s
4096	1	1	1	4	32	1.00032s
4096	1	1	1	4	64	0.87744s

link_cell_pairs_cuda_k1()						
GRID Size			Block Size			Time
x	y	z	x	y	z	
4096	1	1	2	1	1	14.5981s
4096	1	1	2	1	2	9.53277s
4096	1	1	2	1	4	5.71722s
4096	1	1	2	1	8	3.30419s
4096	1	1	2	1	16	1.92418s
4096	1	1	2	1	32	1.09680s
4096	1	1	2	1	64	0.79372s
4096	1	1	2	2	1	11.2675s
4096	1	1	2	2	2	6.93750s
4096	1	1	2	2	4	4.03961s
4096	1	1	2	2	8	2.30932s
4096	1	1	2	2	16	1.25796s
4096	1	1	2	2	32	0.78957s
4096	1	1	2	2	64	0.70742s
4096	1	1	2	4	1	8.69114s
4096	1	1	2	4	2	5.13231s
4096	1	1	2	4	4	2.93261s
4096	1	1	2	4	8	1.58245s
4096	1	1	2	4	16	1.00735s
4096	1	1	2	4	32	0.89514s
4096	1	1	2	4	64	0.91423s

link_cell_pairs_cuda_k1()						
GRID Size			Block Size			Time
x	y	z	x	y	z	
4096	1	1	4	1	1	10.2009s
4096	1	1	4	1	2	6.24003s
4096	1	1	4	1	4	3.61101s
4096	1	1	4	1	8	2.06161s
4096	1	1	4	1	16	1.12471s
4096	1	1	4	1	32	0.79485s
4096	1	1	4	1	64	0.77657s
4096	1	1	4	2	1	7.67479s
4096	1	1	4	2	2	4.48657s
4096	1	1	4	2	4	2.55216s
4096	1	1	4	2	8	1.36684s
4096	1	1	4	2	16	0.85296s
4096	1	1	4	2	32	0.74399s
4096	1	1	4	2	64	0.75610s
4096	1	1	4	4	1	5.83987s
4096	1	1	4	4	2	3.34591s
4096	1	1	4	4	4	1.86432s
4096	1	1	4	4	8	1.27235s
4096	1	1	4	4	16	1.01603s
4096	1	1	4	4	32	1.00009s
4096	1	1	4	4	64	1.09611s

Figure 3: Kernel *link_cell_pirs_cuda_k1()* execution time.

The CUDA version of DL_POLY, that was developed at ICHEC, was designed for the Fermi GPU architecture. The following list shows the most compute intensive components of DL_POLY that have been enabled on the Fermi GPU architecture.

- constraints_shake
- ewald_spme_forces
- link_cell_pairs
- metal_ld_compute
- spme_container
- spme_forces
- two_body_forces

Profiling and optimizing DL_POLY

The overall goal of this project was to profile the CUDA version of DL_POLY on NVIDIAS's newest GPU architecture called the Kepler K20. Based on the analysis of the initial performance of DL_POLY on the K20, the further aim of this project was to investigate basic optimizations targeting the K20 architecture.

Kepler architecture	GRID Size			Block Size			Time (seconds)	Speed-up
	x	y	z	x	y	z		
Default version	4096	1	1	1	1	32	1.917000	2.706905
Optimized version	4096	1	1	2	2	64	0.708189	

Figure 4: *link_cell_pirs_cuda_k1()* performance on Kepler architecure

It should be pointed out that the two GPU cards used in this project (Tesla M2090 Fermi and Tesla K20 Kepler architecture) are considerably different and therefore a particular set of optimizations for one architecture may not work well on the other architecture. Therefore, my work in this project has been to analyse the kernels in DL_POLY and to find the optimal combination of the block and grid dimensions in terms of performance. To do this, I have used several advanced tools to profile DL_POLY on GPUs. For example, throughout my project I have used NSIGHT to help search for the optimal CUDA parameters for the new K20 architecture.

