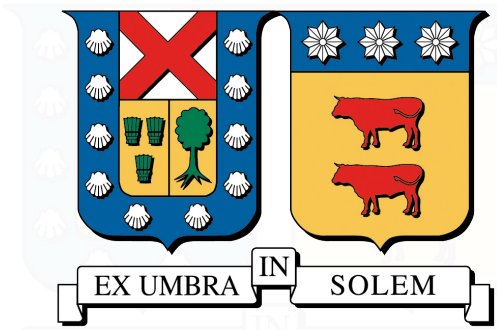
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INGENIERÍA MECÁNICA
VALPARAÍSO - CHILE



ESTUDIO NUMÉRICO DE LA ELECTROSTÁTICA EN BIOMOLÉCULAS DE GRAN TAMAÑO, MEDIANTE HERRAMIENTAS DE COMPUTACIÓN DE ALTO DESEMPEÑO.

MATÍAS BASTIÁN MARTÍNEZ ALVARADO

MEMORIA PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA MECÁNICA

PROFESOR GUÍA	:	PHD. CHRISTOPHER COOPER
PROFESOR CORREFERENTE	:	PHD. ALEJANDRO PACHECO
PROFESOR CORREFERENTE	:	PHD. HORACIO GUZMÁN

NOVIEMBRE 2020

Quisiera agradecer a Dios por estar conmigo a lo largo de este proceso y por poner en mi camino a las personas indicadas. A mi familia y mis amigos, entre ellos José Allendes, quién me apoyó mucho durante la realización de esta tesis. Al profesor Cooper y la gente del departamento de ingeniería mecánica, por su total apoyo en los momentos complicados y por tenerme la paciencia que creí que no merecía. A Horacio Guzmán por la colaboración que dio como resultado la publicación con la que hoy opto al grado de magíster. En general, a todas las personas con las que me he cruzado en este tiempo. Todas han influido en mí de alguna u otra manera y me han hecho ser la persona que está sentada escribiendo este documento.

Índice de Contenidos

1. Marco Teórico **1**
1.1. Modelo de solvente implícito . . . 1
1.1.1. Electrostática en el vacío . . . 1
1.1.2. Electrostática en un dieléctrico continuo . . . 4
1.1.3. Efecto de una solución electrolítica . . . 7
1.1.4. Cálculo de energías con el modelo de solvente implícito . . . 13
1.1.5. Cálculos usando el modelo de solvente implícito . . . 15
1.2. Método de Elementos de Borde (BEM) . . . 17
1.2.1. Derivación del problema de Laplace . . . 17
1.2.2. Operadores de capa . . . 20
1.2.3. Discretización y sistema lineal . . . 24
1.2.4. Métodos de multipolo . . . 24
1.3. Treecode . . . 25
1.3.1. Treecode en BEM . . . 29
1.4. Computación paralela . . . 29
1.4.1. Computadores de memoria compartida: SMP . . . 31
1.4.2. Computadores de memoria distribuida: DSM . . . 33
1.4.3. Modelos de programación paralela . . . 34
1.4.4. Programación en SMP y OpenMP . . . 35
1.4.4.1. OpenMP . . . 36
1.4.4.2. Características de OpenMP . . . 37
1.4.4.3. Distribución de trabajo . . . 38
1.4.4.4. Modelo de memoria de OpenMP . . . 39
1.4.4.5. Sincronización . . . 40
1.4.4.6. Número de threads y números de thread . . . 41
1.4.4.7. Correctitud: Condición de carrera . . . 42
1.4.4.8. Desempeño . . . 42
1.5. Proteínas y virus . . . 44
1.5.1. Aminoácidos y enlace peptídico . . . 44
1.5.2. Estructura de proteínas . . . 45
1.5.3. Virología . . . 48
1.5.3.1. Estructura viral . . . 49
1.5.3.2. La construcción de Caspar-Klung . . . 50

2. Metodología **53**

2.1. Paralelización de PyGBe 53
2.1.1. Procesos a paralelizar 54
2.1.1.1. Cálculo del RHS 55
2.1.1.2. Cálculo del P2P 55
2.1.1.3. Cálculo del M2P 56
2.1.1.4. Cálculo del energía de solvatación 57
2.1.1.5. Cálculo del energía de Coulomb 57
2.2. Parametrización y mallado de estructuras 57
2.2.1. Parametrización 57
2.2.2. Mallado 59
2.3. Gradientes de pH 60

3. Resultados y discusión **63**
3.1. Resultados de paralelización 63
3.2. Zika: Gradientes de salinidad y contribución de residuos 67
3.2.1. Motivación 67
3.2.2. Preparación de la estructura 68
3.2.3. Cápside expuesta a gradientes de salinidad 69
3.2.4. Contribución de aminoácidos en energía de solvatación 73
3.3. Triatoma: Energías libres en el desensamble 77
3.3.1. Motivación 77
3.3.2. Preparación de la estructura 77
3.3.3. Cálculo de energías libres para estructura sin desensamblar . . 79

4. Conclusiones **84**

Bibliografía **85**

Índice de Tablas

1.1. Ley de Amdahl para $f_{\text{par}} = 1$. 43
1.2. Ley de Amdahl para $f_{\text{par}} = 0{,}9$. 43
1.3. Ley de Amdahl para $f_{\text{par}} = 0{,}8$. 43

3.1. Resultados de tiempo y speedup para distintos números de threads en el cálculo del RHS para el caso de una lisosima. 64
3.2. Resultados de tiempo y speedup para distintos números de threads en el cálculo del P2P para el caso de una lisozima. 64
3.3. Resultados de tiempo y speedup para distintos números de threads en el cálculo del M2P para el caso de una lisozima. 65
3.4. Resultados de tiempo y speedup para distintos números de threads para la función `M2P_sort` aislada. 65
3.5. Resultados de tiempo y speedup para distintos números de threads en el cálculo de la energía de solvatación para el caso de una lisozima. . 65
3.6. Resultados de tiempo y speedup para distintos números de threads en el cálculo de la energía de Coulomb para el caso de una lisosima. . . . 67
3.7. Parámetros para cada una de las regiones para simulaciones de gradientes de salinidad. 71
3.8. Resultados de energía de solvatación para el ZIKV expuesto a diferentes gradientes de salinidad. Las referencias de los sub-índices son respecto a la figura 3.4. 72
3.9. Diferencias entre simulaciones para los casos con y sin capa de Stern. 75
3.10. Valores de ΔG_{carga} y $\Delta G_{\text{carga}}/\text{N}^\circ$ Átomos al apagar diferentes aminoácidos en la cápside del ZIKV. 76
3.11. Influencia del gradiente de pH en la energía electrostática libre, con pH neutro interiormente y variaciones en el exterior. 82
3.12. Influencia del gradiente de pH en la energía electrostática libre, con pH = 8,5 en la región exterior y con variación en la región interna. . . 82
3.13. Energía libre electrostática de la cápside en proceso de desensamblado al añadir 800 cargas negativas dentro de la cápside. 83
3.14. Energía libre electrostática de la cápside en proceso de desensamblado al añadir 1600 cargas negativas dentro de la cápside. 83

Índice de Figuras

1.1. Condición de salto para superficie cargada. Cooper, C. C. (2015). Jump condition for surface charge. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. 3
1.2. Definición de superficies. Cooper, C. C. (2015). Surface definition. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. . . 11
1.3. Representación de una proteína en un solvente. Cooper, C. C. (2015). Representation of a protein in a solvent. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. 12
1.4. Tratamiento de la integral singular. Cooper, C. C. (2015). Treatment of the singular integral. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. 19
1.5. Expansión de multipolos en Treecode. Cooper, C. C. (2015). Multipole expansion in the treecode. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. 25
1.6. Pasos del Treecode. Cooper, C. C. (2015). Steps of the treecode. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors. 27
1.7. Diagrama de bloques de un procesador basado en caché de doble núcleo genérico. Massachusetts Institute of Technology. (2008). Block diagram of a generic, cache-based dual core processor [Figura]. En Using OpenMP. 33
1.8. Computadores de memoria distribuida y compartida. Massachusetts Institute of Technology. (2008). Distributed- and shared-memory computers [Figura]. En Using OpenMP. 34
1.9. Modelo fork/join soportado por OpenMP. Massachusetts Institute of Technology. (2008). The fork-join programming model supported by OpenMP [Figura]. En Using OpenMP. 38
1.10. Esquema general de un aminoácido. 44
1.11. Representación genérica de los componentes principales de un virus. . 49
1.12. Estructura de Caspar-Klung para $T = 4$ 50

1.13. Virus caracterizados por diferentes número de triangulación. De izquierda a derecha: virus del mosaico del tabaco (PDB ID: 1A34, $T = 1$), virus del mosaico clorótico de la habichuela (PDB ID: 1IHM, $T = 3$) y omegatetravirus (PDB ID: 1ZA7, $T = 4$). Imágenes fueron obtenidas con el software VIPER en la página http://viperdb.scripps.edu/. . . . 52

2.1. Representación grafica del virus Triatoma. Image from the RCSB PDB (rcsb.org) of PDB ID 3nap Squires, G., Pous, J., Rozas-Dennis, G.S., Costabel, M.D., Agirre, J., Marti, G., Navaza, J., Guerin, D.M.A., Rey, F.A. The crystallographic structure of Triatoma Virus (TrV) highlights the Dicistroviridae and Picornaviridae family differences. To be published. (Squires et al., 2013) 58
2.2. Diagrama de flujo del proceso llevado a cabo desde los archivos obtenidos en el Protein Data Bank hasta que están listos para ser utilizados por PyGBe. 60
2.3. Representación de la cápside de un virus con extensiones en su estructura. La línea roja representa la circunferencia de radio promedio r_p y la línea azul representa la del radio promedio de la estructura modificada r_m. 61

3.1. Distribución de tiempo que toman los distintos procesos para diferentes números de threads involucrados en el cálculo. 66
3.2. Representación grafica del virus 5IRE. Image from the RCSB PDB (rcsb.org) of PDB ID 5IRE (Sirohi, D., Chen, Z., Sun, L., Klose, T., Pierson, T., Rossmann, M., Kuhn, R.) (2016) The cryo-EM structure of Zika Virus. (Sirohi et al., 2016) 67
3.3. Representación del virus Zika. Image from the RCSB PDB (rcsb.org) of PDB ID 5ire Sirohi, D., Chen, Z., Sun, L., Klose, T., Pierson, T. C., Rossmann, M. G., & Kuhn, R. J. (2016). The 3.8 resolution cryo-EM structure of Zika virus. Science, 352(6284), 467-470. 69
3.4. Esquema del ZIKV para modelamiento de diferencias de salinidad dentro y fuera de la macromolécula. 70
3.5. Relación entre la concentración de sal en el suero fisiológico y el recíproco de la longitud de Debye de la solución κ. 71
3.6. Energía de solvatación con variación de salinidad en la región externa y concentración $C_3 = 0{,}15[M]$ fija en la región interna. 73
3.7. Energía de solvatación con variación de salinidad en la región interna y concentración $C_0 = 0{,}15[M]$ fija en la región externa. 74
3.8. Esquema termodinámico de energías de solvatación, de Coulomb y de cargas. 75

3.9. Representación de la cápside en un modelo de solvente implícito con tres regiones. La figura 3.9a muestra la región dentro de la cápside (Ω_1), fuera de la cápside (Ω_2) y la interfaz donde están ubicados los átomos de la cápside (Ω_3). La figura 3.9b muestra una representación alternativa añadiendo una región extra con cargas puntuales para representar el cambio después de la deprotonación del material genético. American Chemical Society. (2019). Representation of the capsid in an implicit-solvent model which comprises three regions [Figura]. En Free Energies of the Disassembly of Viral Capsids from a Multiscale Molecular Simulation Approach. 79
3.10. Energía libre electrostática con variación de pH en la región externa y $pH_{int} = 7$ fijo en la región interna. Los valores numéricos se encuentran expresados en la tabla 3.11 80
3.11. Energía libre electrostática con variación de pH en la región interna y $pH_{int} = 8{,}5$ fijo en la región externa. Los valores numéricos se encuentran expresados en la tabla 3.12 81

1 Marco Teórico

1.1. Modelo de solvente implícito

Las biomoléculas de gran tamaño en las que estamos interesados se encuentran disueltas en medios acuosos, y es por eso que se hace necesario incluir estos medios en cualquier modelo que se utilice para simular el comportamiento de estas estructuras. En particular, el modelo de solvente implícito o *Implicit solvent model*, representa este medio acuoso o solvente como un medio dieléctrico continuo. A continuación haremos un repaso de la teoría que hay detrás de estos sistemas electrostáticos.

1.1.1. Electrostática en el vacío

Campo eléctrico y potencial electrostático

El campo eléctrico $\mathbf{E}$ es un campo vectorial dado por la fuerza eléctrica por unidad de carga en un punto dado. Si ponemos una carga de prueba q, suficientemente pequeña como para no perturbar el resto de campo, estaría siendo afectada por una fuerza $\mathbf{F} = q\mathbf{E}$. La ley de Gauss, una de las ecuaciones de Maxwell, relaciona el campo eléctrico con las distribución de carga ρ mediante la expresión

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \tag{1.1}$$

donde ϵ_0 es la permitividad en el vacío. Si presentamos la ley de Gauss en su forma integral, sobre un volumen Ω y aplicamos el teorema de la divergencia en el lado izquierdo de la igualdad, obtenemos

$$\oint_{\Gamma} \mathbf{E} \cdot \mathbf{n} d\Gamma = \int_{\Omega} \frac{\rho}{\epsilon_0} d\Omega \tag{1.2}$$

donde Γ es la superficie que encierra a Ω, y $\mathbf{n}$ es un vector unitario normal a la superficie Γ. El término en el lado izquierdo de la ecuación es conocido como el flujo eléctrico.

La electrostática no considera la contribución del campo magnético en la ecuación de Maxwell y por lo tanto, el campo eléctrico es irrotacional, es decir

$$\nabla \times \mathbf{E} = 0 \tag{1.3}$$

por lo tanto, existe un campo escalar ϕ tal que

$$-\nabla\phi = \mathbf{E} \tag{1.4}$$

Usando las ecuaciones (1.1) y (1.4), obtenemos la bien conocida ecuación de Poisson para electrostática

$$\nabla^2\phi = -\frac{\rho}{\epsilon_0} \tag{1.5}$$

y le damos a ϕ el nombre de potencial eléctrostático.

Si la distribución de carga consiste solo en una carga puntual, esto es

$$\rho = q\delta(\mathbf{r}') \tag{1.6}$$

entonces, la solución analítica de la ecuación (1.5) es

$$\phi(\mathbf{r}) = \frac{q}{4\pi\epsilon_0|\mathbf{r}-\mathbf{r}'|} \tag{1.7}$$

la cual puede ser generalizada para cualquier distribución como

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_\Omega \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\Omega' \tag{1.8}$$

Condiciones de salto del campo eléctrico y del potencial en superficies cargadas

En electrostática es común encontrar superficies cargadas. Consideremos el sistema de la figura 1.1, donde $\mathbf{E}_1$ y $\mathbf{E}_2$ son los campos eléctricos evaluados dentro y fuera de la superficie Γ que tiene una densidad $\sigma(\mathbf{r})$. Si usamos la ecuación de gauss (1.2) para evaluar $\mathbf{E}_1$ y $\mathbf{E}_2$, la única diferencia en la carga encerrada por esas superficies está dada por $\sigma(\mathbf{r})$, por lo tanto, restando estas dos evaluaciones obtenemos la condición de salto para la componente normal del campo eléctrico a través de Γ:

$$(\mathbf{E}_1 - \mathbf{E}_2)\cdot\mathbf{n} = \frac{\sigma}{\epsilon_0} \tag{1.9}$$

La ecuación (1.9) nos dice que la componente del campo eléctrico normal a una superficie cargada sufre una discontinuidad al atravesar dicha superficie, y que esa discontinuidad es igual a la densidad superficial de carga dividida por ϵ_0.

Dado que $\mathbf{E}$ es un campo irrotacional, la integral de línea sobre l depende solamente del potencial eléctrico en los puntos inicial y final

$$\int_A^B \mathbf{E}\cdot\mathbf{l}dl = -(\phi(\mathbf{r}_B) - \phi(\mathbf{r}_A)) \tag{1.10}$$

donde $\mathbf{l}$ es un vector unitario tangente a l. Entonces, si l es una curva cerrada, la integral en la ecuación (1.10) es 0. Consideremos la curva cerrada l que consta de cuatro partes: la primera es paralela a $\mathbf{n}$ y cruza Γ desde dentro hacia afuera, la segunda es paralela a la superficie de afuera, la tercera parte cruza la superficie de vuelta hacia adentro y la cuarta y última parte es paralela a la superficie y vuelve al origen. La ecuación (1.10) nos dice que la integral de línea del campo $\mathbf{E}$ a través de

toda l es nula, y sabemos que las integrales de las secciones que cruzan la superficie son iguales a la condición de salto de la ecuación (1.9) con signos opuestos. Luego, se cumple

$$\mathbf{E}_1 \cdot \mathbf{t} = \mathbf{E}_2 \cdot \mathbf{t} \tag{1.11}$$

donde $\mathbf{t}$ es algún vector paralelo a la superficie Γ.
La componente del campo eléctrico tangencial a una supeficie cargada es continua al atravesar dicha superficie y aunque la superficie cargada pueda generar una discontinuidad en el campo eléctrico el potencial electrostático es continuo.

Energía potencial electrostática

El trabajo hecho para mover una carga entre los puntos A y B está dado por

$$\begin{aligned} W = -\int_A^B \mathbf{F} \cdot \mathbf{l} dl &= -q \int_A^B \mathbf{E} \cdot \mathbf{l} dl \\ &= q \int_A^B \nabla\phi \cdot \mathbf{l} dl \\ &= q \int_A^B d\phi = q(\phi(\mathbf{r}_B) - \phi(\mathbf{r}_A)) \end{aligned} \tag{1.12}$$

donde $\mathbf{l}$ es un vector tangente a l. Entonces el trabajo de traer una carga i desde el infinito, donde el potencial eléctrico tiende a cero, hasta el punto $\mathbf{r}_i$ es

$$W_i = q_i \phi(\mathbf{r}_i) \tag{1.13}$$

Podemos generalizar la expresión (1.13) para obtener la energía potencial para cualquier distribución

$$W = \frac{1}{2} \int_\Omega \rho(\mathbf{r}) \phi(\mathbf{r}) d\Omega \tag{1.14}$$

El término $1/2$ en la ecuación (1.14) evita contar las interacciones dos veces. Por ejemplo, si la distribución de cargas fuese dos cargas puntuales, la energía potencial electrostática está dada por $q_1 q_2 / 4\pi\epsilon_0$ y no por $q_1 q_2 / 4\pi\epsilon_0 + q_1 q_2 / 4\pi\epsilon_0$.

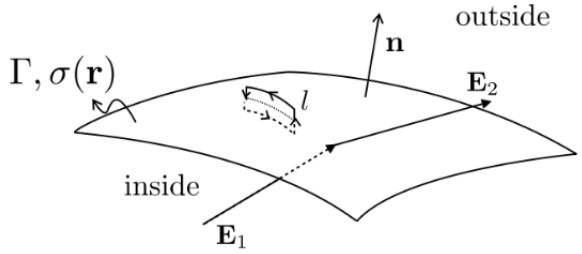


Figura 1.1: Condición de salto para superficie cargada. Cooper, C. C. (2015). Jump condition for surface charge. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

Expansión en multipolos

El potencial electrostático puede ser analizado usando expansiones de multipolo con armónicos esféricos

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_{l=0}^{\infty}\sum_{m=-l}^{l}\frac{4\pi}{2l+1}q_{lm}\frac{Y_{lm}(\theta,\varphi)}{|\mathbf{r}|^{l+1}} \tag{1.15}$$

donde

$$q_{lm} = \int_{\Omega} Y^*_{lm}(\theta',\varphi')|\mathbf{r}'|^l\rho(\mathbf{r}')d\Omega' \tag{1.16}$$

son los momentos de multipolo, Y_{lm} es la función armónica esférica de grado l y orden m, y Y^*_{lm} es el conjugado de Y_{lm}. Este análisis es útil para saber, por ejemplo, qué tanto se parece una distribución de carga a una carga puntual, dada la importancia del momento de monopolo (q_{00}). Otra distribución usada es la de dipolo, que aparece en la ecuación (1.15) para $l = 1$,

$$\mathbf{p} = \int_{\Omega}\mathbf{r}'\rho(\mathbf{r}')d\Omega' \tag{1.17}$$

El sistema más simple con un dipolo consiste en dos cargas puntuales de signo contrario, en este caso, el momento de dipolo es

$$\mathbf{p} = q\mathbf{d} \tag{1.18}$$

donde $\mathbf{d}$ es el vector posición entre las dos cargas. Si $|\mathbf{d}|$ es suficientemente pequeño, el momento de dipolo va a dominar la expansión armónica esférica, y se puede representar el sistema como un dipolo puntual. El potencial electrostático de dos cargas infinitesimalmente cercanas con signos opuestos, centrada en $\mathbf{r}'$ es

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\frac{\mathbf{p}\cdot(\mathbf{r}-\mathbf{r}'))}{|\mathbf{r}-\mathbf{r}'|^3} \tag{1.19}$$

en presencia de un campo externo, las fuerzas electrostáticas intentarán orientar el momento de dipolo para que esté paralelo al campo eléctrico.

1.1.2. Electrostática en un dieléctrico continuo

Polarización y desplazamiento eléctrico

La electrostática en el vacío es ideal para hacer simulaciones entre dos cargas que están realmente separadas por vacío, como en dinámica molecular (MD), donde cada molécula es modelada como cargas puntuales y dipolos en el vacío. Sin embargo, al querer utilizar esta teoría en sistemas que cuentan con muchas moléculas, hacer un modelamiento de esta forma se vuelve muy costoso en términos de computación. De esta forma, se hace útil aproximar estos sistemas mediante una teoría macroscópica de electrostática, donde el campo eléctrico actúa en un material dieléctrico, lo que aproxima los efectos microscópicos de las moléculas y los átomos del medio.
En un medio compuesto por muchos átomos y moléculas, si no se aplica ningún

campo eléctrico, todos los momentos de multipolo son nulos. Sin embargo, al aplicar un campo eléctrico, el momento molecular que predomina es el dipolo. Esto produce una polarización eléctrica $\mathbf{P}$ en el medio, que es la densidad de momento de dipolo por unidad de volumen. Este concepto generaliza la ecuación (1.19) en

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_\Omega \frac{\mathbf{P}(\mathbf{r})\cdot(\mathbf{r}-\mathbf{r}'))}{|\mathbf{r}-\mathbf{r}'|^3}d\Omega' \tag{1.20}$$

que puede ser escrita como

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_\Omega \mathbf{P}(\mathbf{r}')\cdot\nabla'\left[\frac{1}{|\mathbf{r}-\mathbf{r}'|}\right]d\Omega' \tag{1.21}$$

Podemos calcular el potencial electrostático desde un punto de vista macroscópico superponiendo el potencial debido a una distribución de carga (Ecuación (1.8)) y el potencial debido a la distribución de dipolos en el medio (Ecuación (1.21)), lo cuál nos da

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_\Omega \left(\frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + \mathbf{P}(\mathbf{r}')\cdot\nabla'\left[\frac{1}{|\mathbf{r}-\mathbf{r}'|}\right]\right)d\Omega' \tag{1.22}$$

Usando integración por partes para el segundo término del integrando, y asumiendo que la única condición de borde es que $\mathbf{P}$ está acotada cuando $\mathbf{r}' \to \infty$, podemos reescribir la ecuación (1.23) como

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_\Omega \left(\frac{1}{|\mathbf{r}-\mathbf{r}'|}\left[\mathbf{P}(\mathbf{r}') - \nabla'\cdot\mathbf{P}(\mathbf{r}')\right]\right)d\Omega' \tag{1.23}$$

Esta última ecuación es idéntica a la ecuación (1.8)) para una distribución de carga $\rho - \nabla\cdot\mathbf{P}$, donde el término $-\nabla\cdot\mathbf{P}$ alguna veces es llamado carga de polarización (ρ_b). En este contexto, la ley de Gauss quedaría

$$\nabla\cdot\mathbf{E} = \frac{1}{\epsilon_0}\left[\rho - \nabla\cdot\mathbf{P}\right] \tag{1.24}$$

definimos el desplazamiento eléctrico $\mathbf{D}$,

$$\mathbf{D} = \epsilon_0\mathbf{E} + \mathbf{P}. \tag{1.25}$$

Usando $\mathbf{D}$, podemos reescribir la ley de Gauss como

$$\nabla\cdot\mathbf{D} = \rho, \tag{1.26}$$

la cual es la versión macroscópica de la ley de Gauss. La ecuación (1.26) tiene la forma integral

$$\oint_\Gamma \mathbf{D}\cdot\mathbf{n}d\Gamma_= \oint_\Omega \rho d\Omega. \tag{1.27}$$

Dieléctrico lineal

Asumimos un dieléctrico isotrópico que reacciona linealmente a un campo eléctrico aplicado,

$$\mathbf{P} = \epsilon_0\chi_e\mathbf{E}, \tag{1.28}$$

donde χ_e es la susceptibilidad eléctrica del medio. Esta relación constitutiva es ampliamente usada, pero falla cuando el campo eléctrico se vuelve demasiado grande. Podemos reescribir la ecuación (1.25) como

$$\mathbf{D} = \epsilon \mathbf{E} \tag{1.29}$$

donde

$$\epsilon = \epsilon_0(1 + \chi_e), \tag{1.30}$$

es la constante dieléctrica del medio. Usando esta última ecuación y la ecuación (1.4) obtenemos la versión macroscópica de la ecuación de Poisson de elecrostática

$$\nabla^2 \phi - \nabla \cdot \mathbf{E} = -\frac{\rho}{\epsilon} \tag{1.31}$$

Discontinuidades superficiales en medio dieléctrico

En electrostática, muchas veces, el dominio contiene más de un medio dieléctrico. Podemos derivar las condiciones en la interfaz de dos materiales dieléctricos comenzando desde la ecuación 1.27, usando el mismo razonamiento que llevó a la ecuación 1.9 y la ecuación 1.11. Eso nos lleva a

$$(\mathbf{D}_2 - \mathbf{D}_1) \cdot \mathbf{n} = \sigma$$

$$\mathbf{E}_1 \cdot \mathbf{t} = \mathbf{E}_2 \cdot \mathbf{t} \tag{1.32}$$

En esta ecuación, la carga superficial σ no incluye la carga de polarización superficial.

Cálculos de energía en medio dieléctrico

El cálculo de energía en al ecuación 1.14 es válida en vacío, pero para otros dieléctricos, necesitamos tomar en cuenta la energía usada para polarizar el medio. El trabajo hecho por un cambio infinitesimal en la distribución de carga macroscópica, que no afecta el potencial electrostático es

$$\delta W = \int_\Omega \delta\rho(\mathbf{r})\phi(\mathbf{r}) d\Omega \tag{1.33}$$

Sabemos de la ecuación 1.27 que $\delta\rho = \nabla \cdot (\delta \mathbf{D})$, por lo tanto podemos reescribir como

$$\delta W = \int_\Omega \delta \mathbf{D} \cdot \mathbf{E} \Omega \tag{1.34}$$

después de integrar por partes y haciendo $\mathbf{E} = -\nabla\phi$. La energía total está dada por la integración de esta ecuación, como

$$W = \int \delta W = \int \int_\Omega \delta \mathbf{D} \cdot \mathbf{E} d\Omega \tag{1.35}$$

Hasta ahora, para llegar a esa última ecuación no hemos usado ninguna información relacionada con las propiedades del medio, sin embargo, para continuar debemos usar a ley constitutiva linear, que nos permite escribir la ecuación

$$\delta\mathbf{D}\cdot\mathbf{E} = \frac{1}{2}\delta(\mathbf{D}\cdot\mathbf{E}) \tag{1.36}$$

Usando esta relación en la ecuación 1.35 obtenemos la siguiente expresión para la energía

$$W = \frac{1}{2}\int_{\Omega}\mathbf{E}\cdot\mathbf{D}d\Omega \tag{1.37}$$

En la ecuación 1.37, podemos reemplazar $\mathbf{E} = -\nabla\phi$ y $nabla\cdot\mathbf{D} = \rho$, y luego integrar por partes y usar condición de desvanecimiento en los bordes para obtener

$$W = \frac{1}{2}\int_{\Omega}\phi(\mathbf{r})\rho(\mathbf{r})d\Omega \tag{1.38}$$

Esta expresión es válida solo para un dieléctrico lineal y es idéntica a la ecuación 1.14.

1.1.3. Efecto de una solución electrolítica

El fluido fisiológico consiste en agua con sal. Cuando está disuelta, la sal produce iones que contribuyen al campo eléctrico y se mueven libremente en el agua. Un campo aplicado afectará la distribución espacial de los iones de sal, haciendo que la distribución de densidad de carga de la ecuación 1.31 depende del potencial electrostático ($\rho = \rho(\phi)$). Hill esbozó el análisis matemático de este sistema (Mason, 1960), que lleva a la ecuación de Poisson-Boltzmann. Ahora revisaremos la derivación.

Función de distribución radial

Consideremos N partículas que se mueven libremente en un medio continuo con volumen V. Si situamos el origen en un partícula específica i, podemos escribir la densidad promediada en el tiempo o concentración ($c(\mathbf{r})$) como

$$c(\mathbf{r}) = \frac{N}{V}g(r) = c_0 g(r) \tag{1.39}$$

La función de distribución radial ($g(r)$) es de la desviación de la densidad numérica de la media (c_0), una distancia r de la partícula i. Esta es una forma conveniente de representar la densidad porque la distribución promediada en el tiempo tenderá a ser esféricamente simétrica respecto a la partícula de referencia i.
El número de partículas en un volumen dV_α ubicado en $\mathbf{r}_\alpha$ es

$$N_\alpha = c(\mathbf{r}_\alpha)dV_\alpha \tag{1.40}$$

por tanto, N_α/N representa la probabilidad de una partícula de estar en dV_α. Para encontrar la probabilidad de que cualquiera de las partículas esté en dV_α, necesitamos muestrear las N partículas, cada una con una probabilidad N_α/N, dejando

$$N\frac{N_\alpha}{N} = N\frac{c(\mathbf{r}_\alpha)dV_\alpha}{N} = c_0 g(|\mathbf{r}_\alpha|)dV_\alpha \tag{1.41}$$

Ahora consideremos que tenemos dos especies, α y β en un medio continuo. La probabilidad de encontrar una partícula α en dV_α es $c_\alpha dV_\alpha$. Por otro lado, la probabilidad de encontrar una partícula β en dV_β a una distancia $r_{\alpha\beta}$ de la partícula α, es $c_\beta g_{\alpha\beta}(r_{\alpha\beta})dV_\beta$. Luego, la probabilidad de encontrar α en dV_α y β en dV_β es la multiplicación de ambas probabilidades

$$c_\alpha c_\beta g_{\alpha\beta}(r_{\alpha\beta})dV_\alpha dV_\beta \tag{1.42}$$

El argumento que llevó a la ecuación 1.42 puede ser generalizado para cualquier número de especies como

$$c_\alpha c_\beta c_\gamma ... g_{\alpha\beta\gamma...}(r_{\alpha\beta}, r_{\alpha\gamma}, ...)dV_\alpha dV_\beta dV_\gamma ... \tag{1.43}$$

Si consideramos que $\alpha, \beta, \gamma, ...$, son partículas de las misma especie, la ecuación 1.43 se convierte en

$$c^n g_n(r_{12}, r_{13}, ..., r_{1n})dV_1 dV_2 ... dV_n \tag{1.44}$$

y corresponde a la probabilidad de encontrar la partícula 1 en dV_1, la partícula 2 en dV_2 (a una distancia r_{12} de la partícula 1), la partícula 3 en dV_3 (a una distancia r_{13} de la partícula 1), ... y la partícula n en dV_n (a una distancia r_{1n} de la partícula 1). La probabilidad de encontrar un sistema de N partículas con energía $E = p_m^2/2m + U$ está dada por la distribución de Maxwell-Boltzmann

$$P(E) = \frac{e^{-E/k_B T} dqdp}{\int e^{-E/k_B T} dqdp} \tag{1.45}$$

donde p_m es el momentum, q es la configuración especial, U la energía potencial, k_B la constante de Boltzmann y T la temperatura. Si integramos sobre el momentum y reemplazamos dq por la posición de cada partícula $dV_1 dV_2 dV_3 ... dV_N$, obtenemos

$$P(U) = \frac{e^{e^{-U/k_B T}} dV_1 dV_2 ... dV_N}{Z_N} \tag{1.46}$$

donde Z_N es la integral de configuración

$$Z_N = \int e^{e^{-U/k_B T}} dV_1 dV_2 ... dV_N \tag{1.47}$$

sobre todas las posibles configuraciones.
La probabilidad de encontrar partículas desde la 1 hasta la n en dV_1 a dV_n, independiente de la posición de las $N - n$ partículas restantes es

$$\frac{dV_1 dV_2 ... dV_n \int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N}{Z_N} \tag{1.48}$$

sin embrago, dado que las partículas son indistinguibles entre ellas, cualquier partícula puede ser 1, 2, ..., n, y debemos considerar todas las permutaciones

$$P(U) = \frac{N!}{(N-n)!} \frac{dV_1 dV_2 ... dV_n \int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N}{Z_N} \tag{1.49}$$

La ecuación 1.49 es la probabilidad de encontrar la partícula 1 en dV_1, la partícula 2 en dV_2, ... y la partícula n en dV_n. Esto equivale a la ecuación 1.44, e igualando las obtenemos

$$\rho^n g_n(r_{12}, r_{13}, ..., r_{1n}) = \frac{N!}{(N-n)!} \frac{\int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N}{Z_N} \tag{1.50}$$

Para n pequeño, podemos usar la aproximación

$$\frac{N!}{c^n (N-n)!} = V^n \left(1 + O\left(\frac{1}{N}\right)\right) \tag{1.51}$$

para obtener una expresión para la función de distribución radial

$$g_n(r_{12}, r_{13}, ..., r_{1n}) = \frac{V^n \int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N}{Z_N} \tag{1.52}$$

Es conveniente definir una función w_n tal que

$$g_n = e^{-w_n/k_B T} \tag{1.53}$$

que aplicada a la ecuación 1.52 da

$$e^{-w_n/k_B T} = \frac{V^n \int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N}{Z_N} \tag{1.54}$$

El significado físico de w_s se vuelve evidente cuando potamos el logaritmo de la ecuación **??** y derivamos respecto a la posición de la partícula i, donde $1 \leq i \leq n$, llevando a

$$-\nabla_i w_n = \frac{-\int e^{-U/k_B T} (\nabla_i U) dV_{n+1} dV_{n+2} ... dV_N}{\int e^{-U/k_B T} dV_{n+1} dV_{n+2} ... dV_N} \tag{1.55}$$

que es la fuerza promedio en la partícula i. La ecuación 1.55 revela que a función w_n es el potencial de la fuerza promedio en la partícula i.

