DADES DEL PROJECTE

Títol del Projecte: GRID Superscalar aplicat a la química computacional.

*Nom de l'estudiant:*Joan Díaz Capell

Titulació: Enginyeria Informàtica

Crèdits: 37.50

Director/Ponent: Ignasi Belda Reig / Eduardo Ayguadé Parra

Departament: Arquitectura de Computadors

MEMBRES DEL TRIBUNAL*(nom i signatura)*

President: Jordi Torres Viñals.

Vocal: Juan Luis Esteban Ángeles.

Secretari: Eduardo Ayguadé Parra.

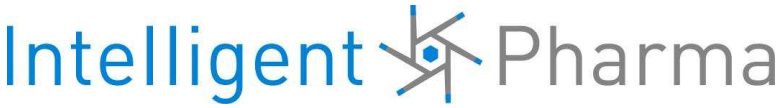
QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

GRID Superscalar aplicat a la química computacional

Joan Díaz Capell



Índex

Capítol 1

Introducció

La **química computacional** és la branca de la química que se n'encarrega de la utilització d'entorns computacionals per la resolució de problemes químics. Un dels problemes químics típics en la recerca de nous fàrmacs és la predicció de com s'uneix la molècula que ha d'actuar com a fàrmac amb la seva diana terapèutica (normalment una proteïna). I una aproximació comuna per a la resolució d'aquest tipus de problemes és el *docking*, en contraposició a altres aproximacions com la dinàmica molecular [1]. Tot i això, el docking dista encara bastant de ser una ciència exacta [2, 3, 4], i és per aquest motiu que, hi ha gran interès, tant al món acadèmic com a l'industrial, de millorar aquestes eines. Per tant, l'**objectiu del present projecte és crear un aplicatiu marc que doni suport computacional i estadístic a la recerca en la millora d'eines de docking al grup de recerca en química computacional de l'empresa Intelligent Pharma, S.L**.

Un dels problemes en el que es troben els nostres experts quan utilitzen les aplicacions de *docking*, que tenen un cost computacional molt elevat, és que l'execució d'una d'aquestes aplicacions pot tardar varies hores i normalment hom no vol fer només una simple execució sinó que en necessita varies per poder comparar resultats, estudiar els diferents casos, etc. i el fet d'executar de forma seqüencial els diferents *dockings* fa que aquest treball sigui realment molt costos i lent. Un altre problema en el qual ens trobem és que un cop executats els diferents experiments, aquestes aplicacions de *docking* guarden els resultats en fitxers plans de text de gran mida que son poc còmodes de tractar degut al gran volum de dades que contenen, llavors per treballar amb aquests fitxers de sortida, l'usuari es veu obligat a recollir manualment les dades que li interessen si vol contrastar les dades o fer algun tipus de càlcul estadístic. Per tant, veiem que si es vol arribar a executar un procediment típic com el que hem explicat breument, que podria consistir de: varies execucions de *dockings*, comprovar i recollir els resultats dels fitxers de sortida i analitzar les dades d'un mateix experiment o comparar dades entre varis experiments; això ens portaria, de ben segur, varies setmanes de treball i a més estaríem usant varis programes diferents a la vegada per poder fer totes les tasques.

Així doncs, el nostre objectiu ha consistit en introduir una d'aquestes eines de *docking* dins d'una aplicació informàtica. Aquesta aplicació mitjançant eines/mecanisme de paralització (en el nostre cas GRID Superscalar[1]) és capaç d'executar varis d'aquests *dockings* en paral·lel, en una mateixa màquina i/o en varies màquines a la vegada de forma distribuïda, ja sigui

[1]En parlarem en el capítol 4

dins d'un grid o d'un clúster. Així, en el temps que anteriorment es tardava en calcular un *docking* amb la nostre aplicació l'usuari serà capaç d'executar-ne varis, generant uns beneficis temporals, que per simplificar, direm que són proporcional al número de màquines que l'usuari disposa i al número de processadors que aquestes tinguin. I no només això, sinó que a més l'aplicació també disposarà d'un suport per emmagatzemar les dades obtingudes en cada una d'aquestes execucions, i posa a disposició de l'usuari diferents eines per gestionar les dades recollides o per realitzar anàlisis i comparacions estadístiques. Per tant, el que hem fet és que tant la gestió de les execucions, la gestió de la base de dades i els càlculs estadístics estiguin és una sola aplicació, centralitzant diversos recursos, i que a més a aquesta s'hi pugui accedir a través d'una interfície amigable i lliure de dependències arquitectòniques per l'usuari, és a dir una interfície web.

Per tant, aquesta aplicació pretén ser una eina per a químics i està destinada en tot moment a facilitar la seva feina dins del camp del *docking* de varies maneres:

- Afegim **comoditat** alhora de llençar els càlculs, ja que aquesta funció es fa a través d'una interfície còmoda.

- Ajudem a **recol·lectar i contrastar** els resultats ja que els fitxers de sortida són *parsejats* i emmagatzemats automàticament a una base de dades i a més ara aquestes dades poden ser visualtizades amb un cop d'ull a través de la mateixa interfície.

- Posem a disposició de l'usuari un seguit d'**anàlisis i comparacions estadístiques** les quals poden ser visualitzades gràficament amb facilitat.

- Eliminem la **diversificació** d'aplicacions, perquè tot està centralitzat en un mateix software, al qual si pot accedir a través d'una web.

- Augment de l'**eficiència** (sense perdre eficàcia), com ja hem comentat abans, ara som capaços de realitzar en una sola tasca el que abans en necessitaven moltes més i això augmenta el rendiment de l'usuari que treballa amb aquest tipus d'eines de *docking*. A més, de forma transparent a l'usuari hem intentat treure encara més rendiment dels recursos informàtics:

  - Hem optimitzat al màxim l'aplicació de *docking* que utilitzem.

– S'ha estudiat la millor configuració del Grid Superscalar perquè els *dockings* s'executessin amb el mínim cost temporal sense sobrecarregar les màquines.

Perquè ens poguem acabar de situar com s'utilitza la nostre aplicació i ser una mica més concrets, em de tenir clar que l'ús principal per la qual està dissenyada és, en un inici, per realitzar investigació i desenvolupament dins la pròpia empresa. És a dir, pretenem ser capaços d'executar un gran número de càlculs de *docking* el més ràpid possible mitjançant l'aplicació *AutoDock*[2], i llavors un cop realitzades aquestes simulacions, comparem i analitzem els resultats amb els valors experimentals — aquests són, per dirho d'altre manera, valors experimentals realitzats en laboratoris bioquímics —. Així doncs, aquest procés ens permet realitzar tot un seguit de millores i optimitzacions sobre l'aplicació de *docking*, l'*AutoDock*, per intentar que aquest sigui capaç de realitzar simulacions cada cop més reals i precises. Aquest procés l'anomenem *Benchmarking* o banc de proves.

Un exemple molt conegut que succeïx en aquests tipus d'aplicacions que provoca que aquestes simulacions siguin **menys** reals, és el fet que per realitzar la simulació s'eliminen totes les molècules d'aigüa que puguin contenir les dues molècules que s'han d'ancorar una amb l'altra, és ben cert que en la realitat existeix aquest fenomen, però per realitzar les simulacions s'extreu l'aigua perquè sinó els algorisme de cerca de *docking* per trobar les millors conformacions, augmentarien massa el temps i podrien arribar a perdre molta eficàcia, és a dir l'espai de cerca es faria tant gran que no arribarien a trobar bones conformacions. Per aquest motiu s'extreu l'aigua perquè així, almenys, encara que la simulació de l'ancoratge entre les dues molècules no sigui el més real possible, i per tant poc eficient, si que és eficaç, és a dir l'algorisme és capaç de trobar bones conformació amb un temps relativament assequible, però sense aigües. Ara mitjançant l'aplicació que hem desenvolupat en aquest projecte, podem investigar en la creació d'altres eines destinades, per exemple, a afegir aigua en el medi i fer que les simulacions de l'*AutoDock* s'aproximin més a la realitat.

Un cop vist de forma més específica l'ús de l'aplicació desenvolupada, a continuació seguirem analitzant les eines que actualment hi han al mercat destinades a realitzat el mateix o quelcom semblant a la nostre. Primer de tot, hem de tenir en compte que ja existeixen eines d'aquest tipus en el mercat, moltes d'elles estan dedicades única i exclusivament a la part de

[2]En el capítol 5 explicarem més explícitament com treballa aquesta aplicació.

"GRID computing" i a la distribució del treball per augmentar el rendiment ("throughput") d'algun programa. Aquest fet, és ben normal dins de l'entorn en el que ens trobem, i això succeeix degut a que la gran majoria de les aplicacions que s'utilitzen en l'investigació acostumen a ser eines amb uns costs computacionals realment elevat, i per tant el primer pas que hom realitzar per duu a termes investigacions a més gran escala és, clarament, la distribució en paral·lel de les diferents execucions, sinó l'investigació es faria inviable. Existeixen varies eines d'aquests estil que es dediquen a paral·lelitzar i distribuir el treball en "grids" com en "clouds" de forma molt robusta i dedicada, amb interfícies gràfiques amigables i amb independència del programa que es vulgui executar. Com per exemple l'eina **Fura** de GridSystems[3], el portal web **P-GRADE** (http://www.p-grade.hu/) del "*Computer and Automation Research Institute Hungarian Academi of Sciences*"[4], **Condor** de l'universitat de Wisconsin, etc. Totes aquestes eines són realment molt bones alhora de treballar amb "grids" i/o "clouds", però cap d'aquestes aplicacions és específica per ésser executada en cap camp en concret, estan preparades per suportar execucions de qualsevol aplicació i prou.

A part d'aquestes eines especialitzades en temes de "GRID computing" també hem de dedicar un petit espai a parlar d'altres les quals s'han fet en torn del *docking*, com és el nostre cas. En aquest apartat tenim varis programes dels quals, destaquem el **FightAIDS@Home**[5], una eina desenvolupada per "The Olson Laboratory" en la qual permet que tota persona es pugui instal·lar en el seu ordinador un software que s'executa quan el/s processador/s de l'ordinador estan en estat ociós, el qual mitjançant l'*AutoDock* ajuda en la recerca i el descobriment de nous fàrmacs per la SIDA i d'altres tipus d'investigacions al voltant del HIV. No tots aquests programes estan dedicats a l'investigació pura i dura, com la nostra, també hi ha d'altres els quals estan dedicats al cribatge virtual ("Virtual Screening"), un procés que es dedica al descobriment de fàrmacs ("drug discovery") com l'eina **DOVIS** de "BioMed Central Ltd" i d'altres més que entorn aquesta temàtica intenten augmentar el rendiment de l'*AutoDock* per poder treballar amb bases de dades de molècules que permeten tenir més possibilitats alhora d'encaminar un procés per dissenyar i comercialitzar un nou fàrmac.

Tot i que l'aplicació que hem desenvolupat també és experta en *docking* i també augmenta el rendiment de l'*AutoDock*, no és tant escalable com les

[3]http://www.gridsystems.com/
[4]http://www.lpds.sztaki.hu/
[5]http://fightaidsathome.scripps.edu/

demés aplicacions dedicades a "GRID computing" ja que no permetem l'execució de cap altre programa que no sigui l'*AutoDock*; ni tant específica en una sola malaltia com la de FindingAIDS@Home. Però a canvi, permet total llibertat alhora de generar recerca entorn qualsevol diana terapèutica; està focalitzada, a priori, en investigació i desenvolupament i no al descobriment de fàrmacs com la majoria. A més de tot això, el que ens fa ser diferents de les demés i per tant aquest és l'aspecte que més valorem de l'aplicació per ser el seu punt fort, és que oferim: tota una part dedicada al suport i gestió d'una base de dades amb els diferents resultats obtinguts en cada execució de l'*AutoDock*, diversos càlculs estadístics (d'anàlisis i comparació) amb tot el volum de dades emmagatzemat en la base de dades i a més, també com els altres, oferim l'execució d'un programa de *docking* de forma distribuida i paral·lela en un "grid" o clúster i, esperem que d'aquí poc temps també es pugui executar en "clouds"[6] per poder donar una eficiència computacional encara més elevada que l'actual.

Per seguir un ordre, per poder diferenciar entre la part més tecnològica de la química i d'altres, i facilitar la lectura d'aquesta memòria, hem estructurat el treball dividint-lo en 4 part ben diferenciades de la següent manera:

1. **Introducció** inclou un capítol dedicat a explicar el procés químic de *docking*.

2. **Tecnologia**, la qual consta dels capítols 3, 4, 5 i 6.

3. **Resultats** i exemples d'estudis aconseguits mitjançant l'aplicació, capítol 7.

4. **Conclusions** obtingudes, recollides en l'últim tema.

[6]En el capítol 8 s'explica de forma més detallada.

Capítol 2

Docking

2.1 Introducció

El docking[1] computacional molecular és una tècnica per predir si una molècula encaixarà amb una altre molècula receptora, normalment una proteïna, i en cas de fer-ho, ens diu també com ho farà. Les prediccions de dockings entre proteïna-proteïna, proteïna-ADN i proteïna-lligand, ja estan implementades, encara que les tècniques usades en cada àrea són molt variades. El docking proteïna-lligand es realitza modelant l'interacció entra la proteïna i el lligand: si la geometria de les dues molècules és complementària i es donen interaccions favorables, el lligand encaixarà potencialment la proteïna in vitro o in vivo. Normalment, els programes de docking també retornen l'energia lliure d'unió.

En aquest projecte, el docking és la part més important, ja que tota la nostre aplicació gira en torn a aquest càlcul químic i a més és l'encarregat d'afegir un alt cost computacional a l'aplicació. Per aquesta raó, aquesta és la part que intentarem optimitzar (veure la secció 5.2) i paral·lelitzar (veure el capitol 3). De tota manera, hi ha una gran varietat de programes de docking. Pel nostre cas, usarem l'AutoDock (veure capítol 5), el qual ens dona una sèrie de ventatges que ja explicarem en el seu degut moment.

Normalment la documentació del docking divideix aquesta metodologia en dues parts independents:

- Els algorisme de cerca.
- Les funcions d'avaluació.

Els espais de cerca on el docking busca les millors conformacions del lligand són multidimensionals i multimodals, per tant, calen algorismes de cerca robustos i potents. L'algorisme de cerca usat, busca la conformació d'aquest lligand que té una millor puntuació quan és classificat mitjançant una funció d'avaluació. Actualment també hi ha un altre mètode de docking que ha sorgit fa relativament poc: la reavaluació o avaluació consensuada [5, 6]. La reavaluació només realitza una funció d'avaluació per liderar la cerca, però llavors, les conformacions obtingudes són reavaluades utilitzant varies altres avaluacions. Aquesta aproximació s'utilitza per reduir el nombre de falsos positius.

Els algorismes de cerca han d'explorar de manera eficaç tot el que la funció d'avaluació abasteixi en un temps de CPU raonable. Per aquesta raó, les

[1] ancoratge

cerques sistemàtiques de les conformacions van ser abandonades ràpidament en detriment de mètodes estocàstics com és el cas de Monte Carlo, "simulated annealing" i algorismes evolutius; o altres estratègies com construccions incrementals (mitjançant triangulacions de Delaunay) o distàncies geomètriques[7]. Actualment es creu que les avaluacions semblen ser més exigent que no pas la cerca i per altra part, no queda clar si els problemes de les cerques estan del tot resolts, per aquest motiu sistemàticament s'examina com d'exhaustives són les cerques dins del context del docking [4, 2], encara que un estudi recent conclou que tant l'avaluació com la cerca en el docking són igual d'exigents [3]. Quan un experiment de docking falla, és important saber si és degut per una limitació de la cerca o de l'avaluació. De totes maneres, això no es representa amb un simple anàlisis, ja que no es possible seguir el RMSD(Root Mean Sqared Deviation[2]) des d'una estructura referencial durant l'experiment de docking, i tampoc és possible realitzar altres experiments de docking equivalents amb funcions d'avaluació diferents.

La funció d'avaluació ideal ha de calcular de forma exacta les energies lliures d'unió però, generalment no hi ha cap funció d'avaluació que s'hi acosti tant. Les funcions d'avaluació estan basades en l'assumpció que l'afinitat d'unió pot ser descrita com la suma dels paràmetres independents. Per aquesta raó, totes les funcions d'avaluació tenen una certa dependència a la mida de l'avaluació: com més gran sigui la molècula, major és la probabilitat de que es puntuï (avaluï) favorablement. Després, hem de tenir en compte que les funcions d'avaluació menyspreen, en gran mesura, efectes entròpics en l'avaluació de les interaccions entre un receptor rígid i un sol lligand i d'altres factors més complexos. Per tant, a més de les consideracions anteriors, hem de tenir en ment que bons complements entre receptor i lligand són segur prerequisit per l'unió, però no un criteri suficient.

L'energia lliure d'unió (ΔG), es pot definir com l'energia associada amb el treball invers necessari d'unir dues molècules, i pot ser interpretada com una costant d'equilibri (K) que descriu el comportament d'un sistema dinàmic depenent de la temperatura (T), *ex:* quina proporció de molècules s'uniran i quina proporció, complementaria, no ho farà. (En l'equació 2.1, R és la constant universal del gas).

$$\Delta G = -R \cdot T \cdot lnK \tag{2.1}$$

L'energia lliure d'unió és en resum l'entalpia d'un complexe — l'afinitat d'un lligand amb el seu receptor — i l'entropia — una penalització que es

[2]Desviació del valor o mitja quadràtica

paga perquè un sistema unit té menys *mobilitat* i d'altres paràmetres relacionats amb la desolvatació —. La penalització es paga degut a la pèrdua de mobilitat dels lligands correlacionats amb el seu número d'enllaços flexibles. De totes maneres, en el càlcul d'un docking, aquesta penalització té el mateix valor per totes les conformacions examinades—la molècula examinada pel programa de docking, és sempre la mateixa—, llavors aquests programes eviten el càlcul d'aquest paràmetre durant la cerca de conformacions. L'equació 2.2 mostra una nova definició de l'energia lliure d'unió, on $E_{intermolecular}$ representa la contribució entàlpica en una formació complexa més altres contribucions entròpiques minoritàries relacionades amb els efectes de la desolvatació, i $E_{torsional}$ representa la penalització entròpica degut a les capacitats de torsió del lligand.

$$\Delta G = E_{intermolecular} + E_{torsional} \tag{2.2}$$

Després d'això, els algorismes de docking han d'assegurar que entre la millor predicció de les conformacions no hi han conformacions amb enfrontaments. Per aquesta raó, els programes de docking incorporen dins dels càlculs de l'energia de conformació un paràmetre que conté l'energia interna per totes les conformacions d'una molècula, l'equacio 2.3 il·lustra el concepte d'*energia de docking*, la funció que els programes de docking realment volen optimitzar.

$$E_{docking} = E_{intermolecular} + E_{intramolecular} \tag{2.3}$$

L'últim, però no menys important són les molècules d'aigua, que podrien tenir un paper clau en els events d'unió. Una possible estratègia que inclueix tant els efectes dissolvents com els patrons de flexibilitat, és l'ús de simulacions de molècules dinàmiques amb un camp de força [8]. Malauradament, aquests tipus de càlculs només es poden tractar amb unes poques configuracions inicials del lligand receptor degut al alt cost computacional.

2.2 Estat de l'art

En aquesta secció realitzarem un repàs a l'estat de l'art del docking i ens centrarem en el docking proteïna-lligand — el tipus de docking usat en el nostre cas—. Per aquesta raó, les aplicacions i adaptacions tals com proteïna-proteïna o proteïna-DNA no seran examinades. El repàs està separat en quatre seccions. El primer de tots és un anàlisis dels diferents algorismes de cerca usats en els programes de docking, en la següent secció fem

un estudi de varies funcions d'avaluació, després comparem diferents programes de docking que hi han, especialment aquells en els que l'AutoDock hi està implicat i a finalment investiguem algunes aplicacions biomèdiques de docking.

2.2.1 Algorismes de Cerca

Un dels primers estudis comparatius sobre els algorismes de cerca va ser publicat per Ewing i Kuntz [4] el 1997. El seu objectiu en aquest estudi era trobar el millor algorisme de cerca per incorporar-lo en el seu programa de docking DOCK 4.0. DOCK 4.0 representa el problema de docking amb grafs unidireccionals: hi ha un graf per cada proteïna diana i un graf que representa el lligand. Per tant, la tasca de l'algorisme de cerca és trobar la millor sol·lució pels dos grafs. Per realitzar aquest objectiu, s'analitzaven cinc algorismes diferents de cerca, incloent la cerca exhaustiva. A més, cada una dels algorismes de cerca utilitza altres algorisme per determinar si l'ajust és el més òptim. Finalment, es va escollir l'algorisme de cerca "single graph matching", el qual en el seu algorisme per determinar l'ajust òptim utilitza el número d'ajustos i l'actual orientació de cada molècula alhora.

En un document més recent i complet de Halperin *et al.* [7], els autors divideixen els algorismes de cerca en dues parts: (1) aquells algorismes que recorren l'espai de sol·lució de forma entera d'una manera predefinida; i (2) aquells algorismes que només recorren parts de la sol·lució de forma, en part aleatòria i en part guiats per algun criteri. El segon grup bàsicament és Monte Carlo, "simulated annealing, algorismes evolutius, etc. Òbviament, per molècules de gran flexibilitat el primer grup d'algorismes són inacceptables a nivell de temps de computació. Les aplicacions més actuals de docking tals com DOCK [4, 9], FlexX [10], GOLD [11], AutoDock [12] o ICM [13], tots utilitzen algorismes de cerca com els descrits en el segon grup.

DOCK 4.0.1 utilitza un algorisme d'ajust esfèric per adequar els àtoms del lligand en una esfera. Per tant, DOCK 4.0.1 utilitza un algorismes incremental per trobar l'ajust òptim, situant els lligands flexibles en l'esfera. De forma similar FlexX usa un altre algorisme incremental per ajustar els lligands flexibles amb els elements complementaris de la superfície de la proteïna. Per altra banda, tant GOLD com AutoDock utilitzen algorismes genètics per evolucionar la població aleatòria inicial de conformacions que s'uneixen amb la superfície. Al final del procés d'evolució, els dos programes arriben a la millor conformació trobada durant el procés.

2.2.2 Funcions d'avaluació

Els algorismes de cerca produeixen un número enorme de sol·lucions potencials. Per distingir les bones conformacions d'entre les dolentes, el procés de docking necessita eines que siguin capaces de donar un càlcul aproximat de l'avaluació de totes les conformacions. De forma teòrica, l'energia lliure pot ser un indicador força fiable alhora de discriminar i revisar sol·lucions. No obstant, no és factible usar aquest enfoc en cerques de docking — una de les raons és que una part important de les contribucions de l'entropia són constants per totes les conformacions de la mateixa composició i a més, és car realitzar càlculs acurats de les energies lliures —.

Diferents programes implementen el seu propi algorisme d'avaluació, i en la majoria dels casos estem exposats a que aquests algorismes arribin a donar una sol·lució incorrecte i per tant, ens trobaríem davant de falsos positius. Després d'analitzar diferents algorismes d'avaluació, una de les conclusions a les que arriba Ferrara *et al.* [14] és que AutoDock té implementada la millor, ja que presenta un percentatge d'encert d'entre el 80% i el 90%, depenent de la diana.

2.2.3 Comparació entre diferents programes de docking

Per exemple, en l'article de Taufer *et al.* [15], els autors desenvolupen una nova proposta de docking basada en molècules dinàmiques. Per demostrar que el seu algorisme era competitiu, van comparar el seu rendiment i precisió contra l'AutoDock, DOCK, FlexX, ICM i el GOLD. Excloent el seu mètode de les comparacions, podem observar que l'AutoDock és el segon millor programa de docking dels cinc anteriors — ICM és el millor d'ells. Vam exloure el seu mètode de la següent comparació seguint el consell de Cole *et al.* [16], ja que en l'article de Cole *et al.*, l'autor aporta un seguit de normes/consells per comparar programes de docking.

Una altre comparació interessant que es realitza periòdicament sobre programes de docking és el CAPRI, sigles de "Critical Assessment of PRediction of Interactions" [17], unes sèries d'events que s'estan portant a terme on els investigadors de tot el món intenten ancorar els mateixos complexes, proveïts pels assessors. Cada sèrie es realitza cada sis mesos i cada una conté entre una i sis dianes complexes proteïna-proteïna, les estructures de les quals han sigut recentment determinades de forma experimental. De tota manera, les comparacions de docking publicades per CAPRI inicialment no són aplicables

directament fins que el problema és un docking del tipus proteïna-proteïna. En aquest cas, l'AutoDock no es compara perquè no està preparat per treballar en aquest tipus d'entorns. Així doncs, d'aquestes comparacions podem veure com rendeixen els competidors de l'AutoDock, com és el cas de l'ICM.

Per totes les cites, revisions, comparacions i pel nombre de publicacions biomèdiques que utilitzen l'AutoDock, considerem l'AutoDock com un software de mitja/alta qualitat. Els tres avantatges principals que té són, (1) és gratuït en recerca acadèmica; (2) podem introduir-li lliurement algorismes i/o modificacions químico-físsiques; i (3) i a més ens proveeix d'eines que ens ajuden a preparar els lligands per ser ancorats. Aquesta darrera qüestió és important en casos on la molècula presenta una gran flexibilitat, la segona qüestió ens és de gran ajuda també, ja que s'hi han introduït varies modificacions, algunes d'optimització que s'expliquen en l'apartat 5.2 i d'altres realitzades per l'empresa que no s'expliquen en aquest projecte.

2.2.4 Aplicacions de docking en la biomedicina

En els últims anys, el procés de docking s'ha usat de forma extensa en la biomedicina, ja sigui per descobrir nous fàrmacs, per portar a terme dissenys basats en l'estructura de molecules bioactives o per altres aplicacions [18, 19, 20, 21, 22, 23, 24, 25]. AutoDock és un dels programes més usats d'entre aquest tipus d'aplicacions, de fet, segons Sousa *et al.* [26], AutoDock és el programa de docking més citat en la literatura científica. Els autors utilitzen l'"ISI Web of Science" per contar el nombre de vegades que es cita alguna de les referències originals dels 22 programes de docking. En el 2005 l'autoDock era el més citat amb una 27% de les citacions seguit per GOLD (15%), FlexX (11%), DOCK (6%) i ICM (6%).

