Solució d'alt rendiment per a química computacional

Albert Herrero Rosello
Màster en Enginyeria Computacional i Matemàtica
Computació d'altes prestacions

Ivan Rodero Castro
Josep Jorba Esteve

14 de juny de 2017

Fitxa del treball final

Títol:	Solució d'alt rendiment per a química computacional
Nom de l'autor:	Albert Herrero Rosello
Nom del consultor/a:	Ivan Rodero Castro
Nom del PRA:	Josep Jorba Esteve
Data de lliurament(mm/aaaa):	06/2017
Titulació o programa:	Màster en Enginyeria Computacional i Matemàtica
Idioma del treball:	Català
Paraules clau	cuda, openmp, simd

Resum

El propòsit d'aquesta document és explicar el procediment seguit en l'acceleració de PharmScreen, un software científic desenvolupat per l'empresa Pharmacelera. PharmScreen és un eina de virtual screening dedicada al descobriment de nous compostos per a la realització de nous medicaments. Gràcies al seu algoritme d'alta precisió, basat en camps d'interacció, el software pot trobar possibles compostos actius dotats d'una alta diversitat química.

En el treball s'expliquen les tècniques, tecnologies (Cuda, OpenMP, SEE/AVX) i eines utilitzades en el procés d'acceleració i es presenta un amplia gama dels resultats que s'obtenen quan s'executen les solucions en diferents plataformes.

Abstract

The purpose of this report is to explain the PharmScreeen acceleration procedure, a scientific software developed by the company Pharmacelera. PharmScreen is a virtual screening tool involved in early stage drug discovery. Thanks to its highly accurate algorithm, based on interaction fields, the software can find potential active compounds endowed with high chemical diversity.

The document explains techniques, technologies (Cuda, OpenMP, SEE/AVX) and tools used in the acceleration process and presents a broad range of results obtained when running the solutions on different platforms.

Agraïments

Vull dedicar aquest document a totes les persones que m'han ajudat i m'han donat suport durant el desenvolupament del treball. En especial als meus familiars que m'ha donat ànims en tot moment per continuar endavant i acabar la feina. Als amics que han aguantat explicacions tècniques sobre el treball que estava fent sens entendre molt be el que els deia. Als amics que m'han donat forces per tirar endavant en els moments difícils. A Enric Herrero i Enric Gibert que han supervisat la feina i m'ha donat diferents punts de vista per encarar els problemes. A Javier Vázquez i Alessandro Deplano que m'han ajudat a entendre els conceptes teòrics en els que es fonamenta tot el treball. A Ivan Rodero que ha tutelat el treball i m'ha donat accés a recursos per poder desenvolupar-ho. A totes ells perquè, sense el seu recolzament, hagués sigut una tasca molt difícil de realitzar. Per tot això, gràcies companys.

Índex

Índex de figures

Índex de taules

Capítol 1

Descripció del projecte

1.1 Introducció

La química computacional o química digital és una branca de la química que fa servir ordinadors per ajudar a resoldre problemes químics. Es fan servir els resultats de la química teòrica, incorporant-los en algun software per fer els càlculs de les estructures i les propietats de molècules i cosos sòlids. La química computacional és àmpliament utilitzada en el disseny de nous medicaments i materials.

En els camps de la medicina, la biotecnologia i la farmacologia, el descobriment de fàrmacs és el procés mitjançant el qual es descobreixen nous medicaments. Històricament, els medicaments han sigut descoberts a través de la identificació de l'ingredient clau en remeis tradicionals o per un descobriment fortuït. Posteriorment però, apareix el que es coneix com a farmacologia clàssica, que consisteix en seleccionar biblioteques químiques de molècules petites i productes o extractes naturals per tal d'identificar substàncies que tinguin un efecte terapèutic desitjable.

Des de la seqüenciació del genoma humà que va permetre la clonació ràpida i la síntesi de grans quantitats de proteïnes purificades, s'ha convertit en una pràctica comuna l'ús de cribratge[i] d'alt rendiment de grans biblioteques de compostos contra objectius biològics aïllats.