Ecuación de Poisson-Boltzmann

Podemos usar una función de distribución radial para describir la distribución de iones de sal en un solvente, asumiendo que los iones son esferas duras y que el solvente no tiene comportamiento molecular: es continuo incluso a pequeñas distancias. Para obtener la ecuación de Poisson-Boltzmann necesitamos considerar que la fuerza de los iones es puramente de naturaleza electrostática, significando que la fuerza potencial es

$$w = \phi q \tag{1.56}$$

donde ϕ es el potencial electrostático y q es la carga del ión. Hay algunos supuestos hechos cuando se escribe la ecuación 1.56. En general, los iones no solo sienten fuerzas debido a la electrostática sino también debido a los efectos de solvatación de van de Waals (entropía); sin embargo, son interacciones de corto rango, y la ecuación 1.56 se vuelve exacta para sistemas altamente diluidos (donde los iones están altamente separados). En la ecuación 1.56 estamos además despreciando los coeficientes de

actividad, implicando una mezcla ideal de iones. En una mezcla ideal, la interacción entre cada par de especies químicas es la misma, haciendo la entalpía de mezcla nula. El coeficiente de actividad es una forma de medir qué tan alejada está una mezcla de la ideal.
Usando la función de distribución radial en la ecuación 1.31 para N_s especies de iones obtenemos

$$\nabla^2\phi = -\frac{\rho}{\epsilon} = -\frac{1}{\epsilon}\sum_i^{N_s} q_i c_i(\mathbf{r}) = -\frac{1}{\epsilon}\sum_i^{N_s} q_i c_0 g_i(r) = -\frac{1}{\epsilon}\sum_i^{N_s} q_i c_0 e^{-\phi q_i/k_B T} \tag{1.57}$$

donde qi y c_{0i} son la carga y la densidad promedio de iones de la especie i. En el caso de la sal, por ejemplo cloruro de sodio, hay dos tipos de iones: el ion de sodio con carga e^+ y el ion cloruro con carga e^-. Estos iones usualmente vienen de la misma fuente, así que es seguro asumir que igual densidad promedio c1,57 se convierte en

$$\begin{aligned}\nabla^2\phi = -\frac{n_0}{\epsilon}\left(e^+e^{-\phi e^+/k_B T} + e^-e^{-\phi e^-/k_B T}\right) &= -\frac{e^+ n_0}{\epsilon}\left(e^{-\phi e^+/k_B T} - e^{\phi e^+/k_B T}\right) \\ &= \frac{2n_0 e^+}{\epsilon}\sinh\left(\frac{\phi e^+}{k_B T}\right)\end{aligned} \tag{1.58}$$

en casos en que $\phi q << k_B T$, la aproximación de primer orden de la exponencial es

$$e^{-\phi q/k_B T} \approx 1 + \frac{\phi q}{k_B T} \tag{1.59}$$

y podemos escribir una forma linealizada de la ecuación 1.57

$$\nabla^2\phi = \frac{\phi}{\epsilon k T}\sum_i^{N_s} n_{0i} q_i = \kappa^2\phi \tag{1.60}$$

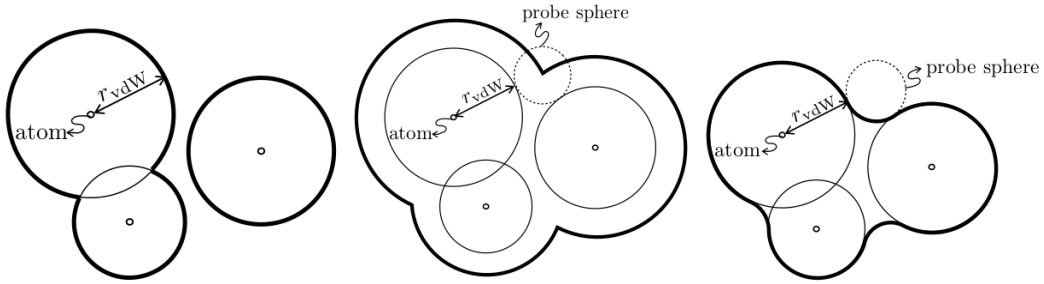
el parámetro de apantallamiento κ es el inverso del largo de Debye y para el caso de la sal que llevó a la ecuación 1.58, se convierte en

$$\kappa^2 = \frac{2n_0 e^+}{\epsilon k T} \tag{1.61}$$

Como ejemplo, la concentración normal de suero de sal es al rededor de 150mM, lo que da $\kappa = 0{,}125[\text{Å}^{-1}]$

Modelo de solvente implícito en sistemas moleculares

El modelo de solvente implícito usa la teoría de electrostática continua en medios dieléctricos para representar el solvente al rededor de una proteína. Con este enfoque no hay necesidad de contar explícitamente cada molécula de agua en el sistema y la polarización del solvente es considerada por la constante dieléctrica, disminuyendo el esfuerzo computacional. Este modelo ha sido usado para cálculos en una variedad de aplicaciones, como el calculo de la constante de disociación ácida, afinidad de uniones, energías de solvatación y catálisis entre otros. Existen varias revisiones exhaustivas



(a) van de Waals surface (b) Solvent-accessible surface (c) Solvent-excluded surface

Figura 1.2: Definición de superficies. Cooper, C. C. (2015). Surface definition. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

sobre este tema (Orozco y Luque, 2000) (Roux y Simonson, 1999)(Sharp y Honig, 1990) (Simonson, 2003) (Tomasi et al., 2005) (Lu et al., 2008) (Xu y Cai, 2011) (Bardhan, 2012) (Baker, 2004), mostrando la gran cantidad de trabajo que se ha hecho usando modelos de solvente implícito para sistemas moleculares.

Electrostática continua en sistemas biomoleculares

En una configuración biológica, las biomoléculas generalmente son disueltas en agua con sal. Una importante familia de biomoléculas son las proteínas, que son cadenas de aminoácidos que son usualmente halladas ensambladas de formas muy específicas. Dentro de la proteína hay una distribución de carga provocada por la carga dependiente del pH de los aminoácidos y las cargas parciales de los enlaces químicos. Afortunadamente, la información de la estructura cristalina de muchas proteínas de interés están disponibles libremente en la Protein Data Bank (PDB), y los modelos de campos de fuerza pueden calcular los radios de van de Waals y la carga en la posición de cada átomo.
Una forma de representar el sistema proteína-solvente es considerando el solvente como un dieléctrico continuo. EN el contexto de la electrostática, podemos identificar dos regiones eléctricas en una proteína disuelta: una región de solvente y una región de proteína. La región de solvente consiste en agua y sal, tiene una constante dieléctrica típica del agua ($\epsilon \approx 80$), y los iones de sal son considerados por la ecuación de Poisson-Boltzmann, en nuestro caso, en la forma linealizada de la ecuación 1.60. La región de proteína tiene una constante dieléctrica mucho más baja ($\epsilon \approx 2-4$) porque la reorientación de sus grupos dipolares está mucho más restringida (Gilson y Honig, 1986) (Gilson y Honig, 1987) y consiente una distribución de carga, usualmente modelada como cargas puntuales en las ubicaciones de los átomos.

La figura 1.2 muestra tres definiciones de interfaz comunes, todas ellas dependen de el radio de van de Waals del átomo, calculado por un campo de fuerza. La superficie de van der Waals de las figura 1.2a considera como región de proteína todo lo que está dentro del radio atómico de cada átomo. Esta definición es usada en métodos generalizados de Born, donde la proteína es aproximada como un set de esferas. La figura 1.2 muestra la superficie accesible por el solvente o *solvent-accesible surface* (SAS), la cual se genera trazando el centro de una probeta esférica del tamaño de una

molécula de agua (1,4Å) al rodar en torno a la proteína. Finalmente, la superficie de exclusión de solvente *solvent-excluded surface* (SES) mostrada en la figura 1.2c define los más cercano que la molécula de agua puede estar de la proteína. En este trabajo, usamos la SES, calculada usando el software gratuito nanoshaper. (Decherchi y Rocchia, 2013)
Hay otra superficie molecular bien conocida no mostrada en la figura 1.2 llamada superficie Gaussiana. Esta superficie está definida implícitamente por

$$F(\mathbf{r}) = \sum_{i=1}^{N_q} e^{-|\mathbf{r}-\mathbf{r}_i|/r_W r_{vdW,i}^2} - e^{-1/r_W} \tag{1.62}$$

donde $\mathbf{r}$ es la superficie, $\mathbf{r}_i$ es la posición del átomo i, $r_{vdW,i}$ el radio de van de Waals del átomo i y r_W el tamaño de la probeta esférica o molécula de agua.

La figura 1.3 muestra una proteína disuelta. La región de proteína (Ω_1) tiene una permitividad ϵ_1, cargas puntuales en la ubicación de los átomos y el potencial electrostático representado por ϕ_1. Por otro lado, en la región de solvente (Ω_2) el potencial electrostático es ϕ_2, permitividad ϵ_2 y tiene iones de sal que producen un largo de Debye κ^{-1}. En la interfaz Γ, en este caso la SES, se encuentran dos condiciones. La primera es que el potencial electrostático debe ser continuo. La segunda es que dado que no hay cargas explícitas en la SES, la ecuación 1.32 indica que el desplazamiento eléctrico debe ser continuo. Matemáticamente, puede ser escrito como

$$\begin{aligned} \nabla^2\phi_1(\mathbf{r}) &= -\sum_k \frac{q_k}{\epsilon_1}\delta(\mathbf{r},\mathbf{r}_k) \qquad \text{en el soluto } (\Omega_2) \\ \nabla^2\phi_2(\mathbf{r}) &= \kappa^2\phi_2(\mathbf{r}) \qquad \text{en el solvente } (\Omega_1) \\ \phi_1 &= \phi_2 \qquad \text{en la interfaz } \Gamma \\ \epsilon_1\frac{\partial\phi_1}{\partial\mathbf{n}} &= \epsilon_2\frac{\partial\phi_2}{\partial\mathbf{n}} \end{aligned} \tag{1.63}$$

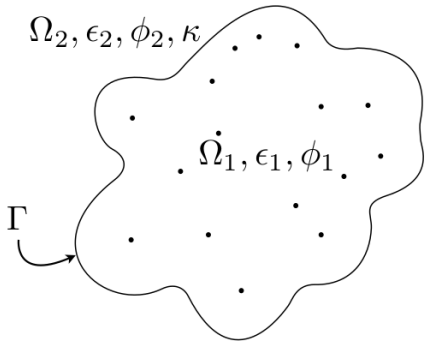


Figura 1.3: Representación de una proteína en un solvente. Cooper, C. C. (2015). Representation of a protein in a solvent. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

Cavidades llenas de solvente y capas de Stern

Dos cosas importantes que el modelo de solvente implícito es capaz de resolver son las cavidades llenas de solvente y capas de Stern. Incluso aunque las proteínas estén plegadas muy fuertemente, el solvente puede quedar atrapado dentro en bolsillos o cavidades. En el contexto del modelo de solvente implícito, las cavidades son regiones con una constante dieléctrica alta dentro de la proteína. Un tratamiento continuo no es ideal cuando la cavidad es pequeñas y solo se pueden acomodar en ella un par de moléculas de agua, pero e este trabajo no consideramos tratamientos especiales para estos casos.
El tratamiento continuo de iones en teoría de Poisson-Boltzmann no permite considerar efectos estéricos, lo que lleva a sobrestimar los iones cerca de las superficies cargadas. La capa de Stern (Stern, 1924) o capa de exclusión de iones es una región delgada (~ 2Å) libre de iones alrededor de la SES que busca modelar este efecto. La práctica común es usar la constante dieléctrica del solvente en la capa de Stern, sin embargo, trabajo reciente extiende esta idea a variar la permitividad e incluso la viscosidad (Bonthuis y Netz, 2013). El modelo de dos regiones de la ecuación 1.63 es incapaz de capturar los detalles como cavidades internas o capas de Stern y más tarde discutiremos cómo incluirlas en el modelo.

1.1.4. Cálculo de energías con el modelo de solvente implícito

Para un sistema con proteínas disueltas, es útil separar la energía libre entre componentes polares y no polares. La energía electrostática es la contribución polar a las energía libre, mientras que la componente no polar considera los efectos de van de Waals y entrópicos.
Nosotros asumiremos que todas las energías libres son de naturaleza electrostática, despreciando la componente no polar. Es una buena aproximación en varios casos especialmente cuando las proteínas tienen regiones cargadas. Por simplicidad, usaremos el término energía para referirnos a la componente polar de la energía libre, a menos que se diga lo contrario.

Contribución polar a la energía

Tenemos sistemas con tres posibles fuentes de energía electrostática: Energía de Coulomb de las cargas puntuales, energía de solvatación causada por la influencia del solvente fuera de la región de proteína y las energía superficial de una superficie con carga o potencial prescrito.

Energía de Coulomb

El modelo de solvente implícito considera la distribución de cargas de la proteína usando las cargas puntuales en las ubicaciones de los átomos. Usando funciones delta

como distribución de carga, la ecuación 1.14 es

$$F_{\text{Coulomb}} = \frac{1}{2}\sum_i \sum_j \frac{q_i q_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{1.64}$$

la cual es la bien conocida energía de Coulomb.

Energía de solvatación

Cuando una proteína es disuelta, el solvente reacciona polarizándose, llevando a o que comúnmente conocemos como energía de solvatación. Si el sistema solo contiene una proteína, la energía de solvatación está dada por la diferencia energética entre la proteína en el vacío, donde solo está el efecto coulómbico de las cargas puntuales, y la proteína disuelta. Si hay otras entidades en el solvente, como otras proteínas y superficies cargadas, también serán afectadas por la proteína disuelta y contribuirán a la energía de solvatación. Esta es la razón por la que en este caso queremos generalizar el concepto de energía de solvatación a la energía generada por la reacción en todos los alrededores de a proteína: solvente y otras entidades. En una configuración unimolecular, esta definición de energía de solvatación coincide con la energía requerida para disolver una molécula. Habiendo dicho esto, la energía de solvatación generada por el potencial de reacción (ϕ_{reac}) dentro de la proteína, esto es

$$F_{\mathbf{solv}} = \frac{1}{2}\int_\Omega \rho\phi_{\text{reac}} = \sum_{k=0}^{N_q} q_k \phi_{\text{reac}}(\mathbf{r}_k) \tag{1.65}$$

donde ρ es la distribución de carga. En este caso, la distribución de carga consiste en cargas puntuales que transforman la integral en una suma. Por otro lado, el potencial de reacción es

$$\phi_{\text{reac}} = \phi_{\text{total}} - \phi_{\text{Coulomb}} \tag{1.66}$$

y podemos interpretarlo como el potencial generado por la frontera de la región molecular (Γ y Ω_1 en al figura 1.3)

Contribución no polar a la energía

La contribución no polar a la energía ($F_{nopolar}$) considera las interacciones de dispersión de corto rango entre moléculas , dadas por las interacciones de va de Waals, y efectos conformacionales representados por la entropía.
Una proteína disuelta ocupa espacio en el solvente y para acomodarlo, las moléculas de agua necesitan ser empujadas fuera de la región de proteína. Además, las moléculas de agua tendrán una estructura bien organizada en las primeras pocas capas de solvatación, cerca de la SES. Estos dos efectos trabajan en contra de la entropía, y necesitan ser considerados para tener una representación completa de a energía libre. Existen modelos bien establecidos para la contribución no polar de la energía libre ((Roux y Simonson, 1999) (Wagoner y Baker, 2006)). La mayoría de estos modelos asumen una relación linear de $F_{nopolar}$ con la superficie molecular, dada la SES:

$$F_{nopolar} = \gamma A \tag{1.67}$$

donde A es la superficie molecular y γ es el coeficiente energético o tensión superficial.

Energía libre de interacción

Cuando existen dos o más cuerpos en el solvente, estos interactúan. Para calcular la energía de interacción, necesitamos tomar la diferencia entre la energía total del sistema en el que están interactuando y la energía total de cada uno de los componentes aislados. Despreciando la contribución no polar, la energía libre total está dada por

$$F_{\text{total}} = F_{\text{Coulomb}} + F_{\text{solvatación}} \tag{1.68}$$

Luego, la energía libre de interacción es

$$F_{\text{interacción}} = F_{\text{total}}^{\text{ensamble}} - \sum_{i=1}^{N_c} F_{\text{total}}^{\text{componentes}} \tag{1.69}$$

donde N_c es el número de componentes en el sistema y $F_{\text{total}}^{\text{componentes}}$ es calculado sobre la componente aislada i.
Cuando el sistema interactuando es un ensamble de dos moléculas unidas, la energía de interacción corresponde a la energía de enlace.

1.1.5. Cálculos usando el modelo de solvente implícito

La solución del acoplamiento entre las ecuaciones de Poisson y la de Poisson-Boltzmann (ecuación 1.63) en configuraciones moleculares ha sido de interés en la comunidad de biofísicos por un largo tiempo. En 1934, Kirkwood llegó a una expresión cerrada para la energía de solvatación de una molécula esférica, basada en armónicos esféricos (Kirkwood, 1934). Esta expresión puede manejar una distribución arbitraria de cargas puntuales dentro de la molécula y una capa de Stern. Desafortunadamente, las formas analíticas están disponibles solo para esferas y geometrías más complicadas requieren aproximaciones numéricas.

Solución numérica de las ecuación de Poisson-Boltzmann

Ha habido un trabajo substancial en intentar resolver el sistema acoplado de las ecuaciones de Poisson y Poisson-Boltzmann para geometrías realistas. EL primer cálculo numérico en una configuración biomolecular fue hecha en 1982 por Warwicker y Watson (Warwicker y Watson, 1982) usando el método de diferencias finitas, sin embargo, consideraron un solvente libre de sal. También usando diferencias finitas, el grupo de Honing comenzó a desarrollar Delphi en 1985 (Gilson et al., 1985), el cual se convirtió en un estándar en la comunidad y es aún tiene soporte activo, y más tarde Bashford escribió el código de diferencias finitas `MEAD` para hacer cálculos de pK_a (Bashford y Gerwert, 1992) (Bashford, 1997). Otro software numérico que está en activo desarrollo en `APBS` (Baker et al., 2001), que hace sus cálculos usando tanto diferencias finitas como elementos finitos. La ventaja principal de usar un método de elementos finitos sobre uno de diferencias finitas es que la malla se puede amoldar a

las superficie molecular, dejando de lado la forma escalonada de la SES inherente al método de diferencias finitas. Siguiendo esta línea, el grupo liderado por Wei introdujo el paquete `MIBPB` (Geng et al., 2007), un solver de diferencias finitas con un tratamiento especial de la interfaz llamada *the matched interface and boundary*, donde son capaces de forzar explícitamente las condiciones de interfaz, logrando una precisión de segundo orden.
Por otro lado, la naturaleza elíptica de la ecuación 1.63 permite su formulación en términos de una integral a lo lago de la interfaz, lo que motivó varios desarrollos usando el método de elementos de borde. Esta formulación fue hecha por primera vez por Shaw (Shaw, 1985), expresando la integral en término de la carga de superficie de polarización, las cual es válida para un solvente sin sal. Más tarde, Yoon y Lenhoff (Yoon y Lenhoff, 1990) consideraron la forma linealizada de la ecuación de Poisson-Boltzmann en el solvente, y Juffer y sus colegas (Juffer et al., 1991) encontraron una formulación con operadores hipersingulares. Incluso aunque los operadores hipersingulares pueden ser incómodos y la ecuación presentada por Juffer contiene más términs, la matriz final tiene un numero de condición que no depende del número de elementos de discretización, mientras que en la formulación de Yoon y Lenhoff el condicionamiento depende del tamaño (Liang y Subramaniam, 1997). Nuestro trabajo está basado en el acercamiento de Yoon y Lenhoff porque usamos precondicionadores y tamaños en los que es una alternativa competitiva.
Un ejemplo de una implementación del método de elementos de borde para la ecuación 1.63 es el trabajo del grupo liderado por White y Tidor, donde usaron `FFTSVD`, un código basado en elementos de borde acelerado por `FFT` para diseño de fármacos (Altman et al., 2006). Además, Lu y sus colegas desarrollaron el software `AFMPB` en 2006 (Lu et al., 2006), y más recientemente Krasny y Genge desarrollaron un código de método de elementos de borde con aceleración por treecode llamado `TABI` (Geng y Krasny, 2013). En 2013, Cooper y su equipo desarrollaron `PyGBe` (Cooper y Barba, 2013), un código que utiliza el método de elementos de borde en conjunto acelerado por treecode y capaz de correr en GPU.
Las ventajas principales de utilizar métodos integrales de borde sobre diferencias finitas o elementos finitos es que la representación de la superficie molecular es más fiel a la realidad, las condiciones de borde en el infinito coinciden perfectamente y menos grados de libertad. Sin embargo, las técnicas de elementos de borde entregan matrices densas, requiriendo técnicas de aceleración especiales para obtener tiempos de cómputo razonables. Algunas ventajas de los métodos basados en volúmenes, como elementos finitos y diferencias finitas, son que pueden resolver la ecuación de Poisson Boltzmann no linealizada y fácilmente lidiar con cavidades y capas de Stern.

Modelo generalizado de Born

El modelo generalizado de Born considera el soluto como un set de esferas que interactúan con cargas centrada y usan resultados analítcos de Kirkwood (Kirkwood, 1934) para encontrar la energía de solvatación. La forma mejor conocida de este modelo fue presentada por Still y sus equipo (Still et al., 1990) que es

$$F_{GB} = -166\left(1 - \frac{1}{\epsilon_{\text{solvatación}}}\right)\sum_i \sum_j \frac{q_i q_j}{f_{GB}} \tag{1.70}$$

con

$$f_{GB} = (|\mathbf{r}_i - \mathbf{r}_j|^2 + \alpha_{ij}^2 e^D)$$
$$\alpha_{ij} = (a\alpha_i\alpha_j)^{0,5}, \qquad \text{y}$$

$$D = |\mathbf{r}_i - \mathbf{r}_j|/(2\alpha_{ij})^2 \tag{1.71}$$

donde α_i es un parámetro llamado radio de Born. Este parámetro depende del caso y usualmente se consigue experimentalmente.

1.2. Método de Elementos de Borde (BEM)

El método de elementos de borde (BEM) resuelve ecuaciones integrales de borde numéricamente (BIE). Esto es útil en el contexto de PDEs cuando el operador diferencial tiene una solución fundamental y pueden ser escritos como un (BIE). La idea de representar PDEs como BIEs viene de los tempranos 1900s, con el trabajo de Betti, Somigliana y Fredholm ((Betti, 1872) (Somigliana, 1885) (Fredholm, 1903)) que se basó en el trabajo previo de Green (Green, 1828). De hecho, el trabajo de Fredholm es considerado el punto de partida de BEM, al ser él la primera persona en aplicar datos de borde en una potencial aplicación teórica. El trabajó usando técnicas analíticas que fue validado para geometrías simples. Las primeras soluciones numéricas a la ecuación de Fredholm aparecieron en los 60 (Jaswon, 1977) y desde entonces, BEM ha evolucionado a ser una técnica ampliamente usada.
Muchas referencias presentan una buena introducción a BEM. Estas pueden ser encontradas en monografías de Atkinson, McLean y Steinbach, ((Atkinson, 1997) (McLean y McLean, 2000) (Steinbach, 2007)) que se involucran en detalle con la matemática detrás del método. Revisiones no tan exhaustivas han sido realizadas por Brebbia (Brebbia y Dominguez, 1994) y Katsikadelis (Katsikadelis, 2002).

1.2.1. Derivación del problema de Laplace

Una PDE simple que puede ser resuelta con el método BEM es la ecuación de Laplace. Para una cantidad escalar ϕ en un dominio Ω con bordes Γ y condiciones de borde conocidas ϕ_0 o $\frac{\partial\phi_0}{\partial n}$, la ecuación de Laplace se puede escribir

$$\nabla^2\phi = 0 \tag{1.72}$$
$$\phi = \phi_0 \text{ o } \frac{\partial\phi}{n} = \frac{\partial\phi_0}{\mathbf{n}} \tag{1.73}$$

La formulación débil de la ecuación (1.73) con la función de prueba w es

$$\int_\Omega \nabla^2\phi(\mathbf{r'}_\Omega)w(\mathbf{r'}_\Omega)d\Omega' = 0 \tag{1.74}$$

donde el punto de evaluación $\mathbf{r}_\Omega$ está ubicado en el dominio Ω, y la integral suma sobre las cantidades con el signo prima.

El resultado de aplicar un operador diferencial linear en su función de Green en el espacio libre, es menos la función delta de Dirac ($-\delta(\mathbf{r})$). Por ejemplo,

$$\nabla^2 G_L(\mathbf{r}, \mathbf{r'}) = -\delta(\mathbf{r}) \tag{1.75}$$

define la función de Green en el espacio libre para el operador de Laplace, que es

$$G_L(\mathbf{r}, \mathbf{r'}) = \frac{1}{4\pi|\mathbf{r} - \mathbf{r'}|}. \tag{1.76}$$

Usando la función de Green de espacio libre como función de prueba en la ecuación (1.74), obtenemos

$$\int_\Omega \nabla^2 \phi(\mathbf{r'}) G_L(\mathbf{r}, \mathbf{r'}) d\Omega' = 0 \tag{1.77}$$

Podemos manipular la integral en la ecuación (1.77) para obtener

$$\int_\Omega [\nabla \cdot (\nabla \phi(\mathbf{r'}_\Omega) G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Omega)) - \nabla \phi(\mathbf{r'}_\Omega) \nabla G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Omega)] d\Omega' = 0 \tag{1.78}$$

y manipulando el segundo término de la ecuación (1.78) obtenemos

$$\int_\Omega [\nabla \cdot (\nabla \phi(\mathbf{r'}_\Omega) G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Omega)) - \nabla \cdot (\phi(\mathbf{r'}_\Omega) \nabla G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Omega)) + \phi(\mathbf{r'}_\Omega) \nabla^2 G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Omega)] d\Omega' = 0 \tag{1.79}$$

El operador divergencia afecta a los primeros dos términos de la ecuación (1.79), por lo tanto podemos aplicar el teorema de la divergencia en ellos. Además, usando la ecuación (1.75), obtenemos

$$\int_\Gamma \nabla \phi(\mathbf{r'}_\Gamma) \cdot \mathbf{n} G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Gamma) d\Gamma' - \int_\Gamma \phi(\mathbf{r'}_\Gamma) \nabla G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Gamma) \cdot \mathbf{n} d\Gamma' - \int_\Omega \phi(\mathbf{r'}_\Omega) \delta(\mathbf{r}_\Omega) d\Omega' = 0 \tag{1.80}$$

donde $\mathbf{n}$ es el vector normal a Γ apuntando hacia afuera del dominio Γ, y $\mathbf{r}_\Gamma$ está localizado en la frontera Γ. Podemos reescribir la ecuación (1.80) como

$$\phi(\mathbf{r}_\Omega) = \int_\Gamma G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Gamma) \frac{\partial}{\partial \mathbf{n}} \phi(\mathbf{r'}_\Gamma) d\Gamma' - \int_\Gamma \phi(\mathbf{r'}_\Gamma) \frac{\partial}{\partial \mathbf{n}} G_L(\mathbf{r}_\Omega, \mathbf{r'}_\Gamma) d\Gamma' \tag{1.81}$$

La derivación de la ecuación (1.77) a la ecuación (1.80) también sirve para demostrar la segunda identidad de Green, que para $\phi(\mathbf{r'})$ y $G_L(\mathbf{r}, \mathbf{r'})$ es

$$\int_\Omega (\nabla^2 \phi G_L - \phi \nabla^2 G_L) d\Omega = \int_\Gamma G_L \frac{\partial}{\partial \mathbf{n}} \phi - \phi \frac{\partial}{\partial \mathbf{n}} G_L d\Gamma. \tag{1.82}$$

En la ecuación (1.81), $\mathbf{r}$ puede estar en cualquier parte del dominio Ω, y $\mathbf{r'}$ está solo en la superficie de frontera Γ. Si tomamos el límite cuando $\mathbf{r} \to \Gamma$, G_L tiene una singularidad cuando $\mathbf{r} = \mathbf{r'}$. Para solucionar este problema, tomamos la integral en la superficie Γ' que es exactamente igual a Γ en todas partes, pero contiene un hemisferio de radio ϵ centrado en $\mathbf{r}$, y dividimos las integrales sobre Γ' en la integral en que Γ y Γ' coinciden, y la integral sobre el hemisferio (Γ'_{hem}). La superficie Γ' evita la singularidad, y tomando $\epsilon \to 0$ recuperamos la superficie original Γ. Esto se puede visualizar en la figura 1.4.

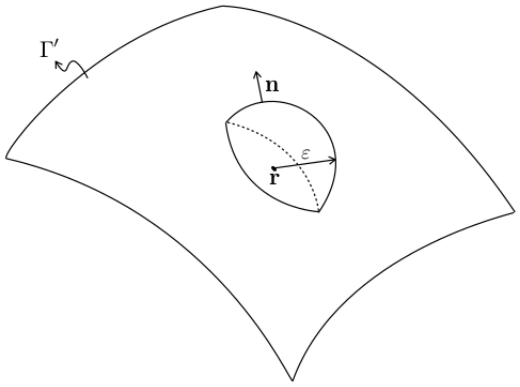


Figura 1.4: Tratamiento de la integral singular. Cooper, C. C. (2015). Treatment of the singular integral. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

La integral sobre el hemisferio Γ'_{hem} es

$$\begin{aligned}\int_{\Gamma'_{hem}} \frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r'})G_L(\mathbf{r},\mathbf{r'})d\Gamma' &= \frac{1}{4\pi}\int_{\Gamma'_{hem}} \frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r'})\frac{1}{|\mathbf{r}-\mathbf{r'}|}d\Gamma' \\ &= \frac{1}{4\pi\epsilon}\int_{\Gamma'_{hem}} \frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r'})d\Gamma',\end{aligned} \tag{1.83}$$

porque sabemos que en el hemisferio Γ'_{hem}, las distancia $|\mathbf{r}-\mathrm{r'}| = \epsilon$ es constante. Tomando el límite cuando $\epsilon \to 0$ para recuperar la superficie Γ original, obtenemos

$$\begin{aligned}\lim_{\epsilon\to 0}\frac{1}{4\pi\epsilon}\int_{\Gamma'_{hem}} \frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r'})d\Gamma' &= \lim_{\epsilon\to 0}\frac{1}{4\pi\epsilon}\frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r})\int_{\Gamma'_{hem}} d\Gamma' \\ &= \lim_{\epsilon\to 0}\frac{1}{4\pi\epsilon}\frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r})2\pi\epsilon^2 = 0,\end{aligned} \tag{1.84}$$

considerando que $\mathbf{r'} \to r$ cuando $\epsilon \to 0$. La aproximación de que el área al rededor de la singularidad es un hemisferio es válida porque para pequeños valores de ϵ y para una superficie suave Γ, la superficie en la vecindad de Γ'_{hem} es aproximadamente plana.
Podemos hacer cálculos similares para las integrales que involucran $\partial G_L/\partial \mathbf{n}$, que en el hemisferio Γ'_{hem} es

$$\begin{aligned}\int_{\Gamma'_{hem}} \phi(\mathbf{r'})\frac{\partial}{\partial \mathbf{n}}G_L(\mathbf{r},\mathbf{r'})d\Gamma' &= -\frac{1}{4}\int_{\Gamma'_{hem}} \phi(\mathbf{r'})\frac{(\mathbf{r}-\mathbf{r'})\cdot\mathbf{n}}{|\mathbf{r}-\mathbf{r'}|^3}d\Gamma' \\ &= -\frac{1}{4}\int_{\Gamma'_{hem}} \phi(\mathbf{r'})\frac{\epsilon}{\epsilon^3}d\Gamma',\end{aligned} \tag{1.85}$$

porque $(\mathbf{r}-\mathbf{r'})$ está siempre alineado con $\mathbf{n}$ y tiene magnitud ϵ. Luego, el límite

cuando $\epsilon \to 0$ es

$$\begin{aligned}\lim_{\epsilon\to 0} -\frac{1}{4}\int_{\Gamma'_{hem}} \phi(\mathbf{r'})\frac{\epsilon}{\epsilon^3}d\Gamma' &= -\frac{\phi(\mathbf{r})}{4\pi}\lim_{\epsilon\to 0}\frac{1}{\epsilon^2}\int_{\Gamma'_{hem}} d\Gamma' \\ &= -\frac{\phi(\mathbf{r})}{4\pi}\lim_{\epsilon\to 0}\frac{1}{\epsilon^2}2\pi\epsilon^2 = -\frac{\phi(\mathbf{r})}{2}.\end{aligned} \tag{1.86}$$

Usando los resultados de la ecuación (1.84) y la ecuación (1.86), podemos escribir el límite de la ecuación (1.81) cuando $\mathbf{r} \to \Gamma$ como

$$\frac{\phi(\mathbf{r}_\Gamma)}{2} + \int_\Gamma \phi(\mathbf{r'}_\Gamma)\frac{\partial}{\partial \mathbf{n}}G_L(\mathbf{r}_\Gamma, \mathbf{r'}_\Gamma)d\Gamma' = \int_\Gamma G_L(\mathbf{r}_\Gamma, \mathbf{r'}_\Gamma)\frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{r'}_\Gamma)d\Gamma', \tag{1.87}$$

donde estas son las integrales de valor principal de Cauchy.
La misma derivación que llevó desde la ecuación (1.74) a la ecuación (1.87) es válida para la ecuación linealizada de Poisson-Boltzmann descrita en la ecuación (1.60), donde el operado diferencial es $(\nabla^2 - \kappa^2)$. La función de Green de espacio libre de la ecuación linealizada de Poisson-Boltzmann es también conocida como el potencial de Yukawa y es

$$G_Y(\mathbf{r}, \mathbf{r'}) = \frac{\exp\left(-\kappa|\mathbf{r} - \mathbf{r'}|\right)}{4\pi|\mathbf{r} - \mathbf{r'}|}. \tag{1.88}$$

1.2.2. Operadores de capa

Los operadores integrales en las ecuaciones (1.87) y (1.81) son recurrentes en técnicas de integrales de frontera. Es por ello que han sido clasificados como operadores de potencial de una capa singular y de potencial de capa doble, y se han estudiado a fondo para asegurar su solubilidad (Steinbach, 2007).