A partir del que hem comentat, podem concloure que actualment, AutoDock s'utilitza en un de cada dos estudis involucrats en el docking. Per aquesta raó, és quasi impossible realitzar una completa revisió dels casos en els que l'Autodock ha sigut usat. Per tant, només hem realitzat una petita repassada i ens hem centrat en el procés el qual usem.

Capítol 3

Daedalus

Daedalus, és el nom que rep l'aplicació que hem dissenyat, però no només això, sinó que a més dins d'aquesta mateixa nomenclatura també hem incluit l'arquitectura la qual s'ha escollit perquè hi anés a sobre, ja que més endavant aquesta mateixa arquitectura ens servirà per altres programes corporatius.

3.1 Arquitectura

Volíem que l'aplicació fora capaç de corre en paral·lel en diferents màquines, per tant també podríem dir que l'aplicació és, en part, distribuïda. A més, ens interessava que es pogués dur a terme tant en un "grid" com en un clúster. Actualment tenim un clúster, totes les màquines són arquitectònicament semblants, disposem de proximitat física i s'ha pogut crear un "NFS"[1] amb totes les màquines, sense grans problemes de seguretat ja que totes les màquines pertanyen a la xarxa privada del PCB[2], on estem ubicats, però és ben segur que en el futur ens interessarà que aquesta mateixa organització es pugui aprofitar per treballar en "grid" ja que a mesura que tinguem més màquines noves segur que aquest conjunt començarà a ser heterogeni i s'hauran de recompilar els programes per altres arquitectures, ens serà més difícil treballar amb "NFS" degut a qüestions de seguretat... Tot plegat, el fet de treballar en "grid" ens simplificaria la tasca, llavors tindríem binaris per cada màquina, discos propis pels "workers", etc. Així doncs, la pròpia arquitectura de l'aplicació està pensada i preparada per donar aquest pas, perquè com estem parlant de programes amb uns costos computacionals molt elevats, és ben segur que necessitarem un gran número de processadors per poder executar tants treballs alhora com sigui possible i així poder augmentar el nostre rendiment.

En el cas que ens ocupava, es va usar el clúster de l'empresa i a més a més se li van afegir altres màquines de l'empresa per poder treure més rendiment. Així som capaços d'executar més AutoDocks alhora. Com totes les màquines i el clúster estan en la mateixa xarxa va ser relativament fàcil ferho. Es va crear un servidor "NFS", que tots els diferents "workers" utilitzen per poder-hi treballar, així poden generar fitxer i guardar sortides vàlides d'una execució; després per no carregar una sola màquina amb totes les demandes, un altre màquina fa de servidor d'un nou "NFS" on estan publicats els binaris que els diferents "workers" executen per duu a terme la tasca.

[1] Network File System

[2] Parc Científic de Barcelona, estructura del sistema d'innovació creada per la Universitat de Barcelona, amb el suport de la Fundació Bosch i Gimpera i la Caixa Catalunya.

A més, les diferents màquines s'intercanvien informació entre elles mitjançant "ssh", en aquest cas ho varem preferí així, per sobre de la creació d'un "Web service" — que vindria a ser l'altre alternativa més clara pel problema al qual ens afrontàvem — ja fos amb "SOAP"[3] o "REST"[4], ja que són a priori independents de la tecnologia que s'utilitza, el primer perquè va sobre XML i l'altre sobre HTML. De totes maneres, ho varem decidir així perquè: aprofitàvem les connexions de "ssh" que havíem creat pel funcionament del **GRID Superscalar**[5], perquè el "Web service" ens introduïa un "overhead" massa elevat per realitzar les funcions pel qual el necessitàvem, i a més perquè la màquina encarregada de publicar aquest serveis hauria hagut de ser el màster del GRID Superscalar i això no ens sortia a compte, ja que la càrrega que aquest podria arribar a patir seria elevada com ja hem comentat just abans.

A partir d'aquí, podríem separar les diferents màquines en 4 parts diferents, les quals podrien estar solapades en alguns casos (figura 3.1).

- Servidor web
- Servidor GRID Superscalar(master GRID Superscalar)
- Màquines del clúster/"grid" ("workers")
- Servidor base de dades.

Hi ha un petit detall que l'usuari sí que ha de tenir en compte dins d'aquesta arquitectura alhora de generar qualsevol execució, i és el fet que ha de respectar una sèrie de paràmetres dins de l'organització de les dades que s'utilitzen com a fitxers d'entrada per l'AutoDock. Únicament l'usuari ha d'agrupar l'entrada en una mateixa carpeta i dins d'ella ha de generar tantes noves carpetes com AutoDocks s'hagin d'executar. És a dir, en cada una d'aquestes subcarpetes ha d'haver-hi els fitxers que descriuen les dues mol·lecules amb les quals volem realitzar el docking.

3.2 Les tres capes

L'estructura de l'aplicació, l'hem separat en tres parts:

- Capa de presentació

[3]Simple Object Access Protocol
[4]Representational State Transfer
[5]En el capítol 4 es tractarà aquest tema

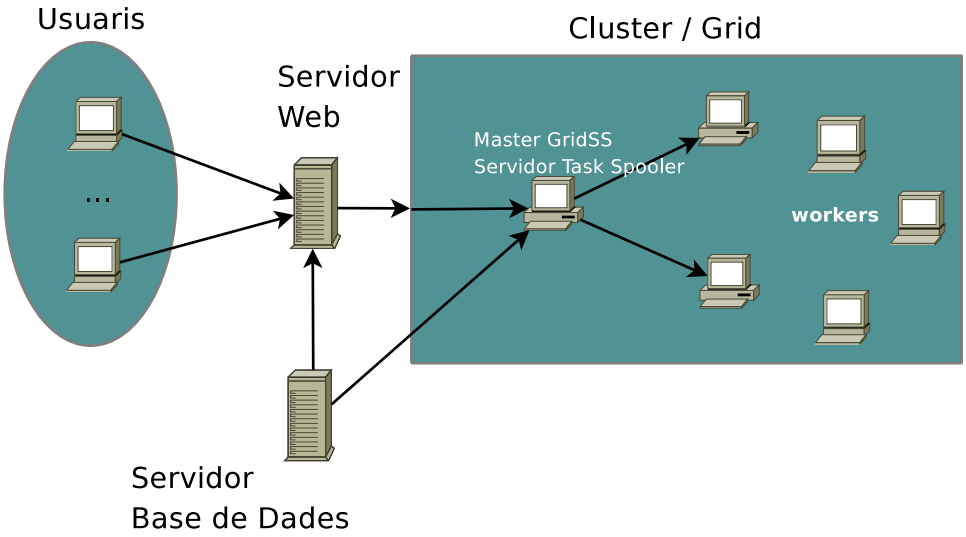


Figura 3.1: Arquitectura del daedalus

- Capa de domini
- Capa de dades

3.2.1 Presentació

Aquesta primera capa, la que és executada per l'usuari, corre sobre un servidor web Apache. La majoria del treball està fet amb PHP sobre HTML, per tant gran part de l'aplicació s'executa en el propi servidor. Tot hi això, en altres apartats de la web, s'hi han afegit altres tecnologies com Java Script per donar-li més dinamisme i interactivitat.

La raó principal per la qual ens varem decidir a crear una interfície web per comunicar-nos amb la capa de domini de l'aplicació, és bastant obvia, ja que aquesta tecnologia ens ofereix poder llançar les execucions des de qualsevol altre ordinador per a qualsevol usuaris. Així, està clar que l'aplicació principal havia de ser remota i a més amb la avantatge que HTML ens ofereix molta independència tecnològica, l'usuari només ha de tenir instal·lat un simple navegador web i prou. En general, s'ha generat una interfície web el més intuïtiva, accessible, etc. possible, perquè fos "user freandly" ja que ha de facilitar als usuaris, en aquest cas químics/químics computacionals ha poder-se moure amb comoditat per la web, de tal manera aquesta primera

capa estarà realitzant els objectius per la qual s'ha dissenyat.

A diferència d'altres aplicacions, la comunicació entre aquesta capa i la de domini es fa mitjançant "ssh", aprofitant els mètodes que el propi PHP porta per executar comandes a través d'aquest protocol/eina. Però tot i tenir aquesta connexió tant peculiar entre capes, el disseny està fet perquè sigui transparent a l'usuari — mitjançant el previ intercanvi de claus privades — i la comunicació no deixí de ser bidireccional. Bàsicament, aquesta capa interacciona amb la següent capa a través de "ssh" amb una cua que hi ha instal·lada en la màquina que fa de master del GRID Superscalar, així podem estar segur que no es fa un ús abusiu dels recursos de les màquines del clúster, donant així el màxim del rendiment de la tecnologia de la qual es disposi en aquell moment alhora d'executar un experiment. Un cop llançada l'aplicació, aquest queda encuada esperant a poder ser executada, i per altre banda, l'usuari pot veure en tot moment en quina posició de la cua es troben els seus experiments, si ja han sigut executats o si en aquest precís instant estan sent executats, tot això a través de la mateixa web.

A més, per protegir i restringir l'ús de la web només als usuaris que hi han de tenir accés, hem afegit un mètode d'identificació d'usuaris. Com no necessitem un control molt acurat dels usuaris, és a dir, no hi han diferents grups, no ens fa falta un usuari administrador, ni tenir cap tipus de sessió oberta ni la necessitat de que els usuaris siguin capaços de registrar-se des de la web; ens va semblar que la millor opció era utilitzar el mòdul **libapache2-mod-auth-mysql** de l'Apache, el qual serveix per tenir un control d'identificació dels usuaris on, tant la seva identificació com el seu "password" estan emmagatzemats en una base de dades MySQL. D'aquesta manera el propi Apache s'encarrega de que no hi hagi cap forat de seguretat, no cal obrir i tancar sessions com es fa amb el control d'usuaris de PHP, etc. Sinó que realitzem el control a nivell de fitxers i no de sessions.

Aquest mètode, consisteix en protegir uns certs directoris on s'hosteja la web dins del servidor i restringir l'accés si l'usuari en qüestió no té un usuari i un "password" associat. Tota la informació corresponent al nom de l'usuari i al password que aquest té estan emmagatzemat en una base de dades que és consultada pel pròpi Apache. Tots els caps necessaris, nom de la base de dades, taula associada, directoris protegits, etc. s'indica en el fitxer de configuració *apache2.conf*.

La web està formada per 4 parts:

1. Experiment Launcher
2. Status
3. Data Base
4. Statistics

La **primera** part permet llançar un experiment, tal com el seu nom indica. Perquè això pugui succeir s'hi han d'especificar una sèrie de camps que l'AutoDock necessita per poder ser executat i després hem de marcar en quines màquines voldrem que s'executi i quants treballs voldrem que s'hi envii per màquina. Per defecte, porta marcada la configuració més òptima que va ser provada i estudiada prèviament. Aquest aspecte està explicat en l'apartat 5.2.2.

La **segona** part és l'encarregada de mostra l'estat en el que es troben les diferents execucions. A part els usuaris a través d'aquest apartat poden mirar què va treure per pantalla l'execució d'algun càlcul, és a dir l'output que es podria veure directament per la sortida estàndard i/o la sortida d'error, podem eliminar treballs de la cua i inclús podem posar un treball a la primera posició.

La **tercera** part tenim la gestió de la base de dades, a partir d'aquí podem eliminar execucions que s'hagin realitzar i veure els resultats obtinguts per les diferents execucions de l'AutoDock en un experiment concret, a més, alhora de visualitzar aquests resultats podem decidir quins camps volem veure o si directament els volem veure tots, també tenim l'opció d'ordenar cada camp de forma ascendent o descendent de forma independent. En aquest cas aquesta capa interaccions directament amb la capa de dades més perquè bàsicament l'operació que més s'executa són simples consultes que no modifiquen l'estat de les dades, per les operacions d'eliminació d'entrades en les taules, s'ha decidit fer-ho des de l'interfície web per raons d'eficiència temporal, ja que connectar-nos de nou per "ssh" amb el domini, que és l'unitat de càlcul més intensiva de l'aplicació global no ens sortia a compte.

La **quarta** i última part, tenim tot una sèrie de càlculs i estudis estadístics que es realitzen amb les dades que s'han extret de les diferents execucions. En aquest cas, podem realitzar un seguit d'anàlisis sobre els resultats d'un mateix experiment o realitzar comparacions entre dos experiments diferents.

3.2.2 Domini

Aquesta capa realment la podríem separar en dues parts ben diferenciades: la part de **càlcul** i la de **parseig** de les dades obtingudes.

A més, per sobre d'aquestes dues parts hi hauria una fina capa que seria l'encarregada d'interaccionar amb la capa de presentació, aquesta estaria formada pel servidor de la cua de la qual parlàvem abans, i a més és l'encarregada de decidir quan tota aquesta capa es pot posar en funcionament i quant no. D'aquesta manera, guanyem certa independència i per tant una mica més d'escalabilitat dins d'aquesta capa.

La part de **càlcul** seria la primera de les dues parts i a més seria la més important de tota l'aplicació, aquí és on és llança la part que crida a GRID Superscalar i fa que els seus "workers" comencin a executar AutoDocks a totes les màquines especificades anteriorment. Això genera un gran nombre de càlculs i a causa del gran cost computacional que es genera varem decidir implementar tota aquesta capa bàsicament amb C++ per intentar treure el màxim rendiment a nivell temporal i espacial, però de totes maneres, aquesta part vindria a ser el coll d'ampolla de l'aplicació i és per això que l'hem tractat d'optimitzar al màxim. En el capítol 5.2 està explicat amb més detall com s'ha optimitzat l'AutoDock. Igualment, tot i el que s'ha comentat anteriorment, aquesta capa no deixa de ser una part senzilla a nivell d'enginyeria del software, perquè realment fora de l'AutoDock i el GRID Superscalar no s'hi han afegit gaire classes més al disseny, el transcurs de l'aplicació sempre és el mateix, la majoria de les excepcions són externes a l'aplicació en sí i això fa que siguin intractables i que no es puguin definir camins alternatius d'execució. Això no vol dir que l'aplicació sigui inestable, però pot passar que alguna execució de l'AutoDock falli o que alguna màquina caigui i GRID superscalar modifiqui el seu comportament, errors típic que tant el GRID Superscalar com l'Autodock ja s'encarreguen de tractar.

La següent part s'executa just quan el GRID Superscalar i els seus "workers" ja han fet el seu treball, llavors tots els fitxers de sortida generats per l'AutoDock són **parsejats**. Es recullen tot un seguit de dades les quals els nostres experts han cregut que són significatives i dignes de formar part d'estudis posteriors. Aquestes dades són enviades a la següent capa, la capa de dades.

3.2.3 Dades

En aquesta última capa tractem tot l'emmagatzematge de dades que la pròpia aplicació hagi de realitzar. En aquest cas, estem parlant de les dades de sortida que genera l'AutoDock i que ja han sigut parsejades prèviament per poder ser tractades en aquest apartat. Totes aquestes dades van directes a un servidor SQL, que en el nostre cas serà MySQL, pel fet de ser gratuït, "open sorce" i per oferir-nos uns serveis més que suficients i de gran qualitat gràcies a la tecnologia **InnoBD** que ens dóna una gran integritat referencial, a més de disposar d'eines de gestió com phpMyAdmin que al nivell al qual ens trobem són més que suficients per duu a terme les tasques pel qual el necessitem.

En aquesta capa, bàsicament el que realitzarem és: un vincle entre l'aplicació i la base de dades i tot un seguit de "querys" per anar introduint les dades a la base de dades. Tot plegat es farà mitjançant les llibreries MySQL++ les quals ens permeten fer les accions anomenades justs abans amb tota comoditat integrant llenguatge SQL dins del codi C++.

Tot seguit podem veure el disseny de la base de dades (figura 3.2). En aquesta mateixa figura 3.2, hem marcat las claus primàries amb les sigles **PK** i les claus forànies amb **FK**, juntament amb el nom de l'atribut i la taula a la qual referencien. A més, totes les claus forànies estan definides amb: "DELETE ON CASCADE". Així cada cop que eliminem un experiment s'eliminaran també totes les entrades on aquest apareix, primer s'eliminen els resultats i aquests després eliminen totes les entrades on estiguin de les següents taules. Aquest és un fet molt important que hem de vigilar que es doni correctament, ja que amb el gran volum de dades que tractem no ens pot donar el cas que hi hagin resultats que no pertanyin a cap experiment i es mantenguin emmagatzemats en la base de dades, ja que això faria augmentar el tamany de la base de dades molt ràpidament de forma innecessària i a més sense cap tipus de control, per tant hem vigilat de respectar al màxim l'integritat referencial (que com ja hem dit abans InnoBD ja ens és de gran ajuda) i la robustesa de les dades.

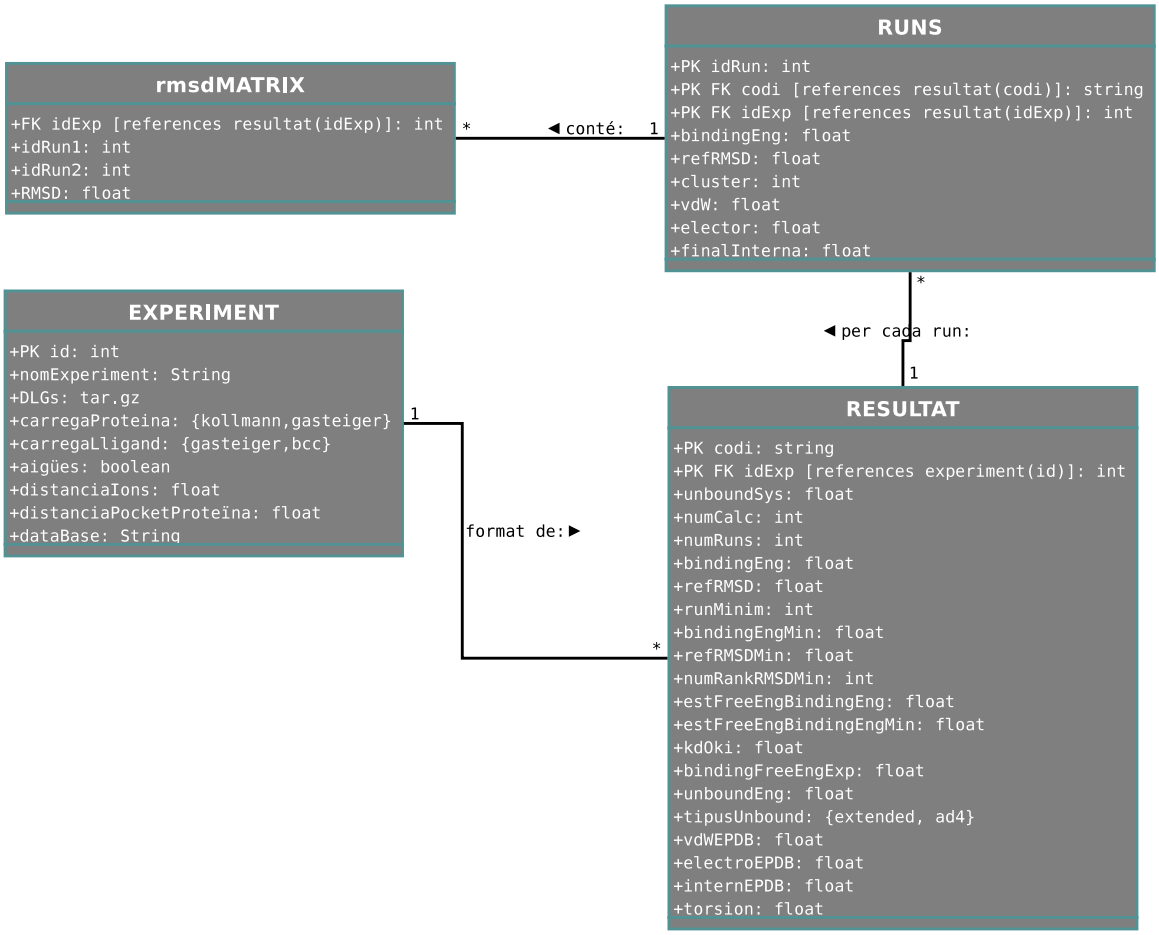


Figura 3.2: Base de dades usada pel daedalus

Capítol 4

GRID Superscalar

4.1 Introducció

GRID Superscalar és una eina desenvolupada pel BSC (*Barcelona Supercomputing Center*) que facilita el pas de convertir una aplicació seqüencial a una altre de paral·lela, la qual pot ser executada també en un GRID o en el nostre cas en un clúster.

Per aquesta eina en el BSC, s'han basat en el món de l'arquitectura de computadors i el disseny de microprocessadors[27]. Ja que, a nivell conceptual un processador no és molt diferent d'un "grid" (disposa d'unitats de càlcul, unitats d'emmagatzematge i una xarxa que les interconnecta)[28]. Les diferències s'aprecien en la quantitat de temps que fa falta per realitzar una operació en aquest entorn. En un processador les operacions s'executen en un temps de l'ordre de nanosegons, quan en un "grid" una operació pot tardar segons, minuts o inclús hores. Tècniques com l'execució en desordre o el renombrament d'operands (registres en el cas dw processadors o fitxers en el cas de GRID Superscalar) s'apliquen en el runtime d'aquesta eina per a l'execució en paral·lel de l'aplicació. Com a resultat per a l'usuari final, és que la seva aplicació s'ha executat en el "grid" aprofitant el paral·lelisme disponible, quan per a ell és com si s'hagués executat de forma seqüencial en la seva pròpia màquina.

En el cas que ens ocupa, nosaltres hem usat aquesta eina en un clúster, però l'efecte que ha causat ha sigut el mateix, ja que ens ha permès llançar en paral·lel en les diferents màquines de les que disposàvem una aplicació, l'AutoDock(veure capítol 5), sense haver de prendre en cap cas decisions en torn a la divisió, ni a la distribució del treball a realitzar en l'execució paral·lela, ni es va haver de realitzar grans modificacions del codi inicial del programa; cosa que sí que s'ha de fer per adaptar un programa seqüencial a paral·lel si s'utilitzen eines com OpenMP o MPI [29] directament.

Per treballar amb el GRID Superscalar l'usuari ha de proporciona a aquesta eina com a mínim tres fitxers:

- **Programa principal**: l'algorisme que es vol resoldre.
- **Fitxer IDL**: interfícies de les funcions (o tasques)que es volen executar en el "grid"/clúster. És a dir: nom de les funcions i paràmetres, afegint la direcció del paràmetre (si és d'entrada, sortida o d'entrada/sortida).
- **Implementació** de les funcions descrites en el IDL.

A partir d'aquests tres fitxers, es realitza el desplegament o deployment — construcció dels diferents binaris a les maquines del "grid" — i finalment l'execució del nostre programa principal. Això desencadena que el runtime entri en acció, ja que les funcions del programa principal generen crides al runtime enlloc d'executar realment la funció. El runtime realitza un anàlisi de dependències entre les diferents funcions per tal de construir un graf de dependències de les funcions. D'aquest graf, les tasques que tinguin les seves dependències resoltes (és a dir, que no s'hagi d'executar una altre tasca abans que ella) són candidates a ser executades en el "grid"/clúster. GRID Superscalar escull la més convenient i també escull un recurs en el "grid"/clúster, intentant minimitzar el temps d'execució total i llança l'execució remota en el "grid"/clúster fent ús del middleware corresponent. Aquelles funcions que no tinguin dependències entre elles es poden executar a la vegada, sent aquest el mètode que GRID Superscalar fa servir per paral.lelitzar les aplicacions.

Per acabar d'introduir-nos i entendre el comportament d'aquesta eina, hem de tenir clar una distinció que GRID Superscalar fa i és que aquesta eina de paral·lelització distingeix entre les màquines destinades només a executar les diferents tasques que s'hagin de realitzar — grup que anomena *"workers"* — i la màquina que anomena *"master"* — que també podria formar part dels *"workers"* si així ho desitgem — la qual és l'encarregada de llançar el runtime de GRID Superscalar i per tant té el programa principal de l'aplicació.

4.2 Model de programació

Tal com hem nombrat anteriorment, per poder usar aquesta eina, primer de tot hem de generar com a mínim els tres fitxers anteriors. En el nostre cas, s'han creat altres fitxers de suport a aquest tres, perquè el problema a resoldre ho requeria i per realitzar una programació modular que ens dona més escalabilitat, independència...

4.2.1 Programa principal

En aquesta part, GRID Superscalar ens proporciona un conjunt de primitives molt reduïdes perquè l'adaptació del codi sigui el més senzill possible. Bàsicament ens dona una parell de primitives per indicar quan el runtime s'ha d'iniciar (GS_On) i quan s'ha de finalitzar (GS_Off); d'altres pel control de fitxers quan el runtime de GRID Superscalar està en marxa (GS_Open, GS_Close, GS_FOpen i GS_FClose) —les quals tenen unes funcionalitats semblants a les que el llenguatge de programació C ofereix— i unes poques més

per tenir més control sobre les diferents tasques com GS_Barrier, GS_Throw, GS_Speculative_End, etc.

En aquest cas, el nostre programa principal, li passem com a paràmetres: el path on es troba el fitxer que necessita l'AutoDock com a entrada —cada experiment està separat en un directori diferent— el path de sortida que usarà l'AutoDock, el path on es troba el fitxer que conté les dades experimentals de l'unió[1] i l'identificador que té l'experiment que s'està executant en la base de dades. Llavors el runtime de GRID Superscalar conté un bucle que dona tantes voltes com execucions s'hagin de fer de l'AutoDock, és a dir, aquest número és igual al número de directoris que hi han en el path del primer paràmetre comentat just abans; i en cada volta llança una nova tasca pels workers on indiquem *"hardcoded"* el path absolut de l'AutoDock i els fitxers d'entrada i sortida que aquesta eina necessita. Aquesta tasca està definida en el fitxer IDL.

4.2.2 Fitxer IDL

En aquest fitxer s'hi defineixen les capçaleres de les funcions que es volen executar en el "grid"/clúster. Aquest format amb el que s'especifiquen aquestes capçaleres és el mateix que s'utilitza en el llenguatge IDL de CORBA [30]. S'hi ha d'especificar el nom de la funció, amb tots els paràmetres que necessita, amb el tipus i la direcció de cada paràmetre; si el fitxer és d'entrada (només es llegeix dins de l'operació), si és de sortida (només s'hi escriu) o si és d'entrada/sortida. Les dependències de dades es tenen en compte només en els paràmetre definits com a tipus File. Per tant, en aquests fitxers IDL només s'hi han d'especificar les funcions que vulguem executar en "grid". Per altres funcions locals, es compilaran i s'executaran com es feia anteriorment.