El descobriment de nous medicaments implica la identificació de *hits*[ii], de química mèdica i de l'optimització d'aquests *hits* per augmentar l'afinitat, la selectivitat (reduir efectes secundaris), l'eficàcia/potència, l'estabilitat metabòlica i la biodisponibilitat oral. Un cop identificat un compost que compleixi tots aquests requeriments es comença amb el procés de desenvolupament del fàrmac per a, posteriorment, passar als assajos clínics.

El descobriment de nous medicaments doncs, és un procés intensiu que implica grans inversions per part de les empreses de la industria farmacèutica, així com dels governs nacionals. No obstant això, malgrat els avenços de la tecnologia i del coneixement dels sistemes biològics, el descobriment de fàrmacs continua sent un procés costós, difícil i llarg amb un taxa de descobriments terapèutics molt baixa.

[i] Recerca sistemàtica indiscriminada que hom aplica a un conjunt d'elements per tal de descobrir-hi els que es troben afectats d'alguna particularitat.

[ii] Quan es parla de hits es fa referència a les molècules que compleixen característiques que s'estan buscant.

En aquest sentit, la idea principal que hi ha darrera d'aquest treball és la d'accelerar i optimitzar un software científic dedicat a la identificació de compostos candidats per al desenvolupament de nous medicaments. S'ha realitzat un estudi sobre el rendiment i eficàcia del *software* quan es desplega en diverses plataformes amb hardware especific d'última generació dissenyat per a l'execució de càlculs científics. El software en qüestió és un eina de *virtual screening*[iii] anomenada *PharmScreen*[iv], producte propietat de l'empresa *Pharmacelera*.

1.2 Actors

El treball realitzat està emmarcat dins l'àmbit industrial i concretament en l'àrea del *I+D*. És tracta d'un treball d'investigació continu que involucra principalment dos actors: *Pharmacelera* i el grup B*iologia Computacional i Disseny de Fàrmacs*.

1.2.1 Grup Biologia Computacional i Disseny de Fàrmacs

El grup *Biologia Computacional i Disseny de Fàrmacs (CBDD)*, és un grup de recerca del departament de *Fisicoquímica* de la Universitat de Barcelona (UB). La recerca que es desenvolupa està orientada cap a l'estudi de les relacions estructurals, dinàmiques i reactives de biomolècules amb la seva activitat biològica. En aquest marc, el seu treball compren una gama variada de sistemes, que varia des de molècules petites (lligands, fàrmacs, etc) fins a biomacromolècules (proteïnes, àcids nuclèics) o bé la interacció entre lligand i biomacromolècula en el marc del disseny de fàrmacs.

- **Molècules petites**. En aquest punt l'interès principal es dirigeix cap a la caracterització i predicció de diverses propietats, incloent estats conformacionals, tautomèrics i d'ionització, així com l'estudi de forces intermoleculars que determinen el proces de reactivitat i reconeixement molecular.

- **Biomacromolècules**. L'objectiu és esbrinar la relació entre l'estructura de la proteïna, la seva flexibilitat dinàmica i la seva funció biològica. Actualment s'està estudiant l'hemoglobina *truncada N* de *M. tuberculosis* i la seva implicació com a sistema de defensa del microorganisme enfront de l'estrès nitrosatiu. Tanmateix s'està considerant formes anòmales de dúplexes de DNA formades per bases no naturals, com seleno derivats de les bases naturals o benzo- i nafto-derivats de les bases del DNA.

- **Disseny de fàrmacs**. L'objectiu comprèn el desenvolupament de noves eines computacionals en disseny de fàrmacs i el disseny basat en estructures en sistemes d'interès terapèutic. Entre els primers, cal esmentar el desenvolupament de mètodes per determinar la conformació bioactiva, la predicció de centres d'unió vàlides en disseny de fàrmacs o la millora de programes de docking[v]. Entre els segons, el treball contempla el desenvolupament d'inhibidors en sistemes com acetilcolinesterasa, glicogen sintasa quinasa 3, o fosfodiesterasa 7.