Espacios de Sobolev

El espacio de Sobolev de orden entero r basado en $L_p(\Omega)$ se define como

$$W_p^r(\Omega) = \{\psi \in L_p(\Omega) : \partial^\alpha \psi \in L_p(\Omega), |\alpha| \leq r\} \tag{1.89}$$

donde $L_P(\Omega)$ es el espacio de funciones ψ para las cuales la p-ésima potencia del valor absoluto es integrable

$$||\psi||_p = \left(\int_\Omega |\psi|^p d\Omega\right)^{\frac{1}{p}} < \infty, \tag{1.90}$$

y $W_p^r(\Omega)$ está equipada con la norma

$$||\psi||W_p^r(\Omega) = \left(\sum_{|\alpha|\leq r}\int_\Omega |\partial^\alpha \psi(\mathbf{r})|^p d\Omega\right)^{\frac{1}{p}}. \tag{1.91}$$

El espacio de Sobolev de orden real $s = r + \mu$, donde r es un entero y $0 < \mu < 1$, basado en $L_p(\Omega)$ corresponde a

$$W_p^s(\Omega) = \{\psi \in W_p^r(\Omega) : |\partial^\alpha \psi|_{\mu,p,\Omega} < \infty, |\alpha| = r\}, \tag{1.92}$$

con la norma

$$||\psi||W_p^s(\Omega) = \left(||\psi||^p_{W_p^r(\Omega)} + \sum_{|\alpha|=r} |\partial^\alpha \psi|^p_{\mu,p,\Omega} \right)^{\frac{1}{p}} \tag{1.93}$$

donde

$$|\psi|_{\mu,p,\Omega} = \left(\int_\Omega \int_\Omega \frac{|\psi(\mathbf{r}) - \psi(\mathbf{r'})|}{|\mathbf{r} - \mathbf{r'}|^{n+p\mu}} d\Omega d\Omega \right)^{\frac{1}{p}} \tag{1.94}$$

es la semi norma de Slobodeckij.
El orden de un espacio de Sobolev puede también ser negativo. $W_p^{-r}(\Omega)$ admite la siguiente representación

$$\psi = \sum_{|\alpha|\leq r} \partial^\alpha f_\alpha, f_\alpha \in L_p(\Omega) 2{,}94) \tag{1.95}$$

con la norma

$$||\psi||_{W_p^{-r}(\Omega)} = \text{ínf} \left(\sum_{|\alpha\leq r|} ||f_\alpha||^p_{L_p(\Omega)} \right). \tag{1.96}$$

Dado que nosotros usamos la formulación débil de la PDE, mostrada en la ecuación (1.74), se requiere ϕ tal que $\phi \in L_2$, y para $p = 2$ llamaremos al espacio de Sobolev $W_2^s = H^s$
Para un orden r entero, el producto interno de H^r es

$$(\phi, \psi)_{H^r(\Omega)} = \sum_{|\alpha|\leq r} \int_\omega \overline{\partial^\alpha \phi(\mathbf{x})} \partial^\alpha \psi(\mathbf{x}) d\Omega, \tag{1.97}$$

y para orden real s es

$$(\phi, \psi)_{H^s(\Omega)} = (\phi, \psi)_{H^r(\Omega)} + \sum_{|\alpha|=r} \int_\Omega \int_\Omega \frac{\overline{(\partial^\alpha \psi(\mathbf{r}) - \partial^\alpha \psi(\mathbf{r'}))}(\partial^\alpha \psi(\mathbf{r}) - \partial^\alpha \psi(\mathbf{r'}))}{|\mathbf{r} - \mathbf{r'}|^{n+2\mu}} d\Omega d\Omega. \tag{1.98}$$

En las definiciones desde las ecuación (1.89) a la (1.98), la derivada $\partial^\alpha \psi$ es tomada en forma débil, esto es,

$$\int_\Omega \phi D^\alpha \zeta d\Omega = \int_\Omega \psi \zeta d\Omega \forall \zeta \in C^\infty \tag{1.99}$$

donde D^α es el operador de derivada clásica, luego ψ es la derivada de orden α de ϕ. En los casos en que $\psi \in C^\alpha$, $\partial^\alpha \psi$ coincide con la derivada clásica.
También podemos definir el espacio de Sobolev $\tilde{H}^s(\Omega)$ como el cierre de $C_0^\infty(\Omega)$ respecto de la norma en el espacio de $H^s(\Omega)$, $(||\psi||_{H^s(\Omega)})$.

Operador Traza

El operador traza γ_0 toma la distribución $\psi(\mathbf{r}_\Omega)$ válida en todo el dominio y la evalúa en la frontera

$$\gamma_0 \psi(\mathbf{r}_\Omega) = \psi(\mathbf{r}_\Gamma). \tag{1.100}$$

Si $s > 1/2$, entonces γ_0 tiene una extensión única a un operador acotado

$$\gamma_0 : H^s(\mathbb{R}) \to H^{s-1/2}(\mathbb{R}^{n-1}). \tag{1.101}$$

El operado γ_0 también es conocido como el operador traza de Dirichlet, y podemos encontrarlo en el segundo término de la ecuación (1.87), donde $\phi(\mathbf{r}) = \phi(\mathbf{r}_\Gamma)$.

El término en el lado derecho de la ecuación (1.87) contiene la derivada normal de la traza. Este operador combinado es conocido como la traza de Neumann, γ_1:

$$\gamma_1 = \partial/\partial\mathbf{n}\gamma_0 : H^s(\mathbb{R}^n) \to H^{s-3/2}(\mathbb{R}^{n-1}) \tag{1.102}$$

Con esta nueva notación, podemos reescribir la ecuación (1.74) usando la segunda identidad de Green para las distribuciones ψ y ϕ como

$$\int_\Gamma \gamma_1\phi(\mathbf{r'})\gamma_0 w(\mathbf{r'})d\Gamma - \int_\Gamma \gamma_0\phi(\mathbf{r'})\gamma_1 w(\mathbf{r'})d\Gamma - \int_\Omega \phi(\mathbf{r'})\nabla^2 w(\mathbf{r'})d\Omega = 0 \tag{1.103}$$

lo que es válido y tiene su solución única para $\phi(\mathbf{r}_\Omega) \in H^1$ y $w(\mathbf{r}_\Omega) \in H^1$. Luego, $\nabla^2 w(\mathbf{r}_\Omega) \in \tilde{H}^{-1}, \gamma_0\phi(\mathbf{r}_\Omega) \in H^{1/2}$ y $\gamma_1\phi(\mathbf{r}_\Omega) \in H^{-1/2}$.

Operador de potencial de capa singular

Para una función de densidad $\psi \in H^{-1/2}$ en una superficie de Lipshitz Γ que encierra un dominio Ω, el operador de potencial se capa singular evaluado en $\mathbf{r}_\Omega$ es

$$\phi(\mathbf{r}_\Omega) = \tilde{V}_L^{\mathbf{r}_\Omega}((\psi(\mathbf{r}_\Gamma))) = \int \psi(\mathbf{r}_\Gamma)G_L(\mathbf{r}_\Omega, \mathbf{r}_\Gamma)d\Gamma, \tag{1.104}$$

y considerando la ecuación (1.81), la distribución resultante $\phi(\mathbf{r}_\Omega)$ está en H^1. Por lo tanto, la ecuación (1.104) define un mapeo lineal

$$\tilde{V} : H^{-1/2}(\Gamma) \to H^1(\Omega), \tag{1.105}$$

delimitada por

$$||\phi||_{H^1(\Omega)} = ||\tilde{V}_L^{\mathbf{r}_\Omega}(\psi(\mathbf{r}_\Gamma))||_{H^1(\Omega)} < c||\psi||_{H^{-1/2}(\Gamma)}. \tag{1.106}$$

Dado que $\tilde{V}_L^{\mathbf{r}_\Omega}(\psi(\mathbf{r}_\Gamma)) \in H^1$, la traza de Dirichlet de $\tilde{V}$ está bien definida y delimitada, definiendo un nuevo operador de potencial de capa singular

$$V = \gamma_0\tilde{V} : H^{-1/2} \to H^{1/2} \tag{1.107}$$

que es

$$\phi(\mathbf{r}_\Gamma) = V_L^{\mathbf{r}_\Gamma}(\psi(\mathbf{r})_\Gamma) = \int_\Gamma \psi(\mathbf{r'}_\Gamma)G_L(\mathbf{r}_\Gamma, \mathbf{r'}_\Gamma)d\Gamma. \tag{1.108}$$

El hecho de que el operador V resulte en una distribución en $H^{1/2}$ concuerda con la ecuación (1.87), donde el término $\phi(\mathbf{r})/2$ también están en $H^{1/2}$ porque está evaluado en la frontera, y la distribución $\partial\phi/\partial\mathbf{n} \in H^{-1/2}$.

Operador de potencial de capa doble

Para una función de densidad $\psi \in H^{1/2}$ en una superficie de Lipshitz Γ que encierra un dominio Ω, el operador de potencial de doble capa evaluado en $\mathbf{r}_\Omega$ es

$$\phi(\mathbf{r}_\Omega) = W_L^{\mathbf{r}_\Omega}(\psi(\mathbf{r}_\Gamma)) = \int_\Gamma \psi(\mathbf{r}_\Gamma)\frac{\partial}{\partial \mathbf{n}} G_L(\mathbf{r}_\Omega, \mathbf{r}_\Gamma) d\Gamma, \tag{1.109}$$

y considerando la ecuación (1.81), la distribución resultante $\phi(\mathbf{r}_\Omega)$ está en H^1. Por lo tanto, la ecuación (1.109) define un mapeo lineal

$$\tilde{W} : H^{1/2}(\Gamma) \to H^1(\Omega), \tag{1.110}$$

delimitada por

$$||\phi||_{H^1(\Omega)} = ||W_L^{\mathbf{r}_\Omega}(\psi(\mathbf{r}_\Gamma))||_{H^1(\Omega)} < c||\psi||_{H^{1/2}(\Gamma)}. \tag{1.111}$$

Dado que $\tilde{W}_L^{\mathbf{r}_\Omega}(\psi(\mathbf{r}_\Gamma)) \in H^1$, la traza de Dirichlet de W está bien definida y delimitada, definiendo un nuevo operador de potencial de capa doble

$$V = \gamma_0 W : H^{1/2} \to H^{1/2} \tag{1.112}$$

Vimos en la ecuación (1.86) que el término $\phi/2$ aparece al tomar la traza de Dirichlet del potencial de capa doble. Es útil definir el operador K como

$$\phi(\mathbf{r}_\Gamma) = K_L^{\mathbf{r}_\Gamma}(\psi(\mathbf{r}_\Gamma)) = \int_\Gamma \psi(\mathbf{r'}_\Gamma)\frac{\partial}{\partial \mathbf{n}} G_L(\mathbf{r}_\Gamma, \mathbf{r'}_\Gamma) d\Gamma. \tag{1.113}$$

para escribir $\gamma_0 W$ como

$$\gamma_0 W = \frac{1}{2} + K \tag{1.114}$$

El hecho de que el operador W resulte en una distribución en $H^{1/2}$ concuerda con la ecuación (1.87), donde el término $\phi(\mathbf{2})/2$ también está en $H^{1/2}$ porque está evaluado en la frontera, y la distribución $\phi \in H^{1/2}$.
Podemos reescribir la ecuación (1.87) usando la notación de operadores que se ha introducido, como sigue

$$\left[\frac{\mathbb{I}}{2} + K_L^{\mathbf{r}_\Gamma}\right](\phi_\Gamma) = V_L^{\mathbf{r}_\Gamma}\left(\frac{\partial}{\partial \mathbf{n}}\phi_\Gamma\right), \tag{1.115}$$

donde $\mathbb{I}$ es el operador identidad. En general, podemos formular el sistema en la ecuación (1.115) como

$$Ax = f. \tag{1.116}$$

Si la condiciones de borde de Neumann son forzadas, $A = \left[\frac{\mathbb{I}}{2} + K_L^{\mathbf{r}_\Gamma}\right]$, $x = \phi_\Gamma$, y $f = V_L^{\mathbf{r}_\Gamma}(\frac{\partial}{\partial \mathbf{n}}\phi_\Gamma)$, y si las condiciones de borde de Dirichlet son forzadas, $A = V_L^{\mathbf{r}_\Gamma}$, $x = \frac{\partial}{\partial n}\phi_\Gamma$, y $f = \left[\frac{\mathbb{I}}{2} + K_L^{\mathbf{r}_\Gamma}\right](\phi_\Gamma)$. Considerando que todos los operadores involucrados en la ecuación (1.115) son elípticos, lo que los hace coercitivos, y cerrados, el teorema de Lax-Milgram asegura la solvabilidad del sistema en la ecuación (1.116). Este teorema dice que si A es acotada, de forma bilineal, y coercitiva, para cualquier f existe una única y acotada solución x.

1.2.3. Discretización y sistema lineal

Para obtener la forma discreta de los operadores de capa de la ecuación (1.108) y la ecuación (1.114), discretizamos la superficie en paneles triangulares plano, y asumimos un valor constante de ϕ y $\partial\phi/\partial\mathbf{n}$ en cada panel. La forma discretizada de los operadores de capa singular y doble son

$$V^{\mathbf{r}_i}_{L,\text{disc}}\left(\frac{\partial}{\partial\mathbf{n}}\phi(\mathbf{r}_\Gamma)\right) = \sum_{j=1}^{N_p}\frac{\partial}{\partial\mathbf{n}}\phi(\mathbf{r}_{\Gamma_j})\int_{\Gamma_j} G_L(\mathbf{r}_i,\mathbf{r}_{\Gamma_j})d\Gamma_j \tag{1.117}$$

$$K^{\mathbf{r}_i}_{L,\text{disc}}\left(\phi(\mathbf{r}_\Gamma)\right) = \sum_{j=1}^{N_p}\phi(\mathbf{r}_{\Gamma_j})\int_{\Gamma_j}\frac{\partial}{\partial\mathbf{n}}G_L(\mathbf{r}_i,\mathbf{r}_{\Gamma_j})d\Gamma_j \tag{1.118}$$

donde N_p es el número de elementos de discretización de Γ, y $\phi(\mathbf{r}_{\Gamma_j})$ y $\frac{\partial}{\partial\mathbf{n}}\phi(\mathbf{r}_{\Gamma_j})$ son los valores de ϕ y $\frac{\partial\phi}{\partial\mathbf{n}}$ en el panel Γ_j.
En este trabajo, obtenemos un sistema lineal mediante la colocación de la ecuación en el centroide de cada panel Γ_i. Por ejemplo, en el problema de Laplace, el sistema sería

$$\left[\frac{\mathbb{I}}{2} + K^{\Gamma}_{L,\text{disc}}\right][\phi] = \left[V^{\Gamma}_{L,\text{disc}}\right]\left[\frac{\partial}{\partial\mathbf{n}}\phi\right] \tag{1.119}$$

donde los elementos de las matrices son

$$V^{\mathbf{r}_i}_{L,ij} = \int_{\Gamma_j} G_L(\mathbf{r}_i,\mathbf{r}_{\Gamma_j})d\Gamma_j \tag{1.120}$$

$$K^{\mathbf{r}_i}_{L,ij} = \int_{\Gamma_j}\frac{\partial}{\partial\mathbf{n}}G_L(\mathbf{r}_i,\mathbf{r}_{\Gamma_j})d\Gamma_j \tag{1.121}$$

con $\mathbf{r}_{\Gamma_i}$ en el centro de Γ_i

1.2.4. Métodos de multipolo

Una desventaja de BEM es que las matrices obtenidas son densas. Esto significa que resolver el sistema lineal con eliminación Gaussiana requiere $\mathcal{O}(N^3)$ cómputos y $\mathcal{O}(N^2)$ espacios de memoria. Usando un solver iterativo, como el método de residuos mínimos generalizados (GMRES por su nombre en inglés),los cálculos están dominados por un producto matriz-vector, y el escalamiento decae a $\mathcal{O}(N^2)$ por iteración. Este escalamiento en los cálculos y la memoria, hace que BEM sea factible para algunos miles de elementos de borde, lo que está bastante lejano a los tamaños de malla que se requieren para aplicaciones reales.
Las estrategias de aceleración usan la naturaleza de decaimiento del kernel para calcular una aproximación de bajo rango de la matriz de BEM. Los más usados comúnmente son los métodos de multipolos, métodos basados en FFT y la aproximación cruzada adaptativa (ACA).
Los métodos de multipolo fueron desarrollados originalmente para acelerar problemas de N-cuerpos, cálculos de complejidad $\mathcal{O}(N^2)$ en simulación de partículas (Barnes y Hut, 1986) (Greengard y Rokhlin, 1987), donde el potencial debido a *sources*

puntuales (fuentes) de masa son calculadas en la ubicación de los *targets* (objetivos). Estos métodos aproximan las interacciones entre *sources* y *targets* lejanos con una expansión de multipolo usando expansiones de Taylor (Lindsay y Krasny, 2001) (Li et al., 2009) o esféricos armónicos (Greengard y Rokhlin, 1987) (Greengard et al., 1998) (Greengard y Huang, 2002). En el contexto de BEM, los algoritmos de multipolo son útiles porque usualmente aproximamos integrales con cuadraturas Gaussianas, que convierten productos matriz-vector de la matriz de BEM en un cálculo de N-cuerpos. Los dos algoritmos principales de este tipo son el treecode (Barnes y Hut, 1986) y el método rápido de multipolos (FMM) (Greengard y Rokhlin, 1987).

1.3. Treecode

Treecode es un algoritmo capaz de ejecutarse en complejidad $\mathcal{O}(N \log N)$ a partir de patrones $\mathcal{N}^{\in}$ del tipo

$$V(x) = \sum_{j=1}^{N} q_j \psi(\mathbf{x}_i, \mathbf{y}_j) \tag{1.122}$$

donde q_j es el peso, ψ el kernel, $\mathbf{y}_j$ la posición de los *sources* y $\mathbf{x}_i$ la de los *targets*.

Un octree, árbol octal u octárbol es una estructura de datos en forma de árbol en la cual cada nodo interno tiene exactamente 8 hijas. Estas estructuras se usan mayormente para partir un espacio tridimensional, dividiéndolo recursivamente en ocho octantes. En treecode, agrupamos los *sources* $\mathbf{y}_j$ de acuerdo a su posición espacial en celdas o cajas de un octárbol. Teniendo esto, calculamos las contribuciones cercanas de forma directa, y aproximamos las contribuciones distantes escribiendo expansiones en series representando los *sources* al rededor de los centros de las cajas y_c, como se muestra en la figura 1.5.

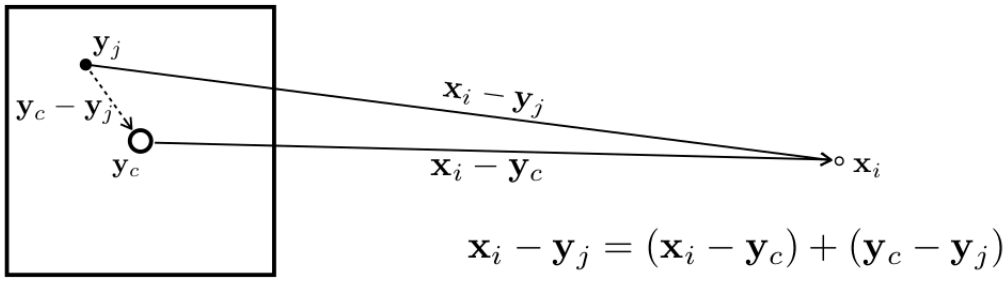


Figura 1.5: Expansión de multipolos en Treecode. Cooper, C. C. (2015). Multipole expansion in the treecode. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

Las expansiones son basadas usualmente o en expansiones de Taylor o en esféricos armónico. Nosotros usamos las expansiones de Taylor, que son las siguientes

para ψ

$$\psi(\mathbf{x}_i, \mathbf{y}_j) = \sum_{||\mathbf{k}||=0}^{P} \frac{1}{\mathbf{k!}} D_{\mathbf{y}}^{\mathbf{k}}(\mathbf{x}_i, \mathbf{y}_j)(\mathbf{y}_c - \mathbf{y}_j)^{\mathbf{k}} \tag{1.123}$$

donde P es el orden de la expansión, $D_{\mathbf{y}}^{\mathbf{k}} = D_{y_1}^{k_1} D_{y_2}^{k_2} D_{y_3}^{k_3}$ el operador derivada, y $\mathbf{k} = (k_1, k_2, k_3)$. Definimos las operaciones en $\mathbf{k}$ como $\mathbf{k!} = k_1!k_2!k_3!$ y $\mathbf{y}^{\mathbf{k}} = y_1^{k_1} y_2^{k_2} y_3^{k_3}$. Reescribiendo los términos en la ecuación (1.123), encontramos la siguiente expresión

$$V(\mathbf{x}_i,) = \sum_{||\mathbf{k}||=0}^{P} \frac{1}{\mathbf{k!}} D_{\mathbf{y}}^{\mathbf{k}}(\mathbf{x}_i, \mathbf{y}_j)(\mathbf{y}_c - \mathbf{y}_j)^{\mathbf{k}} \tag{1.124}$$

donde

$$a^{\mathbf{k}}(\mathbf{x}_i, \mathbf{y}_c) = \frac{1}{\mathbf{k!}} \mathbf{D}_{\mathbf{y}}^{\mathbf{k}} \psi(\mathbf{x}_i, \mathbf{y}_c) \tag{1.125}$$

son los coeficientes de la expansión de Taylor y

$$m_c^{\mathbf{k}} = \sum_j q_j (\mathbf{y}_c - \mathbf{y}_j)^{\mathbf{k}} \tag{1.126}$$

son los multipolos. En la ecuación (1.124), el cálculo de los multipolos puede ser hecho de forma independiente de $\mathbf{x}_i$, eliminando así el escalamiento $O(N^2)$.
Para cada punto *target*, debemos decidr si una caja de *sources* está suficientemente distante como para que la expansión en series de Taylor sea aceptable. Si es el caso, usamos la ecuación (1.124); si no es así, entramos un nivel más allá en el octárbol y hacemos la misma pregunta sobre las cajas *hijas*. Hacemos este proceso recursivamente hasta que una de las cajas de más bajo nivel sea alcanzada, y se computa la influencia de aquellos *sources* de manera directa. Esta decisión se toma basándose en el criterio de aceptación de multipolo, o *multipole-acceptance criterion* θ (MAC), que debe ser más grande que el ratio entre el tamaño de la caja y la distancia entre el *target* y el centro de la caja, $\frac{r_b}{r} < \theta$. Valores comunes para θ son 1/2 y 2/3.
El algoritmo de treecode puede ser dividido en cinco partes: generación del árbol, partícula a multipolo o *particle-to-multipole*, multipolo a multipolo o *multipole-to-multipole*, multipolo a partícula o *multipole-to-particle* y partícula a partícula o *particle-to-particle*. La figura 1.6 muestra estos pasos para un grupo de *sources* $\mathbf{y}_j$, representados por puntos negros sólidos, interactuando con dos puntos *target* $\mathbf{x}_i$ representados por cruces.

Generación del árbol

Los *sources* son agrupados en cajas de una estructura de octárbol, asegurándose de que las cajas de menor nivel tengan menos de algún número crítico de partículas. A este parámetro llamaremos NCRIT. En la figura 1.6 dada como ejemplo, los *sources* $\mathbf{y}_j$ son agrupados en un árbol de dos niveles. El nivel más bajo tiene cuatro cajas que están contenidas dentro de una caja más grande del nivel más alto, centrado en y_c.

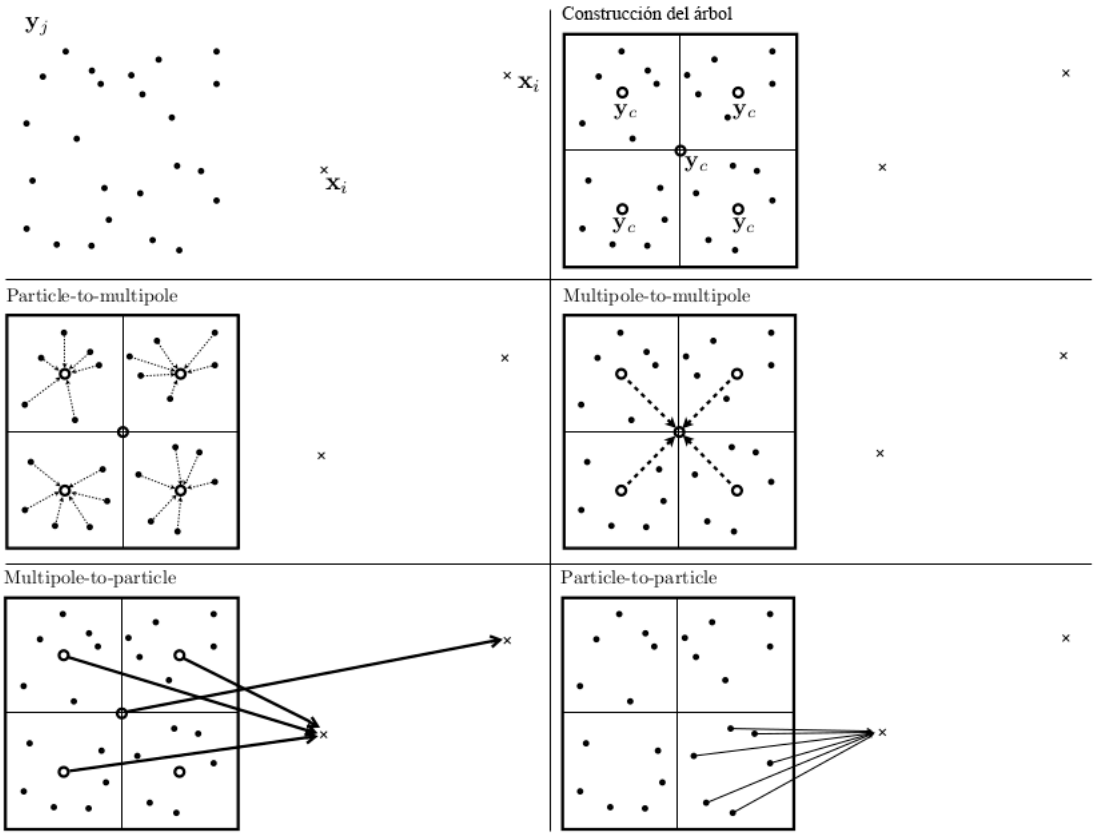


Figura 1.6: Pasos del Treecode. Cooper, C. C. (2015). Steps of the treecode. [Figura]. En Biomolecular electrostatics with continuum models: a boundary integral implementation and applications to biosensors.

Partícle-to-multipole (P2M)

Los multipolos son calculados para las cajas de más bajo nivel usando la ecuación (1.126). Los *sources* son expandidos en torno al centro de la caja correspondiente.

Multipole-to-multipole (M2M)

Los multipolos son calculados para las cajas restantes. En el caso de la figura 1.6, solo la caja grande que contiene a todos los *sources* necesita que su multipolo sea calculado. Para eficiencia, calculamos estos multipolos cambiando la expansión de Taylor desde las cajas hijas, lo que resulta en

$$m_p^{\mathbf{k}} = \sum_{\text{hijas}} \sum_{k_1',k_2',k_3'=0}^{k_1'\leq k_1, k_2'\leq k_2, k'3\leq k_3} m_c^{\mathbf{k'}} \binom{k_1}{k_1'}\binom{k_2}{k_2'}\binom{k_3}{k_3'} (\mathbf{y}_{cp}-\mathbf{y}_{cc})^{\mathbf{k}-\mathbf{k'}} \tag{1.127}$$

donde $m_p^{\mathbf{k}}$ es el multipolo de orden $\mathbf{k}$ de la caja madre, $\mathbf{k'} = (k_1', k_2', k_3')$, $\mathbf{y}_{cp}$ es la ubicación de la caja madre, y $\mathbf{y}_{cc}$ es la ubicación de la caja hija.

Multipole-to-particle (M2P)

La ecuación (1.124) es calculada por las interacciones partícula-caja que satisfacen el criterio θ. La representación en la figura 1.6 muestra que el *targets* más a la derecha está suficientemente lejana como para interactuar con la caja grande, mientras que el *target* más cercano a los *sources* no cumple con el criterio θ con la caja inferior derecha.

En nuestra implementación, usamos de manera recursiva las relaciones de Krasny y su equipo para calcular los coeficientes de Taylor en la ecuación (1.125) (Lindsay y Krasny, 2001)(Li et al., 2009). Usando la notación corta $\Delta\mathbf{x} = (\Delta x_1, \Delta x_2, \Delta x_3) = \mathbf{x}_i - \mathbf{y}_c$, $\mathbf{e}_1 = (1,0,0)$ $\mathbf{e}_2 = (0,1,0)$ $\mathbf{e}_3 = (0,0,1)$ y $||\mathbf{k}|| = k_1 + k_2 + k_3$, la relación recursiva para el potencia de Laplace es

$$a^{\mathbf{k}}(\mathbf{x}_i,\mathbf{y}_c) = \frac{1}{||\mathbf{k}||\cdot|\Delta\mathbf{x}|^2}\left[-(2||\mathbf{k}||-1)\sum_{j=1}^{3}\Delta x_j a^{\mathbf{k}-\mathbf{e}_j} - (||\mathbf{k}||-1)\sum_{j=1}^{3} a^{\mathbf{k}-2\mathbf{e}_j}\right] \tag{1.128}$$

Para el potencial de Yukawa, la relación recursiva es

$$\begin{aligned} a^{\mathbf{k}}(\mathbf{x}_i,\mathbf{y}_c) = &\frac{1}{||\mathbf{k}||\cdot|\Delta\mathbf{x}|^2}\left[-(2||\mathbf{k}||-1)\sum_{j=1}^{3}\Delta x_j a^{\mathbf{k}-\mathbf{e}_j} - \right. \\ &\left.(||\mathbf{k}||-1)\sum_{j=1}^{3} a^{\mathbf{k}-2\mathbf{e}_j} - \kappa\left(\sum_{j=1}^{3}\Delta x_j b^{\mathbf{k}-\mathbf{e}_j} + \sum_{j=1}^{3} b^{\mathbf{k}-2\mathbf{e}_j}\right)\right. \end{aligned} \tag{1.129}$$

donde κ es el coeficiente en el exponente del potencial de Yukawa y,

$$b^{\mathbf{k}} = -\frac{\kappa}{||\mathbf{k}||}\left(\sum_{j=1}^{3}\Delta x_j a^{\mathbf{k}-\mathbf{e}_j} + \sum_{j=1}^{3} a^{\mathbf{k}-2\mathbf{e}_j}\right) \tag{1.130}$$

Particle-to-particle (P2P)

Las interacciones entre pares de partículas cuando estas están demasiado cercanas como para pasar el criterio θ. En la figura 1.6 esto ocurre para las interacciones entre los *sources* en la caja inferior derecha con los *targets* más cercanos a ella.

1.3.1. Treecode en BEM

Existen muchas opciones para solvers estándares de sistemas lineales lineales para calcular las soluciones de los generados por BEM. Usamos el método GMRES, un solver iterativo de subespacio de Krylov que acepta matrices no simétricas. En GMRES, los tiempos de cómputo están dominados por multiplicaciones matriz-vector, un patrón de complejidad $\mathcal{O}(N^2)$, que son ejecutadas n veces (usualmente $n << N$). En aplicaciones reales, evaluamos la mayor parte de las integrales de la matriz de BEM con cuadraturas Gaussianas, que transforman cada integral en una suma. Por ejemplo, para el operador de potencial de capa singular en la ecuación 1.119, cada elemento de la matriz se convierte en

$$V^{\Gamma}_{L,i,j} = \int_{\Gamma_j} G_L(\mathbf{r}_{\Gamma_i}, \mathbf{r}_{\Gamma_j}) d\Gamma_j = \sum_{k=1}^{N_k} w_k G_L(\mathbf{r}_{\Gamma_i}, \mathbf{r}_{k\Gamma_j}) \tag{1.131}$$

donde N_k es el número de puntos de Gauss, ubicado en $\mathbf{r}_{k\Gamma_j}$ y con peso w_k. Si multiplicamos la matriz $\left[V^{\Gamma}_{L,\text{disc}}\right]$ con el vector $\mathbf{m}$, cada elemento del vector resultante es computado como

$$V^{\Gamma}_{L,i,j} m_j = \sum_{j=1}^{N_p} m_j \sum_{k=1}^{N_k} w_k G_L(\mathbf{r}_{\Gamma}, \mathbf{r}_k \Gamma_j) = \sum_{l=1}^{N_p, N_k} M_l G_L(\mathbf{r}_{\Gamma}, \mathbf{r}_l), \tag{1.132}$$

lo cual es un problema de N-cuerpos de la forma de la ecuación (1.122), donde M_l tiene contribuciones del vector $\mathbf{m}$ y los pesos de integración Gaussiana w_k, y $\mathbf{r}_l$ es la posición de los nodos de gauss $\mathbf{r}_k\Gamma_j$ de todos los paneles Γ_j. Podemos usar el algoritmo treecode para acelerarlo, lo que significa que no existe necesidad de guardar la matriz explícitamente en memoria y los cómputos escalan con complejidad algorítmica $\mathcal{O}(N \log N)$ en vez de $\mathcal{O}(N^2)$.
En el contexto del BEM, los *sources* $\mathbf{y}_j$ en la ecuación (1.122) son los puntos de cuadratura Gaussiana, los *targets* $\mathbf{x}_i$ corresponden a los puntos de colocación y el kernel ψ es la función de Green en el espacio libre.

1.4. Computación paralela

Con la tecnología se genera un ciclo en que una máquina más poderosa siempre lleva a nuevos tipos de aplicaciones que a su vez trae la necesidad de sistemas poderosos. El resultado es un progreso tecnológico continuo de los sistemas computacionales que son altamente complejos. Están hechos de múltiples componentes, o unidades funcionales que pueden operar simultáneamente y tener tareas específicas, como sumar dos enteros o determinar si un valor es mayor a cero. Como resultado, un

computador, podría ser capaz de hallar un dato en memoria y multiplicarlo por algún número y evaluar alguna condición booleana todo al mismo tiempo. Este es un nivel realmente bajo de procesamiento paralelo y usualmente es llamado "paralelismo a nivel de instrucción", o ILP por sus siglas en inglés. Un procesador que soporta este tipo de paralelismo tiene una arquitectura superescalar. Hoy en día es común en microprocesadores de propósito general, incluso en aquellos usados en laptops y PCs. El ordenamiento cuidadoso de estas operaciones puede mantener los componentes de la máquina ocupados. La mayor parte del trabajo de encontrar un ordenamiento óptimo de las operaciones es realizada por el compilador. Para lograr esto, los creadores de los compiladores desarrollaron técnicas para determinar dependencias entre operaciones y encontrar un ordenamiento que use eficientemente el paralelismo a nivel de instrucción y guarda varias unidades funcionales y caminos a memoria ocupados con trabajo útil. Compiladores modernos ponen un esfuerzo considerable en optimizaciones de este nivel. Lamentablemente, varios estudios (Jouppi y Wall, 1989) han mostrado que aplicaciones típicas no suelen contener más de tres o cuatro instrucciones que pueden ser dadas a un computador al mismo tiempo en esta manera, por lo que no resulta rentable invertir en acelerar este tipo de procesamiento a través del hardware.