NVIDIA Nsight is an Eclipse-based development platform for heterogeneous computing which includes powerful debugging and profiling tools. Using this platform the DL_POLY code has been analysed. The profiling tool provides a visual output with useful information for each kernel: execution time, number of invocations, grid and block size, etc. It also provides information on the duration of each kernel on the GPU device.

Once the most compute intensive kernels have been found, it is necesary to analyse the source code and to find the grid size and block size parameters and their default values. Some of these parameters can be modified as environment variables. The table in the Figure 2 shows the environment variables that can be used to change the parameters of the analyzed kernels.

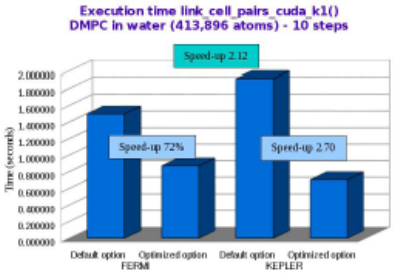


Figure 5: *link_cell_pirs_cuda_k1()* performance on Kepler and Fermi architecures

The use of scripts allows me to easily search through the CUDA parameter space and to get the execution time of each kernel for different configurations. For example, for the kernel *link_cell_pairs_cuda_k1()* the Figure 3 shows the times obtained for the different configurations on the Tesla K20 Kepler GPU. The example corresponds to the simulation of DMPC (dimyristoylphosphatidylcholine) in water (413,896 atoms), a visualization of which you can see in the Figure 1 (this simulation is carried out for 10 time steps and the time corresponds to the sum of all the kernel invocations).

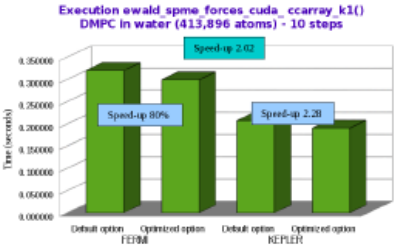


Figure 6: *ewald_spme_forces_cuda_ccarray_k1()* performance on Kepler and Fermi architecures.

The red box denotes the default configuration for this example and the green box denotes the optimal one. The execution times are 1.9170 seconds and 0.7081 seconds, respectively. This means that, as the table in the Figure 4 shows, the speed-up is 2.7X

which is a very significant result because it means that for this particular kernel the performance can be almost three times faster if non-default combinations of the parameters are selected.

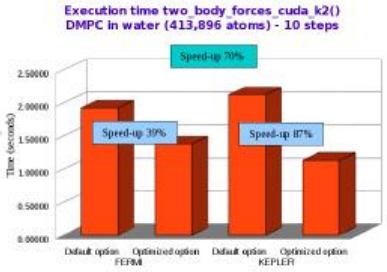


Figure 7: *two_body_forces_cuda_k2()* performance on Kepler and Fermi architecures.

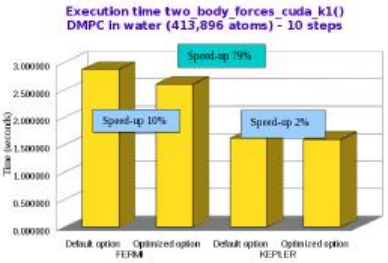


Figure 8: *two_body_forces_cuda_k1()* performance on Kepler and Fermi architecures.

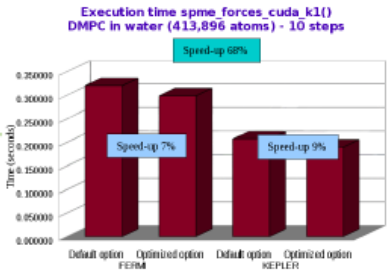


Figure 9: *spme_forces_cuda_k1()* performance on Kepler and Fermi architecures.