[iii] El virtual screening és una tècnica utilitzada en la recerca de noves molècules candidates per al disseny de nous medicaments. Consisteix en comparar una base de dades de molècules amb una molècula de referència i seleccionar les molècules que més s'assemblin.

[iv] PharmScreen és un eina de virtual screening dedicada al descobriment de nous compostos per a la realització de nous medicaments. PharmScreen augmenta dràsticament els *hits* trobats en les primeres fases d'investigació gràcies a un algoritme d'alta precisió basat en els camps d'interacció que generen un conjunt més divers de compostos potencialment actius.

[v] El docking és una tècnica que es fa servir en la química computacional i que permet predir la configuració més estable entre una molècula i un receptor.

El grup de recerca col·labora directament en el desenvolupament de l'eina *PharmScreen*. El principal responsable en la col·laboració és Javier Luque.

- **Javier Luque**: Javier és un professor de la Universitat de Barcelona (UB) i responsable del grup *Biologia Computacional i Disseny de Fàrmacs*. Té una àmplia experiència en el camp de la investigació i desenvolupament de programari de disseny de fàrmacs. És autor de més de 350 publicacions que han estat referenciades més de 11.250 vegades. Actualment Javier està proporcionant la seva experiència per tal d'ajudar a *Pharmacelera* en el desenvolupament de noves metodologies per a la cerca de noves molècules candidates en el desenvolupament de nous fàrmacs.

1.2.2 Pharmacelera

Pharmacelera és una empresa dinàmica i innovadora que proporciona solucions de hardware i software per al disseny de fàrmacs assistit per ordinador, amb productes com **PharmScreen**, **PharmaBox** o **PharmQSAR**. Els productes desenvolupats per *Pharmacelera* trenquen l'equilibri entre la precisió i el temps (costos computacionals) en les primeres etapes de descobriment de fàrmacs mitjançant el desenvolupament de nous models moleculars i mitjançant l'ús d'acceleradors de maquinari per executar aquests models de manera eficient.

L'empresa està formada per un equip multidisciplinari de professionals amb una trajectòria excepcional en els seus respectius camps:

- **Enric Gibert**: Enric és el director d'operacions de *Pharmacelera*. És enginyer informàtic amb una àmplia experiència en el desenvolupament de programari, en investigació i innovació a través dels seus anys de treball en els laboratoris d'*Intel*. Enric té un títol en Enginyeria i Arquitectura per La Salle (Universitat Ramon Llull), doctorat en Arquitectura de Computadors i Tecnologia de Compilació per la Universitat Politècnica de Catalunya (UPC), i disposa d'un títol de Formació empresarial per l'IESE (Universitat de Navarra).

- **Enric Herrero**: Enric és el director de tecnologia de *Pharmacelera*. Va treballar diversos anys en els laboratoris d'*Intel* fent el disseny de maquinari i desenvolupament de metodologies i eines d'avaluació per a plataformes de computació heterogènia. Té un títol d'Enginyer Elèctric per la Universitat Politècnica de Catalunya (UPC) i és doctor en Arquitectura i Disseny de Maquinari Informàtic per la Universitat Politècnica de Catalunya (UPC).

- **Alessandro Deplano**: Alessandro és químic computacional en *Pharmacelera*. És doctor en Química Mèdica per la Universitat de Cagliari en Itàlia i disposa d'un títol de màster en Química i Tecnologia per la Universitat de Cagliari.

- **Javier Vázquez**: Javier és químic computacional en Pharmacelera. Javier és llicenciat en Farmàcia per la Universitat de Barcelona i actualment està portant a terme un programa de doctorat industrial entre la Universitat de Barcelona i Pharmacelera dirigit per Javier Luque i Enric Herrero.