En los 80s, varios vendedores, produjeron computadores que explotaban otro tipo de arquitectura paralela. Construyeron máquinas consistentes en múltiples procesadores completos con una memoria compartida común. Estás máquinas de multiprocesador o de paralelismo a nivel de memoria compartida podían procesar programas con una variedad de necesidades de memoria, y eran perfectas para varios balances de carga diferentes. Como resultado, se volvieron populares en el mercado, donde se han mantenido populares hasta ahora. Ambos tipos de computadores, con pequeño y gran tamaño de memoria compartida (en términos de número de procesadores) han sido desarrollados. Usualmente encontramos dos o cuatro CPUs conectadas a la misma memoria compartida, pero existen sistemas que contienen más de mil CPUs trabajando y el número que puede ser configurado sigue creciendo. La tecnología usada para conectar los procesadores y la memoria ha sido mejorada significativamente desde los primeros días de desarrollo. Recientes desarrollos en tecnología de hardware ha vuelto este tipo de arquitectura de paralelismo importante para la computación de corriente principal.

En décadas pasadas, los componentes usados para construir máquinas de escritorios o para fines más avanzados han estado decreciendo en cuanto a tamaño continuamente. Un poco antes de 1990, Intel anunció que la compañía había puesto un millón de transistores en un solo chip (el i860). Unos pocos años después, varias compañías habían puestos 10 millones de ellos en un chip. En la actualidad, el progreso tecnológico ha estado acotado por la ley empírica de Moore, que expresa que aproximadamente cada 2 años se duplica el número de transistores en un microprocesador. Al volverse más cortos los caminos de datos, el ritmo al cual las instrucciones son enviadas ha aumentado, por lo que hacer crecer la rapidez global de los procesos se ha vuelto una fuente mayor de avance en cuanto a desempeño del procesador. Este enfoque tiene limitaciones inherentes, particularmente respecto al consumo de potencia y emisión de calor, que es cada vez más difícil de disipar.

Por lo mismo, recientemente, las arquitecturas de los computadores han comenzado a enfatizar otras estrategias para aumentar el desempeño en cuanto a hardware,

haciendo mejor uso del espacio disponible en el chip. Dada la poca usabilidad de agregar más unidades funcionales, se ha vuelto a las ideas de los 1980s: múltiples procesadores que compartan memoria son configurados en una sola máquina y, cada vez más, en un chip. Esta nueva generación de computadores paralelos en cuanto a memoria compartida es barata y útil para uso de propósitos generales. El diseño de algunos computadores recientes permite a un solo procesador ejecutar múltiples lineas de instrucciones de forma intercalada. Multihebra o multihread simultanea, por ejemplo, intercala instrucciones de múltiples aplicaciones en un intento de no usar más de los componentes de hardware dados en un tiempo. Por ejemplo, el computador puede sumar dos valores de un set de instrucciones y, "al mismo tiempo", ir a buscar un valor de la memorias que sea necesario para lograr una operación de un set distinto de instrucciones. Un ejemplo es la tecnología de Intel llamada hyperthreading™. Otras plataformas van más allá y replican parte sustancial de la lógica de un procesador en un único chip y se comporta como una máquina paralela de memoria compartida. Este enfoque es conocido como multinúcleo o multicore. Plataformas de multihebra simultanea, máquinas multicore y computadores paralelos de memoria compartida proveen soporte de sistema para la ejecución de varias corrientes de instrucción independientes o hebras (threads). Más aún, estas tecnologías pueden ser combinadas para crear computadores que pueden ejecutar un alto número de threads.
Dadas las limitaciones de las estrategias alternativas para crear más computadores potentes, muy probablemente el uso del paralelismo para hardware de propósitos generales se vuelva más común en el futuro cercano. En la actualidad los PCs y laptops ya son multicore o multithread y muy pronto, los procesadores tendrás más núcleos y posiblemente la capacidad de ejecutar múltiples corrientes de instrucciones en cada núcleo. En otras palabras, la tecnología multicore se está democratizando y volviéndose disponible para todo el mundo. Es vital que el software logre sacar el mayor provecho al hardware paralelo.

1.4.1. Computadores de memoria compartida: SMP

El término "computador de memoria compartida" o SMP (*shared-memory parallel computer*) fue acuñado originalmente para designar un sistema de multiprocesador simétrico, un computador paralelo de memoria compartida cuyos procesadores comparten memoria de una forma en que cada uno puede acceder a cada ubicación en memoria con la misma velocidad; esto es, tienen un tiempo acceso a memoria uniforme (UMA). Muchas máquinas de memoria compartida pequeñas son simétricas en este sentido, sin embargo, las que son más grandes no suelen satisfacer esta definición ya que aunque la diferencia sea muy pequeña, algunos procesadores pueden estar más "cerca" de la memoria y por lo tanto pueden acceder más rápido. Decimos que estas máquinas son de acceso no uniforme con caché coherente (ccNUMA). Los primeros intentos de construir ccNUMAs fueron realizados por Kendall Square Research y Denelcor. Hoy en día los vendedores de hardware más grandes ofrecen alguna forma de computadores paralelos de memoria compartida, con tamaños desde los dos hasta los cientos, y en algunos casos miles, de procesadores.
Usaremos el acrónimo SMP para referirnos a todos los sistemas de memoria compartida, incluyendo los cc-NUMA.

La memoria Caché

El hecho de que algunas partes de la memoria de las SMP no sea compartida puede ser algo confuso. Uno de los desafíos más grandes al enfrentar este hardware es la discrepancia entre la rapidez del procesador y de la memoria. Los procesadores han ido volviéndose cada vez más rápidos, pero mientras más rápido pueden ejecutar instrucciones, más rápido necesitan tener acceso a la data necesaria. Desafortunadamente, la velocidad con la que pueden leer y escribir datos no ha crecido al mismo ritmo. En respuesta, los vendedores ha construido computadores con sistemas de memoria jerárquica, en los cuales, una pequeña memoria (muy rápida y muy cara) llamada caché suministra al procesador con la data e instrucciones a ritmos veloces (Patterson et al., 1990). Cada procesador de una SMP necesita su propio caché privado para ser alimentado rápidamente, y por lo tanto, no toda la memoria es compartida. La figura 1.7 muestra el ejemplo de un procesador basado en caché dual-core genérico. Hay dos niveles de caché. El término *nivel* es usado para denotar qué tan lejos, en término de tiempo de acceso, el caché está del CPU. Mientras más alto el nivel, más se demora en alcanzar el caché en ese nivel. En el nivel 1 distinguimos un caché para datos, uno para las instrucciones, y el *translation lookaside buffer* o búfer de traducción anticipada (TLB). Estos tres cachés son privados para cada núcleo, por lo que otros núcleos no pueden acceder a estos espacios de memoria. La figura muestra solo un caché en el segundo nivel. Generalmente este es más grande que cada uno de los cachés del primer nivel y es compartido para ambos núcleos. Además está unificado, lo que significa que puede contener tanto instrucciones como datos.
La data es copiada al caché desde la memoria principal: bloques de ubicaciones en memoria consecutivos son transferidos al mismo tiempo. Dado que el caché es bastante pequeño en comparación a la memoria principal un nuevo bloque puede desplazar la data que previamente había sido copiada. Una operación puede ser realizada casi inmediatamente si los valores que necesita están disponibles en el caché. Pero si no lo están habrá un retraso mientras los datos necesarios son recibidos desde la memoria principal. Por lo tanto, es importante manejar el caché cuidadosamente. Dado que ni el programador ni el compilador puede poner datos directamente o removerlos desde el caché, es útil aprender a programar el código estructuralmente para asegurarse indirectamente que el caché está siendo utilizado de manera correcta o eficiente.

Implicaciones del caché privado

En un sistema uniprocesador, los valores nuevos calculados por el procesador son escritos de vuelta al caché, donde se quedan hasta que su espacio es requerido para otros datos. En este punto, cada valor que no se haya transferido a la memoria principal se copia a ella. Esta estrategia no funciona para una máquina SMP. Cuando el procesador de una SMP guarda los resultados de los cálculos locales en su caché privado, los valores nuevos son accesibles solo para ejecutar código en ese procesador. Si no se toman precauciones extra, no estarán disponibles para las instrucciones que se ejecuten en algún otro lugar de la SMP hasta después que el bloque de datos sea desplazado del caché y el cuándo de esto puede no ser algo tan claro. De hecho, dado que los valores antiguos pueden aún estar en otros cachés privados, la ejecución de

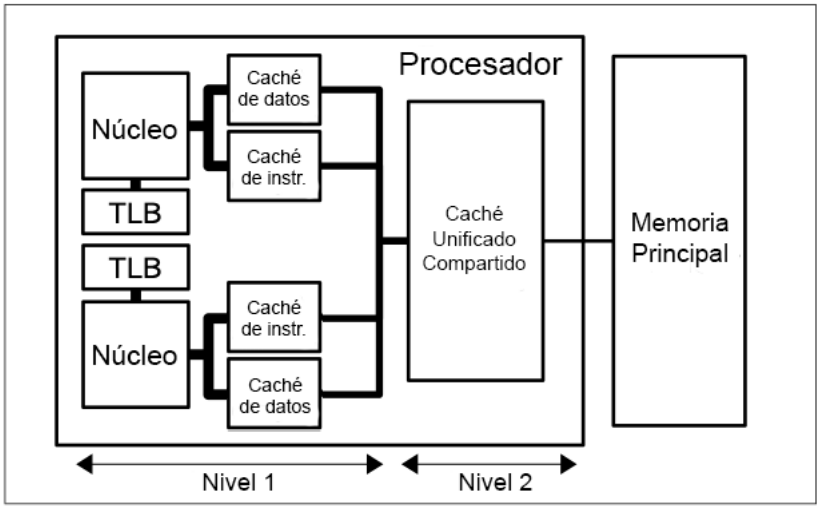


Figura 1.7: Diagrama de bloques de un procesador basado en caché de doble núcleo genérico. Massachusetts Institute of Technology. (2008). Block diagram of a generic, cache-based dual core processor [Figura]. En Using OpenMP.

código en otros procesadores puede continuar usándolos ahí.
Esto es conocido como el problema de consistencia de memoria. Varias estrategias han sido desarrolladas para ayudar a sobrellevar este problema. Su propósito es asegurar que los datos actualizados que han sido puestos en un procesador sean mostrados al programa corriendo en otros procesadores, y para hacer disponibles los datos actualizados si es necesario. Un sistema que provee esta funcionalidad de transparencia se le asigna la cualidad de tener "coherencia de caché".

1.4.2. Computadores de memoria distribuida: DSM

Como se han producido muchos tipos de SMP, también se ha producido máquinas que conectan múltiples computadores independientes mediante una red. Los primeros ejemplos fueron la serie iPSC de Intel (Bomans y Roose, 1989) y máquinas construidas por nCUBE y Meiko (Barton et al., 1994). La memoria es asociada con cada uno de los computadores individuales en la red y es distribuida a través de la máquina. Estos sistemas de memoria distribuida son llamados computadores paralelos masivos o *massively parallel computers* (MPP) porque pueden muchos sistemas singulares pueden ser puestos de esta forma.
Muchos MPP son usados hoy en día, especialmente para computación científica. Si los computadores de memoria distribuida están diseñados con soporte adicional que les permite compartir memoria entre procesadores, también son SMP según nuestra definición. Estas plataformas son usualmente llamadas computadores de memoria distribuida o *shared-memory computers* (DSM) para enfatizar la naturaleza distintiva de esta arquitectura. Cuando los computadores de memoria distribuida son construidos usando estaciones de trabajo estándar o PCs, usualmente se les llama clusters. Los clusters que están usualmente compuestos de SMPs, son mucho más baratos de construir de los MPP propiamente tal. Esta tecnología ha madurado tanto que los clusters son comunes en universidades y laboratorios como en varias compañías.

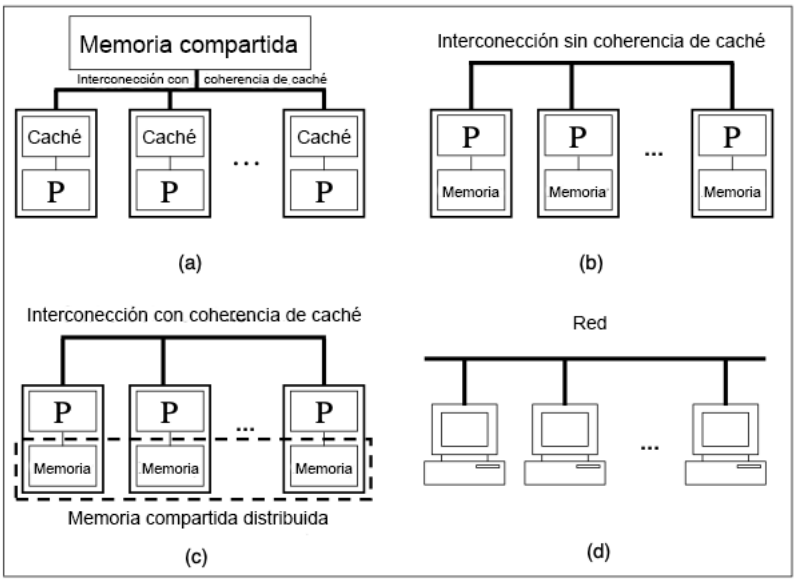


Figura 1.8: Computadores de memoria distribuida y compartida. Massachusetts Institute of Technology. (2008). Distributed- and shared-memory computers [Figura]. En Using OpenMP.

Así, aunque las SMP son las de uso más extendido, existen muchos otros tipos de máquinas paralelas en el mercado.
La figura 1.8 muestra la diferencia entre estas arquitecturas: en (a) vemos un sistema de memoria compartida donde los procesadores comparten memoria principal pero tienen su propio caché privado; (b) representa una MPP en que la memoria está distribuida entre los procesadores o nodos del sistema. La plataforma en (c) es idéntica a (b) excepto por el hecho de que las memorias distribuidas son accesibles para todos los procesadores. El clúster en (d) consiste en un set de computadores independientes en una red.

1.4.3. Modelos de programación paralela

Así como hay varios clases de hardware paralelo, también existen varios tipos distintos de modelos para programación paralela.

Memoria compartida

El modelo de memoria compartida asume, como su nombre implica, que los programas serán ejecutados en uno o más procesadores que comparten toda la memoria disponible. Los programas de memoria compartida son típicamente ejecutados por un numero de hebras independientes, las hebras comparten la data pero también pueden tener datos privados adicionales. Los acercamientos de memoria compartida deben proveer, en adición a un rango de instrucciones normales, una señal para iniciar los distintas hebras, asignando trabajo a cada uno de ellos y coordinando su acceso a los datos compartidos, incluyendo asegurarse de que ciertas operaciones son realizadas por solo una hebra al mismo tiempo. Muchos compiladores proveen un *flag* o opción,

para la paralelización automática. Cuando se escoge esta opción, el compilador analiza el programa, buscando conjuntos independientes de instrucciones, y en particular, buscar loops cuyas iteraciones son independientes una de la otra. Entonces, usa la información para generar código paralelo explícito. Unas de las maneras en que se puede lograr es generando directivas de la API OpenMP o Pthreads, que podría habilitar al programador para poder ver y posiblemente mejorar el código resultante. La dificultad de descansar en el compilador para detectar y explotar el paralelismo en una aplicación es que puede necesitar información extra para poder hacer un buen trabajo. Por ejemplo, puede necesitar conocer los valores que serán asumidos por los loops conectados o el rango de valores de los índices de un arreglo; pero esto usualmente es desconocido antes de la ejecución. Para preservar el uso correcto, el compilador tiene que asumir de forma conservadora que un loop no es paralelo cuando no puede probar lo contrario. Sin ir más lejos, mientras más complejo el código, más probable que esto ocurra.

Memoria distribuida

Un modelo diferente ha sido propuesto para sistemas de memoria distribuida. Generalmente se le conoce como "paso de mensajes", este modelo asume que los programas serán ejecutados por uno o más procesos, cada uno de los cuales tiene su propia dirección de memoria (Gropp et al., 1999). El enfoque de paso de mensajes en programación paralela debe proveer un medio para iniciar y gestionar los procesos participantes, junto con las operaciones para enviar y recibir mensajes, y posiblemente para realizar operaciones especiales a lo largo de los datos distribuidos entre los procesos diferentes. El modelo puro de paso de mensajes, asume que el proceso coopera para intercambiar mensajes cuando quiera que uno de ellos necesite la data producida por algún otro. Sin embargo, algunos modelos recientes están basado en "comunicación unilateral". Esto asume que un proceso puede interactuar directamente con la memoria a través de la red para leer y escribir datos donde se requiera en una máquina. Para este tipo de programación se desarrolló el *message passing interface* o MPI, con el fin de facilitar la programación en MPPs, donde múltiples procesos se ejecutan independientemente y comunican datos bajo necesidad mediante el paso de mensajes. La API fue diseñada para ser altamente expresiva y para permitir la creación de código paralelo eficiente, así como ser ampliamente implementable. Como resultado de su éxito en este campo, es la PI más utilizada para programación paralela en la comunidad, donde las MPPs y los clusters son comunes.

1.4.4. Programación en SMP y OpenMP

Con la propagación de las SMP a precios asequibles fue necesario asegurar que todo el poder de estas máquinas pudiera ser explotado en aplicaciones individuales. Los compiladores siempre han sido responsables de adaptar un programa para hacer mejor uso del paralelismo interno de la máquina. Desafortunadamente, es muy difícil para ellos hacerlo en el caso de un computador con múltiples procesadores o núcleos. La razón es que los compiladores deben identificar corrientes independientes de instrucciones que pueden ser ejecutadas en paralelo. Las técnicas para extraer

esas corrientes de instrucciones de un programa existen y para programas simples, puede valer la pena intentar opciones automáticas de paralelización. Sin embargo, el compilador usualmente no tiene suficiente información para decidir si es posible dividir un programa en alguna forma u otra. Además no puede hacer grandes cambios al código, como reemplazar un algoritmo que no encaja bien con la paralelización. De esa manera, el compilador va a requerir ayuda del usuario.
OpenMP es una interfaz de programación de aplicaciones (API). Las características que presenta están basadas en los esfuerzos previos por facilitar la programación paralela en memoria compartida. En vez de un estándar oficial, es un acuerdo alcanzado por los miembros del ARB (Architecture Review Board)[1], que comparten un interés en un acercamiento a la computación paralela de forma portátil, amigable con el usuario y eficiente. OpenMP está pensado para encajar en las implementaciones en un amplio rango de arquitecturas SMP. Como sus predecesores, OpenMP no es un lenguaje de programación en si mismo. En vez de eso, es una notación que puede ser agregada a otro lenguaje de programación como Fortran, C, o C++ para describir cómo el trabajo será compartido entre los threads que se van a ejecutar en diferentes procesadores o núcleos y para ordenar el acceso a la data compartida como es necesario. La inserción correcta de las características de OpenMP en un programa serial, va a permitir que la mayoría de las aplicaciones se beneficien de la arquitectura paralela de memoria compartida. En la práctica, muchas aplicaciones tiene un paralelismo considerable que puede ser explotado.
OpenMP presenta una serie de factores que lo han vuelto muy exitoso. Uno es el énfasis en la programación paralela estructurada. Otro es que OpenMP es comparativamente fácil de utilizar dado que el trabajo de resolver los detalles del programa paralelo se deja al compilador. La mayor de sus ventajas es el hecho de haber sido ampliamente adoptado, de manera que una aplicación de OpenMP correrá en diferentes plataformas. Con el desarrollo fuerte de grandes y pequeñas SMP y otros hardware multihebra, la necesidad de un programa estándar de memoria compartida que sea fácil de aprender y simples es aceptada en la industria.

1.4.4.1. OpenMP

La API OpenMP fue desarrollada para permitir la programación paralela a nivel de memoria compartida de una manera portátil. Apunta a apoyar la paralelización de aplicaciones de variadas disciplinas y sus creadores tenían la intención de proveer un enfoque que fuera relativamente fácil de aplicar. La API está diseñada para permitir un acercamiento incremental de paralelización de código ya existente, en el cual porciones de un programa son paralelizadas. Esto es un contraste marcado a la conversión todo o nada de un programa entero en un solo paso que típicamente es requerido por otros paradigmas de paralelización. La idea fue el permitir a los programadores el trabajar con un solo código fuente: si un único set de archivos fuente contiene el código tanto para la versión en serie y paralela del programa, entonces la mantención es mucho más simple.
Una hebra, o thread es una entidad de ejecución que es capaz de ejecutar independien-

[1] The OpenMP Architecture Review Board. https://www.openmp.org//about/about-us/. Accedido el 17 de Marzo de 2021.

temente una corriente de instrucciones. OpenMP construye un gran cuerpo de trabajo que soporta la especificación de programas para la ejecución por una colección de threads cooperativos (Andrews, 2000). Los sistemas operativos ejecutan un programa, primero asignando algunos recursos al proceso, incluyendo páginas de memoria y registros para mantener los valores de los objetos. Si varios threads colaboran para ejecutar el programa, entonces van a compartir estos recursos, incluyendo la dirección de espacio del proceso correspondiente. Los threads individuales necesitan solo unos pocos recursos para si mismos, como son un contador de programa y un área en memoria para almacenar variables que son específicas para ellos, incluyendo los registros y un stack o pila. Múltiples threads pueden ser ejecutados en un mismo procesador o núcleo vía cambio de contextos, aunque esto no sería paralelismo propiamente tal. Threads corriendo de forma simultanea en múltiples procesadores o núcleos pueden trabajar concurrentemente para ejecutar un programa paralelo. Programas multithread pueden ser escritos de varias formas, algunos permiten interacciones complejas entre threads. OpenMP intenta facilitar la programación y evitar que el usuario incurra en errores comunes ofreciendo una enfoque estructurado a la programación multithread. Soporta el llamado modelo de programación fork/join, ilustrado en la figura 1.9. Bajo este acercamiento, el programa comienza como un solo thread de ejecución, como si fuera un programa serial. El thread que ejecuta este código es referenciado como el Thread inicial y cuando un constructo paralelo de OpenMP es encontrado por este thread mientras se ejecuta el programa, se crea un conjunto de threads (esto es el fork), se convierte en el thread maestro del conjunto y colabora con los demás para ejecutar el código dinámicamente delimitado por el constructo. Al final del constructo solo continúa el thread maestro original y todos los otros terminan (esto es el join). Cada porción de código encerrada por un constructo paralelo es llamado región paralela.
OpenMP espera que el desarrollador le de especificación de alto nivel acerca del paralelismo del programa y el método de explotación de este paralelismo. Así, provee de notación para indicar las regiones de un programa que deben ser ejecutadas en paralelo; también permite el ser provisto de información adicional sobre cómo debe ser logrado este fin. El trabajo de OpenMP es ordenar los detalles de bajo nivel de crear los threads independientes para ejecutar el código y asignar trabajo de acuerdo a la estrategia especificada por el programador.

1.4.4.2. Características de OpenMP

OpenMP comprende un set de directivas de compilación, librería de rutinas y variables de ambiente para especificar el paralelismo de memoria compartida a Fortran y C/C++. Una directiva de OpenMP es un comentario o pragma con formato especial que generalmente aplica al código ejecutable inmediatamente después de él. Una directiva o rutina de OpenMP generalmente afecta solo a aquellos threads que se la encuentran en su ejecución. Varias directivas son aplicadas a un bloque estructurado de código, una secuencia de declaraciones ejecutables con una sola entrada al principio y una sola salida al final en programas de Fortran y una declaración ejecutable en C/C++. En otras palabras, el programa puede no ramificarse dentro o fuera de un bloque de código asociado a las directivas. OpenMP provee al usuario de formas en

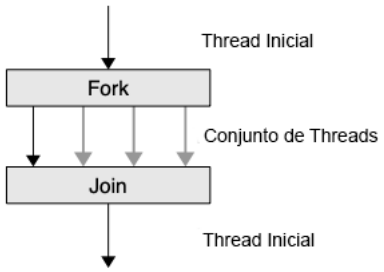


Figura 1.9: Modelo fork/join soportado por OpenMP. Massachusetts Institute of Technology. (2008). The fork-join programming model supported by OpenMP [Figura]. En Using OpenMP.

las que:

- crear conjuntos de threads para ejecución paralela
- especificar como compartir el trabajo entre los miembros de un conjunto
- declarar variables tanto privadas como compartidas
- sincronizar los threads y permitirle a algunos realizar ciertas operaciones exclusivas sin la interferencia de otros threads

Un conjunto o equipo de threads es creado para ejecutar el código en una región paralela de un programa de OpenMP y para lograr eso, el programador simplemente especifica la región paralela insertando la directiva `parallel` inmediatamente antes del código que será ejecutado en paralelo para marcar un comienzo. En el caso de Fortran es necesario también agregar la directiva `end parallel` para cerrar la región paralela. Información adicional puede ser suministrada junto con la directiva `parallel`.

Esto es usado mayormente para habilitar en los threads el tener copias privadas de algunos datos por la duración de la región paralela y para inicializar esos datos. El final de la región paralela es una barrera de sincronización implícita, esto significa que ningún thread puede continuar hasta que todos los threads hayan alcanzado el mismo punto en el programa. Después de eso, la ejecución del programa continua con el thread o los threads que existían previamente. Si un conjunto de threads ejecutando una región paralela se encuentran con otra directiva `parallel`, cada thread en el conjunto crea un nuevo equipo de threads y se vuelve el thread maestro. El anidado permite la realización de programas paralelos de multinivel.

1.4.4.3. Distribución de trabajo

Si el programador no especifica cómo debe compartirse el trabajo entre los threads asignados a una región paralela, cada una ejecutará redundantemente todo el código. Lógicamente esto no acelera el programa. Las directivas de distribución de trabajo de OpenMP son provistas para que el programado pueda establecer como los cálculos en un bloque estructurado de código es distribuido entre las hebras. A menos

que sea explícitamente anulado por el programador, una barrera de sincronización implícita también existe al final de cada constructo de distribución de trabajo. La elección del método puede ser muy influyente en el desempeño del programa. Probablemente el modo más común de distribución de trabajo es hacerlo en un ciclo `DO` en Fortran o `for` en C/C++ entre los threads en un conjunto. Para lograr eso, el programador inserta la directiva apropiada inmediatamente antes del loop en una región paralela. Estas directivas no puedes ser aplicadas a todo tipo de loop en C. Muchos programas, especialmente en aplicaciones científicas, pasan una mayor fracción del tiempo en loops haciendo cálculo en elementos de arreglos y de esta forma esta estrategia es aplicable y muy efectiva. Todos las estrategias de OpenMP para compartir trabajo en loops asignan uno o más sets de iteraciones desconectados a cada thread. El programador puede especificar el método a utilizar para particionar el set. Las estrategia más sencilla asigna a cada thread un lote de iteraciones contiguo. Estrategias más complejas incluyen computar dinámicamente los lotes de iteraciones para cada thread. Si el programador no provee de una estrategia específica, entonces será utilizada una por defecto. No todos los loops pueden ser paralelizados de esta forma. Es posible determinar el número de iteraciones en un loop al entrar, y este número puede no cambiar mientras el loop se ejecuta. Los loops implementados mediante el constructo `while`, por ejemplo, pueden no satisfacer esta condición. Más aún, un loop es adecuado para compartir trabajo entre iteraciones solo si estas son independientes, es decir, que el orden en el que se ejecutan estas iteraciones no tenga incidencias en la salida. Otras cosas pueden afectar la paralelización de un loop como la dependencia de datos. Esta no permite la paralelización de un loop cuando un valor que es escrito en una iteración es también escrito o leído por otra iteración. Mientras más complejo es el loop, más difícil es decidir si existe esta dependencia o no. Otras estrategias pueden ser usadas para asignar trabajo a los distintos threads de una región paralela. Este acercamiento consiste en dar distintos trozos de trabajo a los threads individuales. Este enfoque es adecuado cuando cálculos independientes son ejecutados y el orden en el que se realizan es irrelevante. Es sencillo especificar esto usando la correspondientes directiva de OpenMP. Justo como antes, el programador debe verificar que realmente el loop sea paralelizable.

1.4.4.4. Modelo de memoria de OpenMP

OpenMP está basado en un modelo de memoria compartida, es por eso que por defecto los datos son compartidos entre los threads y son visibles por todos ellos. Sin embargo, a veces se necesitan variables que tengas valores específicos para cada thread. Cuando cada thread tiene su propia copia de una variable, de manera que potencialmente pueda tener un valor diferente para cada uno, denominamos a esa variable una variable privada. Por ejemplo, cuando un equipo de threads ejecuta un loop paralelo, cada thread necesita su propio valor de la variable de iteración. Este caso es tan importante que el compilador lo fuerza. En otros casos el programador debe determinar cuales variables son compartidas y cuales son privadas. La data puede ser declarada compartida o privada respecto a la región paralela o constructo de repartición de trabajo. El uso de variables privadas puede ser benéfico en varias formas. Pueden reducir la frecuencia de actualización de la memoria compartida. Así

pueden ayudar a evitar hot spots o competencias para acceder a ciertos espacios de memoria, que puede ser caro en términos de tiempo si existen múltiples threads intentando acceder. También pueden reducir la probabilidad de que datos remotos accedan en plataformas cc-NUMA y pueden quitar la necesidad de ejecutar algunas sincronizaciones. El lado desfavorable es que aumentará la huella de memoria del programa.
Los threads necesitan un lugar para almacenar los datos privados al momento de ejecución. Para esto cada thread tiene un propia región de memoria especial conocida como el stack del thread. Los desarrolladores de aplicaciones pueden ignorar este detalle de manera segura con una excepción: la mayoría de los compiladores dan al cada thread un espacio por defecto. Pero algunas veces la cantidad de datos que necesitan ser guardados puede ser grande, pues el compilador necesita usar este espacio para guardar otros datos también. Entonces el tamaño por defecto puede no ser suficientemente grande. Afortunadamente, los desarrolladores están provistos usualmente de medios para aumentar el tamaño del stack de un thread. Datos declarados dinámicamente y objetos persistentes requieren un propia área de memoria.
Como se vio en la sección 1.4.1, cada procesador de una SMP tiene una pequeña cantidad de memoria privada llamada caché. Esto puede guiar a problema cuando los datos compartidos están involucrados, pues un valor nuevo de un objeto compartido puede estar en un espacio de cache en vez de la memoria principal, donde otros threads pueden acceder. Afortunadamente no es necesario saber como un sistema especifico lidia con este problema en OpenMP, puesto que esta API tiene sus propias reglas sobre cuando los datos compartidos son visibles o accesibles por los demás threads. Esta reglas dicen que los valores de objetos compartidos deben estar disponibles para los los threads en los puntos de sincronización. Entre los puntos de sincronización, los threads pueden mantener sus valores actualizados ocultos en su propio caché local. Como resultado, los threads pueden tener diferentes valores para objetos compartidos temporalmente. Si un thread necesita un valor que fue creado por otro thread, entonces es necesario insertar un punto de sincronización en el código.
OpenMP también presenta la operación llamada `flush` que asegura que los threads llamándola tienen el mismo valor para objetos compartidos que el que se tiene en la memoria principal. Por lo tanto, nuevos valores de cualquier objeto compartido actualizado por ese thread son escritos en la memoria compartida, y el thread obtiene cualquier valor nuevo obtenido por otro threads para la data compartida que lea. En algunos lenguajes, este `flush` es conocido como valla de memoria, ya que ya que lecturas y escrituras de datos no se pueden cruzarla.

1.4.4.5. Sincronización

La sincronización de threads es algunas veces necesaria para asegurar el ordenamiento apropiado de sus accesos a la data compartida y para prevenir la corrupción de datos.
Asegurar la coordinación requerida de los threads es uno de los desafíos más complicados en el marco de la programación paralela de memoria compartida. OpenMP intenta reducir la probabilidad de errores de sincronización mediante la introduc-

ción de puntos sincronización implícitos. Por defecto, OpenMP hace que los threads esperen al final de un constructo de repartición de trabajo o región paralela hasta que todos los threads hayan terminado. Solo así podrán continuar. Esto es conocido como una barrera. La sincronización de las acciones de un subconjunto de threads es más complicada en OpenMP y requiere programar cuidadosamente porque no existe soporte explícito para ello.
Algunas veces el programado puede necesitar asegurar que solo un thread trabaje en un trozo de código al mismo tiempo. OpenMP tiene varios mecanismos que soportan este tipo de sincronización. Por ejemplo, su un thread intenta ejecutar código protegido por estos mecanismo y que ya ha comenzado ya a ser ejecutado por otro thread, entonces el primero tendrá que esperar a su turno. De forma alternativa, puede ser posible llevar a cabo otro trabajo mientras está esperando. Si es suficiente para proteger las actualizaciones de una variable individual, puede ser más eficiente emplear la directiva `atomic` de OpenMP. Los puntos de sincronización son aquellos lugares en que el código ha sido especificado para detenerse y sincronizarse, sea de manera implícita o explícita. Tienen una función adicional en el código OpenMP: en estos lugares en el código, el sistema se asegura que los threads tienen valores consistentes de objetos de datos compartidos. Los puntos de sincronización de OpenMP incluyen barreras implícitas y explícitas, el comienzo y fin de regiones críticas, puntos en los que se introducen o quitan cerraduras, y cualquier lugar en el que exista la directiva `flush`.

1.4.4.6. Número de threads y números de thread

Para algunas aplicaciones, puede ser importante controlar el número de threads que ejecutan una región paralela. OpenMP permite al programador especificar este número previo a la ejecución del programa vía una variable de ambiente, después de que ha comenzado los cálculos a través de una rutina de librería, o al comienzo de una región paralela. Si es el caso, entonces la implementación debe escoger el número de threads que serán utilizados. Algunos programas necesitan usar el número de threads en un conjunto para setear los valores de ciertas variables. Otros programas van a asignar cálculos a threads específicos. OpenMP asigna números consecutivos, comenzando desde el 0, a cada thread en un equipo para poder identificarlos. Existen rutinas de librería para recibir el número de threads así como para habilitar un thread para poder acceder a su propio identificador numérico.
OpenMP tiene varias características que pueden afectar el número de threads en un equipo. Primero, es posible permitir al ambiente de ejecución variar dinámicamente el número de threads, posiblemente como resultado de otras demandas hechas el los recursos del sistema. El comportamiento por defecto respecto a esto es definido en la implementación. Sin embargo, una vez que el número de threads en un conjunto dado ha creado el número total de ellos no cambiará. En segundo lugar, se permite que una implementación prohíba el paralelismo anidado, o puede ser deshabilitado por el programador.