Per tant, el nostre IDL només conté una sola funció, amb tres paràmetres, com hem explicat en l'apartat anterior. Dos són del tipus File: un té la direcció d'entrada (fitxer d'entrada de l'AutoDock) i l'altre és de sortida (fitxer on anirà la sortida de l'AutoDock), l'altre paràmetre és un string, conté el path absolut del binari de l'AutoDock.

4.2.3 Implementacó de les tasques

Finalment, es realitzen les implementacions de les diferents funcions que puguin haver-hi, les quals s'han de proporcionar en un fitxer separat. Aquesta

[1] Actualment la ki (constant d'inhibició) i la kd (constant de dissociació)

és la part que cada *"worker"* del "grid"/clúster executarà en el seu moment. Per realitzar aquestes funcions també disposem d'un parell de primitives, com el GS_Throw que ja hem anomenat abans i GS_System, equivalent a la funció system que el llenguatge C ofereix, per realitzar una execució d'alguna comanda. Amb aquesta última primitiva podem cridar a un executable extern des del mateix *"worker"*.

Queda clar llavors el que els nostres *"workers"* faran. Agafen els paràmetres de la funció i mitjançant una crida de GS_System llancen una execució de l'AutoDock.

4.2.4 Altres fitxers

La nostre aplicació no acaba un cop s'han realitzat totes l'execució de l'AutoDock que s'hagin de realitzar. Per això, complementem aquest fitxer amb altres dos més.

Un d'ells és l'encarregat de "parsejar" i emmagatzemar els resultats obtinguts de les sortides generades per les diferents execucions de l'AutoDock. Aquesta funció és iniciada pel mateix programa principal quan finalitza el runtime del GRID Superscalar. Per aquest fet deduïm que aquesta tasca ja està fora de GRID Superscalar i per tant s'executarà en local. Es va decidir així perquè aquesta tasca té un cost computacional molt petit i si s'hagés d'executar en el "grid", segurament estaríem malgastant recursos, generaríem fitxers innecessaris i a més pel culpa de l'overhead que es pogués realitzar en els canvis de context en els processadors o amb els propis processos interns del GRID Superscalar, aconseguiríem realitzar de ben segur l'efecte contrari al desitjat, és a dir acabaríem dedicant més temps del que una execució en local podria tardar.

L'altre fitxer és una mena de passarel·la que usem per comunicar-nos amb la base de dades. Aquest fitxer és cridat per l'anterior per anar emmagatzemant totes les dades recaptades al "parsejar" els fitxers de sortida que genera l'AutoDock. Així aconseguim separar entre la part més de "parseig" i la de dades. Qualsevol canvi en les llibreries que usem per connectar el C++ amb MySQL només afectarien a un sol mòdul, a més d'altres canvis com el nom de la base de dades, password, nom de la màquina on es trobi la base de dades, etc.

4.3 Desplegament de les aplicacions

Un cop s'han creat aquest fitxers llavors ja podem començar a realitzar el desplegament o deployment de la nostre aplicació, que com ja hem comentat anteriorment una de les parts importants és la construcció dels diferents binaris a les màquines del "grid". El propi GRID Superscalar té el Deployment Center, una eina per realitzar el desplegament de les aplicacions dins del paquet de components que ofereix, però nosaltres hem generat un desplegament propi, ja que el Deployment Center del BSC usa com a middleware el Globus i aquest no és el nostre cas, ja que usem una altre middleware com ja hem explicat.

En el nostre cas com tots els processadors son homogenis i treballem amb un NFS, totes les màquines agafen un mateix binari de l'espai compartit del NFS. Així doncs, nosaltres per realitzar tot el desplegament el que fem és el següent: Creem un nou experiment, és a dir un directori, i dins d'aquest directori crearem tants directoris com execucions de l'AutoDock tingui l'experiment en qüestió. Aquests nous subdirectoris tenen l'informació d'entrada necessària perquè l'AutoDock es pugui executar. Aquesta informació bàsicament és un fitxer de configuració que conté els paths d'altres fitxers corresponents a diferents tipus d'informació relacionats amb les característiques de les molècules que s'intenten ancorar.

A partir d'aquesta organització l'usuari a través de l'interfície web que oferim, pot escollir tots els paràmetres necessaris perquè es pugui executar amb èxit el programa principal del GRID Superscalar. A més d'això i relacionat amb el deployment, a través de la web també es pot escollir quines màquines formaran el aquest "grid"/clúster on s'executaran els diferents *"workers"* del GRID Superscalar i quants treballs (*"jobs"*) s'executaran en cada un d'ells. Per fer això modifiquem un fitxer del qual encara no havíem parlat, el *project.gsdeploy*. Aquest fitxer XML conté els paths de treball del *"master"* i dels diferents *"workers"*, així com el nom que els identifica, és a dir el nom del host, i el número de *"jobs"* que s'hi enviaran. Per realitzar aquesta tasca, usem l'API DOM en entorn PHP, per "parsejar" i editar fitxers XML.

4.4 Runtime

El *runtime* —que com ja hem explicat anteriorment és executat per la màquina *"master"* en el programa principal de l'aplicació realitzada— de GRID Su-

perscalar té diverses característiques. Realitza un anàlisis de les dependències de les dades entre les operacions que s'han d'executar en el "grid"/clúster (basat en fitxers) i a partir d'això explora el possible paral·lelisme de l'aplicació automàticament de manera transparent a l'usuari, permetent l'execució fora d'ordre de les operacions generades (sempre respectant les dependències de dades). Algunes de les dependències poden ser eliminades amb tècniques de renombrament de fitxers.

Sempre que hi hagin recursos disponibles, el runtime llança a executar, en aquesta cas, en el "grid" tota tasca que no tingui dependències processades (és a dir, que no tingui que esperar a que altres tasques finalitzin per poder ser executades). De les possibles tasques candidates a ser executades, s'avalua el cost de transferir els fitxers que la tasca necessita a la màquina remota on s'ha d'executar i el cost d'execució en la màquina corresponent. Si els fitxers ja estan disponibles en una màquina (perquè una altre execució els havia transferit o generat allà), s'elimina aquest cost de transferència, aprofitant així la localitat dels fitxers. Un realitzades aquestes dues avaluacions s'executarà la tasca que tingui una estimació menor.

De forma transparent a l'usuari, el runtime per a l'execució d'una tasca en concreta, primer transfereix a la màquina seleccionada els fitxers que aquesta necessiti que no tingui disponibles i crear un directori temporal de treball per evitar problemes amb altres tasques que puguin estar executant-se en aquell precís moment. Després de l'execució, els resultats són emmagatzemats en la mateixa màquina que els ha generat, deixant-los d'aquesta manera disponibles per les següents tasques que s'hagin d'executar. Quan ha finalitzat, el runtime actualitza el seu graf de dependències entre tasques eliminant les dependències que tenia la tasca que ha finalitzat, fet que pot deixar altres tasques disponibles per ser executades en el "grid"/clúster. Quan cop realitzades totes les tasques del graf, el propi runtime elimina tots els directoris i fitxers temporals que havia creat.

GRID Superscalar també és capaç de treballar amb discos compartits entre diferents màquines, a través de NFS, i amb rèplica de dades, minimitzant així les transferències necessàries per les execucions. Aquesta ha estat la tècnica que hem usat nosaltres ja que al disposar d'un clúster en l'empresa era l'opció que encaixava millor. Tot i que també es poden usar altres tècniques com el checkpointing a nivell de tasques, que permet reiniciar l'execució des d'un punt intermig de la computació si per qualsevol motiu s'ha tingut que detenir, i la gestió d'exepcions amb primitives que ja hem comentat anteriorment. El runtime de GRID Superscalar funciona sobre diferents

middlewares de “grid” com Globus 2.4, Globus 4.0, Ninf-G2 i ssh/scp. Com ja hem comentat anteriorment, nosaltres usem ssh/scp.

Capítol 5

AutoDock

5.1 Introducció

En aquest apartat descriurem el software de docking que hem utilitzat dins de la nostra aplicació, l'**AutoDock**. Com la part més química ja l'hem tractada anteriorment, aquest capítol ens ha d'ajudar a situar-nos millor en l'entorn computacional en el qual estem i així acabarem d'entendre perquè aquesta aplicació és tan costosa.

Hem escollit aquesta aplicació per executar els docking, perquè és "open source" i això ens permet fer tot tipus de modificacions i optimitzacions en el codi. Fet que ens interessa bastant, ja que així podem adaptar al màxim els nostres projectes i optimitzar-los segons l'arquitectura dels nostre ordinadors. Per tant, a part de les optimitzacions que s'hi han fet per aquest projectes també s'hi han fet d'altres de diferents per altres treballs.

AutoDock incorpora varies opcions en l'algorismes de cerca que utilitza. L'usuari pot escolli entre un algorisme genètic lamarckia, un "simulated annealing" (SA) o inclús un algorisme genètic lamarckia dependent del problema al qual ens afrontem en aquell instant. La configuració habitual que s'acostuma a utilitzar perquè l'AutoDock realitzi aquestes cerques és, un algorisme genètic de cerca, amb una població de 50 individus i un número il·limitat de generacions. L'evolució deixa un número màxim de generacions energètiques, per defecte 250.000, el qual normalment és modificat en funció de la relació qualitat/velocitat desitjada.

Tot i això, AutoDock planteja, per defecte, 10 execucions independents de l'algorisme de cerca, utilitzant en cada una d'elles una llavor aleatòria a partir de la qual es genera el número de generacions, per tant el càlcul de docking d'AutoDock implica un conjunt d'n execucions independents de l'algorisme de cerca. Llavors, al final de les 10 execucions, AutoDock agrupa les millors sol·lucions trobades en cada execució independent, depenent del valor del RMSD[1] i retorna:

- la millor energia de docking de cada grup
- el promig de l'energia de docking de cada grup

Normalment, si l'AutoDock retorna les 10 millors sol·lucions agrupades en pocs grups, per exemple tres o quatre grups, és perquè el procés de docking ha convergit. En canvi, en el cas de molècules molt flexibles, on l'espai de

[1]Root Mean Square Deviation, en el capítol 6 en parlarem amb més detall

cerca és molt gran, el procés de docking no arriba a convergir i l'AutoDock retorna 10 grups, un per cada una de les diferents execucions independents. És ben clar que, per augmentar la probabilitat de trobar el millor ancoratge és necessari incrementar el número d'execucions independents de l'algorisme genètic de cerca. A part d'això, la resta de paràmetres de l'algorisme de cerca poden ser modificats, per exemple: la probabilitat d'encreuament, la probabilitat de mutació, etc.

Per duu a terme un experiment de docking, primer de tot hem de preparar la superfície de potencial molecular de la proteïna. El mateix AutoDock ja ens proveeix d'una eina per fer aquesta tasca, l'AutoGrid. Aquesta eina necessita els següents paràmetres d'entrada: la proteïna diana, en format de fitxer PDB; una *caixa* que inclogui la superfície de potencial molecular; i la resol·lució utilitzada per calcular els potencials. Per tant, AutoGrid computa un conjunt de xarxes ("grids") de potencials, i es defineix una xarxa de potencials per cada un dels àtom definits, per defecte `CHNOS`[2]. Durant el docking, s'utilitza l'interpolació multilineal per avaluar ràpidament l'energia de dispersió/repulsió intramolecular, l'energia d'unió dels hidròsgens, el potencial electrostàtic i l'energia de solvatació de cada àtom del lligand, usant la xarxa de mapes anomenada anteriorment. Per defecte la determinació de l'AutoGrid és de 0.375 Å.

Com a funció d'avaluació de l'algorisme genètic de cerca, AutoDock calcula l'*energia lliure d'unió d'AutoDock*, la qual inclueix tant termes entàlpics com entròpics. Per ser més concisos, aquesta funció la descrivim en la següent equació:

$$\Delta G_{AD} = f_{elec} \sum_{i,j} \frac{q_j q_i}{\epsilon(r_{ij}) r_{ij}} + f_{vdw} \sum_{i,j} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) \tag{5.1}$$

$$+ f_{hbond} \sum_{i,j} \xi(t) \left(\frac{C_i j}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right) + f_{sol} \sum_{i,j} S_i V_j e^{-\left(\frac{r_{ij}^2}{2\sigma^2}\right)} + T_{HBD} + T_{TOR}$$

on,

$$T_{HBD} = \sum_i P_{HBD,i} \tag{5.2}$$

$$P_{HBD,i} = \begin{cases} 0.118 \frac{kcal}{mol} & si\ atom_i = polar\ H\ (H\ en\ un\ vincle\ polar\ covalent) \\ 0.236 \frac{kcal}{mol} & si\ atom_i = O \\ 0.000 \frac{kcal}{mol} & si\ atom_i \neq polar\ H\ o\ O \end{cases} \tag{5.3}$$

[2]Carboni, Hidrogen, Nitrogen, Oxigen i Sofre

Referent a Hetènyi *et al.* [31] o a Morris *et al.* [12] per a la descripció físico-química de l'equació descrita just abans.

L'últim tema al qual hem de tractar, és la traducció que l'AutoDock fa de l'*energia lliure d'unió d'AutoDock* a l'energia lliure d'unió. La primera energia, és la que l'AutoDock intenta minimitzar en la funció d'avaluacions de l'algorisme genètic de cerca. L'altre energia és una estimació del valor *real* de l'energia lliure d'unió. En l'energia lliure d'unió hi ha alguns paràmetres que no depenen de l'estat de la conformació, però sí de la pròpia molècula. Per aquesta raó, l'AutoDock no calcula aquests paràmetres quan està realitzant un docking. Això seria una pèrdua de temps, ja que per tota conformació aquests paràmetre tenen valors constants. De totes maneres, si l'unió de *propensió* s'ha de comparar amb moltes mol·lècules, llavors sí que ens hem de fixar en aquests paràmetres, principalment degut a termes entropics, diferents estructures són més propenses a estar lligades. Per tant, l'únic paràmetre que l'AutoDock té en compte per aquestes tasques és la penalització de torsió (T_{TOR}): la molècula és el més flexible, la *capacitat* d'unió és el pitjor. Per ser més específics, AutoDock penalitza amb 0.3113 $\frac{Kcal}{mol}$ cada unió flexible del lligand, menys les unions en les que hi ha un hidrogen implicat. Aquesta aproximació també l'han cregut convenient altres [32].

5.2 Optimitzacions

Després d'haver implementat tota la nostra aplicació i d'haver realitzat les primeres proves amb ella, ens varem donar compte que el coll d'ampolla era, com ja suposàvem des d'un inici, l'AutoDock. El sol fet de llançar un simple AutoDock per calcular l'ancoratge entre dues molècules (per exemple, una proteïna i una lligand) amb una configuració bastant estàndard de l'algorisme genètic de cerca i on l'AutoDock faci 10 execucions independents[3], aquesta simple execució pot arribar a tardar uns 30 minuts aproximadament. Però aquest temps és només per una sola execució de l'AutoDock, en una experiment normal de comparatives el qual nosaltres realitzem per millorar el procés de doking, es poden arribar a executar fins a 1000 AutoDocks diferents — simulem l'ancoratge entre una proteïna i un lligand diferent cada cop —, és a dir 30 $\frac{minuts}{AutoDock} * 1.000$ AutoDocks = 30.000 minuts, per tant, si només disposéssim d'un sol processador per realitzar aquesta tasca, tardaríem 30.000 minuts en executar tots els AutoDocks, que vindrien a ser uns 20 dies. Massa temps per l'execució d'un experiment, ja que em de tenir en compte que es realitzen varis experiments, i el número d'experiments que es podrien arribar a executar és infinit, per sol fet de que hi ha un número infinit de molècules. A més, si pretenguéssim utilitzar aquesta arquitectura per realitzar d'altres experiments de descobriment de fàrmacs ("drug discovery"), que s'engloben en el procés de "Virtual Screening" que tracte de simular l'ancoratge d'una proteïna amb **milions** de lligands diferents. Estem parlant llavors que s'executaran entre 1.000.000 $\sim$ 5.000.000 d'AutoDocks, per tant si el temps d'una d'execució segueix sent de 30 minuts, estem parlant que l'experiment tardara un temps en torn una magnitud d'anys.

Així doncs, els objectius que ens vam proposar assolir en aquest aspecte són: minimitzar el cost temporal de l'AutoDock a través d'optimitzacions en el seu codi font, en aquest cas sense tenir en compte el cost espacial d'aquestes optimitzacions; i esbrinar quin número màxim de treballs pot enviar GRID Superscalar, a cada màquina del nostre clúster per oferint-nos el màxim rendiment. Aquest últim aspecte també l'hem d'intentar optimitzar al màxim per poder accelerar el procés tant com poguem, ja que nosaltres realitzem els càlculs en paral·lel amb varies màquines alhora.

[3]En el capítol 5 on parlem de l'AutoDock està explicat

5.2.1 Codi font

Per portar a terme aquesta optimització: primer, com l'AutoDock s'executa mitjançant un algorisme genètic el qual s'inicia, per defecte, d'una forma aleatòria mitjançant una llavor, el que vam fer va ser fixar aquesta llavor en un número enter perquè tots els càlculs fossin, dins del que cap, el màxim semblant possible entre ells. Després mitjançant l'eina GPROF de "profiling" es va detectar en quina funció l'AutoDock es passava la major part del temps executant-se, així sabríem quina funció atacar primer per optimitzar-lo; com totes les màquines que tenim son Intel, varem instal·lar el compilador d'Intel, després es van ajustar els "flags" del compilador a les necessitats que requeria el programa i per últim es va forçar optimitzar el codi del programa manualment amb SIMD, d'aquest tipus de tècnica en parlarem més endavant.

Abans de fer res, es van llançar 2 execucions de l'AutoDock amb la llavor ben fixada, un estava compilar amb el g++ de GNU i l'altre amb el icpc d'Intel, tots dos sense tenir cap tipus d'optimització, amb el flag O0 activat. I el de GNU ens va donar un temps de 1h 23min 1seg i el d'Intel 1h 27min 3seg. Després es va executar el programa amb el GPROF i vam poder veure com el 44.50% del temps se'l passava en una funció (eintal.cc).

A partir d'aquí, vam passar-li tota una bateria d'optimitzacions del compilador d'Intel a veure si podíem anar baixant el temps. Però com les màquines eren Intel, el propi compilador per defecte ja posava molts d'aquests flags. Es va provar de posar "-mpcu=pentium4" perquè s'optimitzes específicament per aquestes màquines, amb el flag "-xT" perquè el propi compilador vectoritzés amb instruccions SIMD del tipus SSE3 (que vindrien a ser les instruccions més antigues suportades per les nostres màquines), "-fomit-frame-pointer" que no permet "debugar" però augmenta la velocitat d'execució, "-msee3" que vectoritza amb instruccions sse3, i per últim tot això es va compilar amb la màxima optimització que el propi compilador té, l'O3. Totes aquestes diferents compilacions van donar uns resultats semblants entre elles, que podríem dir que eren iguals ja que diferenciaven de com a molt 1 segon, i amb les magnituds de temps que treballàvem estàvem parlant d'una millora de menys d'un 1%. En resum, vam aconseguir baixar de 1h 27min 3seg a 38min 54seg, és a dir es va reduir el temps d'execució en un 44%. I amb el GRPOF vam poder veure com el temps de la funció "eintcal" nombrada anteriorment, baixava del 44.50% al 39.44%, baixant de 1744.72 segons a 856.19 segons amb el mateix nombre de crides clar, que en aquest cas era de 151009275.

Un cop baixat el temps al màxim mitjançant les opcions que ens donava el compilador d'Intel vam intentar vectoritzar manualment la funció "eintcal" mitjançant la tècnica **SIMD** (Single Instruccion Multiple Data). SIMD és una tècnica que s'utilitza per paral·lelitzar dades en un processador vectorial. És a dir, son instruccions dependent del hardware que treballen sobre vectors de 4 posicions, on cada posició del vector pot ser ocupada per variables o per literals, llavors tal com el seu nom indica, aquesta tècnica disposa de funcions que permeten executar en **una sola instrucció** una operació que implica les 4 posicions d'un vector contra altre vector igual. Això a priori ens permet reduir el temps d'execució d'un bucle de $O(n)$ a $O(\frac{n}{4})$.

Com aquesta funció era complicada, vam intentar vectoritzar-la a mà per veure si realment el propi compilador era capaç de vectoritzar-la sol i, si no era així, llavors segurament podríem aconseguir esgarrapar algunes millores temporals més. Un cop vectoritzada la funció ens vam donar compte de que no érem capaços de baixar el temps, respecte les execucions anteriors. Així doncs vam acabar deduint que el propi compilador d'Intel ja vectoritzava la funció.

Després, es va dur a terme el mateix "modus operandi" pel compilador de GNU. I en aquest cas, vam acabar arribant a la mateixa deducció, ja que el propi compilador amb màxima optimització va baixar el temps fins a 41min 50seg i afegint la nostre vectorització vam tenir un temps de 41min 06seg, que ens donava una millor de 44 segons totalment menyspreable tal com ens havia passat amb el compilador d'Intel.

En aquestes dues gràfiques, la figura 5.1, podem veure la diferencia de temps que hi ha entre les dues execucions i, entre els dos compiladors, les quals veiem que quasi no hi ha cap diferència.

On sí que hi han diferencies és a nivell de resultats, ja que l'AutoDock treballa molt en unitats de coma flotant i com els dos compiladors les tracten de manera diferent fa que surtin resultats diferents. El següent pas doncs va ser analitzar aquestes diferències per esbrinar si alguna de les dues execucions donava un resultat més acurat que l'altre o si realment només es tractava de seguir camins diferents dins de l'algorisme genètic i per tant els dos resultats eren diferents però igual de bons.

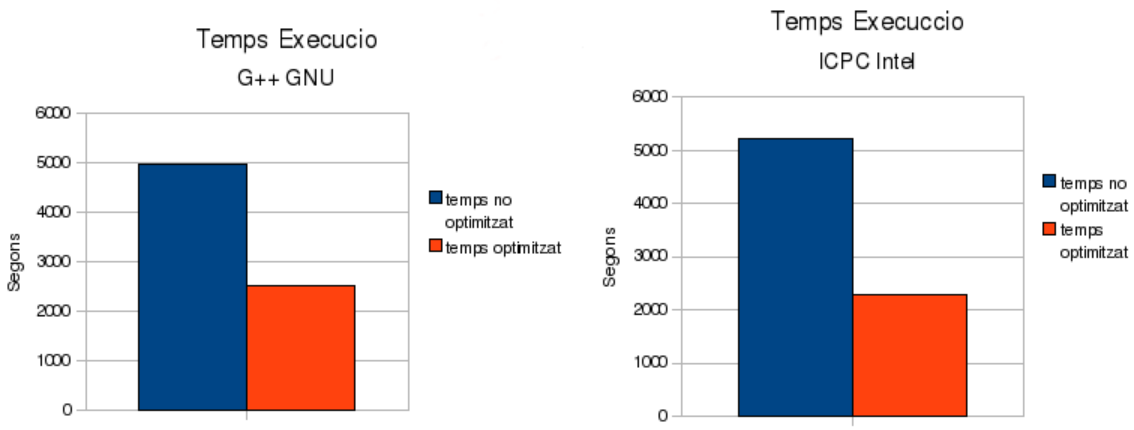


Figura 5.1: Gràfiques de temps entre el compilador de GCC i el de Intel

5.2.2 Limit de treballs

Per duu a terme l'altre treball que varem fer per intentar treure encara més rendiment del gran número d'execucions de l'AutoDock va ser, crear un programa que contingués només la part del domini del daedalus per fer diverses proves i diferents execucions, aquesta nova aplicació es va anomenar *GS_Time* (GRID Superscalar Time). En cada nova execució fèiem que GRID Supescalar enviés un número superior de treballs a la mateixa màquina i alhora, calculàvem el temps que es tardava en fer tota l'execució.

Senta més concrets, aquest experiment el varem fer en una de les màquines del clúster (*hidra-04*), aquesta té 4 processadors i cada un té 2 nuclis, és a dir, disposàvem d'un total de 8 nuclis, que en el nostre cas, aquests són els elements de procés encarregat de realitzar les tasques de computació. L'experiment tractava de variar el número de "workers" amb el que s'executava cada prova i cronometrar el temps que es tardava en realitzar la tasca. La base de dades contra la que s'executava aquesta prova de rendiment, constava de 33 proteïnes i 33 lligands, per tant, com es simulava un dels nostres experiments de comparatives, pel qual està dissenyada principalment la nostra aplicació principal, s'havien d'executar un total de 33 AutoDocks, per fer els diferents dockings entre les dues molècules (proteïna i lligand).

En la taula 5.2 podem veure com varem anar pujant mica a mica el número de treballs que enviava GRID Superscalar, i com clarament el temps també anava baixant fins arribar als 8 treballs. A partir d'aquí, comprovem que el temps ja no disminueix més encara que augmentem el número de tre-

balls a realitzar i a més a més veiem com el temps convergeix, aquest efecte el podem veure de forma més gràfica en la figura 5.2.

Taula 5.1: Temps recollit en les execucions de GS_Time en Hidra-04

Limit de treballs	Temps (hh:mm:ss)
1	25:59:00
2	13:30:02
3	09:10:27
4	07:13:30
5	06:11:23
6	05:21:01
7	04:45:41
8	04:15:53
9	04:17:16
10	04:17:08
11	04:16:27
12	04:17:16

Taula 5.2: Temps obtinguts en les diferents execucions

Per tant, d'aquest estudi en deduïm que a una sola màquina se l'hi han d'enviar com a màxim tants treballs com nuclis tingui aquesta. Enviant-li més treballs que aquests no hi han guanys temporals i a més, es corra el risc de saturar el processador a causa del gran número de canvis de context i l'overhead que es generaria podria arribar inclús a relentitzar lenta l'execució. Per acabar de visualitzar-ho i per contrastar que l'opció més eficient és enviar tants "jobs" com nuclis tingui com la màquina, a continuació vam calcular l'"Speedup"[4] de l'aplicació en aquestes execucions. Per fer-ho usem la formula següent 5.4, juntament amb el càlcul de l'eficiència per tenir una mètrica més del rendiment 5.5.

$$S_p = \frac{T_1}{T_p} \tag{5.4}$$

On:

- p és el número d'elements de procés.