- **Anna Rudzińska**: Anna és una estudiant de màster en bioinformàtica per la Universitat Autònoma de Barcelona. Anna és llicenciada en Biotecnologia per la Universitat de Wroclaw de Ciència i Tecnologia (Polònia).

- **Albert Herrero**: Albert treballa en el camp de la computació d'altres prestacions i és l'encarregat de gestionar i supervisar totes les tasques relacionades en l'optimització i millora de la qualitat del software en *Pharmacelera*. Albert té un títol en enginyeria informàtica de sistemes per la Facultat d'Informàtica de Barcelona (Universitat Politècnica

de Catalunya), un títol de màster en Enginyeria Informàtica per la Universitat Oberta de Catalunya i un títol de màster en Enginyeria Computacional i Matemàtica per la Universitat Oberta de Catalunya i la Universitat Rovira i Virgili (UOC-URV).

1.3 Estructura de la memòria

En aquest apartat es dóna una visió global del document perquè sigui fàcil de consultar i es pugui saber on trobar cada cosa.

- *Descripció.* En aquest bloc es descriu el treball realitzar. Conté una breu introducció al problema tractat, una posada en context que ens situa en el marc de treball en que s'ha realitzat la feina, la justificació de perquè es va decidir realitzar el treball, l'abast, els objectius i la planificació.
- *Tecnologies.* En aquest bloc es descriuen les tecnologies usades en el desenvolupament del treball.
- *Benchmark.* En aquest bloc es fa referència als benchmarks que s'han fet ús en tot el treball.
- *Profiling.* En aquest bloc es fa referència al profiling realitzat en el codi per trobar els punts crítics on aplicar les optimitzacions.
- *Acceleració amb cuda.* En aquest capítol s'explica tot el desenvolupament realitzat amb la tecnologia Cuda.
- *Estudi del nombre de punts del grid.* En aquest capítol es presenta un estudi per identificar el nombre de punts òptims per poder usar Cuda.
- *Acceleració amb OpenMP.* Capítol on s'explica l'acceleració OpenMp aplicada en el software.
- *Vectorització Mopac.* En aquest capítol s'expliquen les tasques realitzades en la vectorització de Mopac.
- *Anàlisis econòmic.* En aquest bloc es fa una una breu valoració dels costos que a suposat fer aquest treball: programari usat, hardware i recursos humans.
- *Valoracions i treballs futurs.* Aquí s'exposen les conclusions globals del treball i es presenten futures línies de millora.
- *Apèndix.* Bloc que conté diferents gràfiques i taules amb els resultats que s'han obtingut en el desenvolupament del treball. Els apartats descrits en l'apèndix complementen la part principal del treball.

1.4 Justificació del projecte

El motiu principal pel qual vaig decidir fer aquest treball va ser la possibilitat i l'interès que tinc en poder treballar en un projecte de I+D. Actualment sóc empleat de *Pharmacelera* i sóc l'encarregat de gestionar i supervisar totes les tasques relacionades en l'optimització i millora de la qualitat del software.

No obstant, aquests interès en el món de la recerca ja ve de fa temps. Va ser en el meu treball final de carrera que vaig començar a tenir els meus primers contactes en el cap de la computació d'altres prestacions, fet em va conduir a fer els meus primers estudis de màster en

la UOC. En aquests estudis vaig fer l'assignatura de '*Computació d'altes prestacions*' on vaig aprofundir més en el camp de l'alta computació i que em va conduir a realitzar el treball final de màster "*Paral·lelització amb GPUs d'una aplicació per a química computacional*". Va ser gràcies aquest treball que vaig poder fer un gir de 360 graus en la meva carrera professional, deixant de banda el món de la consultaria i començar en el món del I+D.

Durant la realització del treball vaig poder contactar amb gent experta en el camp del *HPC* que treballaven en el *BSC* i vaig poder conèixer a Enric Gibert i Enric Herrero, fundadors de *Pharmacelera*. Així doncs, gràcies a aquest treball va ser com des de *Pharmacelera* se'm va oferir la possibilitat de treballar amb ells.