1.4.4.7. Correctitud: Condición de carrera

Una de las dificultades más grandes de la programación paralela en memoria compartida es el esfuerzo requerido para asegurar que el programa esté correcto. En adición a todas las fuentes de error que pueden surgir en un programa serial, los programas de memoria compartida pueden contener nuevos bugs. Afortunadamente, el uso de directivas y un estilo de programación estructurado es suficiente para prevenir muchos problemas, cuando el programador adopta un estilo de programación de bajo nivel, es necesario tener más cuidado. Un problema en particular son las condiciones de carrera, o *race condition*, que puede ser bastante difícil de detectar y manifestarse en un código de memoria compartida a través de la corrupción de datos. Desafortunadamente, el comportamiento en la ejecución de un programa con condición de carrera no es reproducible, es decir, los datos erróneos pueden ser producidos en una ejecución del programa, pero el problema puede no saltar a la luz la próxima vez que se ejecute. El problema emerge cuando dos o más threads acceden a la misma variable compartida sin una correcta sincronización que ordene este acceso, y al menos uno de los accesos es de escritura. Dado que es relativamente fácil crear una versión paralela de un loop anidado sin contar con que múltiples iteraciones referencien al mismo elemento de un arreglo, por ejemplo, el programador debe estar alerta del impacto que esto puede tener. En general mientras más complejo el código, más difícil es garantizar que no se hayan introducido este tipo de errores. El orden de operaciones realmente observado en un código con condición de carrera depende en la carga de un sistema y de el timing relativo de los threads involucrados. Dado que los threads pueden ejecutar sus instrucciones a diferentes velocidades, y que el trabajo del sistema operativo algunas veces afecta al desempeño de uno o más threads, el orden en el cual estos threads alcanzan cierta parte del código puede variar entre una ejecución y otra. En algunos casos el problema ocurre solo para un número específico de threads. Como resultado, un bug de carrera de datos puede escapar de la detección durante el testeo. De ahí que la importancia de evitar este problema prestando mucha atención durante el desarrollo del programa no puede ser sobre-enfatizada. Las herramientas pueden ayudar a apuntar problemas potenciales. Se pueden identificar otras causas potenciales de errores en programas de OpenMP. Por ejemplo, si el programador ha descansado en la ejecución de un programa por un cierto número de threads y si un número diferente es usado para ejecutar el código, puede ser porque los recursos disponibles no son suficientes, entonces el resultado puede ser inesperado. OpenMP generalmente espera que el programador verifique que el ambiente de ejecución es lo que se requiere y provee rutinas de ejecución para esto.

1.4.4.8. Desempeño

Sea T_1 el tiempo de ejecución de una aplicación en un procesador, entonces en una situación ideal, el tiempo de ejecución en P procesadores debiese ser T_1/P. Si T_P denota el tiempo de ejecución en P procesadores, entonces el ratio

$$S = \frac{T_1}{P} \tag{1.133}$$

Tabla 1.1: Ley de Amdahl para $f_{\text{par}} = 1$.

1	1
2	2
3	3
4	4
8	8
16	16
24	24

Tabla 1.2: Ley de Amdahl para $f_{\text{par}} = 0{,}9$.

1	1
2	1.8
3	2.5
4	3.1
8	4.7
16	6.4
24	7.3

es conocido como el speedup del paralelismo y es una medida de el éxito de la paralelización. Sin embargo, un número de obstáculos usualmente tienen que vencerse antes de lograr un speedup perfecto. Virtualmente todos los programas contienen algunas regiones que son adecuadas para paralelización y otras que no lo son. Al aumentar el número de procesadores, el tiempo ocupado en las partes paralelas es reducido, pero la sección serial se mantiene igual. Eventualmente el tiempo de ejecución es dominado completamente por el tiempo tomado para hacer la porción serial, lo cual poner un límite superior en el speedup esperado. Este efecto es conocido como la ley de Amdahl y puede ser formulada como

$$S = \frac{1}{(f_{\text{par}}/P + (1 - f_{\text{par}}))}, \tag{1.134}$$

donde f_{par} es la fracción paralela del código y P es el número de procesadores. En el caso en que todo el código corre en paralelo, $f_{\text{par}} = 1$, el speedup esperado es igual al número de procesadores, como muestra la tabla 1.1. Pero con solo decrecer la sección paralela en un 20 %, esto es, $f_{par} = 0{,}8$, el speedup máximo que se puede esperar de 16 procesadores es 4 y en 24 procesadores de 4,3, como se aprecia en la tabla 1.3. De ahí la importancia de paralelizar la aplicación tanto como sea posible, ya que de por si la creación de threads y definición de variables previas a la paralelización decrecen el speedup de la paralelización.

Tabla 1.3: Ley de Amdahl para $f_{\text{par}} = 0{,}8$.

1	1
2	1.7
3	2.1
4	2.5
8	3.3
16	4.0
24	4.3

Otros obstáculos junto a la forma de perfeccionar el speedup lineal son la introducción de overhead dado por la separación y unión de los threads (fork/join), la sincronización y los accesos de memoria. Por otro lado, la habilidad de hacer calzar más datos en el caché puede compensar algo de overhead. La medida de la habilidad de un programa para disminuir el tiempo de ejecución del código con un aumento de procesadores se conoce como escalabilidad paralela.

1.5. Proteínas y virus

Las proteínas son moléculas biológicas compuestas básicamente por carbono, hidrógeno, oxígeno y nitrógeno. También pueden contener azufre y ciertos tipos de proteínas, fósforo, hierro, magnesio y cobre entre otros elementos. Pueden considerarse como polímeros de pequeñas moléculas llamadas aminoácidos y, por lo tanto, serán monómeros unitarios. Los aminoácidos están conectados por enlaces peptídicos. La unión de una pequeña cantidad de aminoácidos produce péptidos. Si el número de aminoácidos que componen la molécula no es mayor de 10, se llama oligopéptido; si es mayor de 10, se llama polipéptido; si el número es mayor de 50 aminoácidos, el polipéptido ya es una proteína. Por lo tanto, las proteínas son cadenas de aminoácidos plegadas en estructuras tridimensionales que les permiten realizar miles de funciones. La proteína está codificada en el material genético de cada organismo, y la secuencia de aminoácidos se especifica allí, y luego se sintetiza por el ribosoma. La proteína juega un papel vital en biología y es la biomolécula más versátil y diversa. Realizan una gran cantidad de funciones diferentes, incluidas funciones estructurales, enzimáticas y de transporte.

1.5.1. Aminoácidos y enlace peptídico

Son las unidades básicas de las proteínas. Su nombre corresponde a la composición química general que presentan, en la que el grupo amino (-NH2) y otro grupo carboxilo o ácido (-COOH) están conectados al carbono α (-C-). Las otras dos valencias de este carbono están saturadas con átomos de hidrógeno (-H) y un grupo químico variable llamado radical (-R).

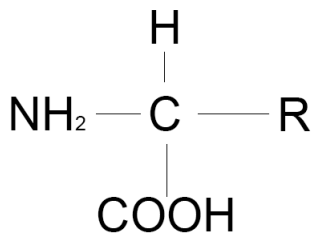


Figura 1.10: Esquema general de un aminoácido.

Tridimensionalmente el carbono α presenta una configuración tetraédrica en la que el carbono se dispone en el centro y los cuatro elementos que se unen a él ocupan los vértices. Cuando en el vértice superior se dispone el -COOH y se mira por la cara opuesta al grupo R, según la disposición del grupo amino (-NH2) a la izquierda o a la derecha del carbono α se habla de α-L-aminoácidos o de α-D-aminoácidos respectivamente. Existen 20 aminoácidos distintos, cada uno de los los cuales está caracterizado por su radical R.
Los aminoácidos se encuentran unidos linealmente por medio de uniones peptídicas. Estas uniones se forman por la reacción de síntesis entre el grupo carboxilo del primer aminoácido con el grupo amino del segundo aminoácido. La formación de un enlace peptídico entre dos aminoácidos es un ejemplo de reacción de condensación. Las dos moléculas están unidas por un enlace covalente CO-NH, pero la molécula de agua se pierde y el producto de la combinación es un dipéptido. El grupo carboxilo libre del dipéptido reacciona de manera similar con el grupo amino del tercer aminoácido, y así sucesivamente para formar una cadena larga. Podemos continuar agregando aminoácidos al péptido, porque siempre hay un extremo NH2 y un extremo COOH. La característica del enlace peptídico es que al ser un doble enlace determina la posición espacial de este en un plano, con distancias y ángulos fijos. Como consecuencia, el enlace presenta rigidez e inmoviliza en el plano a los átomos que lo conforman.

1.5.2. Estructura de proteínas

Todas las proteínas tienen la misma estructura química central, que está compuesta por cadenas lineales de aminoácidos. Lo que hace que una proteína sea diferente de otra es su secuencia de aminoácidos, que se denomina estructura primaria de la proteína. La estructura principal de una proteína determina su función futura, por lo que las proteínas estructurales (como las que forman los tendones y cartílagos) tienen una gran cantidad de aminoácidos rígidos, que establecen fuertes lazos químicos entre sí para dar fuerza a la estructura.
Sin embargo, la secuencia lineal de aminoácidos puede tener múltiples conformaciones en el espacio formado al plegar el polímero lineal. Al repeler el agua los aminoácidos hidrófobos, la atracción de los aminoácidos cargados y la formación de enlaces disulfuro, este pliegue se desarrolla en parte de forma espontánea y también es asistido en parte por otras proteínas. Por tanto, la estructura primaria está determinada por la secuencia de aminoácidos en la cadena de la proteína, es decir, el número de aminoácidos presentes. En el análisis de la estructura secundaria, el orden en que están conectados y la forma en que se pliega la cadena. Además, las posiciones de las proteínas en el espacio son diferentes, por lo que se describe la tercera estructura. Por tanto, la estructura terciaria es cómo se pliega la cadena polipeptídica en el espacio, es decir, cómo se entrelaza una determinada proteína. De manera similar, una proteína no está compuesta en gran medida por una única cadena de aminoácidos, sino que normalmente se combinan múltiples cadenas polipeptídicas (o monómeros) para formar una proteína multimérica más grande. Esto se llama estructura cuaternaria de una proteína, que es la combinación de varias cadenas de aminoácidos (o péptidos) en complejos macromoleculares más grandes. Los enlaces que determinan la estructura primaria son covalentes (enlace amida o enlace peptídico), mientras que la mayoría

de los enlaces que determinan la conformación (estructuras secundaria y terciaria) y la asociación (estructura cuaternaria y quinaria) son de tipo no covalente.

Estructura primaria

La estructura primaria está determinada por la secuencia de aminoácidos en la cadena de la proteína, es decir, el número de aminoácidos presentes y su secuencia de conexión. Las posibilidades de estructuración primaria son prácticamente ilimitadas. Dado que hay 20 aminoácidos diferentes en casi todas las proteínas, el número de estructuras posibles viene dado por la variación de 20 elementos repetidos de n a n, donde n es el número de aminoácidos que componen la molécula de proteína. Dado que los enlaces peptídicos se establecen entre los diferentes aminoácidos que componen una proteína, el resultado es una "columna vertebral" de la que emergen las cadenas laterales de aminoácidos. Los átomos que forman la estructura de la proteína son el nitrógeno del grupo amino (fusionado con el aminoácido anterior), C= (del cual emerge la cadena lateral) y el C del grupo carboxilo (fusionado con el aminoácido) a continuación). Por lo tanto, la unidad de repetición básica que aparece en la estructura de la proteína es -NH-C=-CO-. Conocer la estructura primaria de una proteína no solo es importante para entender su función (ya que ésta depende de la secuencia de aminoácidos y de la forma que adopte), sino también en el estudio de enfermedades genéticas. Es posible que el origen de una enfermedad genética radique en una secuencia anormal. Esta anomalía, si es severa, podría resultar en que la función de la proteína no se ejecute de manera adecuada o, incluso, en que no se ejecute en lo absoluto. La secuencia de aminoácidos está especificada por la secuencia de nucleótidos en el ADN. Existe un sistema de conversión, llamado código genético, que puede utilizarse para inferir el primer código del segundo sistema de código. Sin embargo, ciertas modificaciones en el proceso de transcripción del ADN no siempre permiten esta conversión directamente. Generalmente, la cantidad de aminoácidos que componen una proteína está entre 80 y 300. Los enlaces implicados en la estructura primaria de las proteínas son enlaces covalentes: son enlaces peptídicos.

Estructura secundaria

La estructura secundaria de las proteínas es el plegamiento que la cadena polipeptídica adopta gracias a la formación de puentes de hidrógeno entre los átomos que forman el enlace peptídico. Los puentes de hidrógeno se establecen entre los grupos -CO- y -NH- del enlace peptídico (el primero como aceptor de H, y el segundo como donador de H). De esta forma, la cadena polipeptídica es capaz de adoptar conformaciones de menor energía libre, y por tanto, más estables. Podemos distinguir las siguientes:

- **Hélice-α.** Esta estructura se mantiene gracias a los enlaces de hidrógeno intracatenarios formados entre el grupo -NH de un enlace peptídico y el grupo -C=O del cuarto aminoácido que le sigue. Las cadenas laterales de los aminoácidos se sitúan en la parte externa del helicoide, lo que evita problemas de impedimentos estéricos. En consecuencia, esta estructura puede albergar a cualquier aminoácido, a excepción de la prolina, cuyo C_α no tiene libertad de giro, por

estar integrado en un heterociclo. Por este motivo, la prolina suele determinar una interrupción en la conformación en hélice-α. Los aminoácidos muy polares (LYS, GLU) también desestabilizan la hélice a porque los enlaces de hidrógeno pierden importancia frente a las interacciones electrostáticas. Por este motivo, la estructura en hélice-α es la que predomina a valores de pH en los que los grupos ionizables no están cargados.

- **Láminas-β.** Cuando la cadena principal de un polipéptido se estira al máximo que le permiten sus enlaces covalentes, entonces se adopta una configuración espacial denominada estructura β. En esta estructura las cadenas laterales de los aminoácidos se sitúan de forma alternante a la derecha y a la izquierda del esqueleto de la cadena polipeptídica. Las estructuras β de distintas cadenas polipeptídicas o bien las estructuras β de distintas zonas de una misma cadena polipeptídica pueden interactuar entre sí mediante puentes de hidrógeno, dando lugar a estructuras laminares llamadas por su forma hojas plegadas u hojas β. Cuando las estructuras β tienen el mismo sentido, la hoja β resultante es paralela, y si las estructuras β tienen sentidos opuestos, la hoja plegada resultante es antiparalela.
- **Giros-β.** Son secuencias de la cadena polipeptídica con estructura alfa o beta. A menudo están conectadas entre sí por medio de los llamados giros beta. Son secuencias cortas, con una conformación característica que impone un brusco giro de 180 a la cadena principal de un polipéptido.

Estructura terciaria

Se llama estructura terciaria a la disposición tridimensional de todos los átomos que componen la proteína. La estructura terciaria de una proteína es la responsable directa de sus propiedades biológicas, ya que la disposición espacial de los distintos grupos funcionales determina su interacción con los diversos ligandos. Para las proteínas que constan de una sola cadena polipeptídica (carecen de estructura cuaternaria), la estructura terciaria es la máxima información estructural que se puede obtener. Se pueden distinguir dos tipos de estructura terciaria. Las de tipo fibroso tienen estructuras secundarias que pueden mantener su ordenamiento sin recurrir a grandes modificaciones, solo introduciendo ligeras torsiones longitudinales, como en las hebras de una cuerda. En las de tipo globular no existe una dimensión predominante sobre las demás y su forma es prácticamente esférica. Aquí las cadenas se suceden con estructuras secundarias sin un patrón particular.
Las fuerzas que estabilizan la estructura terciaria de una proteína se establecen entre las cadenas laterales de los aminoácidos que la componen Los enlaces propios de la estructura terciaria pueden ser covalentes y no covalentes.
Existen diferentes regiones en la estructura terciaria de las proteínas, que pueden usarse como unidades autónomas para el plegamiento y / o desnaturalización de proteínas. Estas regiones constituyen el nivel de estructura intermedio entre la estructura secundaria y terciaria, llamadas dominios. Cuando se sintetiza una cadena polipeptídica, los dominios se pliegan por separado. Por lo que produce la estructura terciaria en última instancia es la asociación de diferentes dominios.

Estructura cuaternaria

Cuando una proteína consta de más de una cadena polipeptídica, es decir, cuando es una proteína oligomérica, decimos que tiene estructura cuaternaria. Esta debe considerar el número y la naturaleza de las distintas subunidades o monómeros que integran el oligómero y la forma en que se asocian en el espacio para dar lugar al oligómero.
La estructura cuaternaria se deriva de la combinación de varias cadenas de péptidos, estas cadenas de péptidos se asocian con un multímero cuyas propiedades son diferentes de sus monómeros constituyentes. Las subunidades están asociadas entre sí a través de interacciones no covalentes como enlaces de hidrógeno, interacciones hidrofóbicas o puentes salinos. Si la proteína está compuesta por dos monómeros, el dímero (si los monómeros constituyentes son los mismos) puede ser un homodímero, de lo contrario puede ser un heterodímero. En cuanto a los enlaces covalentes, también pueden existir enlaces disulfuro entre residuos de CYS ubicados en diferentes cadenas. Cuando varias proteínas con estructura terciaria esférica se asocian para formar una estructura cuaternaria, el monómero puede ser exactamente la misma (hexoquinasa), muy parecida (lactato deshidrogenasa), estructuras diferentes pero misma función (hemoglobina) o diferentes en cuanto a estructura y función (aspartato transcarbamilasa). La estructura cuaternaria regula la actividad biológica de la proteína y la separación de subunidades suele dar como resultado la pérdida de funcionalidad. La fuerza para mantener juntas diferentes cadenas polipeptídicas es aproximadamente la misma que la fuerza para estabilizar la estructura terciaria. Las más abundantes son las interacciones débiles (enlaces hidrófobos, polares, electrostáticos y de hidrógeno), aunque en algunos casos, como en las inmunoglobulinas, la estructura cuaternaria se mantiene mediante enlaces disulfuro. El ensamblaje de los monómeros se realiza de forma espontánea, lo que indica que los oligómeros tienen menor energía libre en relación con los monómeros.

1.5.3. Virología

Hace unos cien años, D'Arcy Thompson formuló su interpretación en términos de diagramas de fuerza de la estructura biológica en el sentido de que la forma final que tendrá la estructura es resultado de todas las fuerzas que están actuando sobre ella a través de la morfogénesis (Thompson et al., 1942). Los virus están asociados a toda forma de vida en el planeta y albergan diversidad en términos de organización genética y estrategias de replicación, y se pueden clasificar en términos generales según sus propiedades físicas y biológicas. Por ejemplo, grupos grandes de virus comparten características estructurales generales como simetría, forma, tamaño, la cápside proteica o su forma de replicación. Los virus son nano-objetos con una fenología física extensa, cuyo estudio incluye interacciones fundamentales en la nanoescala, detalles de comportamiento del polímero en sistemas confinados y restringidos, cinética y termodinámica de autoensamblaje de un sistema casi bidimensional, efectos cooperativos en el coensamblaje de ácidos nucleicos y proteínas, y propiedades geométricas y topológicas sorprendentes de cápsides que requieren descripciones de matemática discreta o diferencial.

El tratamiento de las fuerzas involucradas en el crecimiento del virus es particularmente desafiante, debido a la amplitud de las escalas espacio-temporales y fuerzas de interacciones que caracterizan el ensamblaje de un virus. Específicamente, en la escala espacial habitada por la mayor parte de los virus, existe una confluencia de energías características asociadas con una variedad de interacciones incluyendo electrostática, térmica, mecánica, etc. (French et al., 2010). Estas interacciones, por virtud de sus diferentes magnitudes a otras escalas espaciales, generalmente son tratadas dentro de campos separados de las ciencias físicas. Sin embargo, a nanoescala, las interacciones mecánicas, electrostáticas y térmicas, en incluso algunos enlaces químicos biológicamente importantes tienen magnitudes comparables (Phillips y Quake, 2006). Existe entonces un cruce entre los diferentes grados de libertad que, a cualquier otra escala, podrían ser despreciado, pero no a la escala habitada por los virus.
La presente reseña sobre virología está basada en la recopilación hecha por Zandi et al. (Zandi et al., 2020).

1.5.3.1. Estructura viral

Las partículas virales viene en diferentes tamaños y formas y varían mucho en cuanto a cantidad y naturaleza de las moléculas de las que están hechos. El tamaño y la forma particular de una cápside se encuentra en dependencia con muchas variables físicas como el pH, fuerza iónica, presencia de cationes multivalente, presencia de genoma viral y concentración de proteínas.

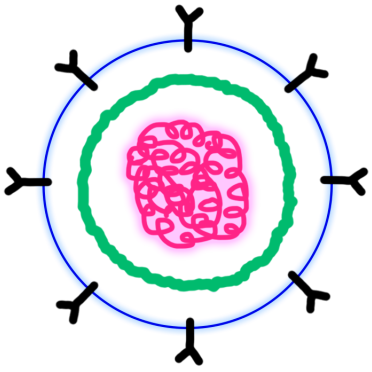


Figura 1.11: Representación genérica de los componentes principales de un virus.

La figura 1.11 representa un esquema de los componentes mayores de un virus. Podemos distinguir el ácido nucleico (rojo), la cápside proteica estructural (verde), en el caso de los virus envueltos podemos distinguir. una membrana lipídica (azul) con incrustaciones de proteínas (negro) usadas para reconocimiento de huésped. Los virus no envueltos no cuentan con esta membrana lipídica. Este es el caso de la mayor parte de los virus icosaédricos monocatenarios de ARN, la clase de virus más abundante en el mundo (Flint et al., 2003). Un grupo aparte dentro de los virus

ilustra un paradigma único en ensamblaje molecular. Estos son los virus que protegen, transportan y entregan su material genético con la ayuda de una caja llamada cápside, construida como una proteína asimétrica con una superficie cerrada, en encapsula los ácidos nucleicos y encimas. Curiosamente, la simetría icosaédrica prevalece entre los virus isométricos (Caspar y Klug, 1962), desde el circovirus más pequeño, de 17 nm de diámetro, hasta el PpV01 de 220 nm de diámetro, un virus de algas marinas y un verdadero gigante en comparación.

1.5.3.2. La construcción de Caspar-Klung

Las cápsides de virus pequeños con material genético relativamente corto en cuanto a su cadena, están compuestas de un número grande de subunidades proteicas idénticas arregladas en forma de hélices o de poliedros regulares (Watson y Crick, 1953). El hecho de que estén presentes en redundancia en forma de copias, es evolutivamente favorable, dado que implica que el genoma solo tiene que codificar un pequeño número de genes diferentes, lo que mantiene el tamaño y costo energético del ensamble del virus, pequeño. Los poliedros regulares tienen simetría icosaédrica dado que un icosaedro tiene el mayor cociente volumen/área de entre todos los poliedros regulares. En términos generales, la mayor asunción en el modelo de Caspar-Klug es la tendencia de los sistemas compuestos de subunidades en interacción de formar el número máximo de enlaces estables entre ellos. Consecuentemente, en una planilla hexagonal bidimensional donde cada sitio tiene seis vecinos cercanos, las subunidades proteicas o capsómeros más cercanos se ubican en posiciones compatibles con la asimetría de la proteína. Plegar esta planilla plana en una superficie cerrada, topológicamente equivalente a una esfera, conduce directamente a su mapeo en una superficie de icosaedro cortando exactamente 12 sectores de 60º de la hoja 2D inicial y cosiendo la malla 2D restante, mientras se conserva exactamente el patrón de enlace de todas las subunidades de proteínas en la red, como se ve en la figura 1.12.

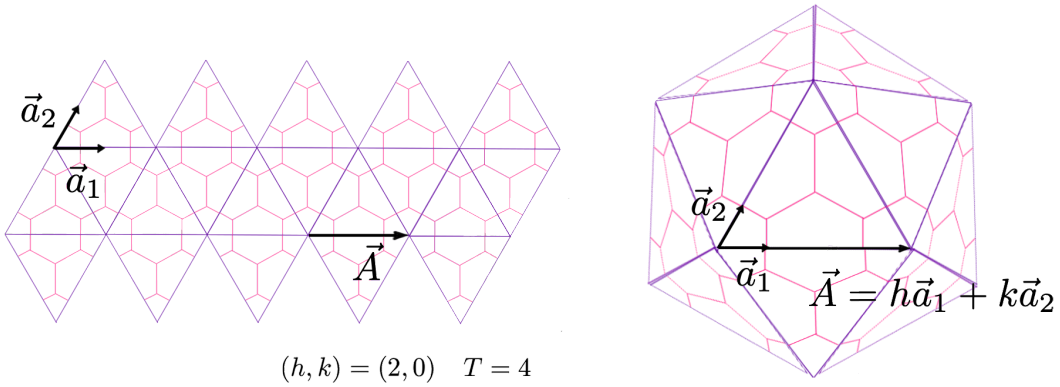


Figura 1.12: Estructura de Caspar-Klung para $T = 4$

Este proceso genera 12 vértices quíntuples a partir de la rejilla original hexagonal, caracterizado por tener cinco vecinos en lugar de los seis originales.
Esta construcción geométrica lleva a una carcasa icosaédrica compuesta de exactamente 60 subunidades, 5 por cada vértice quíntuple, distribuidos en posiciones completamente equivalentes por simetría y preservando el patrón de enlace de la planilla hexagonal 2D. Como muchos virus simples están compuestos de un número mucho más grande de subunidades, Caspar y Klung idearon una extensión de su modelo introduciendo la noción de cuasi equivalencia. Esto puede lograrse cortando secciones triangulares que no preservarían la simetría hexagonal de la plantilla, pero que al coser la malla en este caso no preservaría el patrón de enlace de las subunidades proteicas en la malla 2d de manera exacta, a diferencia del caso anterior. Los sitios de las subunidades de proteínas en la estructura icosaédrica serían en este caso solo cuasi equivalentes, ya que no preservarían la simetría de la red 2D original, pero de esta manera permitirían que la capa se componga de más de 60 subunidades. Este procedimiento implica un número variable de hexámeros que conectan los 12 vértices quíntuples cuantificados por el camino a lo largo de los vectores unitarios $\mathbf{a}_1$, $\mathbf{a}_2$ del retículo 2D que va de un vértice a otro vecino. Cuando esta ruta se puede cuantificar como $h\mathbf{a}_1 + k\mathbf{a}_2$, con h, k enteros no negativos, entonces el número total de subunidades de proteínas en las posiciones cuasi equivalentes sería $60T$, donde

$$T = h^2 + hk + k^2 \tag{1.135}$$

Aquí, T es el número de triangulación del virus y h, k miden el número de pasos en la dirección de los dos vectores unitarios de la rejilla hexagonal que conecta un vértice quíntuple con otro: para llegar de una estructura simétrica quíntuple a otra, es necesario moverse sobre h estructuras simétricas séxtuples a lo largo de una fila de enlaces cercanos contiguos en la rejilla hexagonal, entonces girar 120^{o} y moverse otros k pasos simétricos séxtuples, como en la figura 1.12. Si el icosaedro tiene un número de triangulación más alto, incluso si todas las proteínas son químicamente idénticas, algunos estarán en un ambiente de 5 vecinos (pentámero) y otros estarán en un ambiente de 6 vecinos (hexámeros). Por lo tanto, la posición de cada proteína no es equivalentes las demás, pero por necesidad solo cuasi equivalente. El número de triangulación mide el número de posiciones cuasi equivalentes en la cápside viral. Los virus con diferente número de triangulación difieren en el número de proteínas como se muestra en la figura 1.13, y a medida que aumenta su número de triangulación T, su tamaño también va en aumento.

Una generalización basaba en la teoría de Caspar-Klug fue propuesta para cápsides no isométricas, como el bacteriófago T4 y el virus del VIH, por Nguyen, Bruinsma y Gelbart (Martínez Moncayo, 2016). Además de (h, k), se necesitan dos números enteros más para la construcción de un caparazón esferocilíndrico. La forma de un esferocilindro se determina entonces por dos pares de vectores base $(\mathbf{a}_1, \mathbf{a}_2)$ y $(\mathbf{b}_1, \mathbf{b}_2)$ y dos pares de números enteros (h, k) y (n, m) tales que $A = n(h\mathbf{b}_1 + k\mathbf{b}_2)$ y $B = m(h\mathbf{b}_1 + k\mathbf{b}_2)$.

Aunque muchos tipos de virus pueden describirse mediante la construcción geométrica de Caspar-Klug o sus versiones generalizadas, no todos los virus isométricos se ajustan a los principios de cuasi equivalencia. Para poner la clasificación y los

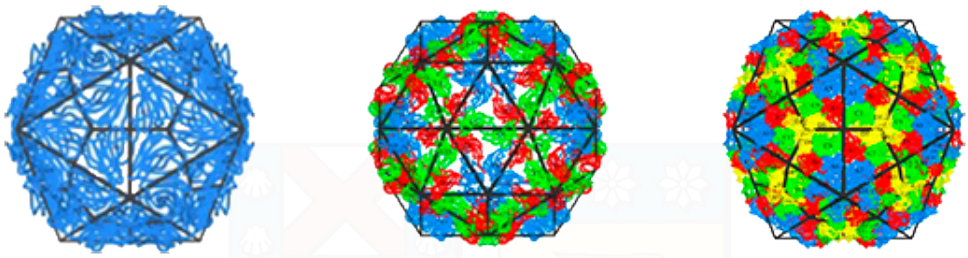


Figura 1.13: Virus caracterizados por diferentes número de triangulación. De izquierda a derecha: virus del mosaico del tabaco (PDB ID: 1A34, $T = 1$), virus del mosaico clorótico de la habichuela (PDB ID: 1IHM, $T = 3$) y omegatetravirus (PDB ID: 1ZA7, $T = 4$). Imágenes fueron obtenidas con el software VIPER en la página http://viperdb.scripps.edu/.

principios estructurales del diseño viral sobre una base más amplia, Lorman y Rochal (Lorman y Rochal, 2007) (Rochal et al., 2016) idearon un enfoque más completo basado en la noción fundamental de que las subunidades de proteínas y la cadena de ácido nucleico se unen espontáneamente para formar un partícula de virus simple porque este es su estado de energía más bajo, es decir, la estructura del virus es el resultado de la minimización de energía (libre). En este enfoque, la construcción geométrica de Caspar-Klug aparece como uno de los posibles estados de simetría, mientras que varios otros, que se sabe que existen en el zoológico de virus y que no son susceptibles de la construcción de Caspar-Klug, han obtenido en este marco generalizado una justificación definitiva para su existencia.

2 Metodología

2.1. Paralelización de PyGBe

PyGBe es un software de código abierto escrito en Python, desarrollado por el profesor Christopher Cooper (Cooper et al., 2016), que aplica el método de elementos de borde para cálculos de electrostática molecular en un modelo de medio continuo, con el fin de obtener valores de energías de solvatación para proteínas modeladas con un número arbitrario de regiones dieléctricas. Originalmente, PyGBe cuenta con aceleración algorítmica a través de treecode (Barnes y Hut, 1986) para acelerar cada iteración de un solver GMRES desde un orden $\mathcal{O}(n^2)$ al orden $\mathcal{O}(n \log n)$ y con aceleración de hardware que le permite utilizar GPU en las partes más intensivas de computación a través del uso del lenguaje CUDA en treecode. Algunas partes del código fueron escritas en C++, empaquetado y linkeado con SWIG.
Desafortunadamente, si la estación de trabajo donde se desea ejecutar el código no cuenta con una tarjeta gráfica potente, o la memoria necesaria, se hace imposible ejecutar el código para biomoléculas de mayor tamaño como lo es un virus. Más aún, si no se cuenta con GPU, el rendimiento de PyGBe puede decaer puesto que no está optimizado para correr en CPU como lo está para su uso en tarjetas gráficas. En este trabajo, hemos tomado el código existente de PyGBe y lo hemos modificado para obtener una versión paralela a nivel de memoria compartida.
El código original de PyGBe ha sido escrito en Python pero dado que está destinado a ser un solver rápido y eficiente, y Python es un lenguaje interpretado y de alto nivel, no logra alcanzar por si solo las velocidades requeridas y es necesario introducir algunas funciones especiales escritas en C++. La ventaja de esta práctica viene del hecho de que C++ es un lenguaje compilado y, por lo tanto, se ejecuta de manera más rápida en comparación a su contraparte interpretada.
Para poder llamar las funciones escritas en C++ desde Python, originalmente se utilizó el código SWIG (Simplified Wrapper and Interface Generator) (Beazley et al., 1995) que través de un archivo de interfaz, es capaz de empaquetar una función escrita y compilada en C++ y enlazarla con una nueva función compatible con Python. Por lo tanto, la paralelización debe ser llevada a cabo a nivel de las funciones de C++, que son las que tienen el grueso del cómputo. Para la paralelización de estos procesos, hemos optado por hacerlo a nivel de memoria compartida usando la API OpenMP (Dagum y Menon, 1998). Antes de poder hacer esto, es necesario considerar que

- SWIG es incapaz de procesar funciones paralelizadas y enlazarlas con Python, por lo tanto, es necesario cambiar la manera en la que este proceso se realiza.

- La forma en que el código de Python llama a las funciones de C++ puede ser ineficiente cuando las últimas han sido paralelizadas.