For this example, kernel *link_cell_pirs_cuda_k1()*, the comparison between the execution times on the Kepler and Fermi architectures is shown in Figure 5. It is interesting to note that the default configuration in the Fermi architecture performes better than the default configuration in the Kepler architecture, but if we select the optimized configuration in Kepler, the speed-up is 2.12X which means that this kernel is 2.12 times faster on Kepler than on Fermi.

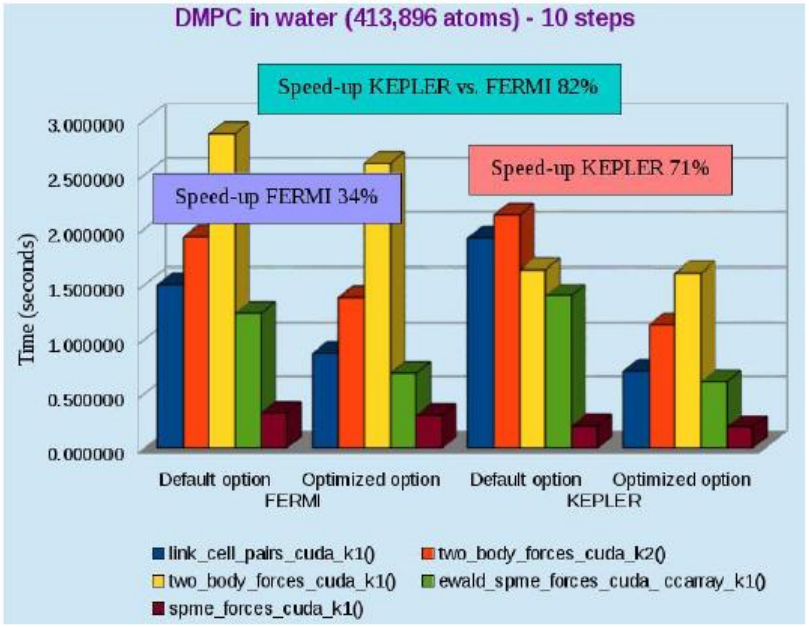


Figure 10: Performance on Kepler and Fermi architecures.

The same analysis has been made for the rest of the kernels and the results are shown in the Figures 6,7,8 and 9. For the kernels in the Figures 6 and 7 (*ewald_spme_forces_cuda_ccarray_k1* and *two_body_forces_cuda_k2*) the optimizations achieved are significant and the analysis results in a more efficient application. However, the differences between the default and the optimized version on the K20 of the kernel shown in the Figures 8 and 9 (*spme_forces_cuda_k1* and *two_body_forces_cuda_k1*), are not as significant as the previous kernels. Even so, these graphs demostrate the improvment that can be achieved by carrying out basic optimizations on the K20 GPU architecture.

Final results

Figure 10 summarizes the global result achieved. For this particular case, the kernels analyzed for the simulation of DMPC in water, DL_POLY performs 82% faster on the K20 architecture than on the Fermi architecture. However, just as significant a finding in this project has been that, due to the architectural differences between the K20 and the Fermi GPU, CUDA configuration parameters that are optimal for one architecture may not necessarily be optimal for the other architecture. It was found during this project that fine-tuning of CUDA configuration parameters when enabling on a new architecture can, in fact, result in a significant performance improvement of the application.

PRACE SoHPCProject Title
Profiling the hybrid Molecular Dynamics code, DL_POLY, on heterogeneous clusters

PRACE SoHPCSite
Irish Centre for High End Computing (ICHEC), Ireland


Maria Aston Serrano

PRACE SoHPCAuthors
Maria Aston Serrano, Spain

PRACE SoHPCMentor
Michael Lysaght, ICHEC, Ireland

PRACE SoHPCContact
Serrano, Maria Aston
Phone: +34 625 175 620
E-mail: m.aston.serrano@gmail.com

PRACE SoHPCSoftware applied
DL_POLY

PRACE SoHPCMore Information
www.stfc.ac.uk/cse/25526.aspx

PRACE SoHPCThanks
First of all, I would like to thank to my project mentor, Michael Lysaght, for his help, his time and all I have learnt over these weeks. I wish to express my sincere gratitude to all the people in ICHEC, specially to Emma, Nix and Simon. Finally, I would like to express my gratitude towards PRACE for this incredible opportunity.