[4]En computació paral·lela, l'*speedup* (acceleració) es refereix a com més de ràpid és un algorisme paral·lel que aquest mateix en seqüencial.

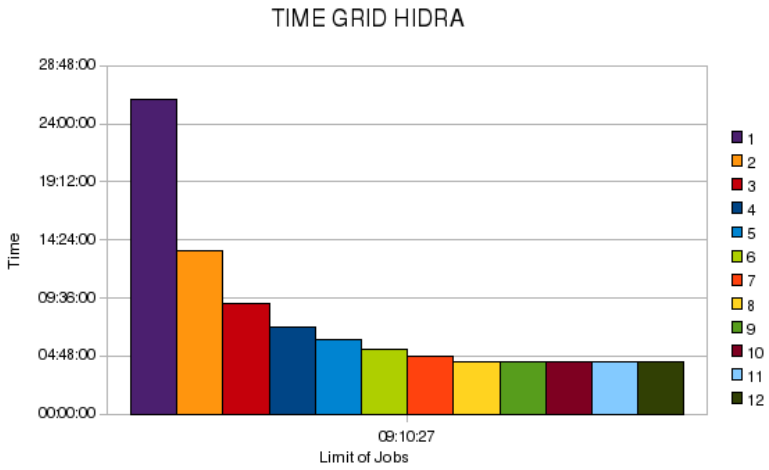


Figura 5.2: Gràfica del temps d'execució de *GS_Time* amb diferents números de "workers"

- T_1 és el temps d'execució de l'algorisme seqüencial.
- T_p és el temps execució de l'algorisme paral·lel amb p elements de procés.

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_1} \tag{5.5}$$

Per tant, aplicant la formula de l'"Speedup"5.4 i d'eficiència 5.5 obtenim la següent taula 5.4 amb la gràfica 5.3.

Al veure aquesta taula, ens varem donar compta que a l'enviar 8 "jobs" a una màquina l'speedup havia baixat més d'un 20% respecte l'ideal i per tant també patíem una disminució igual en l'eficiència que, per ser exactes era de 23.84%, perquè $\frac{speedup}{perfectspeedup}$ és igual a $\frac{6,09}{8} = 23.84$; ja que passàvem a tenir una acceleració de només 6.09 amb 8 elements de procés quan l'speedup ideal en aquesta situació és de 8.

Aquest efecte ens va sorprendre molt, perquè sí això era cert resultava que l'aplicació no era gens escalable, si amb 8 nuclis baixàvem més d'un 20%, llavors amb 32 baixaria encara més i no seria gens eficient. En aquell moment, vam desenvolupar el següent model definit a l'equació 5.6 de [33] on sabíem que en el nostre cas, el temps en el que els nuclis es troben ociosos és igual al temps d'un AutoDock, ja que en l'última sèrie de càlculs si el número

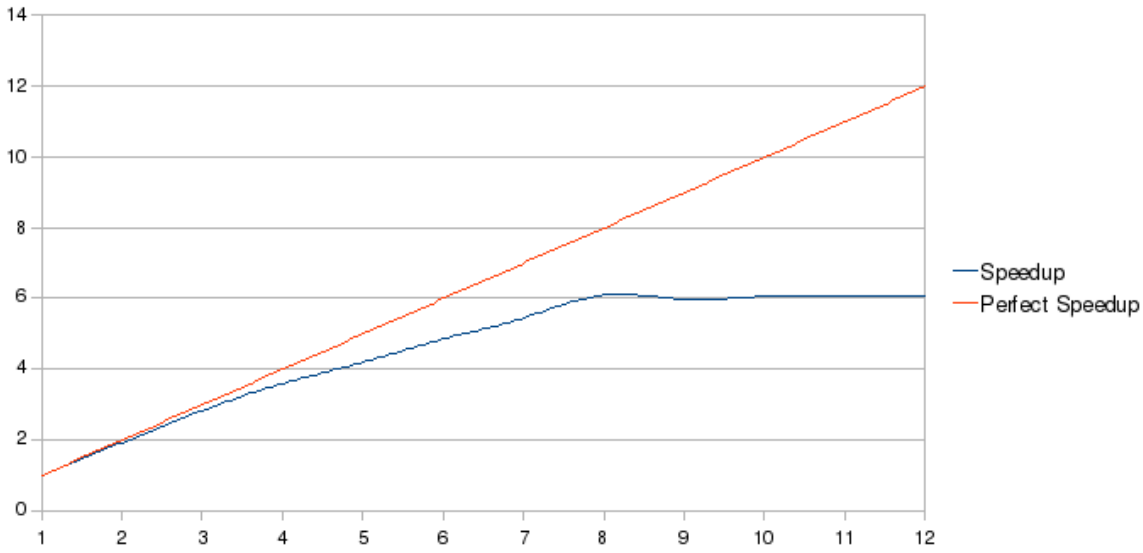


Figura 5.3: Gràfica on comparem l'speedup ideal, amb el que hem calculat a la nostre aplicació.

total de càlculs no és múltiple del número d'elements de procés que destinem al càlcul hi hauran alguns que no tindran feina; i el temps de comunicació és molt més inferior que el temps de computació entre els diferents elements de procés — recordem, que les execució de l'AutoDock són independent entre sí i per tant, no hi ha comunicació entre elles — i com $T_{comp} >> T_{comm}$, llavors T_{comm} és despreciable, així doncs en resultat tenim que $T = \frac{T_{comp}+T_{idle}}{P}$.

$$T = \frac{1}{P}(T_{comp} + T_{comm} + T_{idel}) \tag{5.6}$$

On:

- T és el temps total.
- P és el número d'elements de procés.
- T_{comp} temps de computació.
- T_{comm} temps de comunicació.
- T_{idel} temps que l'element de procés està ociós.

Taula 5.3:

Elements de càlcul(p)	Speedup (S_p)	Speedup perfecte	Eficiència (E_p)
1	1	1	1
2	1.92	2	0.96
3	2.83	3	0.94
4	3.6	4	0.9
5	4.2	5	0.84
6	4.86	6	0.81
7	5.46	7	0.78
8	6.09	8	0.76
9	5.98	9	0.66
10	6.06	10	0.61
11	6.08	11	0.55
12	6.06	12	0.5

Taula 5.4: Resultats obtinguts en les diferents execucions

Acte seguit vam realitzar una nova bateria de càlculs on no volíem que cap element de procés estigués ociós, per tant varem preparar $8*4 = 32$ AutoDocks on a més els 32 AutoDocks feien un mateix càlcul així ens asseguràvem que cap d'ells tardava més que els altres i alterava els resultats. Així si cap processador està ociós el temps totals seria molt pròxim al speedup perquè ara $T = \frac{T_{comp}}{P}$, és a dir el temps disminuiria de forma proporcional al número d'elements de procés que tinguéssim. El resultat que varem obtenir va ser molt semblant, el càlcul va tardar 5 hores i 3 segons amb un speedup de 6.14, seguíem tenint una davallada en l'eficiència superior al 20%. En aquest punt ens varem donar compte que si executàvem 32 AutoDock de 8 en 8 sabíem que, es farien 4 tandes de 8 AutoDocks cada una i com tots els AutoDocks eren igual, tardaven tots el mateix; així doncs, cada una d'aquestes 4 tandes havien de tardar el mateix i això havia de ser $\frac{05:00:03}{4}$, com 5 hores i 3 segons són 300.05 minuts teníem que $\frac{300.5}{4} = 75.01 \frac{minuts}{AutoDock}$ quan en realitat cada un d'aquests AutoDocks sabíem que tardava 57.57 minuts.

En conclusió, vam deduir que com no hi ha comunicació entre els diferents processos, estava clar que teníem problemes amb la memòria, és a dir teníem massa processos accedint a la memòria a la vegada i es realitzaven massa col·lisions i això feia augmentar la latència i per tant com més AutoDock s'executaven en una mateixa màquina més penalitzava aquest fet. Per demostrar-ho vam executar aquest mateix experiment però, enlloc d'enviar 8 "jobs" a una màquina en vam enviar 4 en una i 4 en una altre i com a

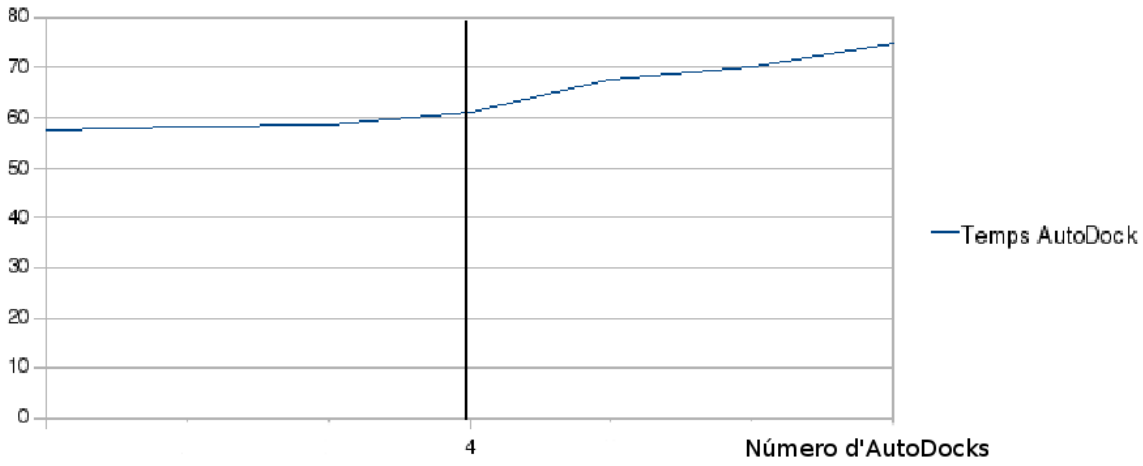


Figura 5.4: Gràfica del temps en realitzar un AutoDock quan la màquina rep entre 1 fins a 8 AutoDocks

resultat l'execució va tardar 3:53:35, és a dir 233.58 minuts i un speedup de 7.89, realment molt proper a l'speedup ideal de 8. El següent pas va ser tornar a enviar aquest mateix càlcul usant 2, 3, 4, 5, 6 i 7 nuclis i observar quan temps es tardava en fer una execució de l'AutoDock tal com havíem fet anteriorment, per veure a partir de quin punt la memòria es saturava i el temps es disparava. En la següent taula 5.6 podem veure els valors que es van recollir de les diverses execucions i en la figuar 5.4 podem veure com el temps de cada AutoDock augmenta i a partir de més de 4 "jobs" el temps es dispara.

Taula 5.5:

Processadors(p)	Speedup (S_p)	Temps Total (min)	Temps per AutoDock (min)
1	1	1842.22	57.57
2	1.98	928.8	58.05
4	3.94	467.72	58.46
5	4.28	430.28	61.27
6	4.76	387.23	67.63
7	5.37	343.05	70.02
8	6.14	300.05	75.01
4 i 4	7.98	233.58	58.4

Taula 5.6: Resultats obtinguts amb el nou model de càlcul

Per tant, si volem treure la màxima eficiència computacional alhora de llançar aquesta aplicació la millor opció és enviar a totes les màquines de les que disposisn tant "jobs" com processadors tingui aquesta. Però si volem és la màxima eficiència temporal omplirem la màquina amb tants càlculs com elements de procés en tingui. Ja que com també hem mostrat a la taula 5.6 al enviar de 8 en 8 els treballs a cada màquina, tot i tardar menys alhora de calcular els AutoDocks, en el comput total seguiex sent més ràpida l'execució que de 4 en 4 on l'speedup és molt proper al ideal encara.

Capítol 6

Estadístiques

Aquesta secció la dedicarem a parlar d'una eina de la qual disposa la nostra aplicació, per realitzar i visualitzar càlculs estadístics, la qual és molt útil, ja que treu moltes hores de treball als químics que utilitzen aquesta eina. Perquè gràcies a la velocitat que som capaços de servir les dades i de realitzar certs càlculs estadístics, fa que l'interacció de l'usuari amb les dades sigui molt més amigable i senzilla, ja que en cap cas l'usuari no ha d'estar navegant entre diferents fitxers de text plens de dades, tal com AutoDock serveix els resultats. A més amb un sol cop d'ull, el científic ja pot començar a treure conclusions dels experiments que s'han calculat prèviament i aquests resultats estadístics que s'extreuen es poden obtenir de forma quasi automàtica just quan l'execució de l'experiment hagi acabat. És a dir perquè ens fem una idea: després d'haver executat tota una bateria de 100 AutoDocks, enlloc d'haver de mirar un a un tots els fitxers de sortida que es generen, tenir que buscar les dades necessàries, analitzar-les, comparar-les amb les altres dades... Realment és una tasca costosa, estem comparant i analitzant 100 fitxers, i cada un d'ells té més o menys 17.000 línies, ens podríem arribar a passar dies només per realitzar la cerca de les dades que ens interessen i després analitzar-les, comparar-les, etc...

Llavors per sol·lucionar-ho aprofitant que tenim totes les dades més important que l'AutoDock genera en la seva sortida, l'eina és capaç de generar anàlisis estadístics en un mateix experiment (histogrames, regressions, percentils,etc.), com comparacions entre dos experiments mitjançant, per exemple, el test de wilcoxon[1].

Per tant, hem dividit la part d'estadístiques en dues parts, anàlisi i comparació. A més també cal comentar que la gran majoria d'aquests càlculs els hem realitzat integrant dins del nostre software crides a programes realitzats amb el llenguatge de programació **R**.

6.1 Anàlisis

A continuació explicarem els càlculs estadístics d'anàlisis d'un experiment que oferim en la nostra aplicació, que podríem separar en 3 grups: histogrames, regressions i d'altres de menys complexos.

[1]En parlarem de forma més detallada en el subapartat 6.2.1

6.1.1 Histogrames

Entenguem histograma com una representació gràfica d'una variable en forma de barres, on la superfície de cada barra és proporcional a la freqüència dels valors representats. En l'eix vertical es representen les freqüències, i en l'eix horitzontal els valors de les variables.

En aquest apartat ens centrem principalment en la variable, RMSD[2], una valor estadístic que retorna el propi AutoDock que és força indicatiu per poder comparar la diferència que hi ha entre varis dockings[3] diferents d'un mateix lligand sobre una proteïna. L'RMSD el que fa és compara per tots els àtoms d'un lligand els valors d'x, y i z que ocupen en l'espai amb l'x, y i z del mateix àtom del lligand en un docking diferent, la funció que defineix aquest valor el podem veure en l'equació 6.1. En el nostre cas sempre comparem la posició de l'àtom que ha calculat l'AutoDock amb la posició que tenia en l'experimental — com ja hem comentat en capítols anteriors, el fitxer on estan les dades referents a l'experimental li passem a l'AutoDock com a paràmetre d'entrada en cada execució —. De tal manera, com més petit és aquest valor millor és la conformació generada per l'AutoDock, així doncs, si l'RMSD és de zero significa que l'ancoratge predit per l'AutoDock és idèntic a l'unió generada en el laboratori entre el lligand i la proteïna i a més tal i com es descriu en diversos articles — per exemple en "How to do an evaluation: pitfalls and traps" [34] o "Comparing protein-ligand docking programs is difficult" [35] — un valor inferior a 2 del RMSD ja és pren com a bo, per sobre de 2 ja es considera erroni i per tant no s'utilitzarà.

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2} \tag{6.1}$$

On δ és la distància entre N parells d'àtoms equivalents.

Dins dels diversos càlculs que oferim, per exemple, tenim el de *RMSD de mínima energia* on només agafem els RMSDs on els diferents "runs" de cada execucions de l'AutoDock ens han donat una energia d'unió més baixa o "binding energy", recordem que com més baixa és l'energia més bo és l'ancoratge, així doncs estem seleccionant els millors resultats que calcula l'AutoDock. Per tant, en l'histograma resultant podrem veure d'entre les diferents execucions que hem realitzat en un mateix experiment, si realment

[2]Root Mean Square Deviation
[3]ancoratge

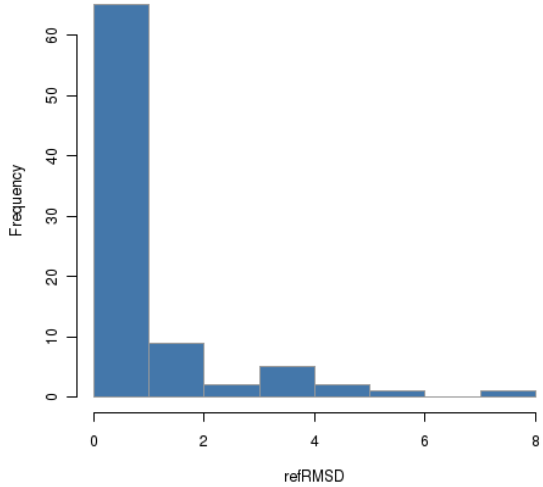


Figura 6.1: Histograma d'exemple de l'*RMSD de mínima energia*

s'ajusten o no a la realitat. En la figura 6.1 podem veure un exemple d'un possible resultat del càlcul de *RMSD de mínima energia*, en aquesta representació gràfica de la variable observem com la majoria d'execucions de l'RMSD està entre 0 i 1 i per tant proper a l'experimental.

6.1.2 Regressions

Sent una regressió lineal o ajust lineal un mètode matemàtic que modelitza la relació entre una variable dependent Y, les variables independents X_i i un terme aleatori ε. Aquest model es pot expressar segona l'equació 6.2. On β_0 és l'intersecció o terme "constant", les β_i són els paràmetres respectius a cada variable independent, i p és el número de paràmetres independents a tenir en compte en la regressió. On ε_i és l'error associat a la mesura del valor X_i suposant que $\varepsilon_i \sim N(0, \sigma^2)$ (mitja zero, variança constant i igual a un σ i $\varepsilon_i \perp \varepsilon_j$ amb $i \neq j$).

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon_i \tag{6.2}$$

Juntament amb la regressió, també es calculen dos paràmetres més: la recta de la regressió, és la recta que s'ajusta millor al núvol de punts o diagrama de dispersió generada per una distribució binomial, normalment expressada com la recta de regressió de Y sobre X 6.3, és a dir X seria la variable independent i Y la dependent de X. I el coeficient de correlació al quadrat (R^2) de les rectes, el qual determina la qualitat amb la que la recta

s'ha ajustat al diagrama de dispersió. si R^2 és 1 o hi està a prop significa que l'ajust és bo; si és 0 o hi està a prop l'ajust és dolent.

$$y = \bar{y} + \frac{\sigma_{xy}}{\sigma_x^2}(x - \bar{x}) \tag{6.3}$$

En aquest aparta s'ofereix la regressió lineal entre les variables *Binding Energy*[4] i *Binding Energy de l'experimental*. Això ho fem per poder comparar aquestes dues variables i saber com d'acurat és el càlcul que hem realitzat. Ja que el primer valor és el resultat que obté l'AutoDock de l'energia d'unió de forma calculada , el valor del qual està extret d'un model matemàtic el qual, no és del tot real; i l'altre valor, l'experimental, està extret directament d'un experiment realitzat en el laboratori, així doncs el valor és més precís i real.

Per poder visualitzar aquestes dades mostrem per pantalla el gràfic resultant, és a dir tot aquest núvol de punts que es forma, amb la seva recta de la regressió corresponent, a part també és pot veure el coeficient de correlació R^2.

Vegem un possible exemple de regressió lineal en la figuara 6.2 on podem veure el diagrama de dispersió entre les dues variables i la recta de la regressió.

6.1.3 Altres

En l'últim apartat d'anàlisis també es poden realitzar altres càlculs més senzills, en el nostre cas donem l'opció a poder saber quin percentatge de les execucions han estat per sota d'un valor donat del RMSD. Així podem saber ràpidament si molts resultats són o no propers al model més real, a l'experimental.

A part dels càlculs que oferim a priori en l'apartat d'anàlisis, que podríem dir que formen un paquet inicial d'estadístics, encara seguim pensant per introduir-hi nous càlculs que puguin donar més significats i informació referent a l'experiment.

[4]Com ja hem anomenat abans, energia d'unió

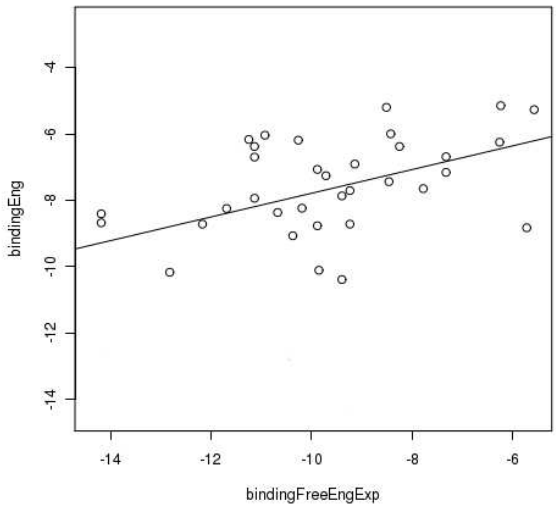


Figura 6.2: Exemple de regressió lineal

6.2 Comparacions

En l'apartat de comparació la nostra aplicació, dona eines per poder comparar dos experiments diferents i així poder ajudar a l'usuari a poder extreure conclusions, a afirmar que dos experiments son diferents o no, a ordenar de forma simètrica valors de dos mateixos experiments, etc. Per realitzar comparacions ens centrarem bàsicament en el test de *Wilcoxon*.

6.2.1 Wilcoxon

La prova de *signes de Wilcoxon* és un mètode no paramètric, alternatiu a la prova t de Student, que compara la mitjana de dues mostres realitzades per determinar si existeixen diferencies entre elles, és a dir, ens diu si dos experiments són iguals o no. En aquest cas, no podem usar el mètode paramètric de la prova de la t de Student perquè per poder-la usar hem de partir de la base que hi ha normalitat en els dos grups de dades que analitzem i tal com hem pogut observar en els histogrames, això no es compleix de cap manera (veure figura 6.1). Per altra banda, amb la prova de la t de Student arribaríem a les mateixes conclusions que amb el test de Wilcoxon si la normalitat de les dades ens la permetessin realitzar.

Per realitzar aquest test hem de partir del plantejament que es disposen

de dues mostres de n observacions cada una, amb x_i com una observació inicial i y_i una altre de final. Amb aquest plantejament tinguem també en compte les següents dues suposicions necessàries per poder realitzar amb èxit el test:

1. Sigui $Z_i = Y_i - X_i$ per "$i = 1, \ldots, n$". On les diferencies Z_i es pressuposen independents.

2. Cada Z_i prové d'una població continua (no tenen perquè ser idèntiques) i simètriques amb respecte a una mediana comú θ.

A partir d'aquest punt passem a realitzar l'hipòtesi $H_0 : \theta = 0$. On l'estadístic W^+ del test és calculat després d'ordenar els valors absoluts $|Z|, \ldots, |Z_n|$. L'ordre de cada $|Z_i|$ ve donat per R_i. Representat per $\varphi_i = I(Z_i > 0)$ on $I(.)$ és un indicador de funció o funció característica. L'estadístic anomenat anteriorment de la prova dels signes de Wilcoxon, W^+, es defineix en l'equació 6.4.

$$W^+ = \sum_{i=1}^{n} \phi_i R_i \tag{6.4}$$

Aquest test, s'utilitza per comparar les diferències entre dos mostres de dades adquirides abans i després del seu tractament i el seu valor s'espera que sigui de zero. Les diferències iguals a zero són eliminades i el valor absolut de les desviacions amb respecte al valor central són ordenades de menor a major. Les dades idèntiques se les assigna un lloc mig de la sèrie. La suma dels rangs es fa per separat per els signes positius i pels negatius. S representa la menor d'aquestes dues sumes. Comparem S amb el valor proporcionat per les taules estadístiques per determinar si refutem o no l'hipòtesis nul·la, segons el nivell de significació escollit. Sí el valor d'acceptar l'hiòtesis nul·la és inferior a 0.05 llavors es rebutja i s'acceptarà l'alternativa, per tant significa que existeixen diferències entre les dues poblacions i llavors pot ser que una sigui inferior a l'altre o viceversa.

Aquest test estadístic l'usem amb els valors de l'energia d'unió ("Binding energy") o de l'RMSD per poder veure si dos experiments són en realitat significativament iguals un de l'altre o no. Un exemple de l'ús que se li pot donar, és quan s'afegeix algun tipus de millora a l'AutoDock o s'afegixen diverses optimitzacions en el procés de docking, com per exemple: millorar les coordenades dels àtoms que formen les molècules, ja que de vegades en els fitxers que descriuen les molècules apareixen angles impossibles de reproduir en la realitat entre dos àtoms. Per tant, el procés seria: és realitzen els

diferents dockings per un experiment, després s'aplica algun mètode d'optimització qualsevol i es tornen a llançar els càlculs de nou. Un cop en aquest punt disposem de tot un seguit de dades, que aparentment poden semblar diferents, però realment no ho podem afirmar, ja que com l'AutoDock té un procés en part aleatori, podria donar-se el cas de tenir diferents valors per les mateixes variables però totes podrien estar patint un mateix increment o decrement. Això faria que els dos resultats fossin en realitat iguals o molt iguals.

Per tant, aquest test l'usem per tenir de forma ben certa si algun tipus de millora que s'ha introduït en el procés de docking millora o no el procés o per contra podria donar el cas que no estiguéssim aconseguint res, és a dir que els canvis o optimitzacions que haguéssim desenvolupat fossin en realitat nocives i per tant no introduïssin cap tipus de millora al procés.

Capítol 7

Resultats

Aquest capítol el separarem en dos grans apartats, en dos tipus de resultats que podem extreure o que hem obtingut en la realització d'aquest projecte:

- Informàtics
- Químics

Quan parlem de la part més informàtica ens referim bàsicament a la nostra aplicació i la seva arquitectura, és a dir a tot el que forma el propi daedalus, en l'aspecte més químic en centrem única i exclusivament amb tot el que està relacionat amb el software de química computacional que hem usat en el projecte, l'AutoDock. Per tant és obvi que no podem ajuntar aquest dos temes en un sol apartat, ja que son dos camps ben diferenciats l'un de l'altre i els resultats adquirits en la part informàtica no tenen res a veure amb els químics, no es poden comparar ni tenen la mateixa significança. Per conseqüent podem dir que, per una part tenim tots els resultats als que hem arribat a nivell informàtic, és a dir la creació de l'interfície web per a la comunicació de l'usuari amb l'unitats de càlcul del daedalus (domini i dades), creació i aplicació d'un full d'estil en cascada (CSS) a aquesta mateixa, resultats obtinguts en execucions realitzades dins del clúster amb GRID Superscalar, recol·lecció de les dades obtingudes en el medi físic mitjançant bases de dades, etc. En canvi, per altra part tenim la vessant més química, en el nostre cas ens referim a tots els resultats que l'AutoDock extreu dels càlculs que realitza en un experiment, però en aquest apartat ens interessa l'informació que aquests resultats contenen — és a dir, quin significat o sentit tenen —, i a més tots els càlculs estadístics que podem arribar a realitzar amb aquest conjunt de dades, aquesta part l'hem inclòs dins l'apartat de resultats químics ja que tot i ser un tema més matemàtic o estadístic, en el fons, no deixem d'estar avaluant, analitzant, comparant dades extretes d'experiments químics.