Para resolver el problema que se tiene con SWIG, hemos optado por utilizar en reemplazo a Cython (Behnel et al., 2011), una extensión del lenguaje de Python que permite hacer declaraciones de tipo explícito y se compila directamente en C. Fue desarrollada por Behnel y su equipo. Este código es capaz de hacer la labor que hasta ahora estaba haciendo SWIG, de compilar funciones en C++ y llamarlas desde Python, pero con la salvedad de que Cython sí permite utilizar funciones paralelizadas. Para poder hacer estos cambios, fue necesario modificar los archivos de interfaz para que sean compatibles con Cython. Originalmente, SWIG utiliza un archivo de extensión `*.i` que debieron ser cambiados por dos archivos, uno de extensión `*.h`, esto es, un header de C++ que contiene a las funciones a exportar y otro de extensión `*.pyx` que hace las veces de archivo de interfaz, definiendo nuevas funciones de Python, cuyo trabajo es pasar los inputs a las funciones de C++ para que estas se ejecuten como siempre y puedan devolver un resultado. Este resultado será el que las funciones de Python podrán tomar para poder devolver a su vez. Como muchas veces la labor de estas funciones, más que devolver un output, era la modificación de los inputs, se tuvo especial cuidado en proveer, desde Python, arreglos que fueran *ctype*, esto es, que valores contiguos del arreglo ocupen espacios contiguos de memoria también, debido a que C++ entiende un arreglo a través de la posición del elemento inicial y tamaño, a diferencia de Python que presenta mayor flexibilidad en cuanto a los espacios de memoria asignados a los arreglos.
Respecto al problema de ineficiencia a la hora de llamar a las funciones de C++, en el código de Python hay varios ciclos iterativos que llaman a las funciones de C desde dentro del mismo ciclo, es decir, la función es llamada múltiples veces. El problema con esta forma de llamar a la función es que ahora que es paralela, se necesita hacer el espacio en memoria para los múltiples arreglos que serán distribuídos entre los diferentes threads por cada vez que sea llamada la función. Las veces que esta función es llamada depende directamente del tamaño de la biomolécula en estudio, y para nuestros casos, el tamaño de los arreglos es grande y lleva a un elevado e innecesario *overhead* que puede solucionarse cambiando las funciones para que en vez de ocurrir en la parte del código de Python, ocurra dentro de la función de C++, de esta forma, los espacios de memoria no necesitan asignarse y desasignarse una vez tras otras y el código se vuelve más eficiente.

2.1.1. Procesos a paralelizar

En PyGBe existen cinco procesos que por naturaleza son altamente paralelizables. Estos procesos son los que toman más tiempo de cálculo y por lo tanto, la paralelización de estos segmentos del código fueron capaces de disminuir el tiempo total de cómputo de forma dramática. Estos cinco procesos son el **cálculo del Right Hand Side**, **cálculo de P2P**, **cálculo de M2P**, **cálculo de energía de solvatación** y **cálculo de energía de Coulomb**. Todos estos procesos tienen sentido dentro del contexto de treecode.

2.1.1.1. Cálculo del RHS

El cálculo del RHS (*right-hand side*) corresponde a la generación del lado derecho del sistema de ecuaciones lineales que necesita ser resuelto por el algoritmo iterativo GMRES. Originalmente en PyGBe, este cálculo se realizaba en la sección de Python, por lo que fue necesario escribir esta función en Cython y enlazarla al código de Python antes de poder paralelizarla. De esta forma introducimos los archivos `rhs.h` y `rhs.pyx` que contienen una nueva función que cumple con esta labor.
En el cálculo del RHS, por cada elemento de superficie i en cada unas de las superficies encerradas por una región y por cada carga j en la región, es necesario obtener los resultados

$$\frac{q_j}{\epsilon|\mathbf{R}_{p_i q_j}|^3} \tag{2.1}$$

donde $\mathbf{R}_{p_i q_j}$ es la distancia entre el centro del elemento de superficie i y la posición de la carga q_j, $\mathbf{n}_i$ es el vector normal al elemento de superficie y ϵ es la permitividad del medio. De esta forma, se vuelve necesario establecer dos ciclos iterativos anidados. Hemos paralelizado el proceso agregando una clausula `#pragma omp for` antes del ciclo exterior. Dependiendo del caso, existirán más o menos elementos de superficie que cargas en la región. Considerando la cantidad de cargas átomos que tiene un virus y los tamaños de malla que usaremos, hemos anidado el ciclo iterativo sobre los elementos de superficie dentro del ciclo iterativo de cargas, de manera que así cada ciclo thread implicado tener más trabajo y no se produzcan desbalances de carga. De esta forma el arreglo de cargas es fragmentado y cada uno de estos fragmentos va a distintos procesadores para que cada uno de ellos haga los cálculos considerando todos los elementos de superficie i. Al terminar, los threads se sincronizan y devuelven los resultados.

2.1.1.2. Cálculo del P2P

Por cada iteración del algoritmo GMRES, la parte más costosa en términos de recursos de cómputo es la multiplicación matriz vector que requiere del cálculo de los potenciales de capa singular y doble. Para esto, en la función `project` de PyGBe, se lleva a cabo este cálculo a través del algoritmo treecode. El cálculo de P2P o *particle to particle* de treecode se refiere al cálculo de la interacción directa entre dos partículas que están lo suficientemente cerca de acuerdo al parámetro de θ elegido como para que la interacción no pueda ser aproximada indirectamente a través de M2P. En este contexto, cuando hablamos de la masa de estas partículas nos referimos al producto entre los pesos de Gauss y el vector en la multiplicación matriz-vector de GMRES. El cálculo de P2P dentro de treecode es hecho en última instancia por la función `P2P_sort` que cuenta, a nivel de programación, con cuatro ciclos iterativos anidados. PyGBe ordena los distintos arreglos de *targets* de manera que aquellos que están en la misma caja de tamaño mínimo o *twig* queden en espacios de memoria contiguos. De esta forma, el primer ciclo anidado itera sobre los distintos *twigs*. Paralelizar el código a nivel de este ciclo puede generar bastante *overhead* debido a que los *twigs* tienen diferentes tamaños y al sincronizarse, los procesadores que terminen primero tendrán que esperar a que termine el último incurriendo así en un desbalance de

carga. El segundo ciclo itera sobre cada *target* en el *twig* abordado. Hemos añadido la cláusula `#pragma omp parallel for` en este ciclo, habiendo dos ciclos anidados dentro de este, como se puede visualizar en el algoritmo 1. Es necesario indicar que no hemos utilizado la cláusula de OMP `collapse` debido a que esta solo se puede utilizar cuando no existe ningún comando entre ciclos, no siendo este el caso, pues al entrar en el ciclo de `targets` es necesario definir variables antes de poder entrar al siguiente en el orden jerárquico.

Algorithm 1: Pseudocódigo de P2P

```
for twig do
    #pragma omp parallel for;
    for target do
        for sources_interacition_list do
            for source do
                calcular y sumar contribución;
            end
        end
    end
end
```

2.1.1.3. Cálculo del M2P

El cálculo de M2P o *multipole to particle* de treecode se refiere al cálculo de la interacción indirecta entre una partícula que está lo suficientemente lejos de de un conjunto de otras como para que sea adecuado aproximar este conjunto de partículas como una caja y sus multipolos; según el criterio tomado (parámetro θ). El cálculo de M2P dentro de treecode es hecho en última instancia por la función `M2P_sort` que cuenta, a nivel de programación, con tres ciclos iterativos anidados. El ciclo más externo itera sobre los distintos *targets*, de modo que en ciclos anidados internos se pueda iterar sobre los distintos multipolos con los que tienen que interactuar. Esta lista de interacción ya ha sido precalculada durante la formación del árbol de treecode. Situamos la cláusula `#pragma omp parallel for` en el ciclo más externo como se ve en el Algoritmo 2.

Algorithm 2: Pseudocódigo de M2P

```
#pragma omp parallel for;
for targets do
    for twig do
        for multipoles do
            calcular y sumar contribución;
        end
    end
end
```

2.1.1.4. Cálculo del energía de solvatación

El cálculo de la energía de solvatación es llevado a cabo por la función `calculate_solvation_energy` que depende de su proceso más costoso computacionalmente: el cálculo del potencial de reacción. Cabe destacar que en el cálculo de energía de solvatación PyGBe solo considera interacciones directas. Para esto, cuenta con la función `direct_c` para realizar el cálculo. Agregamos la cláusula `#pragma omp parallel for` en el ciclo más externo. Para este caso, una vez que los threads han finalizado sus cálculos, todos van a concurrir a sumar sobre el mismo espacio de memoria. Si entre el tiempo en que un thread toma el valor actual de la variable y le agrega el resultado de su cálculo, otro thread quiere hacer lo mismo, se produce una condición de carrera o *race condition*, un problema de sincronización común en computación paralela. Para prevenir que este fenómeno ocurra, usamos la cláusula `omp atomic` para que cuando un thread quiera ocupar un espacio de memoria, las otras no puedan acceder a él. Este comando de sincronización evita llegar a resultados incorrectos a costa de un menor desempeño.

2.1.1.5. Cálculo del energía de Coulomb

Probablemente el caso más simple de paralelizar ha sido el de las energías de Coulomb. Esta energía es calculada por la función `coulomb_energy`, esta a su vez está basada en el cálculo de interacción directa de Coulomb a través de la función `coulomb_direct`. Aquí hemos paralelizado en el ciclo iterativo más externo puesto que todos los ciclos internos son simples y simétricos.

2.2. Parametrización y mallado de estructuras

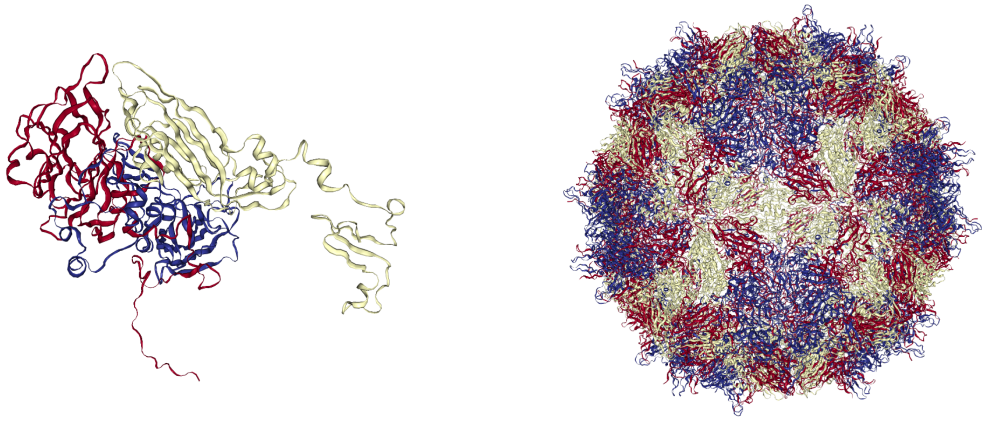
2.2.1. Parametrización

En el contexto de estructuras biomoleculares, el uso del formato PDB (Protein Data Bank) es el más extenso. Este formato provee una representación es estructuras macromoleculares derivada de la difracción de rayos X y estudios de resonancia magnética nuclear y contiene información sobre el tipo de átomo, el residuo al que pertenece, la cadena en la que está, posición en el espacio, entre otros [2]. Para obtener las estructuras moleculares con las que hemos trabajado en esta tesis, hemos recurrido a RCSB PDB (Research Collabortory for Structural Bioinformatics Protein Data Bank), portal que provee de información y estructuras biomoleculares (proteínas, ADN, ARN) 3D descargables en formato `*.pdb`. Cuando hablamos de cápsides virales, en este portal, podemos encontrar diversas formas de descargar las estructuras debido a las distintas simetrías con las que cuentan estructuralmente, teniendo a manera de ejemplo el hexámero, pentámero, la estructura icosaédrica completa, y por supuesto la estructura más básica que es la unidad asimétrica o capsómero. Estas estructuras no contienen el estado de protonación (átomos de hidrógeno) y muchas veces una gran cantidad de átomos pesados están perdidos. Nosotros estamos interesados en hacer

[2] Documentación del formato PDB: http://www.wwpdb.org/documentation/file-format

cálculos de electrostática y por lo tanto necesitamos asignarle una carga a cada uno de los átomos involucrados en estas estructuras. El formato `*.pdb` no contiene esta información y necesitamos asumir un campo de fuerza para poder obtener las cargas de los átomos. En el ámbito de las simulaciones de dinámica molecular de proteínas, el término *campo de fuerza* se refiere a la combinación de fórmulas matemáticas y parámetros asociados que describen la energía en la proteína como función de sus coordenadas atómicas (Guvench y MacKerell, 2008).

Utilizamos el código `pdb2pqr` (Dolinsky et al., 2004) desarrollado por Dolinsky *et al.* para procesar los archivos PDB (Squires et al., 2013) obtenidos del Protein Data Bank. El primer paso en el algoritmo de `pdb2pqr` consiste en la identificación de problemas potenciales con la estructura molecular inicial. Los átomos pesados faltantes son identificados y se determina si es posible reconstruir estos residuos incompletos (si es que no falta más del 10 % de los átomos pesados). Si la reconstrucción es posible, usa la topología estándar de los aminoácidos para determinar la posición de los átomos faltantes. Además, existe la opción de desmontar los átomos reconstruidos y así asegurar que no están siendo puestos dentro del radio de Van de Waals de otros átomos. Posteriormente `pdb2pqr` añade hidrógenos a la estructura molecular, optimizando la red de puentes de hidrógenos global. Por defecto, los átomos de hidrógeno deben pasar por el filtro de desmontamiento en caso de que hayan quedado sobrepuestos entre ellos o con otros átomos pesados. Por último, se asigna las cargas atómicas y radios basados en el campo de fuerza. Este software permite añadir algunos flags en la línea de comando para poder cambiar el campo de fuerza a utilizar o seleccionar algún estado de protonación según un pH específico mediante PropKa. Para nuestros casos de estudio hemos utilizado el campo de fuerza AMBER (amber99) (Wang et al., 2000) sobre la estructura más básica y asimétrica, obteniendo como resultado un archivo de formato PQR con la posición, la carga y el radio de los átomos que componen el monómero estructural de la cápside.



(a) Unidad asimétrica del virus Triatoma **(b)** Bioensamble completo del virus Triatoma

Figura 2.1: Representación grafica del virus Triatoma. Image from the RCSB PDB (rcsb.org) of PDB ID 3nap Squires, G., Pous, J., Rozas-Dennis, G.S., Costabel, M.D., Agirre, J., Marti, G., Navaza, J., Guerin, D.M.A., Rey, F.A. The crystallographic structure of Triatoma Virus (TrV) highlights the Dicistroviridae and Picornaviridae family differences. To be published. (Squires et al., 2013)

La figura 2.1a muestra la estructura básica del virus Triatoma. Este capsómero debe ser replicado y reubicado 60 veces para poder completar el bioensamble completo, como se ve en a figura 2.1b. Para esto usamos transformaciones geométricas para rotar y trasladar las estructuras básicas y poder armar el modelo completo. En caso de requerir modificar el pH en la estructura completa, hacemos las transformaciones a nivel de PDB para poder usar `pdb2pqr` sobre la estructura completa (esto requiere mayor uso de memoria y tiempo.

2.2.2. Mallado

Una vez que tenemos las estructuras parametrizadas en formato PQR procedemos a mallarla. Nuestro objetivo es poder obtener una superficie SES, como la que se describe en la sección 1.1.3, con una molécula de agua de diámetro 1,4 de modo que pueda rodar por fuera de la macromolécula y definir la superficie por la no es capaz de atravesar la estructura. Existen muchos códigos que permiten encontrar esta superficie y generalmente tienen buen desempeño a costa de perder flexibilidad. El software cuyo uso es el más extenso por su eficiencia para obtener la SES a partir de la estructura PQR es MSMS (Sanner et al., 1996). En este paquete, la SES es calculada en dos pasos. Primero, se construye la superficie reducida, que es como el esqueleto de la malla. Luego, después que cada parche de la superficie es identificado, se ejecuta la triangulación. Aunque en un principio quisimos utilizar este código para la generación de las mallas, su uso de memoria sobrepasaba nuestros recursos en los casos de estudio más grandes.

Hemos optado por usar el código `Nanoshaper` (Decherchi y Rocchia, 2013). Este software es capaz de construir y triangular la superficie molecular de sistemas nanométricos de acuerdo a varias definiciones existentes, calcular estimaciones de superficies y volúmenes, y detectar cavidades internas computando su volumen, de manera que se pueda establecer un una cota por debajo de la cual todos los tamaños de cavidades sean despreciables. Lo primero que `Nanoshaper` hace es calcular la forma de la superficie, analíticamente si es posible. En segundo lugar, se encierra la estructura en una caja cúbica y se emiten rayos consistentes con la malla, se ubican las intersecciones con la superficie y se estima el volumen encerrado. En tercer lugar se identifican las cavidades internas y se eliminan de ser necesario. Posteriormente la superficie es triangulada consistentemente con la remoción anterior de cavidades y el área previamente calculada hasta que, finalmente, se proyectan los puntos de la malla en la superficie. `Nanoshaper` admite un archivo de configuración que define el tamaño de la malla, el formato de salida de la malla, el tipo de superficie, la opción de detectar y *llenar de solvente* o ignorar cavidades internas, modificar y el tamaño de la molécula de prueba para la generación de la SES. En general, usamos el formato de malla compatible con `MSMS` que consiste en dos archivos por superficie. El primero contiene la información de todos los vértices de los elementos de la malla y su posición, con la extensión `*.vert`. El segundo tiene la información de los elementos triangulares y de cuáles son los vértices que los componen, con la extensión *.face. Dependiendo de los parámetros utilizados para la generación de la malla se puede volver necesario aplicar un tratamiento a ella para eliminar partes inconexas

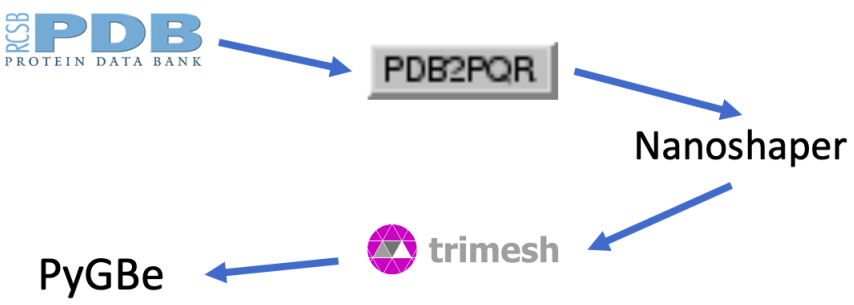


Figura 2.2: Diagrama de flujo del proceso llevado a cabo desde los archivos obtenidos en el Protein Data Bank hasta que están listos para ser utilizados por PyGBe.

(bolsillos, cavidades internas) o para separar una parte interna de otra externa, como ocurre en el caso de aplicar la generación de la SES a una cápside cerrada e hidrofóbica.

La librería Trimesh [3] nos permite tomar una superficie y dividirla según sus partes inconexas. De esta forma, podemos separar la SES de las cavidades que pudieron haber quedado dentro e incluso separa la SES en dos o más partes, dependiendo de lo que se busque. Trimesh toma como input archivos de formato OFF (Object File Format), un formato general para polígonos 3D. Para poder pasar de formato MSMS a OFF simplemente unimos los archivos `*.vert` y `*.face` en un nuevo archivo *.off y hacemos correr el contador de los vértices desde 0, puesto que en `MSMS` el contador parte desde 1. Una vez separadas las mallas, se exporta el resultado en formato `*.off` que posteriormente es separado para volver a obtener los archivos `*.vert` y `*.face`. La figura 2.2 muestra un diagrama de flujo del proceso descrito.

2.3. Gradientes de pH

En el terreno de la química nos encontramos con la expresión pK_a, una medida de acidez que refleja cómo tienden las moléculas de una solución a disociarse. El pK_a describe el nivel de ionización y refleja la aceptación o donación de protones frente a un valor concreto de pH. El paquete `PROPKA`, incluido en `pdb2pqr` permite realizar una protonación acorde a un pH específico mediante la predicción del pK_a (Olsson et al., 2011). En la práctica, esto es posible usando las opciones `-ph-calc-method=propka` (para indicar al software que se desea utilizar `PROPKA`) y `-with-ph=pH`, con un pH específico. Al utilizar estas opciones, `pdb2pqr` genera los archivos PQR a partir de los archivos PDB, pero al agregar los átomos de hidrógeno lo hace de acuerdo a la predicción de `PROPKA`.
Nuestro objetivo en esta sección es poder generar estructuras PQR que tengan un estado de protonación diferente en la parte interna y la parte externa de la cápside viral para poder simular un gradiente de pH en el virus. Para eso, generamos dos ficheros PQR diferentes a partir de la misma estructura PDB pero usando distintos valores de pH en el proceso. Así obtenemos uno con un estado de protonación acorde

[3] Trimesh, by Dawson-Haggerty et al. https://trimsh.org/

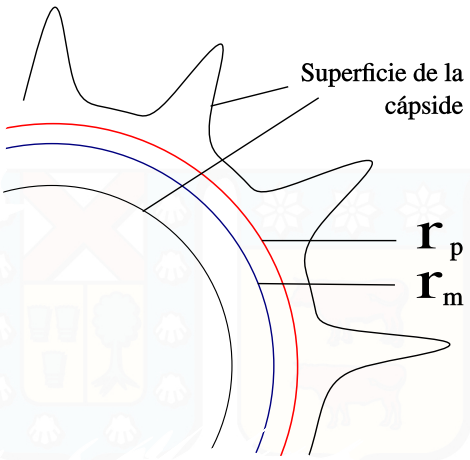


Figura 2.3: Representación de la cápside de un virus con extensiones en su estructura. La línea roja representa la circunferencia de radio promedio r_p y la línea azul representa la del radio promedio de la estructura modificada r_m.

a $pH = pH_{int}$ y otro de acuerdo a $pH = pH_{ext}$. La estructura expuesta al gradiente de pH que queremos generar corresponde a una fusión de las dos estructuras obtenidas anteriormente. Para poder realizar esta fusión, rastreamos la posición de el centro geométrico de la cápside c_0 y calculamos su radio promedio r_p. Muchas veces las cápsides cuentan con extensiones internas o externas que se alejan de la región en que está concentrada la mayor parte de los átomos, como es en el virus Triatoma.

Esto hace que la superficie de la esfera centrada en el centro de la cápside y de radio r_p se aleje de la esfera que pasa por "la mitad" de la biomolécula, quedando desplazada hacia dentro o afuera si las extensiones son internas o externas respectivamente. La figura 2.3 ejemplifica el caso en que las extensiones son externas. Como la cápside es simétrica respecto a su centro, podemos calcular dicho centro sumando las coordenadas de todos los átomos y dividiendo por la cantidad de estos. Posteriormente calculamos la distancia entre este centro y los átomos de uno de los capsómeros. Las distancias están agrupadas excepto por unos cuantos valores que se alejan del montón; estas distancias anómalas corresponden a las correspondientes a los átomos de las crestas que presenta la cápside. Procedemos a eliminar estos puntos. Con estos átomos fuera, podemos encontrar el radio promedio de la estructura modificada r_m. Teniendo estos datos tomamos los dos archivos generados por `pdb2pqr` y los sometemos al siguiente proceso. Sea $\mathbf{r}_i$ la posición del carbono alpha del residuo i de la cápside, podemos calcular la distancia entre este y c_0 según

$$r_i = \sqrt{(\mathbf{r}_i - \mathbf{c}_0) \cdot (\mathbf{r}_i - \mathbf{c}_0)} \tag{2.2}$$

de esta manera procedemos a comparar la distancia de cada carbono alpha respecto al centro con el radio promedio de la estructura modificada r_m. Si $r_i > r_m$ se toma el residuo completo del archivo asociado a la región exterior (pH_{ext}) y si $r_i < r_m$

se toma el residuo completo del archivo asociado a la región interior (pH_{int}). Una vez teniendo todos los residuos de la estructura procedemos a listarlos en un nuevo archivos `pqr` que contiene la información del gradiente de pH a través de su estado de protonación.

3 Resultados y discusión

El modelo de solvente implícito generalmente es utilizado para cálculos de energías de solvatación y enlace y es aplicable en áreas como ingeniería de proteínas y diseño de fármacos. La mayor parte de los códigos disponibles que modelan el solvente de esta forma usan discretizaciones del medio en términos de volúmenes como `APBS` (Baker et al., 2001), `Delphi` (Gilson y Honig, 1987), `MEAD` (Bashford y Gerwert, 1992) (Bashford, 1997), `MIBPB` (Geng et al., 2007), entre otros. Existen también trabajos e investigación basados en métodos de elementos de borde como `AFMPB` (Lu et al., 2006), `TABI` (Geng y Krasny, 2013) que presentan una discretización en términos de superficies. Particularmente, tenemos el software PyGBe, desarrollado por Cooper *et al.* (Cooper et al., 2014)
Para este trabajo hemos hecho una versión paralela a nivel de memoria compartida de PyGBe y en este capítulo presentamos los resultados obtenidos en cuanto a la paralelización en sí y al simular distintos virus usando este software.
Primero nos enfocamos en el virus Zika (ZIKV), que posee una cápside de aproximadamente un millón y medio de átomos. Para este virus mostraremos qué sucede con la energía de solvatación al modificar las concentraciones de salinidad tanto dentro como fuera de la cápside viral con el fin de conocer más sobre la estabilidad de este virus. Con los ojos en este mismo virus, hemos realizado simulaciones quitando las cargas de ciertos residuos con el fin de estudiar la importancia para la estabilidad de la cápside que aporta cada residuo. Finalmente nos enfocamos en el virus Triatoma (TrV), un virus de un tamaño menor al del Zika. Es sabido que este virus se desensambla si es expuesto a ciertos gradientes de pH (Viso et al., 2018). En este trabajo presentamos resultados de energías libres al exponer la cápside del virus Triatoma a gradientes de pH con el fin de estudiar su estabilidad desde un punto de vista electrostático.

Los resultados de esta tesis han sido realizados en una `CPU` Xeon E5-2580v3 de 12 núcleos con 96GB de memoria y previamente publicados en forma parcial (Martínez et al., 2019).

3.1. Resultados de paralelización

Como se vio en la sección de metodología, para la paralelización de los distintos procesos se hizo uso de herramientas similares; pero por la naturaleza de las funciones que rigen cada uno de estos procesos se obtienen resultados diferentes de

paralelización.

La lisozima es una enzima que actúa como catalizador de reacciones químicas. Esta impide infecciones y está presente en numerosas sustancias segregadas por los seres vivos como las lágrimas, la saliva o la leche. Estructuralmente consiste en una cadena de aminoácidos que cuenta con 1957 átomos. Su, relativamente, pequeño tamaño nos ayudará a obtener resultados de *speedup* del código sin incurrir en grandes costos computacionales. Las simulaciones hechas con esta molécula cuentan con una SES que cubre la superficie externa molecular, una capa de Stern, o de exclusión iónica, y tres cavidades llenas de solvente internas; de manera que en total tiene 30170 elementos.

Cálculo del RHS

N° Threads	Tiempo RHS [s]	Speedup
1	0.1680	-
2	0.0832	2.019
3	0.0586	2.866
4	0.0445	3.775

Tabla 3.1: Resultados de tiempo y speedup para distintos números de threads en el cálculo del RHS para el caso de una lisosima.

La tabla 3.1 muestra los resultados de tiempo obtenidos al utilizar PyGBe paralelizado en la obtención del RHS para una lisozima. Es necesario considerar que el tiempo que se ha medido no solo involucra la ejecución de la función paralela sino también el procesamiento de los datos obtenido de esta, por lo que se ve una caída en el escalamiento a medida que aumenta el número de threads. Aún así, el resultado es bastante similar a lo esperado. Dado el tamaño de la lisozima, las cantidades absolutas de tiempo son muy pequeñas, pero en una estructura del tamaño en el que estamos interesados, con millones de átomos, el tiempo de cálculo de RHS puede fácilmente llegar a un día completo. Esta paralelización puede disminuir drásticamente este tiempo a solo un par de horas.

Cálculo de P2P

N° Threads	Tiempo P2P [s]	Speedup
1	530.06	-
2	289.81	1.83
3	192.66	2.75
4	150.87	3.51

Tabla 3.2: Resultados de tiempo y speedup para distintos números de threads en el cálculo del P2P para el caso de una lisozima.

La tabla 3.2 muestra el tiempo tomado por el programa en calcular la componente directa de los potenciales de capa singular y doble. Se puede apreciar que si bien el *speedup* no es perfecto, se asemeja mucho a lo esperado. Si bien hemos puesto la cláusula de paralelización en el segundo ciclo iterativo para reducir el *overhead*

que habría generado usarlo en el primer ciclo, existe un *overhead* asociado al hecho de que por cada *taget*, se accede a una lista específica de *sources* que pueden tener tamaños diferentes. Esto provoca que la sincronización de los threads no sea perfecta y por lo tanto, el escalamiento se aleje de lo esperado.

Cálculo de M2P

N° Threads	Tiempo M2P [s]	Speedup
1	11.56	-
2	6.7487	1.7
3	5.3389	2.2
4	5.0213	2.3

Tabla 3.3: Resultados de tiempo y speedup para distintos números de threads en el cálculo del M2P para el caso de una lisozima.

N° Threads	Tiempo M2P [s]	Speedup
1	11.56	-
2	5.8098	2.0
3	3.9994	2.9
4	3.0307	3.8

Tabla 3.4: Resultados de tiempo y speedup para distintos números de threads para la función `M2P_sort` aislada.

La tabla 3.3 muestra los tiempos de cómputo de todo el proceso de M2P. A primera vista podría parecer que el escalamiento no se está logrando de manera óptima, pero el proceso de M2P no solo incluye la función paralela `M2P_sort` sino que también la creación de arreglos pertinentes, y el reordenamiento de los arreglos de multipolos. Si solo analizamos el escalamiento de la función `M2P_sort` obtendremos mejores resultados. La tabla 3.4 muestra los resultados de tiempo y escalamiento solo para la función paralelizada y se puede apreciar un escalamiento bastante cercano al perfecto. Estas diferencias son debidas al *overhead* natural de la creación de los threads involucrados.

Cálculo de Energía de Solvatación

N° Threads	Tiempo Energía solvatación [s]	Speedup
1	2.1968	-
2	1.2027	1.8
3	0.8375	2.6
4	0.6975	3.1

Tabla 3.5: Resultados de tiempo y speedup para distintos números de threads en el cálculo de la energía de solvatación para el caso de una lisozima.

La tabla 3.5 muestra el tiempo tardado y escalamiento del proceso de cálculo de Energía de solvatación. Este proceso completo incluye mucho más que el cálculo hecho por la función paralelizada. Esto explica la diferencia entre el escalamiento obtenido y el escalamiento perfecto.

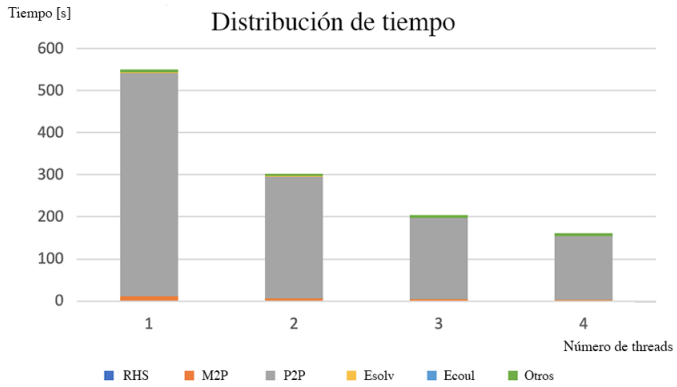


Figura 3.1: Distribución de tiempo que toman los distintos procesos para diferentes números de threads involucrados en el cálculo.

Cálculo de Energía de Coulomb

La tabla 3.6 muestra los resultados de tiempo y escalamiento para el cálculo de la energía e Coulomb. En este caso, como ocurre con el cálculo de la energía de solvatación, el proceso de cálculo de la energía no solo contiene a la función paralelizada sino a otros procesos también (creación de arreglos y reordenamiento de arreglos).

Speedup general

Si bien la mayoría de los procesos más pesados desde un punto de vista de procesamiento se han paralelizado con buenos resultados, el tiempo total resolución incluye otros procesos como la lectura de los datos desde los *inputs*, la conformación del árbol que se usará en treecode, etc.

La figura 3.1 muestra la distribución de tiempo que toman los distintos procesos involucrados para distintos número de threads. Se puede apreciar que comparativamente, el tiempo que toma el proceso P2P es gigantesco. Esto es debido al parámetro θ utilizado, que dentro del algoritmo de treecode, determina si habrá más interacciones bajo la modalidad M2P o bajo la modalidad P2P. En un caso en que se utilice un valor de θ que favorezca más a las interacciones M2P, tenemos de igual forma la garantía de que este proceso está bien paralelizado, siendo los procesos no paralelos, apreciables en color verde en el gráfico, casi despreciables en comparación a los procesos que de hecho son los que presentan un mejor *speedup*. Concluimos entonces que de manera general, la paralelización de PyGBe escala bien en cuanto a tiempos totales de cómputo.

Obtenemos, de esta forma, una versión de PyGBe cuyos procesos más demandantes en términos de cómputo y tiempo han sido satisfactoriamente paralelizados a nivel de memoria compartida.

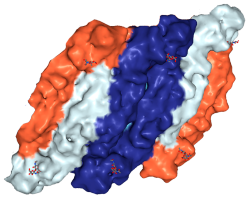
N° Threads	Tiempo Energía Coulomb[s]	Speedup
1	0.0143	-
2	0.0079	1.8
3	0.0051	2.8
4	0.0042	3.4

Tabla 3.6: Resultados de tiempo y speedup para distintos números de threads en el cálculo de la energía de Coulomb para el caso de una lisosima.

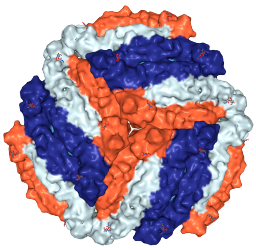
3.2. Zika: Gradientes de salinidad y contribución de residuos

3.2.1. Motivación

La figura 3.3 muestra una representación de la estructura del virus Zika (ZIKV), clasificado dentro de la familia de los flavovirus y tiene una cápside proteica con ARN en su interior. La superficie del virus consiste en 180 copias del capsómero, ordenados en una simetría icosaédrica con 60 unidades asimétricas. En cada unidad asimétrica podemos encontrar tres proteínas o cadenas individuales, A, B y C. Las proteínas de la cápside existen en forma de dimeros (figura 3.2a). Tres dimeros están paralelos entre sí formando una "balsa" (3.2b) que contiene dos unidades asimétricas. En total existen 30 de estas "balsas".



(a) Dímero de ZIKV. Contiene dos monómeros asimétricos.



(b) Estructura en forma de "balsa" que contiene tres dímeros.