1.5 Abast

Quan es parla d'abast es refereix a la suma o col·lecció de productes, serveis i resultats que s'entreguen com a part del projecte, és a dir, el que es farà i el que no es farà. D'aquesta manera es poden definir dos tipus d'abast: abast del projecte i abast del producte.

L'abast del projecte fa referència al conjunt d'entregues que es faran durant l'execució de tot el projecte i es divideix en:

- Una entrega inicial que conté una descripció i una planificació del projecte.
- Dos entregues parcials que contenen part del producte.
- Una entrega final que conté el producte, una memòria i una presentació.

L'abast del producte fa referència al conjunt de tasques realitzades en l'execució del projecte i es divideix en:

- Realitzar profiling del software.
- Instal·lació i configuració del hardware/software.
- Optimització PharmScreen.
- Optimització Mopac.
- Estudi dels resultats

1.6 Objectius

Els principals objectius que es busquen en la realització d'aquest treball són:

- Augmentar la velocitat d'execució del software PharmScreen.
- Millorar el coneixement de la tecnologia CUDA.
- Millorar les meves aptituds de recerca.
- Integrar els coneixements que he adquirit durant el màster.

1.7 Planificació

La planificació del projecte va ser, d'entrada, una aproximació de la feina que s'havia de realitzar ajustada al temps de dedicació per hores/crèdit acadèmic. En aquest sentit però, no es van tenir en compte els inconvenient ni les dificultats que han anat sorgint durant el desenvolupament, fet que va obligar a canviar la planificació inicial del treball eliminant algunes parts, afegint-ne unes altres i donant més temps d'execució en algunes altres. La nova planificació es descriu mitjança la taula de fites i el seu diagrama de Gant associat.

1.7.1 Taula de fites

La taula 1.1 mostra un desglòs de les tasques realitzades:

Tasca	**Data Inici**	**Data Fi**	**Jornades**	**Hores**
Anàlisi i disseny global del treball	**1/11/16**	**11/11/16**	**3.75**	**30**
Entendre el problema	01/11/16	04/11/16	1.25	10
Dissenyar estructura del treball	07/11/16	11/11/16	2.5	20
Preparació entorn	**14/11/16**	**03/03/17**	**1.8**	**15**
Identificar rendiment base	14/11/16	18/11/16	0.25	2
Instal·lació del software de desenvolupament	14/11/16	18/11/16	0.18	1.5
Instal·lació del software de profiling	14/11/16	18/11/16	0.18	1.5
Configurar entorns de treball	**14/11/16**	**03/03/17**	**1.25**	**10**
- Màquina base	14/11/16	18/11/16	0.62	5
- Màquina Amazon Web Services	27/02/17	03/03/17	0.37	3
- Màquina Rutgers	27/02/17	03/03/17	0.25	2
Benchamarks	**21/11/16**	**05/05/17**	**2.5**	**20**
Definició benchamarks PharmScreen	21/11/16	25/11/16	1.25	10
Definició benchamarks Mopac	01/05/17	05/05/17	1.25	10
Profiling	**28/11/16**	**05/05/17**	**2.5**	**20**
Definició benchamarks PharmScreen	28/11/16	02/12/16	1.25	10
Definició benchamarks Mopac	01/05/16	05/05/17	1.25	10
Acceleració amb Cuda	**21/11/16**	**03/03/17**	**25**	**200**
Anàlisi del codi seqüencial	21/11/16	25/11/16	3.75	30
Dissenyar implementació amb Cuda	28/11/16	09/12/16	6.25	50
Implementar codi amb GPU	12/12/16	17/02/17	12.5	100
Executar casos de prova	30/01/17	24/02/17	1.25	10
Anàlisi de resultats	27/02/17	03/03/17	1.25	10
Estudi del nombre de punts del grid	**06/03/17**	**31/03/17**	**11.8**	**95**
Definir cas d'estudi	06/03/17	10/03/17	2.5	20
Implementar estudi	13/03/17	24/03/17	7.5	60
Anàlisi de resultats	27/03/17	31/03/17	1.8	15
Acceleració amb OpenMP	**27/03/17**	**01/05/17**	**16.25**	**130**
Anàlisi del codi seqüencial	27/03/17	31/03/17	2.5	20
Dissenyar implementació amb OepnMP	17/03/17	31/03/17	2.5	20
Implementar codi amb OpenMP	03/04/17	14/04/17	8.75	70
Executar casos de prova	17/04/17	21/04/17	1.25	10
Anàlisi de resultats	24/04/17	01/05/17	1.25	10
Vectorització Mopac	**01/05/17**	**30/05/17**	**8.75**	**70**
Anàlisi amb Intel Advisor	01/05/17	05/05/17	0	5
Identificar punts crítics	01/05/17	05/05/17	0	5
Anàlisi punts crítics	08/05/17	19/05/17	0	20