7.1 Daedalus

Ens referim a aquesta secció com *daedalus* ja que aquí mostrarem tots els resultats que hem pogut extreure i observar de la part més informàtica del projecte. Per fer-ho, ho hem estructurat en tres apartats: primer els resultats aconseguits en la realització de l'interfície web, després fets que hem pogut observar en l'entorn on s'executa l'aplicació i per finalitzar, exemples de com les taules de la base de dades s'han anat omplint mica a mica.

7.1.1 Interície web

Començarem parlant de l'interfície web, que és la part més vistosa de l'aplicació, encara que habitualment sigui l'última part en implementar-se. Com ja hem parlant en capítols anteriors volem que l'interacció amb l'usuari sigui el més senzilla, útil i funcional possible. A més per generar un model el més homogeni possible de la web varem aplicar un full d'estil en cascada (CSS) a la web. Un CSS no deixa de ser més que un llenguatge formal usat per a definir la presentació d'un document estructurat escrit en HTML, XML i per extensió per XHTML, el W3C[1] (*World Wide Consortium*) és l'encarregat de formular l'especificació dels fulls d'estil que servirà d'estàndard per als agents d'usuari o navegadors.

Per això varem separar la web en 4 apartats diferents, les qual veurem tot seguit:

La pàgina principal de l'aplicació, és alhora, la part des d'on llancem els càlculs que vulguem fer. Aquí és on hem d'especificar les diferents direccions dels fitxers i directoris que l'aplicació necessita per ser executada, informació per recordar quin càlcul estem llançant, les màquines en les quals volem que s'executi l'experiment i el número de treballs per màquina que creiem necessaris, per defecte ja porta una configuració prou optima, en l'imatge 7.1 podem veure tot això.

Després, en l'imatge 7.2 podem veure com seria la taula on es pot observar l'estat dels nostres càlculs, la part d'"status". A part d'això aquesta web s'autorefresca sola cada 15 segons, perquè així l'usuari no hagi d'estar refrescant-la manualment cada vegada que vulgui veure com estan evolucionant els seus experiments, a més des de el desplegable d'opcions podem borrar una execucció que al final haviam manat de realitzar però al final ja no ens interessa, podem veure la sortida que ha generat per pantalla alguna execució ja realitzada o informació referent al temps que ha tardat, nom de l'experiment... I a més podem situar un càlcul a dalt de tot de la cua si aquest té més prioritat que un altre.

La següent part és la gestió i consulta de la base de dades, correspon a l'imatge 7.3. Aquí veiem en el requadre de l'esquerra tots els experiments que ja hem executat i a la dreta els diferents camps que podem veure d'aquest càlcul que ja s'ha realitzat. Només hem de marcar les caixetes dels paràmetres que

[1]El W3C (*World Wide Web Consortium*) és un consorci internacional que treballa per a desenvolupar i promocionar estàndards per al Web.

vulguem veure i després seleccionar "RESULTS". A més també podem eliminar algun càlcul realitzat seleccionant-lo i "clickant" sobre "REMOVE", en aquest cas una finestra auxiliar s'obrira per preguntar-nos si realment estem segurs de voler eliminar-ho 7.4.

En l'última part, veiem la part d'estadístiques, imatge 7.5. A la part superior esquerra podem escollir entre la part d'anàlisis i la de comparacions, i com ja passava en la part de la base de dades, a l'esquerra els experiments que ja hem executat. Però a diferència de l'altre apartat, a la dreta tenim tots els càlculs estadístics que oferim depenent de l'apartat en el que estiguem (anàlisis o comparació). Encara que hi hagi aquesta petita diferència el funcionament és el mateix que en la part de la base de dades, és a dir, marquem: un experiment de l'esquerra, els càlculs estadístics que vulguem a la dreta i "clickem" sobre "ANALYZE". Llavors s'obrira una finestra emergent amb la gràfica o el resultat que hagem demanat, una exemple d'una d'aquestes finestres emergents seria l'imatge 7.6.

7.1.2 Entorn d'execució

En aquesta secció, veurem quins efectes té en el sistema una execució de la nostre aplicació, en el nostre cas l'interacció sempre serà amb un sistema Linux, amb la distribució Ubuntu per a servidors.

Una manera molt ràpida i gràfica de veure l'estat real i actual del sistema és mitjançant la comanda *top*[2], la qual realitzada en una màquina que té el rol de "worker", hi podem veure (imatge 7.7), com en aquella màquina, de forma transparent a l'usuari, de cop s'omple d'execucions de l'AutoDock[3].En aquest cas, tal com es pot apreciar en les tres primeres files del top, podem veure com hi havien tres AutoDocks corrents en aquell precís moment, des de feia ja temps.

D'altre banda a través de la comanda *ps*[4] també podem veure tots els processos que hi han actius en el moment. En l'imatge 7.8 mitjançant la comanda *ps -u nomUsuari*, la qual ens mostra els processos actius que té un usuari en concret, podem observar (imatge 7.8 com hi han 3 AutoDocks corrents i els diferents "scripts" de GRID Superscalar[5] tant de la part de

[2]És un programa que ens dona de forma dinàmica una visió en temps real de les execucions del sistema.

[3]Veure capítol 5

[4]Programa que mostra informació sobre una selecció dels processos actius.

[5]Veure capítol 4

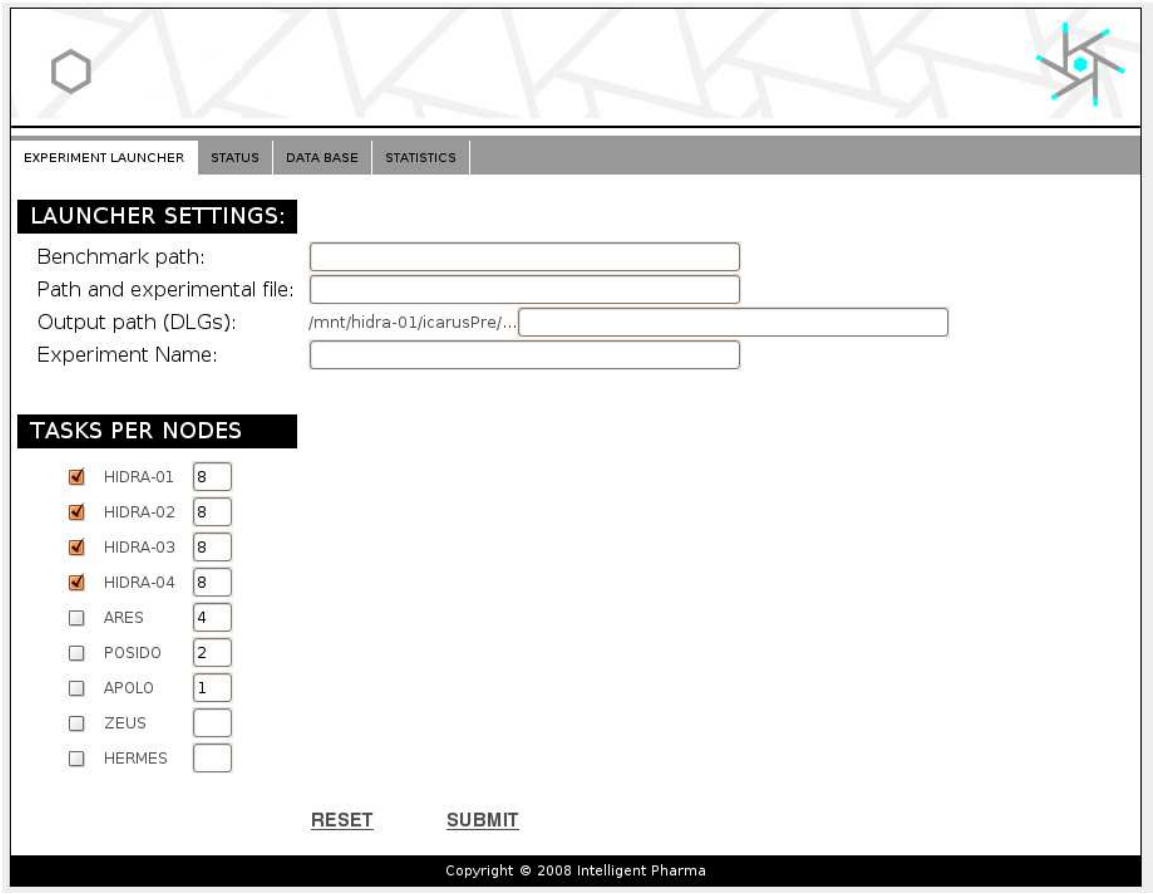


Figura 7.1: Pàgina princial de l'aplicació

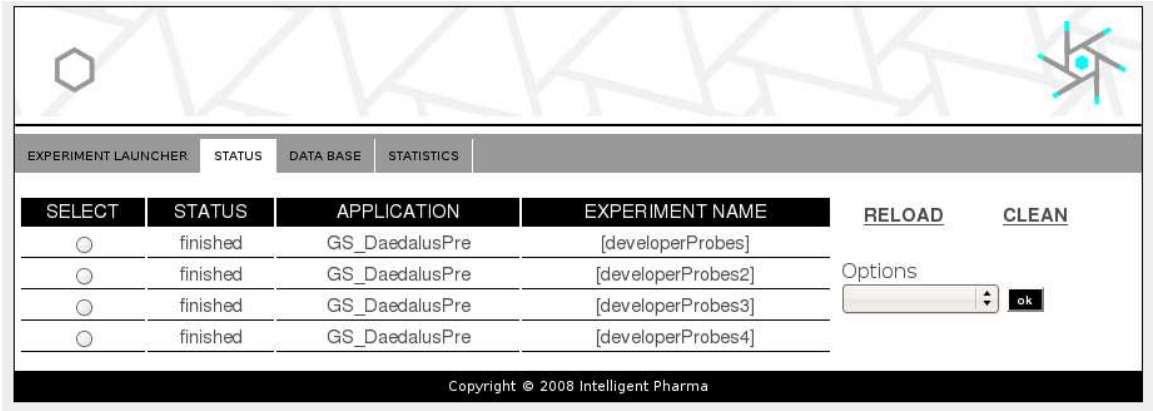


Figura 7.2: Part de l'estatus

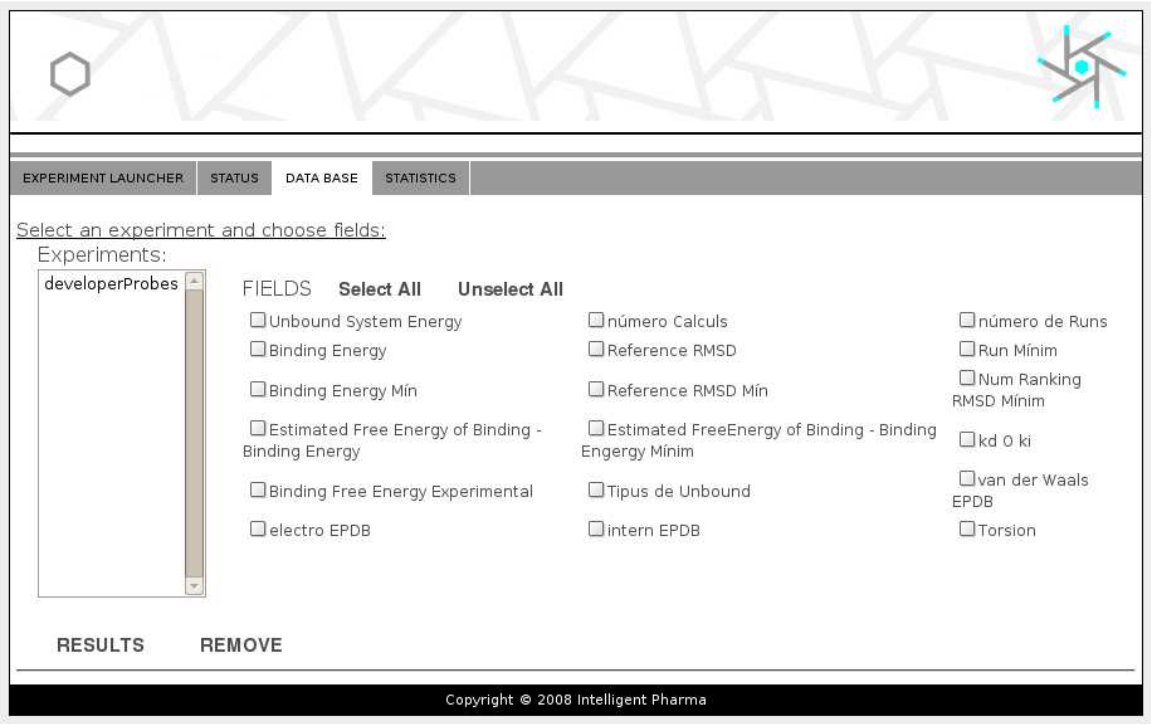


Figura 7.3: Gestió i consulta de la base de dades dels càlculs realitzats

Figura 7.4: Finestra d'alerta en cas d'eliminar una entrada a la taula d'experiments

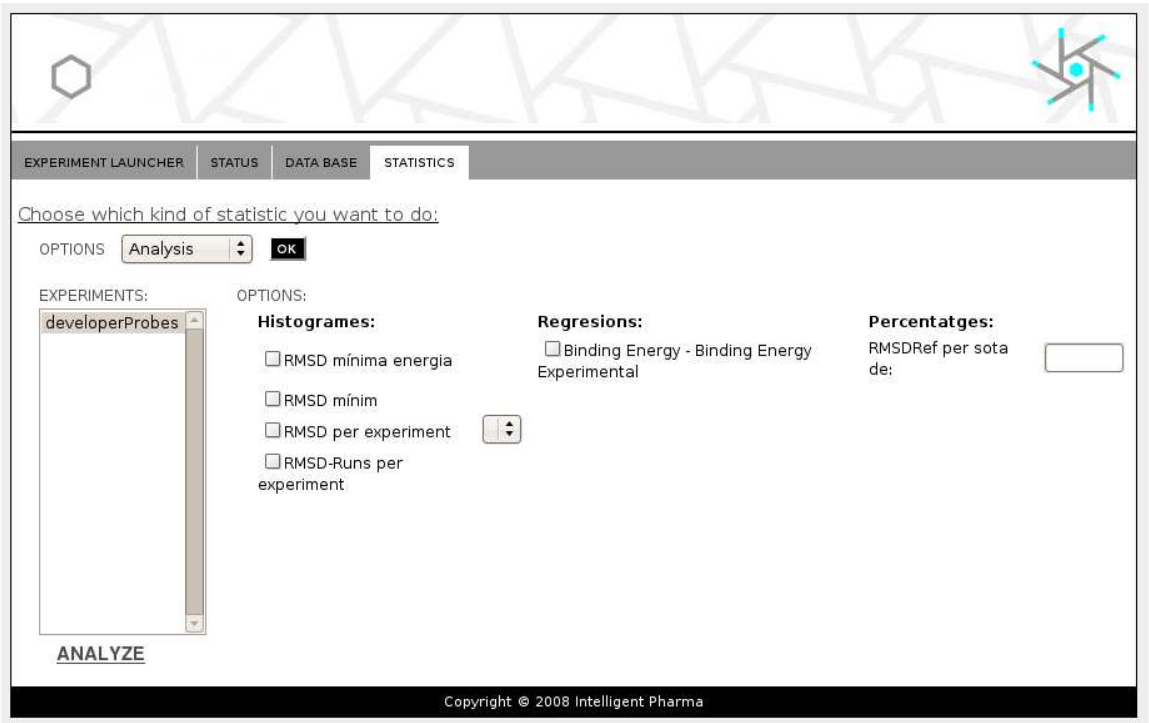


Figura 7.5: Diferents càlculs estadístics que l'aplicació ofereix — part d'anàlisis —.

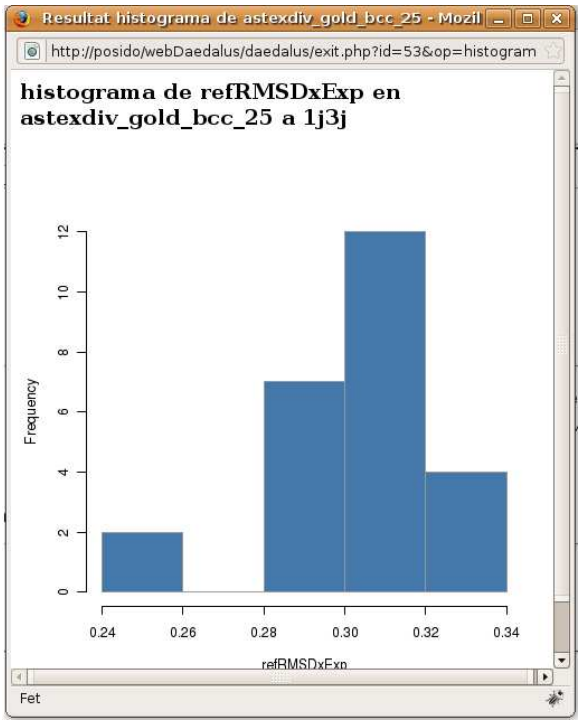


Figura 7.6: Finestra emergent que sorgeix com a conseqüència d'un càlcul estadístic

```
top - 16:21:51 up 92 days,  3:19,  2 users,  load average: 3.00, 3.00, 3.00
Tasks: 145 total,   4 running, 141 sleeping,   0 stopped,   0 zombie
Cpu(s): 14.0%us,  0.0%sy,  0.0%ni, 86.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16464672k total,  6332788k used, 10131884k free,   461876k buffers
Swap: 19535032k total,       88k used, 19534944k free,  5205568k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
30779 joan      25   0  236m  52m 1744 R  100  0.3  14:06.34 autodock4
30810 joan      25   0  236m  48m 1740 R  100  0.3  12:47.88 autodock4
30665 joan      25   0  236m  74m 1740 R   98  0.5  71:20.83 autodock4
    1 root      15   0  5116 1968  572 S    0  0.0   0:03.12 init
    2 root      10  -5     0    0    0 S    0  0.0   0:00.02 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.09 migration/0
    4 root      34  19     0    0    0 S    0  0.0   0:00.09 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.88 migration/1
```

Figura 7.7: Vista de la comanda top executada des del "shell" d'una màquina "worker".

```
  PID TTY          TIME CMD
29859 ?        00:00:00 GS_Time
29864 ?        00:00:00 execute_ssh.sh
29865 ?        00:00:00 execute_ssh.sh
30655 ?        00:00:00 workerGS_script
30659 ?        00:00:00 workerGS_script
30663 ?        00:00:00 workerGS
30664 ?        00:00:00 sh
30665 ?        01:10:16 autodock4
30727 ?        00:00:00 sshd
30728 pts/1    00:00:00 bash
30769 ?        00:00:00 workerGS_script
30773 ?        00:00:00 workerGS_script
30777 ?        00:00:00 workerGS
30778 ?        00:00:00 sh
30779 ?        00:13:02 autodock4
30800 ?        00:00:00 workerGS_script
30804 ?        00:00:00 workerGS_script
30808 ?        00:00:00 workerGS
30809 ?        00:00:00 sh
30810 ?        00:11:43 autodock4
30816 ?        00:00:00 sshd
30817 pts/2    00:00:00 bash
30850 pts/2    00:00:00 ps
```

Figura 7.8: Vista de la comanda "ps -u nomUsuari" executada des del "shell" d'una màquina "worker".

comunicació (ssh) com d'execució. Per altra part, de forma més detallada podem veure l'arbre de processos (imatge 7.9) al executar programa "ps" amb les opcions "axjf". On a més a més del que es veia anteriorment, ara podem veure part dels paràmetres que tenen els "scripts" de GRID Superscalar i les seves dependències.

7.1.3 Base de dades (MySQL)

Tal com hem comentat en capítols anteriors, tot aquest volum de dades s'emmagatzema directament a una base de dades, gestionada pel sistema MySQL. Per realitzar la gestió, visualització i control d'aquestes dades, hem emprat dos mètodes: a través d'una interfície web que ofereix php, que s'nomena phpMyAdmin i a través de la consola directament.

Normalment, per comoditat i rapidesa tots els temes relacionat amb les gestió, consultes, etc. de la base de dades es realitzen mitjançant la consola directament, tot i que a mesura que s'han anat fent cada cop més grans les

Figura 7.9: Vista de la comanda "ps axjf" executada des del "shell" d'una màquina "worker".

taules o per realitzar proves de consultes a la base de dades més complicades... Llavors, sí que hem usat l'interfície web, ja que aquesta interfície és més "amigable" i resulta més còmoda per realitzar consultes i per observar més detingudament els resultats recol·lectats. De fet, hem de remarcar que l'usuari no ha d'arribar mai a entrar en aquesta part, aquestes consultes només es realitzen en el procés de desenvolupament per estar del tot segur que les dades extretes de la sortida de l'AutoDock són correctes, per realitzar proves amb las crides SQL.

A continuació en l'imatge 7.10 podem veure la descripció de la taula "resultat" (ja comentada en el capítol 3) a través de la web phpMyAdmin i en l'imatge 7.11 la mateixa taula des de la consola.

A més, en la següent imatge 7.12 podem veure aquesta mateixa taula, la de resultats, part de les dades recol·lectades i emmagatzemades en la nostra base de dades, a través de la web de phpMyAdmin, al realitzar-hi un experiment amb la base de dades de complexes proteïna-lligand "*Astex divers*".

Camp	Tipus	Ordenaci	Atributs	Nul	Defecte
codi	varchar(20)	latin1_swedish_ci		No	
idExp	int(11)			No	0
unboundSys	float			Si	*NULL*
numCalc	int(11)			Si	*NULL*
numRuns	int(11)			Si	*NULL*
bindingEng	float			Si	*NULL*
refRMSD	float			Si	*NULL*
runMin	int(11)			Si	*NULL*
bindingEngMin	float			Si	*NULL*
refRMSDMin	float			Si	*NULL*
numRankRMSDMin	int(11)			Si	*NULL*
estFreeEngBindingEng	float			Si	*NULL*
estFreeEngBindingEngMin	float			Si	*NULL*
kdOki	float			Si	*NULL*
bindingFreeEngExp	float			Si	*NULL*
tipusUnbound	enum('extended', 'ad4')	latin1_swedish_ci		Si	*NULL*
vdWEPDB	float			Si	*NULL*
electroEPDB	float			Si	*NULL*
internEPDB	float			Si	*NULL*
torsion	float			Si	*NULL*

Figura 7.10: Vista de la taula "resultat" de la base de dades amb phpMyAdmin.

```
mysql> describe resultat;
+-------------------------+-----------------------+------+-----+---------+-------+
| Field                   | Type                  | Null | Key | Default | Extra |
+-------------------------+-----------------------+------+-----+---------+-------+
| codi                    | varchar(20)           | NO   | PRI |         |       |
| idExp                   | int(11)               | NO   | PRI | 0       |       |
| unboundSys              | float                 | YES  |     | NULL    |       |
| numCalc                 | int(11)               | YES  |     | NULL    |       |
| numRuns                 | int(11)               | YES  |     | NULL    |       |
| bindingEng              | float                 | YES  |     | NULL    |       |
| refRMSD                 | float                 | YES  |     | NULL    |       |
| runMin                  | int(11)               | YES  |     | NULL    |       |
| bindingEngMin           | float                 | YES  |     | NULL    |       |
| refRMSDMin              | float                 | YES  |     | NULL    |       |
| numRankRMSDMin          | int(11)               | YES  |     | NULL    |       |
| estFreeEngBindingEng    | float                 | YES  |     | NULL    |       |
| estFreeEngBindingEngMin | float                 | YES  |     | NULL    |       |
| kdOki                   | float                 | YES  |     | NULL    |       |
| bindingFreeEngExp       | float                 | YES  |     | NULL    |       |
| tipusUnbound            | enum('extended','ad4') | YES  |     | NULL    |       |
| vdWEPDB                 | float                 | YES  |     | NULL    |       |
| electroEPDB             | float                 | YES  |     | NULL    |       |
| internEPDB              | float                 | YES  |     | NULL    |       |
| torsion                 | float                 | YES  |     | NULL    |       |
+-------------------------+-----------------------+------+-----+---------+-------+
20 rows in set (0.01 sec)
```

Figura 7.11: Vista de la taula "resultat" de la base de dades de de la consola.

codi	idExp	unboundSys	numCalc	numRuns	bindingEng	refRMSD	runMin	bindingEngMin	refRMSDMin	numRankRMSDMin	estFreeEngBindingEng
1g9v	84	-0.442	14	45	-6.19	4.71	18	-5.54	1.32	4	6.19
1gkc	84	-1.322	27	33	-12.34	0.45	33	-12.34	0.45	1	12.34
1gm8	84	-1.723	18	27	-5.44	1.4	25	-4.86	0.51	1	5.44
1gpk	84	0.734	50	11	-7.32	0.35	14	-7.32	0.33	1	7.32
1hnn	84	0.234	50	7	-5.46	0.74	23	-5.13	0.54	1	5.46
1hp0	84	-0.28	49	5	-6.42	0.34	37	-6.4	0.29	1	6.42
1hq2	84	0.027	50	20	-4.74	0.4	18	-4.7	0.34	1	4.74
1hvy	84	-1.216	1	34	-5.69	2.51	36	-4.79	1.45	4	5.69
1hwi	84	0.23	22	7	-9.69	0.72	20	-9.07	0.7	1	9.69
1hww	84	-0.215	50	28	-7.71	0.54	31	-7.68	0.51	1	7.71
1ia1	84	0.488	50	27	-6	0.41	19	-6	0.39	1	6
1ig3	84	0.752	24	23	-4.55	1.14	21	-3.77	1.01	1	4.55
1j3j	84	0.099	50	39	-6.04	0.32	14	-6.03	0.28	1	6.04
1jd0	84	0.261	22	5	-6.07	1.17	16	-5.98	0.8	1	6.07
1jje	84	-0.415	23	44	-19.82	1.12	9	-19.67	0.93	1	19.82
1jla	84	-0.047	38	34	-8.46	0.6	9	-8.21	0.4	1	8.46
1k3u	84	-0.148	48	32	-6.64	0.26	18	-6.62	0.22	1	6.64
1ke5	84	0.793	28	15	-5.88	0.92	35	-5.5	0.47	1	5.88
1kzk	84	-0.697	8	47	-12.4	0.67	23	-12.2	0.58	1	12.4
1l2s	84	-1.126	50	50	-6.33	0.93	6	-6.16	0.78	1	6.33
1l7f	84	1.029	39	1	-8.26	0.53	6	-8.04	0.36	1	8.26
1lpz	84	0.198	1	9	-7.47	3.56	7	-5.75	2.91	9	7.47
1lrh	84	0.002	50	47	-8.41	0.68	14	-8.35	0.53	1	8.41
1m2z	84	-0.183	50	13	-10.11	0.65	50	-10.09	0.63	1	10.11
1meh	84	0.275	48	33	-6.02	2.72	28	-5.24	1.75	2	6.02
1mmv	84	0.117	25	5	-5.92	0.56	5	-5.92	0.56	1	5.92

Figura 7.12: Vista d'una part dels valors emmagatzemats en la taula "resultat" de la base de dades a través de phpMyAdmin.