Figura 3.2: Representación grafica del virus 5IRE. Image from the RCSB PDB (rcsb.org) of PDB ID 5IRE (Sirohi, D., Chen, Z., Sun, L., Klose, T., Pierson, T., Rossmann, M., Kuhn, R.) (2016) The cryo-EM structure of Zika Virus. (Sirohi et al., 2016)

El virus se transmite principalmente a los humanos a través de la picadura de mosquitos *Aedes*, aunque es de conocimiento general que ya se han reportado casos de propagación del ZIKV a través de las transfusión de sangre y de contacto sexual. La mayoría de las personas no desarrollan los síntomas de la enfermedad pero son portadores de este virus que puede causar microcefalia, defecto congénito en el que la cabeza de un bebé es significativamente más pequeña de lo esperado y defectos congénitos graves en bebés nacidos de madres infectadas. Más allá de los grandes brotes en 2015 y 2016, los años siguientes han sido silenciosos en términos del ZIKV. En los Estados Unidos hubo más de 41000 casos de transmisión en 2016,

menos que 1200 casos en 2017, y solo 220 casos en 2018 [4]. El resto del mundo ha experimentado situaciones similares; sin embargo, gran parte del mundo está en riesgo y mucha gente que viaja también lo está. Los viajeros de áreas endémicas pueden también introducir el virus en nuevos territorios (Leder et al., 2017). La actual impredecibilidad de futuros brotes de ZIKV en es el mayor impedimento para la creación de políticas públicas sanitarias efectivas, vigilancia de la enfermedad y estudio y desarrollo de una vacuna clínica. Son la urgencia sanitaria, los avances en diseño de vacunas basadas en genética y los nuevos descubrimientos en inmunología que han motivado el estudio del ZIKV con el objetivo de poder combatir una amenaza pandémica.

Hemos querido aplicar las herramientas que desarrollamos en PyGBe en el virus Zika, con el fin de enfocarnos en dos tipos de resultados. Primero nos preguntamos cómo cambiaría la estabilidad energética de la cápside viral si es que este fuera sometido a diferencias de salinidad dentro y fuera de su estructura. Luego dado, que la cápside del virus está hecha a base de distintos tipos de aminoácidos, nos preguntamos cuál será el aporte energético de diferentes residuos en la conformación del virus completo y cómo la presencia de estas partes puede afectar su estabilidad.

3.2.2. Preparación de la estructura

Para el trabajo con esta estructura, hemos utilizado la estructura PDB de código 5IRE, en el Protein Data Bank, correspondiente a la estructura de la cápside obtenida por criomicroscopía electrónica. La estructura descargada consiste en el capsómero más básico de la cápside, que contiene 11030 átomos sin considerar hidrógenos. Al usar nuestro script que replica la estructura básica 60 veces para completar el bioensamble completo, obtenemos una estructura grande que cuenta con 661800 átomos en un archivo con formato `*.pdb`. La estructura PDB descargada directamente del Protein Data Bank carecía de algunos átomos pesados necesarios para poder utilizar `pdb2pqr` y por lo tanto fue necesario hacer uso de `pdbfixer`, una herramienta que resuelve este problema, escribiendo la posición de los átomos pesados faltantes. De esta forma es posible usar el software `pdb2pqr` con un campo de fuerza AMBER con el fin de poder utilizarlo en PyGBe. Habiendo agregado los hidrógenos de acuerdo a un estado de pH neutro, el archivo ahora cuenta con 1578250 átomos. Una vez que tenemos listo el archivo `*.pqr` hacemos el mallado de la cápside utilizando `Nanoshaper` en una `CPU` con dos Intel Xeon E5-2580v3 de 12 núcleos con 96GB de memoria. El resultado es una SES hecha con un tamaño de molécula de agua de 1,4, que contiene varias cavidades pequeñas llenas de solvente que debemos remover. Al utilizar `Trimesh` sobre la malla, removemos todos estos bolsillos de solvente para quedar con solo un archivo que contiene toda la parte que cubre a la cápside. De aquí en adelante, el tratamiento dependió del tipo de simulación que estuviéramos interesados en realizar.

[4] Centers for Disease Control and Prevention. Zika virus: statistics and maps. http://shorturl.at/lMY35. Accedido el 15 de Diciembre de 2019.

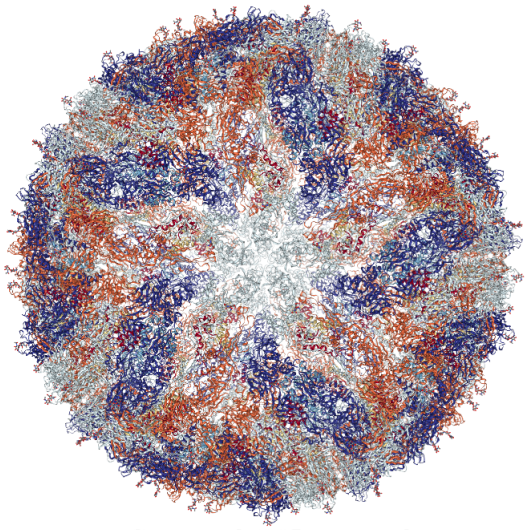


Figura 3.3: Representación del virus Zika. Image from the RCSB PDB (rcsb.org) of PDB ID 5ire Sirohi, D., Chen, Z., Sun, L., Klose, T., Pierson, T. C., Rossmann, M. G., & Kuhn, R. J. (2016). The 3.8 resolution cryo-EM structure of Zika virus. Science, 352(6284), 467-470.

3.2.3. Cápside expuesta a gradientes de salinidad

En la práctica las interacciones electrostáticas generan una capa de Stern, que no permite que las moléculas de agua pasen, pero al mallar con la molécula de agua de tamaño 1,4 para generar la SES, se construye una sola superficie que conecta las partes interior y exterior de la cápside, por lo tanto, no se puede separar en dos superficies de manera simple con el uso de Trimesh. Eso representa un problema para nosotros puesto que uno de nuestros intereses principales es considerar las regiones interior y exterior como distintas para poder usar distintos valores de salinidad en las distintas zonas. Para solucionar este problema, hemos simulado un Stern layer a través de generación de una nueva superficie a la que le hemos sumado 2[Å] (que corresponde al radio de un ión de sal) al radio de cada átomo. Con esta nueva superficie obtuvimos una malla impermeable al agua y separable en dos superficies diferentes: una interna y una externa. Para modelar la biomolécula, usaremos las tres superficies generadas para definir cuatro regiones, como se puede ver en la figura 3.4, donde las superficies generadas con el tamaño de molécula de agua de 1,5 pasan a ser Γ_0 (exterior) y Γ_2 (interior), de manera que la superficie generada inicialmente, Γ_1, pueda quedar encerrada completamente entre Γ_0 y Γ_2. La región exterior de solvente en la que está disuelta la cápside corresponde a Ω_0, que cuenta con unas constante κ y ϵ adecuadas. Posteriormente tenemos una región de pequeño volumen Ω_1 que está bien delimitada por Γ_0 y Γ_2 y que contiene a Γ_1. Esta región es la capa de Stern. Delimitada completamente por Γ_1 tenemos la región Ω_2 que contiene las cargas asociadas a los átomos de la cápside y por último tenemos a Ω_3 delimitada exteriormente por Γ_2 que representa la zona interna de la molécula, que está llena de solvente.

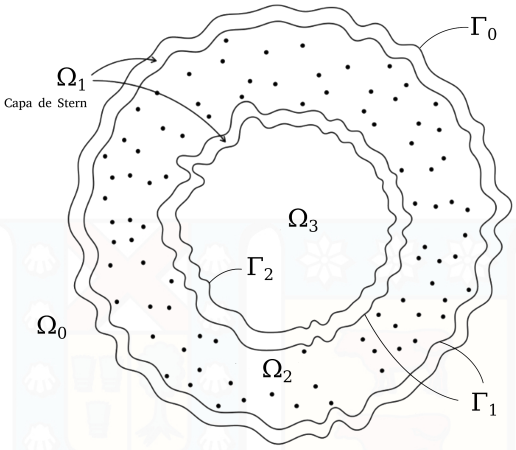


Figura 3.4: Esquema del ZIKV para modelamiento de diferencias de salinidad dentro y fuera de la macromolécula.

Nuestro objetivo ahora es modelar un cambio en la concentración de sal del solvente dentro y fuera de la cápside viral. A nivel de PyGBe existen dos parámetros que definen las distintas regiones del modelo. Estas son la constante dieléctrica ϵ y el inverso de la longitud de Debye κ. Podemos utilizar estos parámetros para representar las diferencias de salinidad. Los rangos de los valores normales de sodio en la sangre humana pueden variar ligeramente entre diferentes laboratorios, pero el rango normal es de 135$[\frac{\text{mEq}}{\text{L}}]$ a 150$[\frac{\text{mEq}}{\text{L}}]$ o unos 150[mM] en términos de molaridad. Gavish *et al.* mostraron que existe una dependencia entre la constante dieléctrica ϵ de una solución salina y su concentración de sal (Gavish y Promislow, 2016), pero las variaciones de ϵ comienzan a notarse después de agregar una cantidad dramática de soluto a la solución ($\sim$ 1[M]). Estos valores se encuentran alejados de las cantidades normales de sodio en la sangre y de las variaciones que queremos investigar, por lo que no hemos hecho variar el valor de ϵ. Por otro lado, tenemos el parámetro κ, correspondiente al inverso de la longitud de Debye.
Para una solución salina con i iones, podemos calcular la longitud de Debye λ_D como

$$\lambda_D = \sqrt{\frac{\epsilon_r \epsilon_0 k_B T}{\sum_i q_i^2 C_i^*}} \tag{3.1}$$

donde ϵ es la constante dieléctrica, ϵ_0 es la permitividad del vacío, k_B es la constante de Boltzmann, T es la temperatura, q_i es la carga de cada ion y C_i^* es la concentración del ión en $[\frac{\text{iones}}{L}]$.

Es así, como para el caso de un medio fisiológico en condiciones normales (C = 150mM), la longitud de Debye es

$$\lambda_D = \sqrt{\frac{80 \cdot 8{,}85 \times 10^{-12}[\frac{C^2}{Jm}] \cdot 1{,}38 \times 10^{-23}[J/K] \cdot 308[K]}{2 \cdot (1{,}6 \times 10^{-19}[C])^2 \cdot 0{,}15[mol/L] \cdot 1000[L/m^3] \cdot 6{,}02 \times 10^{23}[1/mol]}} = 8{,}0[]$$

y por lo tanto la constante κ del medio es $\kappa = 1/\lambda_D = 1/8{,}0[] = 0{,}125[^{-1}]$ que es el valor que usamos para todas las simulaciones en esta tesis a menos que se indique lo

	ϵ	$\kappa[^{-1}]$
Ω_0	80	$\kappa(C_{\text{ext}})$
Ω_1	80	10^{-12}
Ω_2	4	10^{-12}
Ω_3	80	$\kappa(C_{\text{int}})$

Tabla 3.7: Parámetros para cada una de las regiones para simulaciones de gradientes de salinidad.

contrario.

La tabla 3.7 muestra los parámetros a usar en cada una de las regiones representadas en la figura 3.4. Las concentraciones externa e interna, C_{ext} y C_{int}, varían en función del caso de estudio en particular.

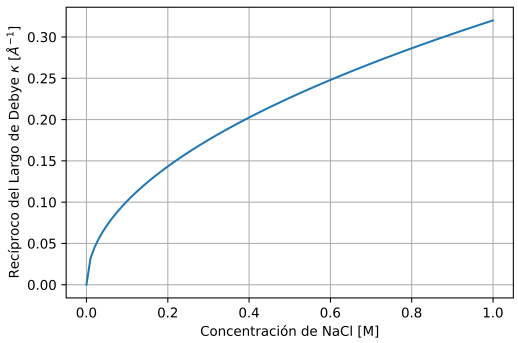


Figura 3.5: Relación entre la concentración de sal en el suero fisiológico y el recíproco de la longitud de Debye de la solución κ.

La figura 3.5 muestra la relación entre la concentración de NaCl en el suero fisiológico y la longitud de Debye de la solución. En términos biológicos, alejarnos demasiado del valor normal de concentración salina en sangre carece de sentido, por lo que nos enfocaremos en estudiar qué sucede con la energía de solvatación si es que:

1. **La concentración interna y externa es la normal.**
   Basándonos en la figura 3.4, $\kappa_0 = 0{,}125[^{-1}]$ y $\kappa_3 = 0{,}125[^{-1}]$
2. **La concentración interna es la normal y no existe sal en la región exterior.**
   Esto es, $\kappa_0 = 0$ y $\kappa_3 = 0{,}125[^{-1}]$
3. **La concentración interna es la normal y la concentración exterior se cuadriplica.**
   Esto es, $\kappa_0 = 0{,}125[^{-1}]$ y $\kappa_3 = 0{,}600[^{-1}]$
4. **La concentración interna se cuadriplica y la concentración externa es la normal.**
   Esto es, $\kappa_0 = 0{,}600[^{-1}]$ y $\kappa_3 = 0{,}125[^{-1}]$
5. **La concentración interna es nula y la concentración externa es la normal.**

C_3 [M]	C_0 [M]	κ_3	κ_0	$\Delta G_{\text{solv}}[kcal/mol]$
0.15	0.15	0.125	0.125	-273843
0.15	0.00	0.125	0.00	-273004
0.15	0.60	0.125	0.250	-274253
0.60	0.15	0.250	0.125	-274165
0.00	0.15	0.000	0.125	-273692

Tabla 3.8: Resultados de energía de solvatación para el ZIKV expuesto a diferentes gradientes de salinidad. Las referencias de los sub-índices son respecto a la figura 3.4.

Esto es, $\kappa_0 = 0[^{-1}]$ y $\kappa_3 = 0{,}125[^{-1}]$

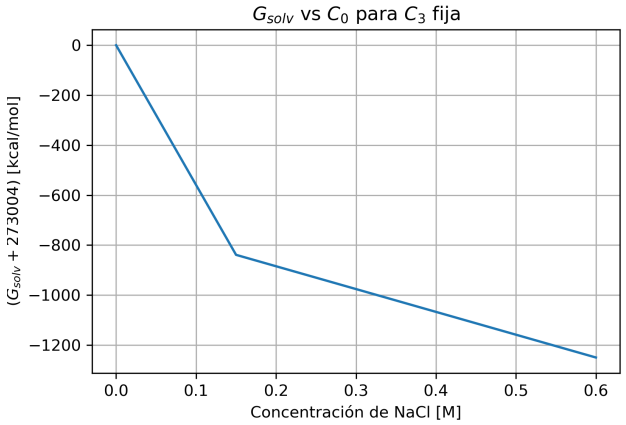
Para brindar perspectiva, exponer la cápside del virus al cuádruple de la concentración normal de sal en la sangre es equivalente a disolver la biomolécula en el Océano Atlántico. La permitividad dentro de la región de proteína fue $\epsilon_{\text{prot}} = 4$ y en el solvente fue de $\epsilon_{\text{solv}} = 80$. `PyGBe` utilizó 19 puntos de cuadratura Gaussiana por elemento cerca del punto de colocación donde se da la singularidad, 1 punto de cuadratura Gaussiana lejos de este punto y 9 puntos por borde de triángulo para la integración semi-analítica. La tolerancia de GMRES fue seteada en 10^{-4} y se hizo uso de solo 2 términos para la expansión de Taylor con un $\theta = 0{,}5$. Las simulaciones fueron hechas utilizando las tres superficies detalladas en la sección 3.2.3 y contaron con un total de 18141380 elementos, tardando al rededor de 4 días cada una y con un uso de memoria de 70GB.
Los casos 1, 2 y 3 presentan la misma concentración interna y van variando

La tabla 3.8 muestra los valores de energía de solvatación obtenidos para cada uno de los casos de estudio. Como se trata de la misma molécula disuelta en distintos medios, la energía de Coulomb es igual a $G_{\text{Coulomb}} = -5696216[kcal/mol]$ en todos los casos.

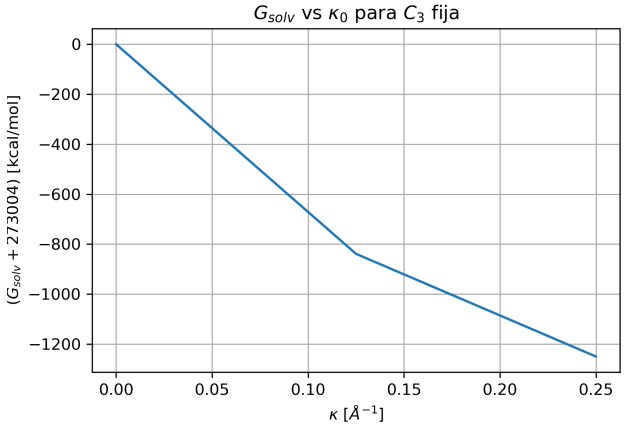
Desde una perspectiva energética, los estados con una energía de solvatación menor son más estables. La figura 3.6 muestra los valores de energía de solvatación a medida que la concentración de sal de la región externa va en aumento, para una salinidad interna fija $C_3 = 0{,}15[M]$. Los resultados muestran que a medida que aumenta la salinidad externamente, el sistema decrece en cuanto a energía de solvatación, y por lo tanto, se vuelve más estable a medida que se agrega más sal en la región externa. También es interesante notar que entre $\kappa_0 = 0$ y $\kappa_0 = 0{,}125$ la energía decae en 839 [kcal/mol] mientras que entre $\kappa_0 = 0{,}125$ y $\kappa_0 = 0{,}250$ decae en 410 [kcal/mol], lo que indica que es posible que al seguir aumentando la concentración salina se pueda llegar un estado de mínima energía potencial antes de la saturación de la solución. De la misma manera, la figura 3.7 muestra los valores de energía de solvatación a medida que la concentración de sal de la región interna va en aumento, para una salinidad externa fija de $C_0 = 0{,}15$[M].

Los resultados muestran que al igual que al hacer variar la salinidad externa, al disminuir la concentración de NaCl en la región Ω_3, la energía de solvatación va en descenso y, por lo tanto, los estados en los que existe mayor concentración salina interna son más estables energéticamente.
A partir de los resultados no es posible decidir si la presencia de un gradiente puede afectar en la estabilidad de la biomolécula, pues en ambos casos la concentración

fija era 0,15[M] valor que se encuentra entre medio de los casos extremos 0[M] y 0,6[M]. Pero sí podemos afirmar que, independiente de si el gradiente es positivo o negativo, la energía de solvatación de la cápside del virus parece ser susceptible a cambios absolutos de la salinidad, tanto adentro como afuera. Podemos concluir que los estados en los que existe mayor concentración de sal, ya sea fuera o dentro, son más favorables energéticamente, independiente del gradiente que se genere.



(a) Energía de Solvatación en función de C_0

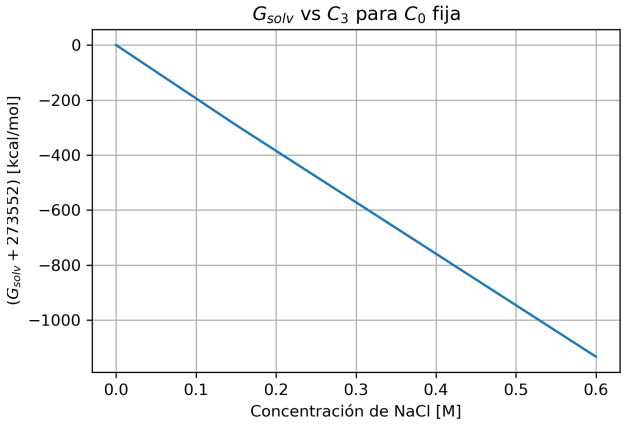


(b) Energía de Solvatación en función de κ_0

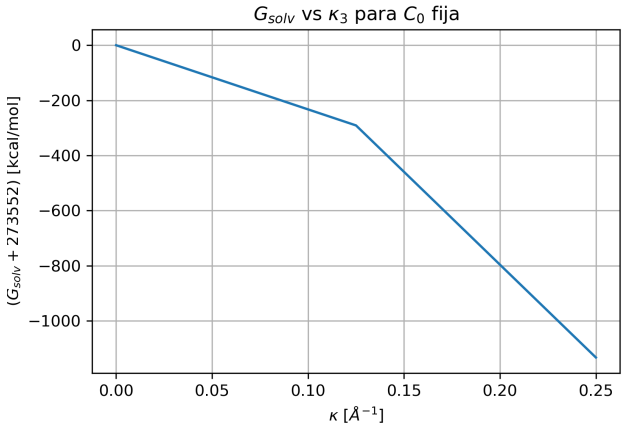
Figura 3.6: Energía de solvatación con variación de salinidad en la región externa y concentración $C_3 = 0{,}15[M]$ fija en la región interna.

3.2.4. Contribución de aminoácidos en energía de solvatación

Para realizar el cálculo de la contribución de un aminoácido en particular a la energía de solvatación de la cápside del virus es necesario calcular la energía de solvatación del sistema en dos casos: considerando y no considerando la presencia de los átomos correspondientes al aminoácido es cuestión. Con este fin, operamos directamente sobre los archivos `*.pqr` y *apagamos* las cargas correspondientes a un residuo en particular haciendo que la carga eléctrica de todos los átomos pertenecientes a ese residuo sean igual a cero.



(a) Energía de Solvatación en función de C_3



(b) Energía de Solvatación en función de κ_3

Figura 3.7: Energía de solvatación con variación de salinidad en la región interna y concentración $C_0 = 0{,}15[M]$ fija en la región externa.

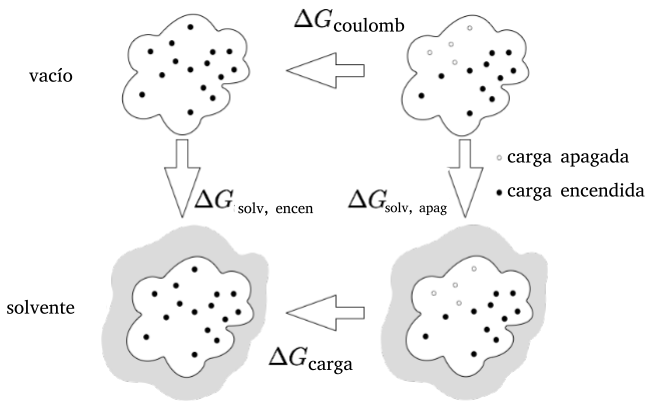


Figura 3.8: Esquema termodinámico de energías de solvatación, de Coulomb y de cargas.

La figura 3.8 muestra un diagrama de las distintas diferencias de energías en el sistema termodinámico. $\Delta G_{\text{solv, encen}}$ y $\Delta G_{\text{solv, apag}}$ son las energías de solvatación de la molécula con todas las cargas encendidas y de la molécula con las cargas de un residuo apagado respectivamente. Estos valores representan la diferencia energética entre el estado en el vacío y el estado disuelto de la cápside viral. Luego tenemos $\Delta G_{\text{coulomb}}$ que es la diferencia de energía de Coulomb entre el estado en que las cargas están apagadas y el estado en el que están encendidas. Finalmente tenemos la diferencia de energía libre de carga ΔG_{carga}, que se puede calcular según

$$\Delta G_{\text{carga}} = \Delta G_{\text{solv, encen}} - \Delta G_{\text{solv, apag}} + \Delta G_{\text{coulomb}} \tag{3.2}$$

En la sección 3.2.3 se detalla la preparación de las distintas mallas a utilizar cuando es necesario que exista una separación entre las regiones interna y externa con el objeto de poder caracterizarlas de manera diferente. Este no es el caso para las simulaciones que queremos efectuar en este apartado, porque no necesitamos establecer diferencias entre el medio interior y exterior del virus.

	N° Elementos	N° Iteraciones GMRES	Tiempo [h]	$\Delta G_{\text{solv}}[kcal/mol]$
Con capa de Stern	18141380	144	92.1	-273843.07
Sin capa de Stern	10788116	139	71.5	-276455.95

Tabla 3.9: Diferencias entre simulaciones para los casos con y sin capa de Stern.

La tabla 3.9 muestra las diferencias entre las simulaciones al considerar la capa de Stern, siguiendo el modelo que se muestra en la figura 3.4 y sin considerar esta capa, esto es, considerando solo dos regiones: una de soluto (proteína) y otra de solvente. Podemos notar que la diferencia entre las energías de solvatación es baja mientras que la ganancia en términos de tiempo es considerable ($\sim$ 1 día), es por esto que hemos decidido efectuar las simulaciones de este apartado utilizando el modelo sin capa de Stern.

Para las simulaciones hemos adoptado valores de la permitividad dentro de la región de proteína de $\epsilon_{\text{prot}} = 4$ y en el solvente fue de $\epsilon_{\text{solv}} = 80$. `PyGBe` utilizó 19 puntos de cuadratura Gaussiana por elemento cerca del punto de colocación donde se da la singularidad, 1 punto de cuadratura Gaussiana lejos de este punto y 9 puntos por borde de triángulo para la integración semi-analítica. La tolerancia de GMRES fue seteada en 10^{-4} y se hizo uso de solo 2 términos para la expansión de Taylor con un $\theta = 0{,}5$. Las simulaciones fueron hechas utilizando solo la superficie SES generada al utilizar el tamaño de molécula de agua de 1,4[Å] y contaron con un total de 10778116 elementos de superficie, con una densidad de malla de 3 elementos por Å^2, tardando al rededor de 3 días cada una y con un uso de memoria promedio de 32GB.
La tabla 3.10 muestra los resultados de la energía de carga ΔG_{carga} obtenidos para cada uno de los casos en que apagábamos los átomos de un residuo aminoácido en particular. También muestra la cantidad de átomos de este aminoácido en la cápside para poder establecer una comparación justa según la energía de carga dividida por el número de átomos involucrados.

	N° Átomos	ΔG_{carga} [kcal/mol]	ΔG_{carga} / N° Átomos [kcal/mol]
ALA	86940	-353846	-4.07
ARG	99360	-751170	-7.56
ASN	45360	-285125	-6.29
ASP	54000	-428177	-7.93
CYS	24846	-94528	-3.81
GLN	48960	-258771	-5.29
GLU	78300	-515125	-6.58
GLY	71820	-388080	-5.40
HIS	58199	-231317	-3.97
ILE	95766	-235575	-2.46
LEU	188100	-461129	-2.45
LYS	126720	-307754	-2.43
MET	58140	-145872	-2.51
PHE	68400	-147652	-2.16
PRO	47880	-104228	-2.18
SER	89280	-515264	-5.77
THR	115920	-572979	-4.94
TRP	56160	-148270	-2.64
TYR	49140	-153530	-3.12
VAL	120960	-334608	-2.77

Tabla 3.10: Valores de ΔG_{carga} y ΔG_{carga}/N° Átomos al apagar diferentes aminoácidos en la cápside del ZIKV.

La arginina (ARG) es el aminoácido que tiene una mayor energía de carga en comparación a los demás residuos, con -751170 [kcal/mol] y aunque existen muchos átomos componiendo las moléculas de este aminoácido, el cociente entre esta energía de carga y el número de átomos sigue siendo unos de los más altos. Este aminoácido cuenta con una cadena lateral que está formada por un grupo guanidino. Esta es la cadena lateral más polar de entre las encontradas en proteínas (Wolfenden, 1983) y por lo tanto es el residuo más común de encontrar en la superficie en un ambiente acuoso. Este grupo guanidino está cargado positivamente y sirve como donador de hidrógenos

en la formación de hasta 5 puentes de hidrógeno. Adicionalmente, la arginina está involucrada en la formación de puentes de sal simples o dobles con el ácido aspártico y el ácido glutámico, que son justamente los otros residuos con la energía por átomo más alta en términos absolutos. Estos puentes son capaces de estabilizar las estructuras terciarias y cuaternarias de numerosas proteínas (Riordan et al., 1977). Nuestros resultados desde el punto de vista electrostático están completamente en sintonía con lo que se puede encontrar en la literatura.

3.3. Triatoma: Energías libres en el desensamble

3.3.1. Motivación

El virus Triatoma (TrV) fue descubierto en el 1984 y pertenece a la familia de virus de insectos Dicistroviridae. El huésped natural de este virus corresponde a invertebrados y es un patógeno conocido del *Triatoma infestans*, o vinchuca, el más grande vector del mal del Chagas, del cual se calcula que existen entre seis y siete millones de personas afectadas según cifras de la OMS. La cápside proteica del virus mide unos $30[nm]$ de diámetro, con una cápside de geometría icosaédrica. (Muscio et al., 1988)
Se ha observado que un cambio en el pH del medio puede gatillar la liberación prematura del material genético contenido por la cápside, sin embargo, los mecanismos a través de los que ocurre este proceso han sido poco conocidos y entendidos. Viso y su equipo (Viso et al., 2018) observaron la formación de una compuerta hidrofóbica en la cavidad a lo largo del eje de simetría de orden 5 de la cápside. Observaron también que protones pueden permear la cápside a través del canal que se forma a través de un mecanismo como el de Grotthuss unidireccionalmente, que es la vía a través de la cual la cápside *siente* la variación de pH. Por último, el flujo de protones provoca un desbalance de cargas que hace que el material genético ejerza una presión y termine rompiendo la cápside.
Se ha demostrado que la cápside del TrV es estable en presencia de pH ácido, pero se desensambla en condiciones alcalinas (Agirre et al., 2013) y el ARN es la molécula que provoca su propia liberación. En esta tesis utilizamos nuestro solver de Poisson Boltzmann paralelizado para simular condiciones de gradiente de pH, con diferentes valores dentro y fuera de la cápside.

3.3.2. Preparación de la estructura

Dado que con nuestra herramienta podemos medir energías libres en distintos estados estáticos, utilizamos estructuras de la cápside en distintas etapas de su desensamble obtenidas mediante simulaciones de multiescala. La obtención de estos estados de la cápside desensamblada fueron obtenidos mediante un procedimiento de simulaciones de *coarse grain* (CG) con el campo de fuerza SIRAH. Con este fin, seguimos las simulaciones y trayectorias reportadas por Viso (Viso et al., 2018).
En su publicación, ellos partieron desde la estructura tridimensional 3NAP, disponible en el Protein Data Bank y usaron el campo de fuerza de GROMOS 53A6 más un

modelo de agua de Single Point Charge (SCP) para detalles atómicos; y el campo de fuerza SIRAH para el grano grueso o *coarse grain*, a través del software GROMACS. (Van Der Spoel et al., 2005). El enfoque multiescala consistió en simular el pentámero de la cápside proteica junto a una delgada capa de agua (1 [nm]) al rededor de ella a nivel atomístico, mientras que el resto de las moléculas fueron simuladas usando el modelo de agua WT4 de SIRAH. Las simulaciones fueron hechas en una solución fisiológica de NaCl a concentración de 0.15 molar y $MgCl_2$ a concentración de 0.015 molar. En el caso de la cápside completa, una tercera capa se agregó con el modelo WLS, que presenta menos detalle que el modelo WT4. Para analizar el desensamble de la cápside, usaron las herramientas de SIRAH para convertir la estructura atomística en CG y disolverla en un cascaron de agua de 2.5 [nm]. Para inducir la desestabilización electrostática, progresivamente fueron sustituyendo moléculas de solvente en el interior con iones de Cl^- y de Na^+ en el exterior, dando origen a las trayectorias de desensamble de la cápside viral. Como PyGBe toma como input trayectorias detalladas átomo por átomo, se hizo uso de herramientas de SIRAH para mapear de vuelta los elementos de CG a átomos. Este trabajo fue realizado por nuestro colaborador, PhD. Horacio Guzmán. Tomamos estos últimos resultados como input para PyGBe, con el fin de estudiar la energía de solvatación del proceso de desensamblado.
Para el caso de esta estructura proteica en particular, debemos distinguir tres regiones:

Ω_1: Región interna, llena con material genético.

Ω_2: Región externa, representado al solvente.

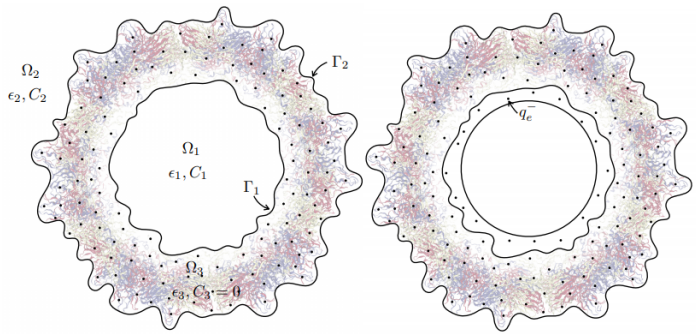
Ω_3: Región de la cápside: membrana proteica entre las regiones interna y externa.

Cada región consta de su constante dieléctrica (ϵ_1, ϵ_2, ϵ_3) y concentración de sal, la cual es nula en la cápside. Por simplicidad, usaremos parámetros que son apropiados para un solvente en la región interna (Ω_1). Si la membrana no es impermeable, Ω_1 y Ω_2 estarían conectados y se consideraría la misma región dieléctrica.