Aplicar millores	08/05/17	19/05/17	0	20
Executar casos de prova	22/05/17	30/05/17	0	10
Anàlisi de resultats	22/05/17	30/05/17	0	10
Preparació informe inicial	**15/11/16**	**30/11/16**	**3.12**	**25**
Redacció i correcció de la memòria	**01/11/16**	**02/06/17**	**16.25**	**130**
Preparació de la presentació	**05/06/17**	**14/06/17**	**2.5**	**20**
TOTAL	**01/11/2016**	**14/06/2017**	**94.4**	**755**

Taula 1.1: Taula de fites.

1.7.2 Diagrama de Gantt

El següent diagrama de Gantt representa l'avanç del projecte a traves del temps. El diagrama representa cadascuna de les tasques de la taula de fites.

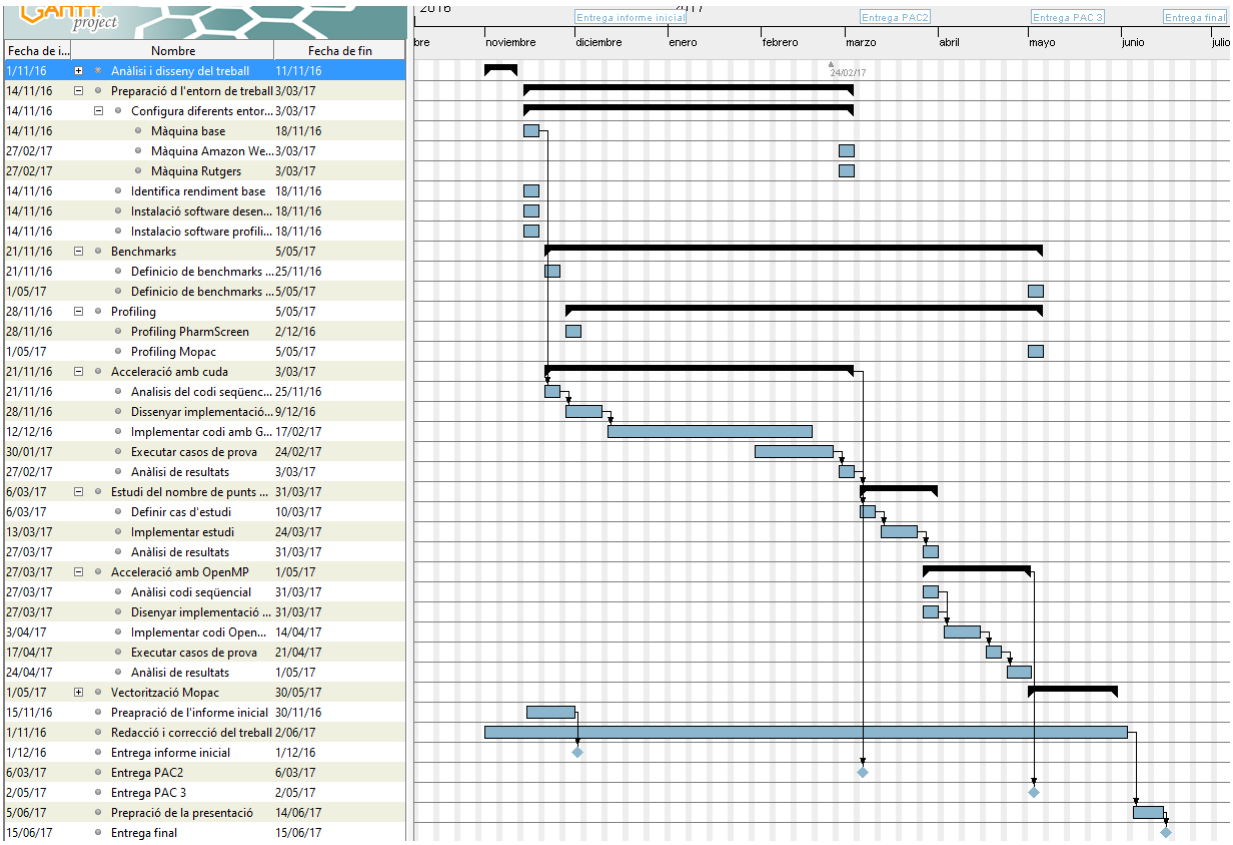

Figura 1.1: Diagrama de Gantt.

Capítol 2

Tecnologies

2.1 PharmScreen

L'ús de mètodes computacionals per a la cerca de molècules actives és un tema central de la químio-informàtica [1]. Un dels seus objectius és permetre el processat d'un gran nombre de compostos processats mitjançant tècniques de cribrat d'alt rendiment, amb l'objectiu d'identificar potencials *hits* contra noves dianes d'interès farmacològic.

La línia d'investigació que segueix *Pharmacelera* es basa en el desenvolupament del software **PharmScreen**, software capaç d'identificar compostos estructuralment diversos amb potencial activitat biològica de manera eficaç i precisa, contribuint a reduir l'elevat cost que comporta l'ús de tècniques experimentals de cribratge.