7.2 Càlcul d'un experiment

Un dels experiments que vam calcular amb la nostra aplicació, tractava d'una base de dades de l'empresa "Astex Therapeutics"[6] anomenada "*Astex diverse*". Aquesta base de dades és usada normalment en els experiments de "*benchmarking*" (comparació o banc de proves) — ús pel que s'ha dissenyat la nostra aplicació —, perquè hi han estructures d'alta qualitat ja que les posicions dels àtoms, tant de l'estructura de les proteïnes com dels lligands estan ben definides. Per tant, podem executar els nostres processos computacionals de docking amb un error mínim dins de la precisió que aquests software puguin donar-nos. A part de la gran qualitat de les estructures, també és útil usar-la en "*benchmarking*" perquè tal com el seu nom indica, és bastant diversa, i això és positiu en aquest tipus d'estudis ja que es usar per posar a prova el dockings que hom realitzi, així doncs, aquesta base de dades dona valors estructurals i experimentals de proteïnes i lligands ben diferents entre ells, així com més diversitat hi hagi més casos diferents es hi hauran, més càlculs diferents es realitzaran i per tant, més errors es podran detectar en els diferents processos de docking que es realitzin.

En definitiva, aquesta "*Astex diverse*" consta de 85 unions, realitzades en un laboratori, entre una proteïna i un lligand, tots dos sempre diferents. A més, un fet força interessants és que d'entre totes aquestes molècules de les que consta aquesta base de dades tenim proteïnes que al trobar-se en mal estat, o al haver-hi en una proporció massa elevada en l'organisme o per altres motius, generen enfermetats com: el càncer (proteïnes "p38 kinase" o "Vascular Endothelial Growth Factor receptor 2"), el SIDA (proteïnes "HIV-1 reverse transcriptase" o "HIV-1 protease"), hepatitis C (proteïna "NS5B polymerase"), la diabetis (proteïna "dipeptidyl peptidase IV") i inclús d'altres proteïnes que ajuden a que certs antibiòtics perquè tinguin més eficàcia (β-lactamasa), ja que a causa de l'ús indiscriminat d'aquests cada cop estant perdent més eficàcia.

Un cop ja sabem el perquè de l'ús d'aquests base de dades "*Astex diverse*", passem a realitzar l'execució dels 85 diferents AutoDocks necessaris, per poder comparar els resultats amb d'altres execucions realitzades amb anterioritat d'aquesta mateixa base de dades amb diferents modificacions introduïdes per millorar el procés de docking, com són: millores en les estructures, adició d'aigües, etc. Un cop realitzats els càlculs, procedim a observar

[6] "Astex Therapeutics" és una empresa biotecnològica dirigida al descobriment i disseny de fàrmacs en oncologia i altres àreas. Fundada l'any 1999 i d'origen Britànic.

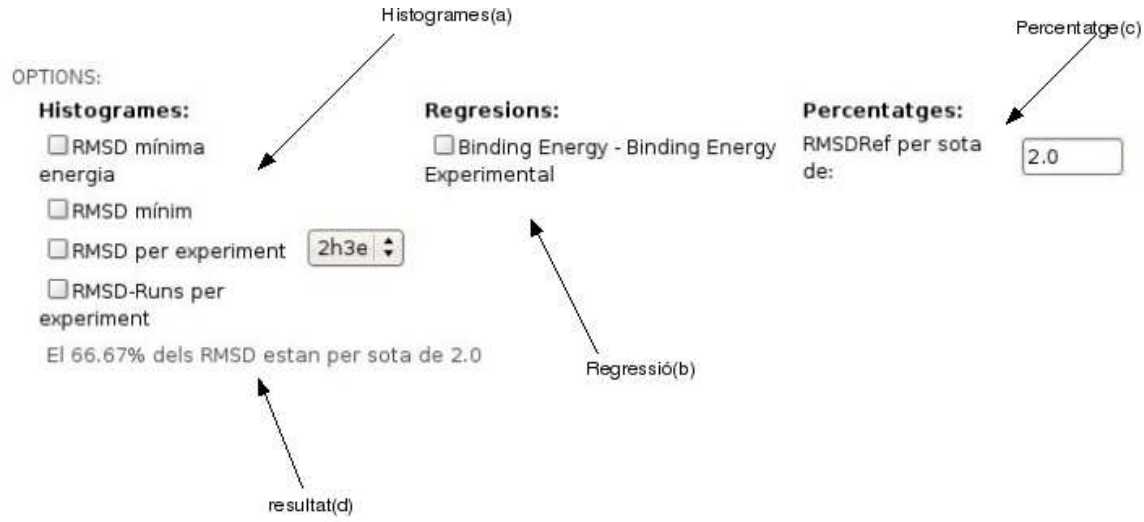


Figura 7.13: Imatge de la part d'opcions d'estadística de la web.

els diferents càlculs estadístics d'anàlisis i de comparació que hem realitzat amb l'execució d'aquest experiment, per ser més exactes treballarem sobre un experiment que vam realitzar amb aquesta base de dades on varem introduir aigües en l'estructura de la proteïna, tot això realitzat per la nostra part mitjançant complexos programes, en els càlculs de docking.

7.2.1 Anàlisis

Aprofitant les eines estadístiques que té la nostra aplicació veiem a continuació alguns resultats que podem extreure de l'execució de l'experiment de la base de dades d'Astex comentada anteriorment. En l'imatge 7.13 podem veure les diferents parts d'anàlisis estadístic implementades actualment.

Vegem doncs a continuació l'histograma d'RMSD de mínima energia (primera opció de l'apartat "a" de la figura 7.13), aquí podem veure els RMSD resultants de les diferents execucions on l'AutoDock ha donat com a millor ancoratge per tenir el valor mínim de l'energia d'unió ("binding energy"), en la figura 7.14 podem veure aquest histograma. Podem veure com 55 de les diferents execucions tenen un RMSD entre 0 i 1 i 15 estan entre 1 i 2, per tant tenim 70 execucions on l'RMSD està per sota de 2.0, que venen a ser les prediccions que podem donar per bones, la resta les hem de rebutjar.

Exactament en aquest cas tenim el 81.18% dels RMSD per sota o igual

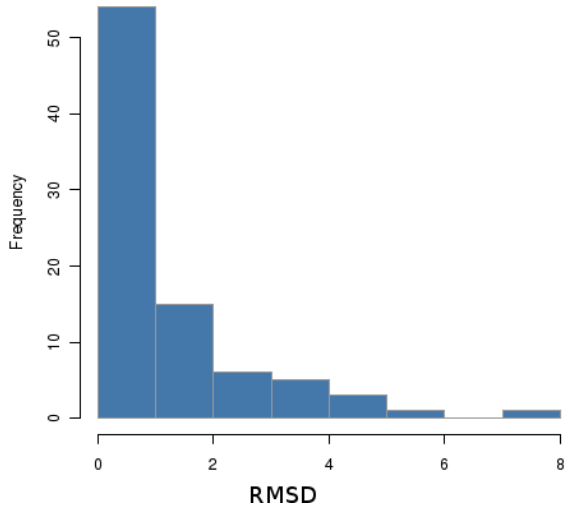


Figura 7.14: Vista de l'histograma *RMSD de mínima energia*.

a 2.0. Això ho sabem amb aquesta precisió gràcies a una altre opció que ofereix la web (apartat "c" de la figura 7.13), aquí li introduïm un valor del RMSD i ens calcula en quin percentil es troben les execucions que estan per sota del valor d'RMSD que hagem introduït.

Després en la segona opció de l'apartat "a" de la figura 7.13 tenim l'histograma d'RMSD mínim, figura 7.15, en aquest càlcul veiem la freqüència dels valors mínims de l'RMSD, en aquest cas veiem que en totes les execucions, menys 6, l'AutoDock a trobat una conformació amb un RMSD inferior a 2.0, en aquesta gràfica hi han més execucions amb un RMSD inferior a 2.0. Tot i que haurien de donar els mateixos valors que els de l'histograma anterior, això no passa degut a que de vegades l'AutoDock troba una bona conformació on l'RMSD és molt baix però la funció d'avaluació ("scoring") no calcula un valor baix de l'energia d'unió. Aquesta comparativa entre aquest dos anàlisis és molt bona si es desitja modificar la funció d'avaluació de l'AutoDock o l'algorisme de cerca.

En la tercera opció de l'apartat "a" de la figura 7.13 podem veure els valors de l'RMSD dels diferents avaluacions ("runs") que ha fet l'AutoDock d'una execució en concret. Recordem que en una execució l'AutoDock per determinar el millor ancoratge primer realitza un número d'execucions interiors independents i d'aquestes primeres selecciona les millors avaluacions, després d'aquest subconjunt de candidats agafa el millor d'ells, el que tingui

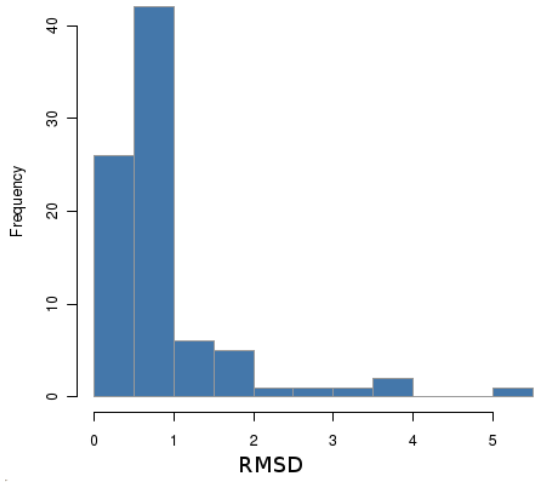


Figura 7.15: Vista de l'histograma *RMSD mínim*.

l'energia d'unió més baixa. En aquest cas, figura 7.16 veiem que hi han 21 avaluacions on RMSD és igual o inferior a 2.0, però per contra hi ha un gran número que estan per sobre, depenent de l'espai de cerca d'aquest problema també ens permet observa si la funció d'avaluació o la cerca són suficientment òptimes.

En l'últim histograma, figura 7.17, tenim l'RMSD de tots els resultats de les millors avaluacions independents realitzades en cada càlcul per l'AutoDock juntes. D'aquesta manera podem veure quina és la tònica que segueix l'algorisme de cerca i la funció d'avaluació, com més eficient sigui la combinació d'aquests dos elements la freqüència d'RMSD baixos hauria d'augmentar.

Seguint amb l'últim apartat que oferim en l'anàlisis, tenim la regressió lineal entre l'energia d'unió que calcula l'AutoDock i aquesta mateixa energia obtinguda de l'experimental, apartat "b" de la figura 7.13. Tant la gràfica resultant com els càlculs que extraiem els podem veure a la figura 7.18. En aquesta figura podem veure com hi ha una gran dispersió en el núvol de punts que formen el diagrama de dispersió, això ens mostra com s'ajusten les energies d'unió predites per l'AutoDock a les de l'experimental, en aquest cas, degut a aquesta dispersió veiem que no són gens iguals els valors d'aquestes dues energies, això és normal, ja que com hem comentat al llarga d'aquesta memòria l'AutoDock posa per sobre l'eficàcia a l'eficiència, sinó si hagués de cercar tots els ancoratges possibles en un espai infinit no acabaria mai. A més, a part de la gràfica de diagrama de dispersió també dibuixem la recta de

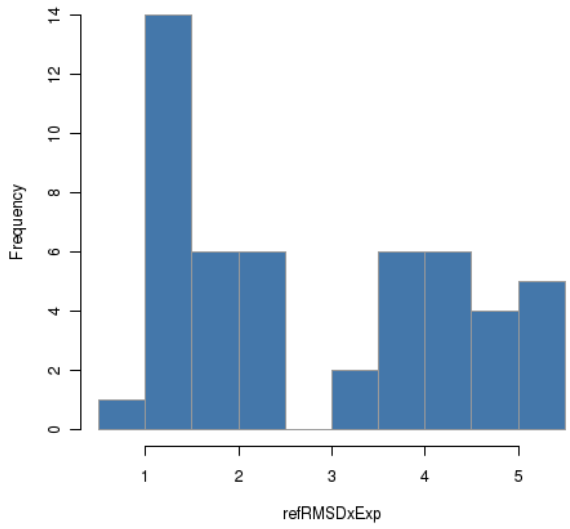


Figura 7.16: Vista de l'histograma *RMSD per experiment*.

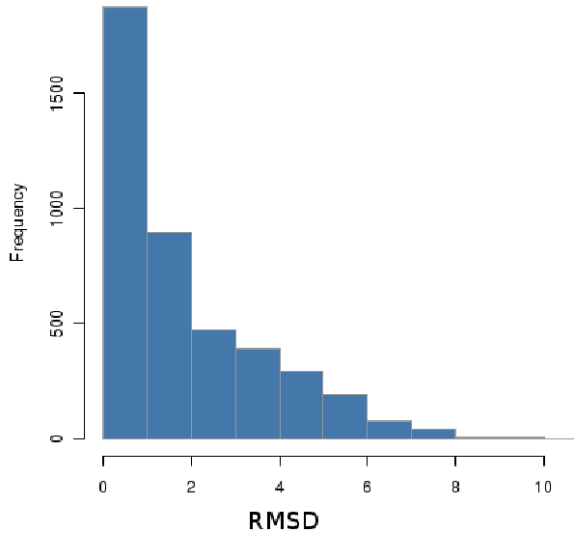
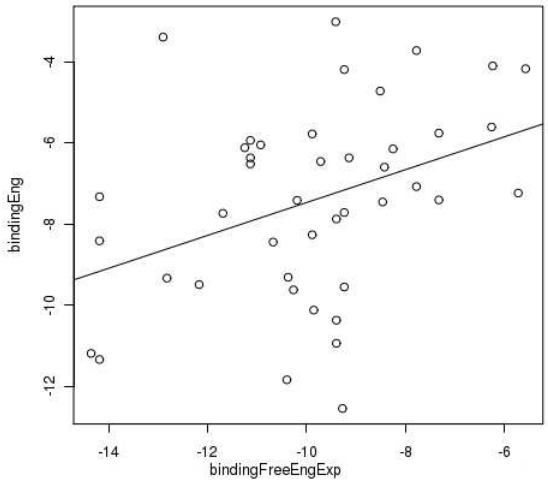


Figura 7.17: Vista de l'histograma *RMSD-Runs per experiment*.




```
Call:
lm(formula = as.formula(paste(names(dades)[1], "~", names(dades)[2])),
    data = dades)

Residuals:
    Min      1Q  Median      3Q     Max
-5.3809 -1.3240  0.2349  1.6059  5.2611

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       -3.4085     1.5680  -2.174   0.0356 *
bindingFreeEngExp  0.4056     0.1550   2.617   0.0124 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.231 on 41 degrees of freedom
Multiple R-squared: 0.1431,     Adjusted R-squared: 0.1222
F-statistic: 6.847 on 1 and 41 DF,  p-value: 0.01238
```

Figura 7.18: Vista de la regressió *Binding Energy - Binding Energy Experimental*.

la regressió, al costat d'aquesta gràfica podem veure l'arrel quadrada del coeficient de correlació R^2, en la figura 7.18 veiem "Multiple R-squared":0.1431 i "Adjusted R-squared":0.1222. Aquest coeficient és un indicador de com de bona és la regressió, és a dir, si la línia de regressió aconsegueix travessar o no una gran número de punts. En el nostre cas com ja hem vist a simple vista el no és gaire bona aquesta recta, el coeficient és molt baix.

Mitjançant aquest apartat d'anàlisis ja podem començar a treure conclusions sobre com ha anat l'execució. En aquest cas particular veiem que s'han obtingut resultats satisfactoris, la majoria dels RMSDs estan per sota de 2.0. Ara falta comparar aquestes dades amb altres execucions realitzades sobre aquesta mateixa base de dades amb alguna modificació per provar si estem millorant això que ja tenim o no.

Per saber si realment estem afegint millores al procés, realitzem una nova execució afegint-hi les millores i mitjançant les eines d'anàlisis que tenim primer, analitzem de forma individual els resultats per veure si s'han obtinguts valors satisfactoris i després passem a comparar els resultats obtinguts en l'anàlisis. Per seguir amb aquest exemple, hem executat la mateixa base de dades amb una millora que hem realitzat que introdueix aigua en el procés de docking , però no deixem que l'aigua es tingui llibertat i es pugui situar d'infinites maneres, nosaltres la situem tenint en compte les diferents interac-

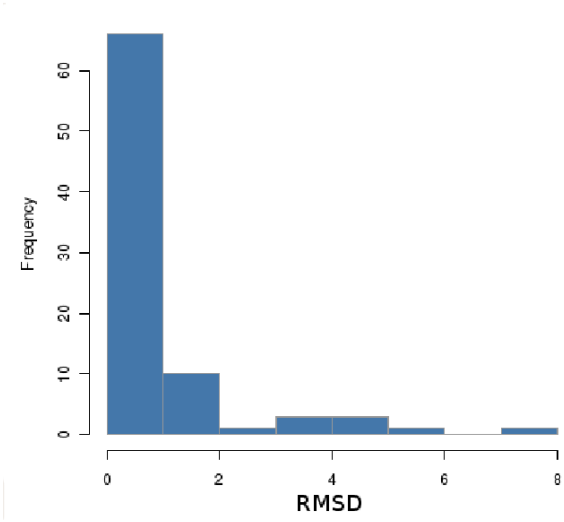


Figura 7.19: Vista de l'histograma *RMSD de mínima energia* amb aigües.

cions entre la proteïna i el lligand.

Vegem els diferents histogrames resultants: RMSD de mínima energia, RMSD mínim, RMSD-Runs per experiment que corresponen a les figures 7.19, 7.20 i 7.21 respectivament.

A més, gràcies a l'eina que calcula quin percentatge de càlculs han obtingut un RMSD inferior a 2.0, ens diu que el 89.41% ho compleixen, per tant sembla ser que a primera vista hem obtingut unes millores d'un 8.23% respecte l'execució anterior. De fet, d'una manera no tant precisa observant els histogrames podem veure com les columnes que representen la freqüència de càlculs que han obtingut un valor d'entre 0 i 2 també a augmentat en detriment de la baixada de les demés columnes. D'altre banda podem anar analitzant de manera més concreta aquests resultats observant l'RMSD de cada experiment individual un a un amb l'opció de *RMSD per experiment* i així també veuríem com afecta la nostra millora de forma més concreta.

De forma visual podem veure que un experiment sembla diferent a l'altres, passem ara a comprar-los per saber si realment són significativament diferents un de l'altre.

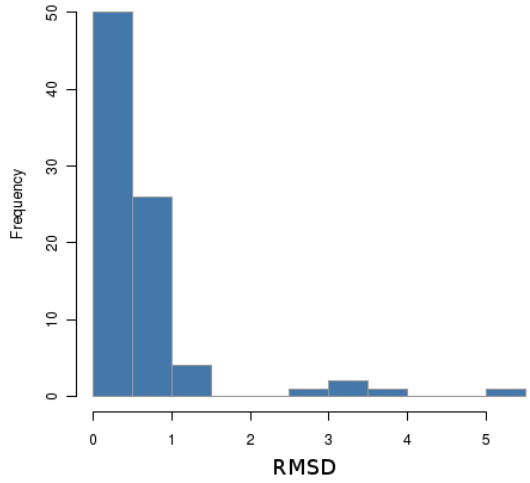


Figura 7.20: Vista de l'histograma *RMSD mínim* amb aigües.

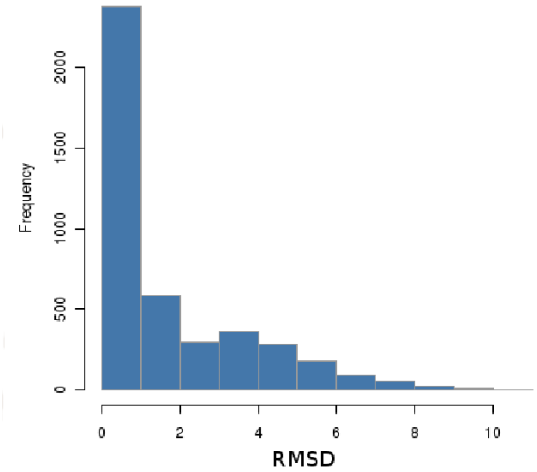


Figura 7.21: Vista de l'histograma *RMSD-Runs per experiment* amb aigües.

7.2.2 Comparacions

Un cop hem analitzat aquests dos experiments i hem observat que sembla existir una petita millora entre els dos, passem a realitzar el test de wilcoxon per assegurar-nos si realment existeixen diferències entre un i l'altre o simplement a causa del procés estocàstic de l'AutoDock hem obtingut resultats diferents però proporcionalment iguals.

En aquest cas, realitzem el test de wilcoxon on l'hipòtesis nul·la és que els dos experiments són iguals i com a hipòtesis alternativa que el primer és menor que el segon. Per tant, com l'experiment amb las nostras millores creem que és menor, llancem el test de wilcoxon amb aquesta població com a primer paràmetre i l'"astex diverse" sense modificar com a segon paràmtera. El resultats que obtenim amb l'R alhora d'executar aquest test usant l'energia d'unió ("Binding Energy") i l'RMSD els podem veure a la taula 7.2.

Taula 7.1:

Dades	W^+	$p - value$	Resultat
Energia d'unió	379.5	1.026e-08	alternative hypothesis: true
RMSD	435	4.088e-07	alternative hypothesis: true

Taula 7.2: Resultat del test de Wilcoxon extret per l'R entre "astex diverse" i "astex diverse" amb aigües

Per tant, podem veure com en el dos casos el $p - value$ és inferior a 0.05 i l'hipòtesis alternativa és certa tal com ens indica l'R. Així doncs no només podem afirmar que els dos experiments són significativament diferents, sinó que a més podem afirmar que l'execució amb la bases de dades d'"astex diverse" modificada per nosaltres on introduïm aigües tenint en compte les interaccions entre la proteïna i el lligand té valors inferiors d'RMSD i per tant hem millorat el procés de docking respecte l'execució anterior, recordem que com més petit és el valor de l'RMSD significa que més s'assembla a l'experimental. Per altra banda, que el test de wilcoxon ens digui que també sigui inferior a nivell d'energia d'unió significa que les unions són més estables, de forma teòrica si l'RMSD baixa l'energia també ho hauria de fer tal com ha passat, però en realitat això depèn de l'eficàcia de la funció d'avaluació de l'AutoDock.

Un cop hem arribat a les conclusions explicades en el paràgraf anterior és força interessant llançar un nou experiment, però en aquest cas deixarem

que els hidrògens de l'aigua es situïn de manera aleatòria, així podrem estar segur que les orientacions dels hidrògens que proposem són correctes. És fa d'aquesta manera perquè anteriorment només situàvem els hidrògens en la conformació que nosaltres creiem convenient de l'aigua, perquè de fet els àtoms de l'oxigen sempre hi són, el que no tenim són el dos àtoms d'hidrogen que falten. Repetim el test de wilcoxon i de forma satisfactòria tornem a rebutjar l'hipòtesis nul·la, tal com es mostra en la taula 7.4.

Taula 7.3:

Dades	W^+	$p - value$	Resultat
Energia d'unió	282	1.85e-10	alternative hypothesis: true
RMSD	1026	0.01191	alternative hypothesis: true

Taula 7.4: Resultat del test de Wilcoxon extret per l'R entre "astex diverse" amb aigües i "astex diverse" amb hidrògens aleatoris

És a dir, les orientacions que em definit pels hidrògens són correctes, ja que obtenim valors de l'RMSD inferiors i per tant més propers a l'experimental altre cop.

Per finalitzar a la figura 7.22 podem veure una comparativa entre els diferents valors de l'energia d'unió entre els resultats obtinguts usant l'"astex diverse" sense modificar i l'"astex diverse" amb aigües. Aquesta gràfica té l'energia ordenada ascendentment i a més, aparella els dos valors obtinguts en un mateix càlcul entre els dos experiments. De tal manera que els punts verds són els casos que els dos ordres coincideixen, els vermells quan és inferiors i els blaus superior. Així sí els càlculs fossin iguals obtindríem com a resultat una línia amb una pendent de 45 ° de punts verds. En aquest cas veiem com no és així i que a més observem uns pocs punts vermells més que blaus, això descriu l'efecte comentat abans, és a dir són diferents.

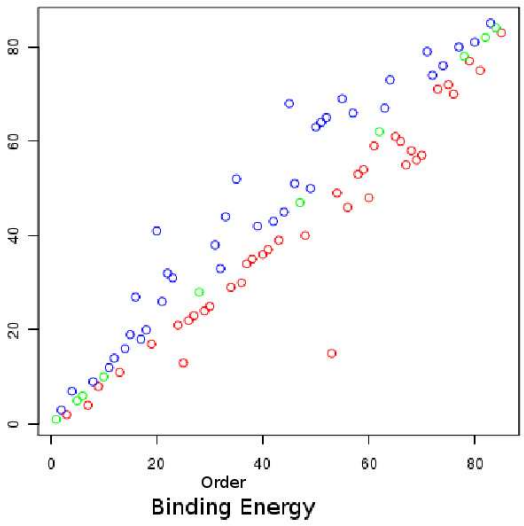


Figura 7.22: Comparativa entre energies ordenades per ordre de magnitud.