Dado el "moviminto" de protones hacia afuera de la cápside, a través del cana de Grotthuss, se genera una carga negativa en la zona interior de la macromolécula. Hemos situado átomos de cloro en esta región para simular el desequilibrio eléctrico que se forma. Las cargas parciales en el soluto son representadas como cargas puntuales estáticas en la ubicación de los átomos, mientras que los iones de sal en el solvente son considerados cargas puntuales móviles de acuerdo a la distribución de Boltzmann. La cantidad de interés más común fue la energía de solvatación, que es el trabajo requerido para traer el soluto desde el vacío hasta el solvente. Considerando que las cargas dentro del soluto son funciones delta de Dirac, podemos escribir para esta energía

$$\Delta G_{\text{solv}} = \int_{\Omega_3} \rho \phi_{\text{reac}} = \sum_{k=1}^{N_q} \phi_{\text{reac}}(\mathbf{r}_k) \tag{3.3}$$

donde $\phi_{\text{reac}} = \phi_3 - \phi_{\text{coul}}$ es el potencial de reacción.
Existe otra fuente de energía en distribución de cargas puntuales del soluto. Luego, la contribución total de energía electrostática a la energía libre es la suma de la energía



(a) Capsid in an implicit solvent (b) Capsid with charge of genetic material

Figura 3.9: Representación de la cápside en un modelo de solvente implícito con tres regiones. La figura 3.9a muestra la región dentro de la cápside (Ω_1), fuera de la cápside (Ω_2) y la interfaz donde están ubicados los átomos de la cápside (Ω_3). La figura 3.9b muestra una representación alternativa añadiendo una región extra con cargas puntuales para representar el cambio después de la deprotonación del material genético. American Chemical Society. (2019). Representation of the capsid in an implicit-solvent model which comprises three regions [Figura]. En Free Energies of the Disassembly of Viral Capsids from a Multiscale Molecular Simulation Approach.

de solvatación y la energía de Coulomb

$$G_{\text{elec}} = G_{\text{solv}} + G_{\text{coul}} \tag{3.4}$$

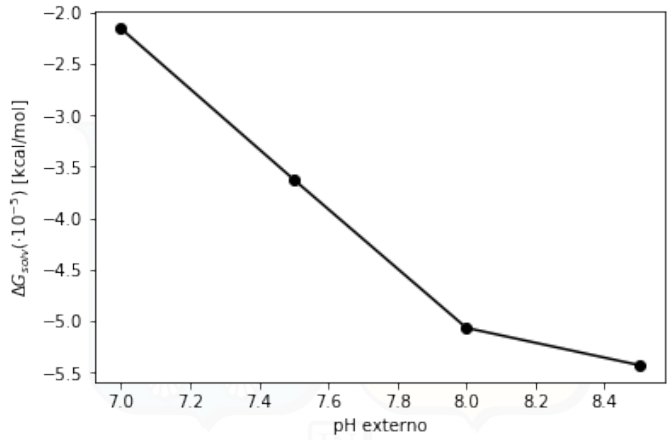
donde

$$G_{\text{coul}} = \frac{1}{2\epsilon_3} \sum_{k=1}^{N_q} \sum_{j=1}^{N_q} q_k \frac{q_j}{4\pi |\mathbf{r}_k - \mathbf{r}_j|} \tag{3.5}$$

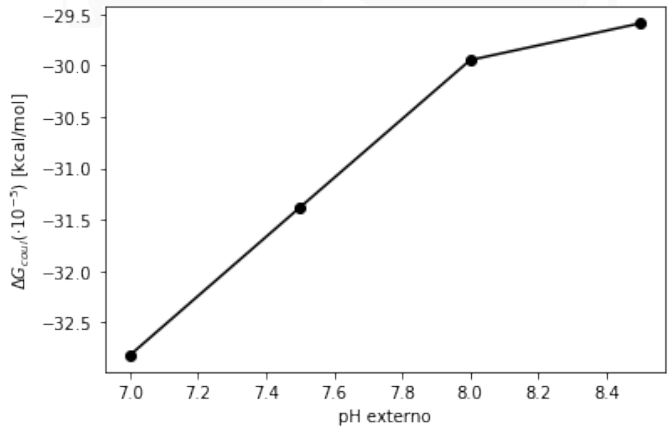
3.3.3. Cálculo de energías libres para estructura sin desensamblar

Nuestro primer enfoque fue realizar cálculos de energía libre para la estructura sin desensamblar y cuando está siendo sometida a un gradiente de pH, con el fin de identificar qué tan susceptible es su estabilidad electrostática a cambios en la concentración de protones tanto dentro como fuera de la molécula. La comunidad de desarrollo de medicamentos para la caracterización experimental de moléculas como virus (VLPs) usa rangos de pH de 6 a 9 para sus experimentos en vacunas (Steppert et al., 2017) y considerando el mecanismo de desensamble propuesto por Viso *et al.* (Viso et al., 2018) estudiamos variaciones de pH entre 7 y 8.5. Inicialmente variando el pH externo manteniendo un pH neutro en la región interna, luego, modificando el pH interno con pH = 8.5 externamente emulando de esta forma el proceso de equilibrio de pH una vez que los protones ya se han desplazado a través del canal mediante el mecanismo de Grotthuss. Estos resultados se encuentran en las tablas 3.11 y 3.12.

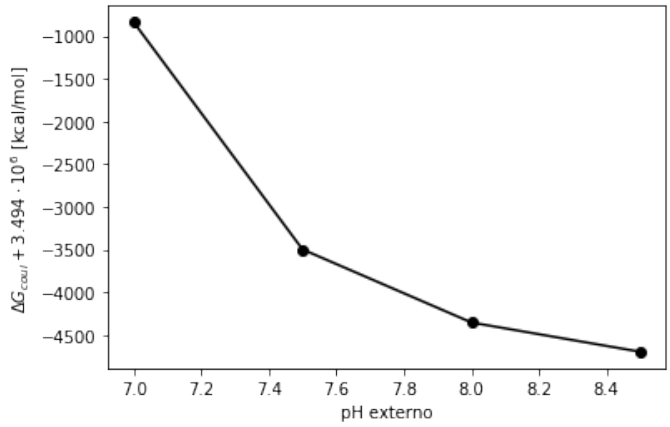
Usualmente, en los cálculos de energía de Coulomb, los campos de fuerza incluyen factores de escala para átomos que están a 3 enlaces o menos. No hemos considerado este factor para nuestros resultados; sin embargo, dado que estamos interesados en diferencias de energía libre, nuestras conclusiones no se ven afectadas pues este error se cancela.



(a) Energía de Solvatación

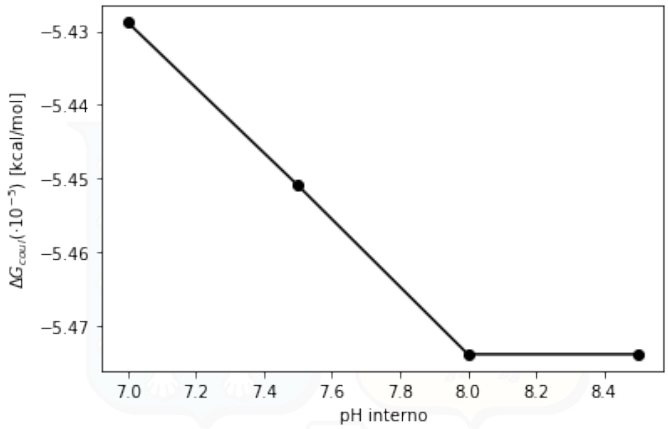


(b) Energía de Coulomb

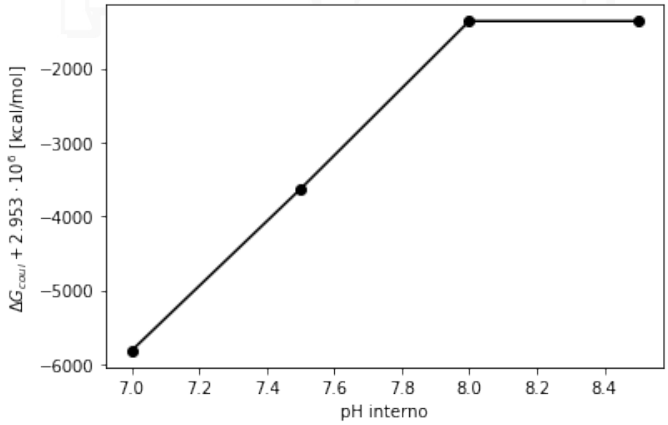


(c) Energía electrostática total

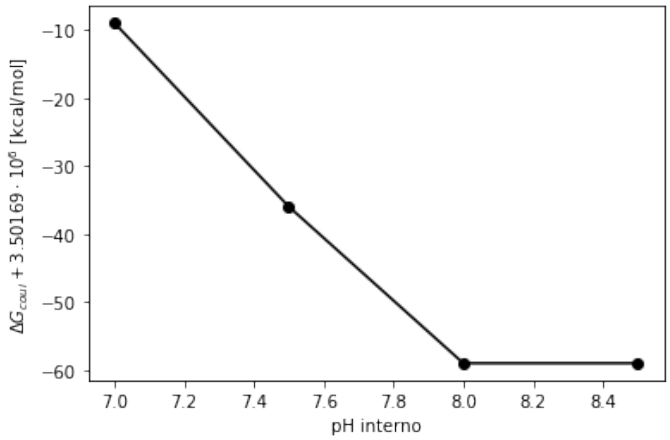
Figura 3.10: Energía libre electrostática con variación de pH en la región externa y $\text{pH}_{int} = 7$ fijo en la región interna. Los valores numéricos se encuentran expresados en la tabla 3.11



(a) Energía de Solvatación



(b) Energía de Coulomb



(c) Energía electrostática total

Figura 3.11: Energía libre electrostática con variación de pH en la región interna y $\text{pH}_{int} = 8{,}5$ fijo en la región externa. Los valores numéricos se encuentran expresados en la tabla 3.12

pH_{int}	pH_{ext}	Energía [kcal/mol]		
		Solvatación	Coulomb	Total
7.0	7.0	-215672	-3282166	-3497838
7.0	7.5	-362387	-3138113	-3500500
7.0	8.0	-506680	-2994676	-3501356
7.0	8.5	-542880	-2958819	-3501699

Tabla 3.11: Influencia del gradiente de pH en la energía electrostática libre, con pH neutro interiormente y variaciones en el exterior.

pH_{int}	pH_{ext}	Energía [kcal/mol]		
		Solvatación	Coulomb	Total
7.0	8.5	-542880	-2958819	-3501699
7.5	8.5	-545094	-2956632	-3501726
8.0	8.5	-547390	-2954359	-3501749
8.5	8.5	-547390	-2954359	-3501749

Tabla 3.12: Influencia del gradiente de pH en la energía electrostática libre, con pH = 8,5 en la región exterior y con variación en la región interna.

Una mirada rápida a la energía libre total en las figuras 3.10c y 3.11c nos convence de que el pH externo tiene un impacto mucho más grande en la estabilidad de la cápside que el pH interno. Estas figuras muestran una caída en la energía libre, sin embargo, la caída es mucho más importante cuando el pH externo crece. Más aún, en la figura 3.11c, la caída es solo del orden de 60 [kcal/mol], que está por debajo de la resolución de nuestro método numérico. Los datos también muestran que no hay una relación clara entre la energía libre y el gradiente de pH a través de la cápside. Por ejemplo, la energía libre es mayor para $\text{pH}_{\text{int}} = 7$ y $\text{pH}_{\text{ext}} = 7{,}5$ en comparación al caso en que $\text{pH}_{\text{int}} = 8$ y $\text{pH}_{\text{ext}} = 8{,}5$ incluso aunque la diferencia de pH es la misma. Y es que existe una asimetría entre los impactos de un cambio en el pH interno y en el externo. Es interesante analizar como los términos de energía de solvatación y la energía de Coulomb contribuyen a la energía libre. Las figuras 3.10 y 3.11 sugieren que estas energías tienden a compensarse, pues las diferencias en la energía de solvatación y Coulomb son grandes y de signo opuesto, pero la energía total sufre cambios pequeños.

Cálculo de energías libres para estructura desensamblada

Un pH alcalino dentro de la cápside deprotona el material genético, generado una fuerza electrostática repulsiva que causa el desensamblado. En forma similar, hemos añadido cargas puntuales negativas al interior de la cápside para simular el genoma deprotonado. Estas cargas puntuales ($q = -q_e$) fueron dispuestas de manera aleatoria en un cascarón esférico cercano a la cápside, como se ve en la figura 3.9b. En estas simulaciones, la nueva región de cascarón esférico no contiene sal ($\kappa = 0$) y posee la constante dieléctrica del solvente ($\epsilon = 80$).

Las tablas 3.13 y 3.14 muestran la contribución electrostática a la energía libre en tres diferentes etapas el desensamblado.

Etapa N°	Energía [kcal/mol]		
	Solvatación	Coulomb	Total
1	-574390	-2954358	-3501749
2	-544237	-2938631	-3482868
3	-289386	-3202676	-3492062

Tabla 3.13: Energía libre electrostática de la cápside en proceso de desensamblado al añadir 800 cargas negativas dentro de la cápside.

Etapa N°	Energía [kcal/mol]		
	Solvatación	Coulomb	Total
1	-574390	-2954358	-3501749
2	-543040	-2891099	-3434139
3	-347984	-3175389	-3523373

Tabla 3.14: Energía libre electrostática de la cápside en proceso de desensamblado al añadir 1600 cargas negativas dentro de la cápside.

1. Estructura de cápside ensamblada con pH= 8,5 en todos lados, representando la cápside en un ambiente alcalino.
2. Estructura de cápside ensamblada con pH= 8,5 en todos lados y cargas negativas dentro del cascarón esférico, para modelar el material genético deprotonado.
3. Estructura desensamblada obtenida por simulaciones de dinámica molecular con aniones de cloro (cargas puntuales negativas) dentro de la cápside y pH= 8,5 en todos lados.

El resultado en la tabla 3.13 corresponde a la simulación con 800 cargas negativas dentro, mientras que la tabla 3.14 presenta los cálculos hechos con 1600 cargas. Las diferencias de energía libre son útiles para el estudio de la estabilidad del sistema en el proceso de desensamblado. El salto de la etapa 1 a la etapa 2 es la diferencia energética de la pérdida de iones H^+ debido al ambiente alcalino. Como es de esperar, la energía libre aumenta, indicando que el sistema es menos estable. Al considerar la contribución de la energía de solvatación de y de Coulomb, esta desestabilización es principalmente debida a la componente coulúmbica y es mayor en el caso de 1600 iones.
Las cargas negativas inducen una repulsión electrostática que ejerce presión sobre la cápside de dentro hacia afuera. De hecho, al comparar las etapas 2 y 3 en las tablas 3.13 y 3.14, la energía libre decae, mostrando que el sistema es más estable al desensamblarse. En este caso, sin embargo, hay un efecto de compensación al aumentar la energía de solvatación y disminuir la energía de Coulomb. En términos absolutos, la contribución de la energía de Coulomb es más grande, confirmando que la repulsión electrostática del material genético cargado lleva al proceso de desensamblado. Estas observaciones concuerdan con el mecanismo propuesto por Viso et al. (Viso et al., 2018).

4 Conclusiones

En los últimos años la computación paralela ha potenciado increíblemente la computación científica, permitiendo acelerar los procesos de cálculo de múltiples aplicaciones. En particular, en este trabajo hemos utilizado la API OpenMP para realizar una paralelización a nivel de memoria compartida del software `PyGBe`; aprovechando así el hardware disponible. Con la paralelización de `PyGBe` se pudo alcanzar tiempos de cómputo reducidos en el cálculo de la electrostática de los virus Zika y Triatoma completos, en distintas configuraciones, resolviendo así la problemática de no tener la capacidad en términos de tiempo y cómputo de realizar estos cálculos en una estación de trabajo.

`PyGBe` es un solver escrito en Python y C que implementa una parametrización de del sistema de ecuaciones diferenciales parciales de Poisson-Boltzmann a través del método de elementos de borde (BEM), que tiene la ventaja de representar de manera precisa la superficie molecular y las condiciones en esta interfaz son forzadas de manera exacta.

Probablemente lo más complejo en la utilización y gestión de datos correspondientes a proteínas es el gran tamaño de estos lo heterogéneo de los formatos para representarlos, y la poca ergonomía al utilizar múltiples programas diferentes para llevar a cabo una cadena de procesos, teniendo que hacer conversiones, normalizaciones o correcciones al pasar de un programa a otro dentro de la misma cadena.

En cuanto al trabajo de paralelización, si bien no se llevó a cabo una paralelización de todo el software, sí se hizo a los procesos más temporalmente costosos, y esto llevo a un ahorro temporal considerable. OpenMP es una API sencilla de usar por lo que probablemente, la mayor complejidad del proceso de paralelización viene más bien por el lado de comprender bien el funcionamiento del código y cómo se están almacenando las variables en la memoria para poder hacer una correcta gestión de estas y hacer las modificaciones correspondientes para la correcta elección de las directivas en pos de un correcto funcionamiento.

Los dos casos de estudio realizados muestran que esta versión paralela de `PyGBe` es capaz de llegar a resultados que son coherentes con la investigación en tiempos considerablemente menores a lo que lo haría la versión serial.

Bibliografía

Agirre, J; Goret, Gaël; LeGoff, M; Sánchez-Eugenia, R; Marti, Gerardo Anibal; Navaza, Jorge; Guérin, DMA; y Neumann, Emmanuelle (2013). Cryo-electron microscopy reconstructions of triatoma virus particles: a clue to unravel genome delivery and capsid disassembly. *Journal of General Virology*, 94(5), 1058–1068. 3.3.1

Altman, Michael D; Bardhan, Jaydeep P; White, Jacob K; y Tidor, Bruce (2006). An accurate surface formulation for biomolecule electrostatics in non-ionic solutions. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference* (pp. 7591–7595).: IEEE. 1.1.5

Altman, Michael D; Bardhan, Jaydeep P; White, Jacob K; y Tidor, Bruce (2009). Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized poisson–boltzmann equation with curved boundary elements. *Journal of computational chemistry*, 30(1), 132–153.

Andrews, Gregory R (2000). *Foundations of multithreaded, parallel, and distributed programming*, volume 11. Addison-Wesley Reading. 1.4.4.1

Atkinson, Kendall E (1997). The numerical solution of boundary integral equations. In *Institute of mathematics and its applications conference series*, volume 63 (pp. 223–260).: Oxford University Press. 1.2

Avendaño Carvajal, Luis Fidel; Ferrés Garrido, Marcela; y Spencer Ossa, Eugenio (2011). *Virología clínica*. Technical report, Mediterráneo,.

Baker, Nathan A (2004). Poisson–boltzmann methods for biomolecular electrostatics. In *Methods in enzymology*, volume 383 (pp. 94–118). Elsevier. 1.1.3

Baker, Nathan A; Sept, David; Joseph, Simpson; Holst, Michael J; y McCammon, J Andrew (2001). Electrostatics of nanosystems: application to microtubules and the ribosome. *Proceedings of the National Academy of Sciences*, 98(18), 10037–10041. 1.1.5, 3

Bardhan, Jaydeep P (2012). Biomolecular electrostatics—i want your solvation (model). *Computational Science & Discovery*, 5(1), 013001. 1.1.3

Barnes, Josh y Hut, Piet (1986). A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096), 446. 1.2.4, 2.1

Barton, Eric; Cownie, James; y McLaren, Moray (1994). Message passing on the meiko cs-2. *Parallel Computing*, 20(4), 497–507. 1.4.2

Bashford, Donald (1997). An object-oriented programming suite for electrostatic effects in biological molecules an experience report on the mead project. In *International Conference on Computing in Object-Oriented Parallel Environments* (pp. 233–240).: Springer. 1.1.5, 3

Bashford, Donald y Gerwert, Klaus (1992). Electrostatic calculations of the pka values of ionizable groups in bacteriorhodopsin. *Journal of molecular biology*, 224(2), 473–486. 1.1.5, 3

Baynes, Brian M; Wang, Daniel IC; y Trout, Bernhardt L (2005). Role of arginine in the stabilization of proteins against aggregation. *Biochemistry*, 44(12), 4919–4925.

Beazley, Dave; Team, SWIG; et al. (1995). Simplified wrapper and interface generator. *See http://swig. sourceforge. net*. 2.1

Behnel, Stefan; Bradshaw, Robert; Citro, Craig; Dalcin, Lisandro; Seljebotn, Dag Sverre; y Smith, Kurt (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2), 31. 2.1

Betti, Enrico (1872). Teoria della elasticita. *Il Nuovo Cimento (1869-1876)*, 7(1), 158–180. 1.2

Bomans, Luc y Roose, Dirk (1989). Benchmarking the ipsc/2 hypercube multiprocessor. *Concurrency: practice and experience*, 1(1), 3–18. 1.4.2

Bonthuis, Douwe Jan y Netz, Roland R (2013). Beyond the continuum: How molecular solvent structure affects electrostatics and hydrodynamics at solid–electrolyte interfaces. *The Journal of Physical Chemistry B*, 117(39), 11397–11413. 1.1.3

Borders Jr, CL; Broadwater, John A; Bekeny, Paula A; Salmon, Johanna E; Lee, Ann S; Eldridge, Aimee M; y Pett, Virginia B (1994). A structural role for arginine in proteins: multiple hydrogen bonds to backbone carbonyl oxygens. *Protein Science*, 3(4), 541–548.

Brebbia, Carlos Alberto y Dominguez, Jose (1994). *Boundary elements: an introductory course*. WIT press. 1.2

Carnie, Steven L y Chan, Derek YC (1993). Interaction free energy between identical spherical colloidal particles: The linearized poisson-boltzmann theory. *Journal of colloid and interface science*, 155(2), 297–312.

Caspar, Donald LD y Klug, Aaron (1962). Physical principles in the construction of regular viruses. In *Cold Spring Harbor symposia on quantitative biology*, volume 27 (pp. 1–24).: Cold Spring Harbor Laboratory Press. 1.5.3.1

Chan, Derek YC y Mitchell, D John (1983). The free energy of an electrical double layer. *Journal of colloid and interface science*, 95(1), 193–197.

Clark, Eliana De Bernardez (1998). Refolding of recombinant proteins. *Current Opinion in Biotechnology*, 9(2), 157–163.

Cooper, Christopher y Barba, Lorena A (2013). Validation of the pygbe code for poisson-boltzmann equation with boundary element methods. 1.1.5

Cooper, Christopher D; Bardhan, Jaydeep P; y Barba, Lorena A (2014). A biomolecular electrostatics solver using python, gpus and boundary elements that can handle solvent-filled cavities and stern layers. *Computer physics communications*, 185(3), 720–729. 3

Cooper, Christopher D; Clementi, Natalia C; Forsyth, Gilbert; y Barba, Lorena A (2016). Pygbe: Python, gpus and boundary elements for biomolecular electrostatics. *J. Open Source Software*, 1(4), 43. 2.1

Dagum, Leonardo y Menon, Ramesh (1998). Openmp: An industry-standard api for shared-memory programming. *Computing in Science & Engineering*, (1), 46–55. 2.1

Decherchi, Sergio y Rocchia, Walter (2013). A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale. *PloS one*, 8(4), e59744. 1.1.3, 2.2.2

Dolinsky, Todd J; Nielsen, Jens E; McCammon, J Andrew; y Baker, Nathan A (2004). Pdb2pqr: an automated pipeline for the setup of poisson–boltzmann electrostatics calculations. *Nucleic acids research*, 32(suppl_2), W665–W667. 2.2.1

Flint, SJ; Enquist, LW; Racaniello, VR; y Skalka, AM (2003). Principles of virology: Molecular biology. *Pathogenesis, and Control (Am. Soc. Microbiol., Washington, DC),*. 1.5.3.1

Fredholm, Ivar (1903). Sur une classe d'équations fonctionnelles. *Acta mathematica*, 27(1), 365–390. 1.2

French, Roger H; Parsegian, V Adrian; Podgornik, Rudolf; Rajter, Rick F; Jagota, Anand; Luo, Jian; Asthagiri, Dilip; Chaudhury, Manoj K; Chiang, Yet-ming; Granick, Steve; et al. (2010). Long range interactions in nanoscale science. *Reviews of Modern Physics*, 82(2), 1887. 1.5.3

Gavish, Nir y Promislow, Keith (2016). Dependence of the dielectric constant of electrolyte solutions on ionic concentration: A microfield approach. *Physical review E*, 94(1), 012611. 3.2.3

Geng, Weihua y Krasny, Robert (2013). A treecode-accelerated boundary integral poisson–boltzmann solver for electrostatics of solvated biomolecules. *Journal of Computational Physics*, 247, 62–78. 1.1.5, 3

Geng, Weihua; Yu, Sining; y Wei, Guowei (2007). Treatment of charge singularities in implicit solvent models. *The Journal of chemical physics*, 127(11), 114106. 1.1.5, 3

Gilson, Michael K y Honig, Barry H (1986). The dielectric constant of a folded protein. *Biopolymers: Original Research on Biomolecules*, 25(11), 2097–2119. 1.1.3

Gilson, Michael K y Honig, Barry H (1987). Calculation of electrostatic potentials in an enzyme active site. *Nature*, 330(6143), 84. 1.1.3, 3

Gilson, Michael K; Rashin, Alexander; Fine, Richard; y Honig, Barry (1985). On the calculation of electrostatic interactions in proteins. *Journal of molecular biology*, 184(3), 503–516. 1.1.5

Green, George (1828). *An essay on the application of mathematical analysis to the theories of electricity and magnetism*. Wezäta-Melins Aktiebolag. 1.2

Greengard, Leslie; Huang, Jingfang; Rokhlin, Vladimir; y Wandzura, Stephen (1998). Accelerating fast multipole methods for the helmholtz equation at low frequencies. *IEEE Computational Science and Engineering*, 5(3), 32–38. 1.2.4

Greengard, Leslie y Rokhlin, Vladimir (1987). A fast algorithm for particle simulations. *Journal of computational physics*, 73(2), 325–348. 1.2.4

Greengard, Leslie F y Huang, Jingfang (2002). A new version of the fast multipole method for screened coulomb interactions in three dimensions. *Journal of Computational Physics*, 180(2), 642–658. 1.2.4

Gropp, William; Gropp, William D; Lusk, Ewing; Skjellum, Anthony; y Lusk, Argonne Distinguished Fellow Emeritus Ewing (1999). *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press. 1.4.3

Guvench, Olgun y MacKerell, Alexander D (2008). Comparison of protein force fields for molecular dynamics simulations. In *Molecular modeling of proteins* (pp. 63–88). Springer. 2.2.1

Jaswon, Maurice Aaron (1977). Integral equation methods in potential theory and elastostatics. 1.2

Jouppi, Norman P y Wall, David W (1989). Available instruction-level parallelism for superscalar and superpipelined machines. *ACM SIGARCH Computer Architecture News*, 17(2), 272–282. 1.4

Juffer, AndréH; Botta, Eugen FF; van Keulen, Bert AM; van der Ploeg, Auke; y Berendsen, Herman JC (1991). The electric potential of a macromolecule in a solvent: A fundamental approach. *Journal of Computational Physics*, 97(1), 144–171. 1.1.5

Katsikadelis, John T (2002). *Boundary elements: theory and applications*. Elsevier. 1.2

Kirkwood, John G (1934). Theory of solutions of molecules containing widely separated charges with special application to zwitterions. *The Journal of Chemical Physics*, 2(7), 351–361. 1.1.5

Leder, Karin; Grobusch, Martin P; Gautret, Philippe; Chen, Lin H; Kuhn, Susan; Lim, Poh Lian; Yates, Johnnie; McCarthy, Anne E; Rothe, Camilla; Kato, Yasuyuki; et al. (2017). Zika beyond the americas: Travelers as sentinels of zika virus transmission. a geosentinel analysis, 2012 to 2016. *Plos one*, 12(10), e0185689. 3.2.1

Li, Peijun; Johnston, Hans; y Krasny, Robert (2009). A cartesian treecode for screened coulomb interactions. *Journal of Computational Physics*, 228(10), 3858–3868. 1.2.4, 1.3

Liang, Jie y Subramaniam, Shankar (1997). Computation of molecular electrostatics with boundary element methods. *Biophysical journal*, 73(4), 1830–1841. 1.1.5

Lindsay, Keith y Krasny, Robert (2001). A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. *Journal of Computational Physics*, 172(2), 879–907. 1.2.4, 1.3

Lorman, VL y Rochal, SB (2007). Density-wave theory of the capsid structure of small icosahedral viruses. *Physical review letters*, 98(18), 185502. 1.5.3.2

Lu, Benzhuo; Cheng, Xiaolin; Huang, Jingfang; y McCammon, J Andrew (2006). Order n algorithm for computation of electrostatic interactions in biomolecular systems. *Proceedings of the National Academy of Sciences*, 103(51), 19314–19319. 1.1.5, 3

Lu, BZ; Zhou, YC; Holst, MJ; y McCammon, JA (2008). Recent progress in numerical methods for the poisson-boltzmann equation in biophysical applications. *Commun Comput Phys*, 3(5), 973–1009. 1.1.3

Martínez, Matías; Cooper, Christopher D; Poma, Adolfo B; y V Guzman, Horacio (2019). Free energies of the disassembly of viral capsids from a multiscale molecular simulation approach. *Journal of Chemical Information and Modeling*. 3

Martinez, Matias; Vargas-Guzman, Horacio; y Cooper, Christopher D (2019). Implicit solvent calculations at large-scale virus-level poisson-boltzmann and multiscale simulations for electrostatics. *Biophysical Journal*, 116(3), 291a.

Martínez Moncayo, Carolina (2016). Virus del moteado de la parietaria (pmov). 1.5.3.2

Mason, Edward A (1960). An introduction to statistical thermodynamics. *Journal of the American Chemical Society*, 82(23), 6209–6209. 1.1.3

McLean, William y McLean, William Charles Hector (2000). *Strongly elliptic systems and boundary integral equations*. Cambridge university press. 1.2

Muscio, OA; La Torre, JL; y Scodeller, EA (1988). Characterization of triatoma virus, a picorna-like virus isolated from the triatomine bug triatoma infestans. *Journal of General Virology*, 69(11), 2929–2934. 3.3.1

Olsson, Mats HM; Søndergaard, Chresten R; Rostkowski, Michal; y Jensen, Jan H (2011). Propka3: consistent treatment of internal and surface residues in empirical p k a predictions. *Journal of chemical theory and computation*, 7(2), 525–537. 2.3

Orozco, Modesto y Luque, F Javier (2000). Theoretical methods for the description of the solvent effect in biomolecular systems. *Chemical Reviews*, 100(11), 4187–4226. 1.1.3

Patterson, David A; Hennessy, John L; y Goldberg, David (1990). *Computer architecture: a quantitative approach*, volume 2. Morgan Kaufmann San Mateo, CA. 1.4.1

Phillips, Rob y Quake, Stephen R (2006). The biological frontier of physics. *Physics Today*, 59(5), 38–43. 1.5.3

Riordan, JF; McElvany, KD; y Borders, CL (1977). Arginyl residues: anion recognition sites in enzymes. *Science*, 195(4281), 884–886. 3.2.4

Rochal, SB; Konevtsova, OV; Myasnikova, AE; y Lorman, VL (2016). Hidden symmetry of small spherical viruses and organization principles in "anomalous" and double-shelled capsid nanoassemblies. *Nanoscale*, 8(38), 16976–16988. 1.5.3.2

Roux, Benoıt y Simonson, Thomas (1999). Implicit solvent models. *Biophysical chemistry*, 78(1-2), 1–20. 1.1.3, 1.1.4

Sanner, Michel F; Olson, Arthur J; y Spehner, Jean-Claude (1996). Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3), 305–320. 2.2.2

Sharp, Kim A y Honig, Barry (1990). Electrostatic interactions in macromolecules: theory and applications. *Annual review of biophysics and biophysical chemistry*, 19(1), 301–332. 1.1.3

Shaw, PB (1985). Theory of the poisson green's function for discontinuous dielectric media with an application to protein biophysics. *Physical Review A*, 32(4), 2476. 1.1.5

Simonson, Thomas (2003). Electrostatics and dynamics of proteins. *Reports on Progress in Physics*, 66(5), 737. 1.1.3

Sirohi, Devika; Chen, Zhenguo; Sun, Lei; Klose, Thomas; Pierson, Theodore C; Rossmann, Michael G; y Kuhn, Richard J (2016). The 3.8 å resolution cryo-em structure of zika virus. *Science*, 352(6284), 467–470. (document), 3.2

Somigliana, Carlo (1885). Sopra l'equilibrio di un corpo elastico isotropo. *Il Nuovo Cimento (1877-1894)*, 17(1), 140–148. 1.2

Squires, Gaeelle; Pous, Joan; Agirre, Jon; Rozas-Dennis, Gabriela S; Costabel, Marcelo D; Marti, Gerardo A; Navaza, Jorge; Bressanelli, Stéphane; Guérin, Diego MA; y Rey, Felix A (2013). Structure of the triatoma virus capsid. *Acta Crystallographica Section D: Biological Crystallography*, 69(6), 1026–1037. (document), 2.2.1, 2.1

Steinbach, Olaf (2007). *Numerical approximation methods for elliptic boundary value problems: finite and boundary elements*. Springer Science & Business Media. 1.2, 1.2.2

Steppert, Petra; Burgstaller, Daniel; Klausberger, Miriam; Tover, Andres; Berger, Eva; y Jungbauer, Alois (2017). Quantification and characterization of virus-like particles by size-exclusion chromatography and nanoparticle tracking analysis. *Journal of Chromatography A*, 1487, 89–99. 3.3.3

Stern, Otto (1924). Zur theorie der elektrolytischen doppelschicht. *Zeitschrift für Elektrochemie und angewandte physikalische Chemie*, 30(21-22), 508–516. 1.1.3

Still, W Clark; Tempczyk, Anna; Hawley, Ronald C; y Hendrickson, Thomas (1990). Semianalytical treatment of solvation for molecular mechanics and dynamics. *Journal of the American Chemical Society*, 112(16), 6127–6129. 1.1.5

Strub, Caroline; Alies, Carole; Lougarre, Andrée; Ladurantie, Caroline; Czaplicki, Jerzy; y Fournier, Didier (2004). Mutation of exposed hydrophobic amino acids to arginine to increase protein stability. *BMC biochemistry*, 5(1), 9.

Thompson, Darcy Wentworth et al. (1942). On growth and form. *On growth and form.* 1.5.3

Tomasi, Jacopo; Mennucci, Benedetta; y Cammi, Roberto (2005). Quantum mechanical continuum solvation models. *Chemical reviews*, 105(8), 2999–3094. 1.1.3

Van Der Spoel, David; Lindahl, Erik; Hess, Berk; Groenhof, Gerrit; Mark, Alan E; y Berendsen, Herman JC (2005). Gromacs: fast, flexible, and free. *Journal of computational chemistry*, 26(16), 1701–1718. 3.3.2

Viso, Juan Francisco; Belelli, Patricia; Machado, Matías; González, Humberto; Pantano, Sergio; Amundarain, María Julia; Zamarreño, Fernando; Branda, Maria Marta; Guérin, Diego MA; y Costabel, Marcelo D (2018). Multiscale modelization in a small virus: Mechanism of proton channeling and its role in triggering capsid disassembly. *PLoS computational biology*, 14(4), e1006082. 3, 3.3.1, 3.3.2, 3.3.3, 3.3.3

Wagoner, Jason A y Baker, Nathan A (2006). Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms. *Proceedings of the National Academy of Sciences*, 103(22), 8331–8336. 1.1.4

Wang, Junmei; Cieplak, Piotr; y Kollman, Peter A (2000). How well does a restrained electrostatic potential (resp) model perform in calculating conformational energies of organic and biological molecules? *Journal of computational chemistry*, 21(12), 1049–1074. 2.2.1

Warwicker, J y Watson, HC (1982). Calculation of the electric potential in the active site cleft due to α-helix dipoles. *Journal of molecular biology*, 157(4), 671–679. 1.1.5

Watson, James D y Crick, Francis HC (1953). The structure of dna. In *Cold Spring Harbor symposia on quantitative biology*, volume 18 (pp. 123–131).: Cold Spring Harbor Laboratory Press. 1.5.3.2

Wolfenden, Richard (1983). Waterlogged molecules. *Science*, 222(4628), 1087–1093. 3.2.4

Xu, Zhenli y Cai, Wei (2011). Fast analytical methods for macroscopic electrostatic models in biomolecular simulations. *SIAM review*, 53(4), 683–720. 1.1.3

Yoon, Byung Jun y Lenhoff, AM (1990). A boundary element method for molecular electrostatics with electrolyte effects. *Journal of Computational Chemistry*, 11(9), 1080–1086. 1.1.5

Zandi, Roya; Dragnea, Bogdan; Travesset, Alex; y Podgornik, Rudolf (2020). On virus growth and form. *Physics Reports*, 847, 1–102. 1.5.3