PRACE SoHPCReferences

[1] DL_POLY:
http://www.stfc.ac.uk/cse/25526.aspx

[2] ICHEC:
http://www.ichec.ie/

REFERENCIAS

[1] Ryckaert, J. P., Ciccotti, G., Berendsen, H. J. C. *Numerical integration of the cartesian equations of motion of a system with constraints; molecular dynamics of n-alkanes*. J.Comp. Phys. 23:327–341, 1977.

[2] http://www.gromacs.org
Fecha última consulta: 17-agosto-2013

[3] García-Risueño, P. *Constraint implementation based on analytical calculations: a posible way to improve widely used solvers*. Technical Report.

[4] Hess, B.; Bekker, H.; Berendsen, H. J. C. and Fraaije, J. G. E. M. *LINCS: A Linear Constraint Solver for Molecular Simulations.* J.Comp. Chem. 18: 1463-1472, 1997

[5] http://www.fftw.org
Fecha última consulta: 17-agosto-2013

[6] http://www.cgl.ucsf.edu/chimera/
Fecha última consulta: 17-agosto-2013

[7] http://avogadro.openmolecules.net/wiki/Main_Page
Fecha última consulta: 17-agosto-2013

[8] http://www.gnu.org/software/ddd/
Fecha última consulta: 17-agosto-2013

[9] http://www.netlib.org/lapack/
Fecha última consulta: 17-agosto-2013

[10] http://www.doxygen.org
Fecha última consulta: 19-agosto-2013

[11] http://www.bevanlab.biochem.vt.edu/Pages/Personal/justin/gmx-tutorials/lysozyme/index.html
Fecha última consulta: 19-agosto-2013

[12] Hockney, R. W., Goel, S. P., Eastwood, J. *Quiet High Resolution Computer Models of a Plasma*. J. Comp. Phys. 14:148–158, 1974.

[13] Swope, W. C., Andersen, H. C., Berens, P. H., Wilson, K. R. *A computer-simulation method for the calculation of equilibrium-constants for the formation of physical clusters of molecules: Application to small water clusters*. J. Chem. Phys. 76:637–649, 1982.

[14] http://www.gromacs.org/About_Gromacs/People
Fecha última consulta: 1-Agosto-2013

[15] http://folding.stanford.edu/English/HomePage
Fecha última consulta: 1-Agosto-2013

[16] http://www.evogrid.org/index.php/Main_Page
Fecha última consulta: 1-Agosto-2013

[17] http://www.scalalife.eu/
Fecha última consulta: 1-Agosto-2013

[18] http://www.spec.org/cpu2006/Docs/435.gromacs.html
Fecha última consulta: 28-agosto-2013

[19] http://www.prace-ri.eu/
Blog del programa de verano (*Summer of HPC*): https://summerofhpc.prace-ri.eu/
Fecha última consulta: 28-agosto-2013

[20] http://www.ichec.ie/
Fecha última consulta: 28-agosto-2013

[21] Lennard-Jones, J. E. Cohesion. Proceedings of the Physical Society **1931**, 43, 461-482

[22] http://www.gromacs.org/Documentation/Installation_Instructions_4.5
Fecha última consulta: 16-agosto-2013

[23] http://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes
Fecha última consulta: 16-agosto-2013

[24] http://es.wikipedia.org/wiki/Enlace_din%C3%A1mico
Fecha última consulta: 16-agosto-2013

[25] http://www.rcsb.org/pdb/home/home.do
Fecha última consulta: 21-agosto-2013