Capítol 8

Conclusions

Hem desenvolupat una aplicació bioinformàtica que executa de forma paral·lela una aplicació de docking anomenada AutoDock, la qual té un gran cost computacional. A més, emmagatzema de forma automàtica les dades extretes per l'AutoDock, ofereix tot un seguit d'eines per gestionar les dades, realitzar diversos càlculs estadístics d'anàlisis i de comparació. Així doncs, d'aquest projecte acabem extraient les següents conclusions:

1. Escalabilitat. S'ha desenvolupat un software escalable que s'utilitza de forma distribuïda per executar una sèrie de processos en paral·lel dins d'un GRID i/o clúster gràcies al GRID Superscalar.

2. Usabilitat. El software dissenyat té una aplicabilitat immediata en la biocomputació, conté varis elements interdisciplinaris com l'informàtica, les matemàtiques aplicades, estadística, intel·ligència artificial, química i bioquímica per solucionar problemes, analitzar dades i realitzar simulacions.

3. Independència. No manté cap tipus de dependència amb cap sistema ja que per accedir-hi només es necessita un navegador web, el qual accedeix a una interfície senzilla, fàcil d'usar i de navegar-hi.

4. Augment de l'eficiència.

    - A nivell computacional, augmentem la capacitat per realitzar càlculs d'alts costos computacionals: distribuint la càrrega entre varies màquines, ajustant al mil·límetre els paràmetres del GRID Superscalar i optimitzant al màxim les aplicacions usades.
    - A nivell humà, reduïm el temps destinat a executar càlculs de docking de mitja i gran envergadura, facilitem l'observació de les dades adquirides i realitzem varis càlculs estadístics amb les dades obtingues.

A més, em de tenir clar que aquesta aplicació no té una aplicació únicament teòrica, sinó tot al contrari, té un ús real en la biomedicina i dins de la nostre empresa, on aquesta aplicació està sent usada ara mateix pels nostres experts en la matèria per realitzar millores en el procés de docking.

La millora del procés de docking és crucial pel disseny de nous fàrmacs, i en particular, en les etapes inicials del disseny. Per exemple, en la base de dades utilitzada en la validació de l'aplicació hi ha casos tant importants per la recèrca biomèdica moderna com: el *receptor de l'hormona del factor de creixement*, una proteïna clau en el desenvolupament tumoral; el *receptor de*

la vitamina D3, implicada en osteoporosi; el *factor leal del B. Antrax*; o la *hidrolasa del virus de la SIDA*.

8.1 Costos

Com és obvi el desenvolupament d'aquest projecte ha generat tota una sèrie de costos, els quals passem a analitzar tot seguit:

A **nivell temporal** aquest projecte es va començar a desenvolupar a finals de Març, i es va finalitzar a mitjans de Novembre amb un total de 152 dies i un total de **850 hores**. En aquestes hores s'hi inclou un curs de formació en GRID Superscalar realitzat pel BSC (Barcelona Supercomputing Center), recerca i documentació, anàlisis i disseny de la base de dades, desenvolupament del software (presentació, domini i dades) i redacció de la memòria del projecte.

A **nivell computacional** contem que hem tingut un cost aproximat de **3250 hores**, 50 hores de treball en l'ordinador personal de l'empresa i 200 hores en el clúster de l'empresa on en cada execució feia un ús mig de 16 processadors, per tant $200 * 16 = 3200$hores. D'aquest número d'hores incloem varies execucions de l'AutoDock, proves individuals dels diferents mòduls que formen el daedalus, diverses proves globals del daedalus on es posava a prova diferents elements com l'inserció a la base de dades, correcte comportament del GRID Superscalar, etc. A més, de tot un seguit d'experiments destinats a optimitzar l'AutoDock i la configuració més acurada possible del GRID Superscalar. Hem de tenir en compte que la gran majoria d'aquestes hores de computació es realitzaven de forma paral·lela al propi desenvolupament o inclús fora de les hores laborals.

A **nivell econòmic** la realització del projecte ha tingut un cost total de **22050.91€**. Aquesta xifra prové de: 1452€ degut als costos computacionals comentats en el punt anterior, 9877.5€ pel desenvolupament més 2200€ per la direcció del projecte, això es calcula mitjançant el temps de treball que hi he destinat segons he comentat al principi d'aquest apartat; 6516.81€ destinats a la meva formació (en el Barcelona Supercomputing Center) i 2004.6€ en costos indirectes, que corresponen al 10% del total.

8.2 Línies de futur

Aprofitant l'escalabilitat de la qual parlàvem anteriorment, aquest aplicació ens permet seguir treballant entorn a ella en diferents projectes. De fet, ja s'ha aprofitat el nucli de daedalus per realitzar altres projectes, alguns on també s'hi executa l'AutoDock i d'altres més complexos. De moment, s'estan duent a terme dos projectes amb el seu motor, d'altres relacionats amb intel·ligència artificial i algorismes genètics encara estan en procés de desenvolupament, i algun més encara està a l'espera de ser desenvolupats.

Un d'ells és el *Selene*, el qual també utilitza l'AutoDock, però en aquest cas té d'altres funcionalitats, enlloc d'estar dissenyat pel *benchmarking* com era el cas del daedalus, en aquest cas està orientada al *Virtual Screening*, és a dir, s'utilitza en el procés de desenvolupament de nous fàrmacs ("drug discovery") a partir del receptor (proteïna), per tant la base és semblant, ja que gran part d'aquest procés gira entorn al docking també. S'ha modificat el tractament de les dades en la base de dades, s'analitzen altres tipus de dades més interessant per aquesta aplicació, en la part d'estadístiques analitzem i comparem tipus de dades diferents, etc..

L'altre que ja està casi finalitzat és l'*Helios*, té el mateix objectiu que l'anterior, és a dir ajudar en el desenvolupament de nous fàrmacs, però en aquest cas aquest ho fa a partir del lligand, a més respecte les altres aplicacions aquesta té la particularitat que no utilitza l'AutoDock i tot hi això segueix tenint un motor molt semblant al del daedalus.

Tot i el treball realitzat, no ens conformem amb el aquest resultat i ja estem treballant amb noves millores per augmentar el rendiment i la capacitat computacional, com és el cas de les següents millores amb les quals ja estem començant a treballar:

Millorar el nucli del daedalus, ja que el principal problema que tenim amb el nucli d'aquesta aplicació és que no pot entrar una nova execució d'un altre experiment fins que tots els AutoDocks anteriors no han acabat. Així doncs, de vegades ens trobem que un nou experiment no pot entrar en execució perquè, posem per exemple que, davant seu hi ha una execució a la qual li falta per acabar n AutoDocks (n és inferior al número de processadors dels qual disposem), i això fa que, $34-n$ processadors estarien desaprofitats (sent 34 el màxim de processador que poguéssim disposar). Una altre situació que també resulta habitual és quan es llança un experiment amb menys execucions de l'AutoDock com processadors disposem, per tant succeirà el mateix

efecte descrit anteriorment. Per tant, una idea en la qual estem treballant és en el fet de convertir el GRID Superscalar en un "daemon", el qual estigués sempre corrent i cada nova execució passaria a encuar-se en una base de dades on el GRID Superscalar estaria consultant contínuament per saber si té noves execucions per realitzar, i així GRID Superscalar no separarà cada experiment com una execució independent, i aconseguiríem introduir noves execucions d'altres experiment tant aviat com hi haguessin processadors disponibles.

Millorar la capacitat computacional, un altre de les millores que estem estudiant és introduir *Cloud computing*[1] al daedalus. Aquest tipus de computació està relacionada amb l'habilitat de proveir una nova capacitat computacional com un servei, permetent als usuaris poder-hi accedir a través d'Internet (en el "cloud", núvol) sense que aquest tingui ni els coneixements, ni les habilitats, ni el control sobre la tecnologia de l'infraestructura a la qual dona suport. Per realitzar-ho, pretenem usar l'infraestructura de *Cloud computing* que ofereix l'empresa *Amazon* amb el servei "Amazon Elastic Compute Cloud (Amazon EC2)". Gràcies a aquest servei podríem disposar de forma ràpida i senzilla d'una gran quantitat de processadors nous on poder executar també l'AutoDock i això ens donaria més capacitat, més paral·lelisme i per tant més velocitat alhora d'executar un càlcul el qual estigues format per una base de dades de, proteïnes i lligands, de l'ordre de milions.

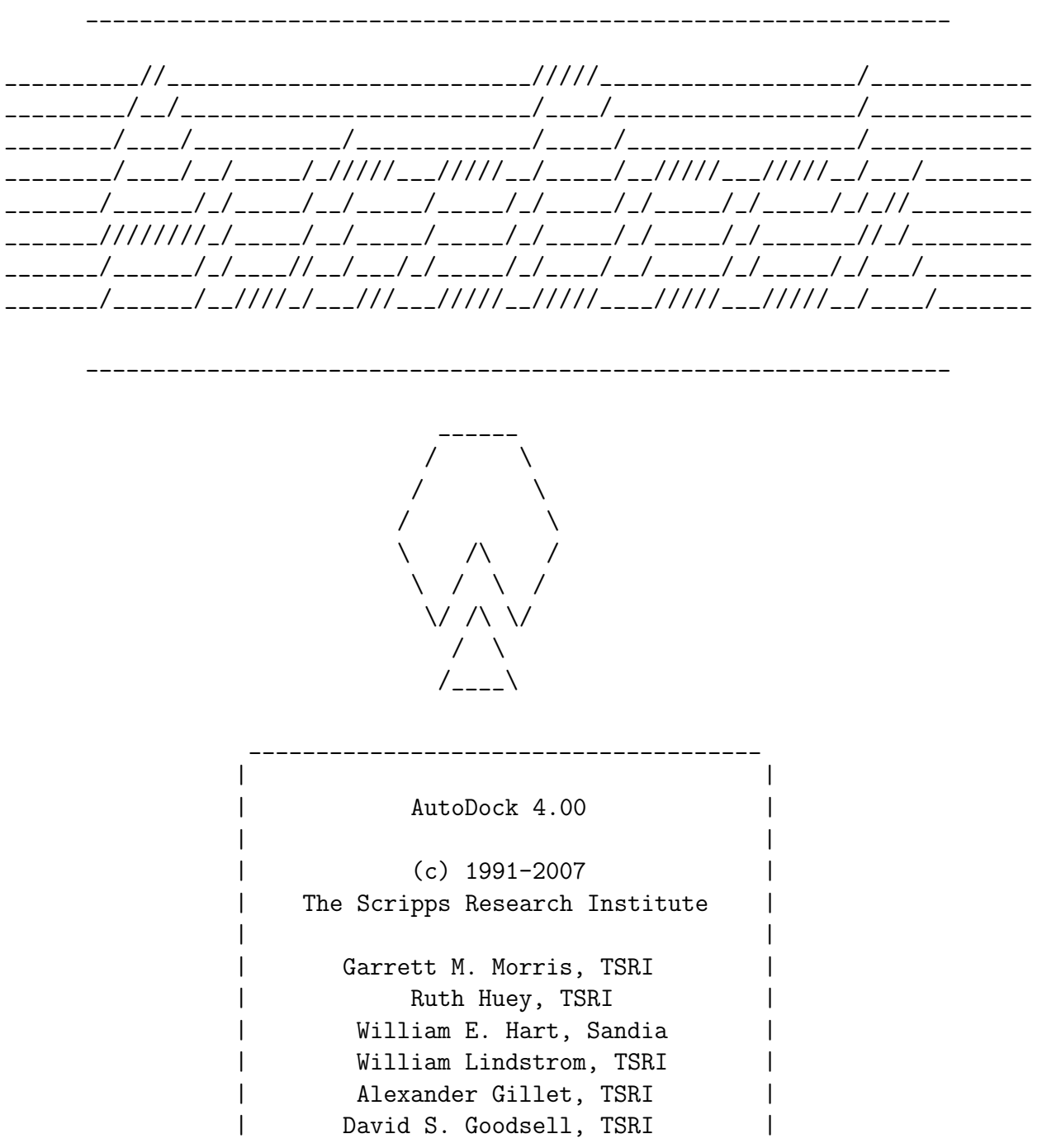
[1]Cloud, núvol en anglès, és una metàfora que representa Internet (basada en com està representada normalment aquesta en els diagrames de xarxes de computadors) i és una abstracció de la complexa infraestructura que té oculta.

Apèndix A

Fitxers AutoDock

A.1 Fitxer DLG

Fragment d'un fitxer DLG, és el fitxer que treu com a sortida una execució de l'AutoDock:



```
                |        Arthur J. Olson, TSRI         |
                |______________________________________|

                 ______________________________________
                |                                      |
                | Automated Docking of Flexible Ligand |
                |  to Flexible Macromolecular Receptor |
                |______________________________________|


        AutoDock comes with ABSOLUTELY NO WARRANTY; for details
        type 'warranty'.  This is free software, and you are welcome
        to redistribute it under certain conditions; type 'copyright'
        for details.


                              $Revision: 1.69 $

                          Compiled on Apr 17 2008 at 09:51:41


This file was created at:                       0:08 25" a.m., 05/28/2008
                  using:                        "hidra-01"

NOTE: "rus" stands for:

      r = Real, wall-clock or elapsed time;
      u = User or cpu-usage time;
      s = System time

All timings are in seconds, unless otherwise stated.


(...)


                Intermolecular Energy Analysis
                ==============================

Atom  vdW+Hb+Elec  vdW+Hbond  Electrosta  Partial          Coordinates
```

Type	Energy	Energy	tic Energy	Charge	x	y	z
1	-0.29	-0.28	-0.01	0.039	31.6570	8.5710	26.13
1	-0.33	-0.29	-0.05	0.157	31.3230	8.1960	27.40
6	-0.32	-0.40	0.08	-0.168	30.4190	9.0730	27.85
5	-0.12	-0.30	0.18	-0.276	30.1470	9.9480	26.98
4	-0.50	-0.29	-0.20	0.189	29.4570	10.7040	27.09
1	-0.35	-0.31	-0.04	0.076	30.8870	9.7460	25.89
1	-0.36	-0.35	-0.02	0.028	30.8960	10.5090	24.72
1	-0.33	-0.31	-0.02	0.043	32.0850	11.0070	24.22
1	-0.36	-0.34	-0.02	0.068	32.0980	11.7910	23.05
3	-0.49	-0.50	0.01	-0.081	33.5840	12.4180	22.44
1	-0.43	-0.40	-0.03	0.090	30.9090	12.0920	22.41
1	-0.53	-0.46	-0.07	0.075	29.7020	11.5950	22.92
1	-0.50	-0.42	-0.07	0.081	29.7050	10.8360	24.09
7	-0.36	-0.36	0.01	-0.359	30.9180	12.8280	21.27
4	0.15	0.16	-0.01	0.217	31.4480	13.6970	21.43
7	0.21	-0.33	0.54	-0.360	28.5470	10.3880	24.58
4	-0.82	-0.28	-0.54	0.217	27.9240	11.1850	24.77
1	-0.26	-0.27	0.01	-0.021	32.5790	7.9360	25.26
1	-0.32	-0.32	-0.01	0.015	32.1930	7.4710	23.98
1	-0.30	-0.28	-0.02	0.037	33.1400	6.8550	23.12
1	-0.25	-0.23	-0.03	0.068	34.4740	6.6820	23.56
1	-0.31	-0.30	-0.01	0.037	34.8200	7.1250	24.83
1	-0.32	-0.31	-0.00	0.015	33.8950	7.7530	25.67
7	0.05	-0.11	0.16	-0.356	35.4700	6.0780	22.79
2	-0.31	-0.17	-0.14	0.208	35.0300	5.5210	21.55
2	-0.27	-0.26	-0.01	0.272	31.8440	7.0080	28.22
7	-0.47	-0.29	-0.17	-0.266	32.3840	6.0480	27.70
5	-0.01	-0.18	0.16	-0.356	31.6730	7.1410	29.54
4	-0.35	-0.20	-0.15	0.162	31.2510	8.0170	29.88
2	-0.42	-0.35	-0.06	0.118	32.0460	6.1200	30.56
2	-0.24	-0.22	-0.01	0.026	33.4590	5.5720	30.35
Total	-9.51	-8.96	-0.55	-0.005			
	vdW+Hb+Elec Energy	vdW+Hbond Energy	Electrosta tic Energy	Partial Charge			

Total Intermolecular Interaction Energy = -10.311 kcal/mol
Total Intermolecular vdW + Hbond Energy = -8.956 kcal/mol

```
Total Intermolecular Electrostatic Energy = -0.550 kcal/mol


epdb: USER    Estimated Free Energy of Binding    =   -8.39 kcal/mol  [=(1)+(2)+(3)-
epdb: USER    Estimated Inhibition Constant, Ki   =  708.00 nM (nanomolar)  [Tempera
epdb: USER
epdb: USER    (1) Final Intermolecular Energy     =   -9.51 kcal/mol
epdb: USER        vdW + Hbond + desolv Energy     =   -8.96 kcal/mol
epdb: USER        Electrostatic Energy            =   -0.55 kcal/mol
epdb: USER    (2) Final Total Internal Energy     =   -0.80 kcal/mol
epdb: USER    (3) Torsional Free Energy           =   +1.92 kcal/mol
epdb: USER    (4) Unbound System's Energy         =   +0.00 kcal/mol
epdb: USER
epdb: USER

(...)


        BEGINNING LAMARCKIAN GENETIC ALGORITHM DOCKING

Run:    1 / 25
Date:   Wed May 28 00:08:38 2008
Output level is set to 1.


Creating an initial population of 150 individuals.

Assigning a random translation, a random orientation and 7 random torsions to each c

Beginning Lamarckian Genetic Algorithm (LGA), with a maximum of 10000000
energy evaluations.

Generation: 100   Oldest's energy: -3.071    Lowest energy: -6.830    Num.evals.: 94
Generation: 200   Oldest's energy: -7.124    Lowest energy: -7.124    Num.evals.: 19
Generation: 300   Oldest's energy: -7.237    Lowest energy: -7.237    Num.evals.: 28
Generation: 400   Oldest's energy: -7.272    Lowest energy: -7.272    Num.evals.: 39
Generation: 500   Oldest's energy: -7.281    Lowest energy: -7.281    Num.evals.: 48
Generation: 600   Oldest's energy: -7.338    Lowest energy: -7.338    Num.evals.: 59
Generation: 700   Oldest's energy: -7.342    Lowest energy: -7.342    Num.evals.: 69
Generation: 800   Oldest's energy: -7.346    Lowest energy: -7.346    Num.evals.: 79
Generation: 900   Oldest's energy: -7.347    Lowest energy: -7.347    Num.evals.: 89
```

```
Generation: 1000   Oldest's energy: -7.355    Lowest energy: -7.355    Num.ev
Generation: 1100   Oldest's energy: -7.355    Lowest energy: -7.355    Num.ev
Generation: 1200   Oldest's energy: -7.355    Lowest energy: -7.355    Num.ev
Generation: 1300   Oldest's energy: -7.360    Lowest energy: -7.360    Num.ev
Generation: 1400   Oldest's energy: -7.364    Lowest energy: -7.364    Num.ev
Generation: 1500   Oldest's energy: -7.364    Lowest energy: -7.364    Num.ev
Generation: 1600   Oldest's energy: -7.364    Lowest energy: -7.364    Num.ev
Generation: 1700   Oldest's energy: -7.377    Lowest energy: -7.377    Num.ev
Generation: 1800   Oldest's energy: -7.377    Lowest energy: -7.377    Num.ev
Generation: 1900   Oldest's energy: -7.384    Lowest energy: -7.384    Num.ev
Generation: 2000   Oldest's energy: -7.392    Lowest energy: -7.392    Num.ev
Generation: 2100   Oldest's energy: -7.392    Lowest energy: -7.392    Num.ev
Generation: 2200   Oldest's energy: -7.392    Lowest energy: -7.392    Num.ev
Generation: 2300   Oldest's energy: -7.392    Lowest energy: -7.392    Num.ev
Generation: 2400   Oldest's energy: -7.395    Lowest energy: -7.395    Num.ev
Generation: 2500   Oldest's energy: -7.395    Lowest energy: -7.395    Num.ev
Generation: 2600   Oldest's energy: -7.395    Lowest energy: -7.395    Num.ev
Generation: 2700   Oldest's energy: -10.064    Lowest energy: -10.064    Num.
Generation: 2800   Oldest's energy: -10.115    Lowest energy: -10.115    Num.
Generation: 2900   Oldest's energy: -10.384    Lowest energy: -10.384    Num.
Generation: 3000   Oldest's energy: -10.384    Lowest energy: -10.384    Num.
Generation: 3100   Oldest's energy: -10.440    Lowest energy: -10.440    Num.
Generation: 3200   Oldest's energy: -10.440    Lowest energy: -10.440    Num.
Generation: 3300   Oldest's energy: -10.462    Lowest energy: -10.462    Num.
Generation: 3400   Oldest's energy: -10.464    Lowest energy: -10.464    Num.
Generation: 3500   Oldest's energy: -10.464    Lowest energy: -10.464    Num.
Generation: 3600   Oldest's energy: -10.464    Lowest energy: -10.464    Num.
Generation: 3700   Oldest's energy: -10.464    Lowest energy: -10.464    Num.
Generation: 3800   Oldest's energy: 119151.305    Lowest energy: -10.464    N
Generation: 3900   Oldest's energy: -10.466    Lowest energy: -10.466    Num.
Generation: 4000   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4100   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4200   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4300   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4400   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4500   Oldest's energy: -10.467    Lowest energy: -10.467    Num.
Generation: 4600   Oldest's energy: -10.468    Lowest energy: -10.468    Num.
Generation: 4700   Oldest's energy: -10.468    Lowest energy: -10.468    Num.
Generation: 4800   Oldest's energy: -10.469    Lowest energy: -10.469    Num.
Generation: 4900   Oldest's energy: -10.469    Lowest energy: -10.469    Num.
Generation: 5000   Oldest's energy: -10.469    Lowest energy: -10.469    Num.
```

```
Generation: 5100   Oldest's energy: -10.469    Lowest energy: -10.469    Num.evals.
Generation: 5200   Oldest's energy: -10.469    Lowest energy: -10.469    Num.evals.
Generation: 5300   Oldest's energy: -10.476    Lowest energy: -10.476    Num.evals.
Generation: 5400   Oldest's energy: -10.476    Lowest energy: -10.476    Num.evals.
Generation: 5500   Oldest's energy: -10.476    Lowest energy: -10.476    Num.evals.
Generation: 5600   Oldest's energy: -10.480    Lowest energy: -10.480    Num.evals.
Generation: 5700   Oldest's energy: -10.480    Lowest energy: -10.480    Num.evals.
Generation: 5800   Oldest's energy: -10.480    Lowest energy: -10.480    Num.evals.
Generation: 5900   Oldest's energy: -10.484    Lowest energy: -10.484    Num.evals.
Generation: 6000   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6100   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6200   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6300   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6400   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6500   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6600   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6700   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6800   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 6900   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7000   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7100   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7200   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7300   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7400   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7500   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7600   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7700   Oldest's energy: -10.487    Lowest energy: -10.487    Num.evals.
Generation: 7800   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 7900   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8000   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8100   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8200   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8300   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8400   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8500   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8600   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8700   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8800   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 8900   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 9000   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
Generation: 9100   Oldest's energy: -10.488    Lowest energy: -10.488    Num.evals.
```

```
Generation: 9200   Oldest's energy: -10.488    Lowest energy: -10.488    Num.
Generation: 9300   Oldest's energy: -10.488    Lowest energy: -10.488    Num.
Generation: 9400   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 9500   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 9600   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 9700   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 9800   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 9900   Oldest's energy: -10.489    Lowest energy: -10.489    Num.
Generation: 10000   Oldest's energy: -10.490    Lowest energy: -10.490    Num
Generation: 10100   Oldest's energy: -10.490    Lowest energy: -10.490    Num
Final-Value: -10.490

Run completed;  time taken for this run:
Real= 2m 59.07s,  CPU= 2m 59.03s,  System= 0.02s

0:11 37" a.m., 05/28/2008
Total number of Energy Evaluations: 10001027
Total number of Generations:        10140


        FINAL LAMARCKIAN GENETIC ALGORITHM DOCKED STATE
        -----------------------------------------------


State:  31.667 9.004 25.225  -0.401 0.004 0.916 8.377   122.54 20.59 -145.47

DOCKED: MODEL        1
DOCKED: USER    Run = 1
DOCKED: USER    DPF = /mnt/hidra-01/joan/Projectes/GS_Daedalus/master/.gs_318
DOCKED: USER
DOCKED: USER    Estimated Free Energy of Binding    =   -8.57 kcal/mol  [=(1)
DOCKED: USER    Estimated Inhibition Constant, Ki   =  523.49 nM (nanomolar)
DOCKED: USER
DOCKED: USER    (1) Final Intermolecular Energy     =  -10.43 kcal/mol
DOCKED: USER        vdW + Hbond + desolv Energy     =   -9.73 kcal/mol
DOCKED: USER        Electrostatic Energy            =   -0.70 kcal/mol
DOCKED: USER    (2) Final Total Internal Energy     =   -0.94 kcal/mol
DOCKED: USER    (3) Torsional Free Energy           =   +1.92 kcal/mol
DOCKED: USER    (4) Unbound System's Energy         =   -0.88 kcal/mol
DOCKED: USER
DOCKED: USER
```

```
DOCKED: USER    NEWDPF move /mnt/hidra-01/oscar/astex_diverse_gast_adt_wat/2bsm/liga
DOCKED: USER    NEWDPF about 31.805300 9.029400 25.303101
DOCKED: USER    NEWDPF tran0 31.667159 9.003948 25.224594
DOCKED: USER    NEWDPF quaternion0 -0.029300 0.000288 0.066904 0.997329
DOCKED: USER    NEWDPF axisangle0 -0.401151 0.003947 0.916003 8.377173
DOCKED: USER    NEWDPF quat0 -0.401151 0.003947 0.916003 8.377173 # deprecated
DOCKED: USER    NEWDPF dihe0 122.54 20.59 -145.47 131.00 -8.46 148.74 -10.94
DOCKED: USER
DOCKED: REMARK  7 active torsions:
DOCKED: REMARK  status: ('A' for Active; 'I' for Inactive)
DOCKED: REMARK    1  A    between atoms: O1_2  and  C2_3
DOCKED: REMARK    2  A    between atoms: C7_8  and  C8_9
DOCKED: REMARK    3  A    between atoms: C9_10  and  C10_11
DOCKED: REMARK       I    between atoms: C10_11  and  N1_13
DOCKED: REMARK    4  A    between atoms: N1_13  and  C11_15
DOCKED: REMARK    5  A    between atoms: C13_20  and  C14_21
DOCKED: REMARK    6  A    between atoms: C17_25  and  O3_26
DOCKED: REMARK    7  A    between atoms: C19_29  and  O4_30
DOCKED: USER                              x       y       z     vdW  Elec       q
DOCKED: USER                           _______ _______ _______ _____ _____    ______
DOCKED: ROOT
DOCKED: ATOM      1  C8  <1> d          31.455   8.522  26.025 -0.29 -0.01    +0.039
DOCKED: ATOM      2  C9  <1> d          31.069   8.121  27.280 -0.29 -0.04    +0.157
DOCKED: ATOM      3  N2  <1> d          30.288   9.083  27.776 -0.40 +0.08    -0.168
DOCKED: ATOM      4  N3  <1> d          30.139  10.036  26.959 -0.30 +0.19    -0.276
DOCKED: ATOM      5  H14 <1> d          29.555  10.869  27.120 -0.31 -0.21    +0.189
DOCKED: ATOM      6  C13 <1> d          30.850   9.801  25.857 -0.31 -0.04    +0.076
DOCKED: ENDROOT
DOCKED: BRANCH   6   7
DOCKED: ATOM      7  C14 <1> d          30.966  10.623  24.736 -0.34 -0.02    +0.028
DOCKED: ATOM      8  C15 <1> d          32.189  11.170  24.394 -0.30 -0.02    +0.043
DOCKED: ATOM      9  C16 <1> d          32.308  12.012  23.277 -0.35 -0.02    +0.068
DOCKED: ATOM     10  I1  <1> d          33.836  12.700  22.856 -0.59 +0.01    -0.081
DOCKED: ATOM     11  C17 <1> d          31.189  12.322  22.519 -0.39 -0.03    +0.090
DOCKED: ATOM     12  C18 <1> d          29.947  11.776  22.870 -0.45 -0.06    +0.075
DOCKED: ATOM     13  C19 <1> d          29.843  10.958  23.994 -0.41 -0.07    +0.081
DOCKED: BRANCH  11  14
DOCKED: ATOM     14  O3  <1> d          31.302  13.116  21.426 -0.50 +0.02    -0.359
DOCKED: ATOM     15  H16 <1> d          31.802  12.607  20.683 +0.14 +0.00    +0.217
DOCKED: ENDBRANCH  11  14
DOCKED: BRANCH  13  16
```

```
DOCKED: ATOM     16  O4  <1> d          28.650  10.462  24.337 -0.35 +0.53
DOCKED: ATOM     17  H18 <1> d          27.921  11.169  24.165 -0.27 -0.63
DOCKED: ENDBRANCH  13  16
DOCKED: ENDBRANCH   6   7
DOCKED: BRANCH   1  18
DOCKED: ATOM     18  C7  <1> d          32.288   7.821  25.119 -0.27 +0.01
DOCKED: ATOM     19  C4  <1> d          33.691   7.972  25.139 -0.30 -0.01
DOCKED: ATOM     20  C3  <1> d          34.512   7.242  24.235 -0.28 -0.01
DOCKED: ATOM     21  C2  <1> d          33.910   6.380  23.288 -0.22 -0.03
DOCKED: ATOM     22  C5  <1> d          32.526   6.272  23.273 -0.33 -0.02
DOCKED: ATOM     23  C6  <1> d          31.720   6.970  24.178 -0.36 -0.00
DOCKED: BRANCH  21  24
DOCKED: ATOM     24  O1  <1> d          34.623   5.637  22.351 -0.09 +0.21
DOCKED: ATOM     25  C1  <1> d          33.926   5.367  21.133 -0.14 -0.24
DOCKED: ENDBRANCH  21  24
DOCKED: ENDBRANCH   1  18
DOCKED: BRANCH   2  26
DOCKED: ATOM     26  C10 <1> d          31.423   6.828  28.026 -0.27 +0.00
DOCKED: ATOM     27  O2  <1> d          31.686   5.793  27.435 -0.66 -0.24
DOCKED: ATOM     28  N1  <1> d          31.438   6.956  29.358 -0.14 +0.13
DOCKED: ATOM     29  H8  <1> d          31.250   7.890  29.748 -0.19 -0.13
DOCKED: BRANCH  28  30
DOCKED: ATOM     30  C11 <1> d          31.706   5.851  30.320 -0.31 -0.05
DOCKED: ATOM     31  C12 <1> d          31.685   6.323  31.775 -0.45 -0.02
DOCKED: ENDBRANCH  28  30
DOCKED: ENDBRANCH   2  26
DOCKED: TORSDOF 7
DOCKED: TER
DOCKED: ENDMDL
-----------------------------------------------------------------------------


(...)


        CLUSTERING HISTOGRAM
        --------------------

-----------------------------------------------------------------------------
     |           |     |           |     |
Clus | Lowest    | Run | Mean      | Num | Histogram
```

```
-ter | Binding   |     | Binding   | in  |
Rank | Energy    |     | Energy    | Clus|    5    10   15   20   25   30   35
_____|___________|_____|___________|_____|____:____|____:____|____:____|____:___
   1 |     -8.68 |  14 |     -8.58 |  24 |########################
   2 |     -6.34 |  15 |     -6.34 |   1 |#
_____|___________|_____|___________|_____|_____________________________________


Number of multi-member conformational clusters found = 1, out of 25 runs.

        RMSD TABLE
        __________

_____________________________________________________________________
     |      |      |           |         |                  |
Rank | Sub- | Run  | Binding   | Cluster | Reference        | Grep
     | Rank |      | Energy    | RMSD    | RMSD             | Pattern
_____|______|______|___________|_________|__________________|___________
   1      1     14       -8.68      0.00      0.66           RANKING
   1      2     16       -8.68      0.15      0.68           RANKING
   1      3     11       -8.67      0.16      0.68           RANKING
   1      4     12       -8.66      0.22      0.66           RANKING
   1      5      7       -8.66      0.13      0.68           RANKING
   1      6     22       -8.66      0.10      0.66           RANKING
   1      7      9       -8.66      0.11      0.67           RANKING
   1      8     21       -8.65      0.17      0.64           RANKING
   1      9     24       -8.58      0.78      0.50           RANKING
   1     10      5       -8.57      0.77      0.50           RANKING
   1     11      8       -8.57      0.70      0.50           RANKING
   1     12     17       -8.57      0.27      0.60           RANKING
   1     13      1       -8.57      0.47      0.62           RANKING
   1     14     13       -8.57      0.56      0.61           RANKING
   1     15     10       -8.57      0.24      0.62           RANKING
   1     16      6       -8.56      0.57      0.46           RANKING
   1     17     19       -8.55      0.76      0.52           RANKING
   1     18      2       -8.54      0.78      0.51           RANKING
   1     19     23       -8.54      0.53      0.60           RANKING
   1     20     25       -8.54      0.81      0.53           RANKING
   1     21      3       -8.54      0.56      0.45           RANKING
   1     22     18       -8.48      0.82      0.68           RANKING
   1     23     20       -8.48      0.85      0.67           RANKING
```

```
   1      24       4        -8.47       0.66       0.58           RANKING
   2       1      15        -6.34       0.00       5.76           RANKING
-----------------------------------------------------------------------
```