En aquest context, el grup de recerca *Biologia Computacional i Disseny de Fàrmacs (CBDD)*, dirigit pel professor Javier Luque de la Universitat de Barcelona ha desenvolupat una metodologia basada en l'ús de models continus de solvatació acoplats a mètodes de química quantica (QM) [2, 3, 4, 5, 6, 7]. Aquesta metodologia permet derivar formalismes per a descompondre l'energia de solvatació en contribucions associades a àtoms i/o fragments, que al mateix temps permet derivar nous descriptors fisicoquímics. D'altar banda, les contribucions atòmiques a la solvatació en diversos solvents poden ser combinades entre si, donant a lloc a contribucions fraccionals a la partició d'un compost entre diversos solvents. La qualitat d'aquests paràmetres ha sigut posada de manifest mitjançant una serie d'estudis preliminars, definint conceptes com dipol hidrofòbic i similitud hidrofòbica, que han permès calibrar el potencial dels descriptors en les relacions estructura-activitat, la similitud molecular i la biodistribució de fàrmacs [8, 9, 10, 11].

La investigació neix amb la intenció d'aplicar els descriptors mencionats en un algoritme de cribratge virtual basat en *lligand*. Amb aquest propòsit es persegueix un doble objectiu cientificotècnic. D'una banda, en l'àmbit científic es pretén desenvolupar e implementar models de representació molecular basats en nous descriptors fisicoquímics, que proporcionin una descripció precisa del reconeixement molecular i la formació de complexos lligand-receptor. D'altra banda, en el marc tecnològic es plantejar el desenvolupament de tècniques computacionals d'alt valor afegit orientades a millorar l'eficiència computacional en dos àmbits de gran rellevància e impacte en el disseny de fàrmacs: *alineament* i *comparació de la similitud molecular*.