A.2 Fitxer DPF

Fitxer DPF que es el que utilitza l'AutoDock com a parmetre d'entrada:

```
outlev 1                                # diagnostic output level
intelec                                 # calculate internal electrostatics
seed pid time                           # seeds for random generator
ligand_types A C HD N NA OA             # atoms types in ligand
fld /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.maps.fld    #grid_
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.A.map        #atom
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.C.map        #atom
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.HD.map       #atom
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.N.map        #atom
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.NA.map       #atom
map /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.OA.map       #atom
elecmap /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.e.map    #elec
desolvmap /mnt/hidra-01/joan/experiments/prova_dock/2brm/protein.d.map    #de
move /mnt/hidra-01/joan/experiments/prova_dock/2brm/ligand.pdbqt    #smallmol
about 3.458 -4.8706 18.7863             # small molecule center
tran0 random                            # initial coordinates/A or random
quat0 random                            # initial quaternion
ndihe 5                                 # number of active torsions
dihe0 random                            # initial dihedrals (relative) or random
tstep 2.0                               # translation step/A
qstep 50.0                              # quaternion step/deg
dstep 50.0                              # torsion step/deg
torsdof 5 0.274000                      # torsional degrees of freedom and coeff
rmstol 2.0                              # cluster_tolerance/A
extnrg 1000.0                           # external grid energy
e0max 0.0 10000                         # max initial energy; max number of retr
ga_pop_size 150                         # number of individuals in population
ga_num_evals 10000000                   # maximum number of energy evaluations
ga_num_generations 27000                # maximum number of generations
ga_elitism 1                            # number of top individuals to survive t
ga_mutation_rate 0.02                   # rate of gene mutation
```

```
ga_crossover_rate 0.8                # rate of crossover
ga_window_size 10                    #
ga_cauchy_alpha 0.0                  # Alpha parameter of Cauchy distribution
ga_cauchy_beta 1.0                   # Beta parameter Cauchy distribution
set_ga                               # set the above parameters for GA or LGA
sw_max_its 300                       # iterations of Solis & Wets local search
sw_max_succ 4                        # consecutive successes before changing rho
sw_max_fail 4                        # consecutive failures before changing rho
sw_rho 1.0                           # size of local search space to sample
sw_lb_rho 0.01                       # lower bound on rho
ls_search_freq 0.06                  # probability of performing local search on inc
set_sw1                              # set the above Solis & Wets parameters
epdb /mnt/hidra-01/joan/experiments/prova_dock/2brm/ligand_reference.pdbqt    #small
compute_unbound_extended             # compute extended ligand energy
ga_run 25                            # do this many hybrid GA-LS runs
analysis                             # perform a ranked cluster analysis
```

Apèndix B

Codis font

B.1 AutoDock vectoritzat

Mostrem a continuació el mòdul eintcal d'AutoDock vectoritzat amb instruccions SIMD (Single Instruccion Multiple Data).

```
/*****************************************************************/
/* VERSION VECTORITZADA AMB ARQUITECTURA I COMPILADOR INTEL v1 SSE2 */
/*****************************************************************/

int const neint_1=NEINT - 1;
int index;
float const rmin_elec2=RMIN_ELEC * RMIN_ELEC;
int Nnb_c;
int index_inb,index_inb2,i,j;
__m128 vec_dx;
__m128 vec_dy;
__m128 vec_dz;
__m128 vec_dx2;
__m128 vec_dy2;
__m128 vec_dz2;
__m128 vec_index;
__m128 vec_r2;
__m128 vec_zero={0,0,0,0};
__m128 vec_SQA_DIV={SQA_DIV,SQA_DIV,SQA_DIV,SQA_DIV};
__m128 vec_rmin_elec2={rmin_elec2,rmin_elec2,rmin_elec2,rmin_elec2};
__m128 vec_neint_1={neint_1,neint_1,neint_1,neint_1};
float dx[4],dy[4],dz[4],r2_tmp[4],index_tmp[4];
double q1q2,nb_desolv;

// Loop over the non-bonds in this nonbond "group", "inb",
for (inb = inb_from;  inb < Nnb_c;  inb++) {
        index_inb=inb*4;
        index_inb2=index_inb+4;

        double e_internal=0;  // e_internal = epair
        double e_desolv=0;    // e_desolv = dpair

        for (i=index_inb,j=0;i<index_inb2;i++,j++) {
                a1 =  nonbondlist[i].a1;
                a2 =  nonbondlist[i].a2;
                t1 = nonbondlist[i].t1; // Xcode-gmm  // t1 is a map_index
```

```
                t2 = nonbondlist[i].t2; // Xcode-gmm  // t2 is a map_index
                nonbond_type = nonbondlist[i].nonbond_type;
                nb_desolv = nonbondlist[i].desolv;
                q1q2 = nonbondlist[i].q1q2;
                dx[j] = tcoord[a1][X] - tcoord[a2][X]; // LC
                dy[j] = tcoord[a1][Y] - tcoord[a2][Y];
                dz[j] = tcoord[a1][Z] - tcoord[a2][Z];
        }

        vec_dx=_mm_load_ps(dx);
        vec_dy=_mm_load_ps(dy);
        vec_dz=_mm_load_ps(dz);

        vec_dx2=_mm_mul_ps(vec_dx,vec_dx);
        vec_dy2=_mm_add_ps(_mm_mul_ps(vec_dy,vec_dy),vec_dx2);
        vec_dz2=_mm_add_ps(_mm_mul_ps(vec_dz,vec_dz),vec_dy2);

        vec_r2=_mm_max_ps(vec_dz2,vec_rmin_elec2);
        vec_index=_mm_add_ps(vec_r2,vec_SQA_DIV);
        vec_index=_mm_min_ps(vec_index,vec_neint_1);
        _mm_store_ps(index_tmp,vec_index);
        _mm_store_ps(r2_tmp,vec_r2);

(...)

        for (i=index_inb,j=0;i<index_inb2;i++,j++) {
                index=(int)(index_tmp[j]);

                index_lt_NEINT = BoundedNeint(index);
                index_lt_NDIEL = BoundedNdiel(index);

                if (B_calcIntElec) {
                        //  Calculate  Electrostatic  Energy
                        double r_dielectric = ptr_ad_energy_tables->
     r_epsilon_fn[index_lt_NDIEL];
                        e_elec = q1q2 * r_dielectric;
                        e_internal = e_elec;
                }
                if  ( r2 < nbc2 ) {
                        e_desolv = ptr_ad_energy_tables->sol_fn[index_lt_NEINT]
   * nb_desolv;
```

```
                        if (B_include_1_4_interactions != 0 &&
   nonbond_type == 4) {
                                //| Compute a scaled 1-4 interaction,
                                e_internal += scale_1_4 *
          (
    ptr_ad_energy_tables->
    e_vdW_Hb[index_lt_NEINT][t2][t1] +
    e_desolv
   );
                        } else {
                                e_internal += ptr_ad_energy_tables->
      e_vdW_Hb[index_lt_NEINT][t2][t1] +
      e_desolv;
                        }
                }

                total_e_internal += e_internal;
#ifdef EINTCALPRINT // eintcalPrint [

        total_e_desolv   += e_desolv;
        total_e_elec     += e_elec;
        double dielectric = ptr_ad_energy_tables->epsilon_fn[index_lt_NDIEL];

        if (B_calcIntElec) {

                e_vdW_Hb = e_internal - e_desolv - e_elec,
                pr( logFile, " %6d   %5d-%-5d  %7.2lf  %+8.3lf  %+8.3lf
%+8.3lf  %+8.3lf   %+8.3lf   %d  %8.3lfn",(int)(inb+1),
(int)(a1+1), (int)(a2+1), (double)sqrt(r2),(double)e_internal,
(double)e_elec, (double)e_vdW_Hb,(double)e_desolv,
(double)ptr_ad_energy_tables->sol_fn[index_lt_NEINT],
(int)nonbond_type, (double)dielectric
                );
        } else {

                e_vdW_Hb = e_internal - e_desolv,
                pr( logFile, " %6d   %5d-%-5d  %7.2lf  %+8.3lf  %+8.3lf
%+8.3lf   %+8.3lf   %d  %8.3lfn",(int)(inb+1),
(int)(a1+1), (int)(a2+1), (double)sqrt(r2),
(double)e_internal, (double)e_vdW_Hb, (double)e_desolv,
                (double)ptr_ad_energy_tables->sol_fn[index_lt_NEINT],
```

```
(int)nonbond_type, (double)dielectric
                );
        }

        total_e_vdW_Hb += e_vdW_Hb;
}

float dx_f,dy_f,dz_f;
// For per completar els enllaços de la resta de la divisió entera
for (inb=inb*4; inb < Nnb; inb++) {

        dx_f = tcoord[a1][X] - tcoord[a2][X];
        dy_f = tcoord[a1][Y] - tcoord[a2][Y];
        dz_f = tcoord[a1][Z] - tcoord[a2][Z];

        r2 = (dx_f*dx_f)+(dy_f*dy_f)+(dz_f*dz_f);
        r2 = ( (r2 < rmin_elec2) ? rmin_elec2 : r2 );
        index = ( (int) ( r2 * SQA_DIV ) );

        if (B_calcIntElec) {
                double r_dielectric = ptr_ad_energy_tables->
      r_epsilon_fn[index_lt_NDIEL];
                e_elec = q1q2 * r_dielectric;
                e_internal = e_elec;
        }
        if  ( r2 < nbc2 ) {
                e_desolv = ptr_ad_energy_tables->sol_fn[index_lt_NEINT]
   * nb_desolv;
                if (B_include_1_4_interactions != 0 && nonbond_type == 4) {
                        //| Compute a scaled 1-4 interaction,
                         e_internal += scale_1_4 *
       (
ptr_ad_energy_tables->
e_vdW_Hb[index_lt_NEINT][t2][t1] + e_desolv
       );
                } else {
                        e_internal += ptr_ad_energy_tables->
      e_vdW_Hb[index_lt_NEINT][t2][t1] + e_desolv;
                }
        }
```

```
        total_e_internal += e_internal;

#ifdef EINTCALPRINT // eintcalPrint [
       total_e_desolv   += e_desolv;
       total_e_elec     += e_elec;
       double dielectric = ptr_ad_energy_tables->epsilon_fn[index_lt_NDIEL];

       if (B_calcIntElec) {

               e_vdW_Hb = e_internal - e_desolv - e_elec,
               pr( logFile, " %6d   %5d-%-5d  %7.2lf  %+8.3lf  %+8.3lf
%+8.3lf  %+8.3lf   %+8.3lf   %d  %8.3lfn",(int)(inb+1),
(int)(a1+1), (int)(a2+1), (double)sqrt(r2),(double)e_internal,
(double)e_elec, (double)e_vdW_Hb, (double)e_desolv,
       (double)ptr_ad_energy_tables->sol_fn[index_lt_NEINT],
(int)nonbond_type, (double)dielectric
               );
       } else {

               e_vdW_Hb = e_internal - e_desolv,
  pr( logFile, " %6d   %5d-%-5d  %7.2lf  %+8.3lf  %+8.3lf
%+8.3lf   %+8.3lf   %d  %8.3lfn",(int)(inb+1), (int)(a1+1),
(int)(a2+1), (double)sqrt(r2), (double)e_internal, (double)e_vdW_Hb,
(double)e_desolv, (double)ptr_ad_energy_tables->sol_fn[index_lt_NEINT],
(int)nonbond_type, (double)dielectric
               );
       }

       total_e_vdW_Hb += e_vdW_Hb;
#endif // eintcalPrint ]

}
```

Bibliografia

[1] Cramer C. J. *Essentials of Computational Chemistry.* John Wiley & Sons, 2002.

[2] G. M. Verkhivker, D. Bouzida, D. K Gehlhaar, P. A. Rejto, S. Arthurs, A. B. Colson, S. T. Freer, V. Larson, B. A. Luty, T. Marrone, and P. W. Rose. Deciphering common failures in molecular docking of ligand-protein complexes. *Journal of Computer-Aided Molecular Design*, 14(8):731–751, 2000.

[3] R. Wang, Y. Lu, and S. Wang. Comparative evaluation of 11 scoring functions for molecular docking. *Journal of Medicinal Chemistry*, 46(12):2287–2303, 2003.

[4] T. J. Ewing and I. D. Kuntz. Critical evaluation of search algorithms for automated molecular docking. *Journal of Computational Chemistry*, 18(9):1175–1189, 1997.

[5] M. K. Holloway, G. B. McGaughey, C. A. Coburn, S. J. Stachel, K. G. Jones, E. L. Stanton, A. R. Gregro, M.-T. Lai, M.-C. Crouthamel, B. L. Pietrak, and S. K. Munshi. Evalutaing scoring functions for docking and designing β-secretase inhibitors. *Bioorganic & Medicinal Chemistry Letters*, 17(3):823–827, 2007.

[6] P. S. Charifson, J. J. Corkery, M. A. Murcko, and W. P. Walters. Consensus scoring: A method for obtaining improved hit rates from docking databases of three-dimensional structures into proteins. *Journal of Medicinal Chemistry*, 42(25):5100–5109, 1999.

[7] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of docking: an overview of search algorithms and a guide to scoring functions. *Proteins: Structure, Function and Genetics*, 47(4):409–403, 2002.

[8] P. Kollman. Free energy calculations: Applications to chemical and biochemical phenomena. *Chemical Reviews*, 93(7):2395–2417, 1993.

[9] T. J. Ewing, S. Makino, A. G. Skillman, and I. D. Kuntz. DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design*, 15(5):3219–3222, 2001.

[10] M. Rarey, B. Kramer, T. Lengauer, and G. Klebe. A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology*, 261(3):470–489, 1996.

[11] G. Jones, P. Willet, R. C. Glen, A. R. Leach, and R. Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology*, 267(3):727–748, 1997.

[12] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. Olson. Automated docking using a lamarckian genetic algorithm and and empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[13] R. A. Abagyan, M. M. Totrov, and D. A. Kuznetsov. ICM: a new method for structure modeling and design: applications to docking and structure prediction from the distorted native conformation. *Journal of Computational Chemistry*, 15(5):488–506, 1994.

[14] P. Ferrara, H. Gohlke, D. J. Price, G. Klebe, and C. L. Brooks III. Assessing scoring functions for protein-ligand interactions. *Journal of Medicinal Chemistry*, 47(12):3032–3047, 2004.

[15] M. Taufer, M. Crowley, D. J. Price, A. A. Chien, and C. L. Brooks III. Study of a highly accurate and fast protein-ligand docking method based on molecular dynamics. *Concurrency and Computation: Practice and Experience*, 17(14):1627–1641, 2005.

[16] J. C. Cole, C. W. Murray, J. W. Nissink, R. D. Taylor, and R. Taylor. Comparing protein-ligand docking programs is difficult. *Proteins: Structure, Function, and Bioinformatics*, 60(3):325–332, 2005.

[17] CAPRI: Critical Assessment of PRediction of Interactions, 2007. EMBL-EBI, http://capri.ebi.ac.uk/Charleston.html.

[18] R. W. DeSimone, K. S. Currie, S. A. Mitchell, J. W. Darrow, and D. A. Pippin. Privileged structures: applications in drug discovery. *Combinatorial Chemistry and High Throughput Screening*, 7(5):473–494, 2004.

[19] C. Selvam, S. M. Jachak, R. Thilagavathi, and A. K. Chakraborti. Design, synthesis, biological evaluation and molecular docking of curcumin analogues as antioxidant, cyclooxygenase inhibitory and anti-inflammatory agents. *Bioorganic Medicinal Chemistry Letters*, 15(7):1793–1797, 2005.

[20] C. Gambacorti-Passerini, M. Gasser, S. Ahmed, S. Assouline, and L. Scapozza. Abl inhibitor BMS354825 binding mode in Abelson kinase revealed by molecular docking studies. *Leukemia*, 19(7):1267–1269, 2005.

[21] M. J. Slusarz, E. Sikorska, R. Slusarz, and J. Ciarkowski. Molecular docking-based study of vasopressin analogues modified at positions 2 and 3 with N-methylphenylalanine: influence on receptor-bound conformations and interactions with vasopressin and oxytocin receptors. *Journal of Medicinal Chemistry*, 49(8):2463–2469, 2006.

[22] A. Lavecchia, S. Cosconati, V. Limongelli, and E. Novellin. Modeling of Cdc25B dual specifity protein phosphatase inhibitors: Docking of ligands and enzymatic inhibition mechanism. *ChemMedChem*, 1(5):540–550, 2006.

[23] D.-Q. Wei, R. Zhang, Q.-S. Du, W.-N. Gao, Y. Li, H. Gao, S.-Q. Wang, X. Zhang, A.-X. Li, S. Sirois, and K.-C. Chou. Anti-sars drug screening by molecular docking. *Amino Acids*, 31(1):73–80, 2006.

[24] B. Jojart and A. Marki. Possible dynamic anchor points in a benzoxazinone derivative-human oxytocin receptor system–a molecular docking and dynamics calculation. *Journal of Molecular Modeling*, 13(1):1–10, 2006.

[25] A. Lauria A, M. Ippolito, and A. M. Almerico. Molecular docking approach on the topoisomerase i inhibitors series included in the nci anti-cancer agents mechanism database. *Journal of Molecular Modeling*, 13(3):393–400, 2006.

[26] S. F. Sousa, P. A. Fernandes, and M. J. Ramos. Protein-ligand docking: current status and future challenges. *Proteins: Structure, Function, and Bioinformatics*, 65(1):15–26, 2006.

[27] D. Goldbert J.L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2002.

[28] P. Bellens et al. R. Sirvent, R. M. Badia. Demostración de uso de grid superscalar. *Boletín de RedIRIS*, 80:1–6, 2007.

[29] Quinn Michael J. *Parallel Programming in C with MPI and OpenMP*. 2004.

[30] PhD Jon Siegel. *Corba 3.0*, chapter OMG IDL Syntax and Semantics, pages 3.1 – 3.74. John Wiley and Sons, 2002.

[31] C. Hetenyi, G. Paragi, U. Maran, Z. Timar, M. Karelson, and B. Penke. Combination of a modified scoring function with two-dimensional descriptors for calculation of binding affinities of bulky, flexible ligands to proteins. *Journal of the American Chemical Society*, 128(4):1233–1239, 2006.

[32] G. Klebe and H. J. Bohm. Energetic and entropic factors determining binding affinity in protein-ligand complexes. *Journal of Receptor & Signal Transduction Research*, 17(1-3):459–473, 1997.

[33] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Publishing Company, 1995.

[34] Paul C. D. Hawkins, Greogory L. Warren, A. Geoffrey, and Anthony Nicholls. How to do an evaluation: pitfalls and traps. 2007.

[35] JC Cole, CW Murray, JW Nissink, RD Taylor, and R Taylor. Comparing protein-ligand docking programs is difficult. *Proteins*, 60(3):325–32, Aug 2005.