2.1.1 Mètode

El protocol de cribratge virtual presentat a continuació descansa sobre dos eixos principals: la *funció d'alineament* i la *similitud molecular*. Fonamentades, totes dues, en els descriptors hidrofòbics derivats a partir de càlculs de solvatació amb models *QM* continus.

La hidrofobicitat d'una molècula es determina típicament a partir del repartiment entre octanol i aigua, *logPo/w*. Aquest paràmetre és una propietat global de la molècula, reflectint el balanç net de les seves interaccions en aigua i en octanol. No obstant això, és convenient descompondre aquesta mesura de hidrofobicitat global en descriptors fraccionaris, atès que això permet avaluar la complementarietat entre la distribució hidrofòbica 3D entre una molècula donada i el seu potencial anàleg.

En aquest sentit, l'esquema de descomposició de l'energia lliure dins de la versió *MST* del model de solvatació *IEF-PCM* permet definir les contribucions atòmiques a *logPo/w*. Per tant, això possibilita la definició de moments capaços de descriure amb precisió la distribució topològica de hidrofilicitat/hidrofobicitat en una molècula, que al seu torn podran emprar-se en l'alineament de molècules i valorar, amb posterioritat, la seva similitud. D'aquesta manera, s'obtenen tensors capaços de reorientar i superposar petites molècules en base a les contribucions atòmiques a la solvatació.

La descripció d'aquests moments hidrofòbics es sustenta en les bases teòriques de la descomposició multipolar del potencial electrostàtic, en què l'elecció del centre d'expansió és fonamental i la caracterització dels coeficients és depenent de la seva localització [12]. Fixar el centre adequat proporciona un punt on el terme líder és dominant al llarg de la major part de l'espai i la contribució del següent terme és mínima. Aquesta possibilitat permet utilitzar l'expansió multipolar per caracteritzar els camps moleculars al voltant d'un únic terme invariant a la translació.

En el cas de molècules iòniques el centre d'expansió s'escull com el punt on el terme líder és la component monopolar, en tant que això minimitza la contribució del dipol, que serà nul aplicat a aquest centre l'equació 2.1. Per contra, en molècules neutres s'escollirà el centre del dipol (eq. 2.2) com a origen de coordenades, atès que això minimitza la contribució del terme quadripolar, sent el dipol el terme líder.

$$q = \frac{\sum(Q_i \cdot X_i)}{Q_{total}} \tag{2.1}$$

$$d = \frac{2}{3p^2}(Q \cdot p - (\frac{p \cdot Q \cdot p}{4p^2}) \cdot p) \tag{2.2}$$

En aquest escenari, utilitzar les contribucions atòmiques a la solvatació permet definir una expansió multipolar del camp hidrofòbic. A més, el mètode continu de solvatació *MST* permet desglossar l'energia lliure de solvatació en tres contribucions 2.3. Aquesta característica ens dóna accés a explorar les relacions de similitud molecular canviant diferents descriptors.

$$\triangle G_{sol} = \triangle G_{ele} + \triangle G_{vW} + \triangle G_{cav} \tag{2.3}$$

Les contribucions no electrostàtiques depenen de la superfície exposada al solut, reflectint la mida i la forma molecular. De fet, hi ha correlació entre aquests dos camps proporcionant informació similar de les característiques estèriques. D'altra banda, la contribució electrostàtica es determina a partir del camp de reacció del dissolvent induït per la distribució de càrrega del solut, proporcionant informació relacionada amb les característiques electrostàtiques. Així mateix, s'observa una manca de correlació respecte a les contribucions no electrostàtiques constatant la no redundància entre els termes electrostàtics i no electrostàtics [13].

Un cop la funció d'alineament ha reorientat els compostos sobre els eixos propicis, es computa la semblança entre molècules emprant el coeficient de *Tanimoto* (eq. 2.4), que és possiblement la funció de similitud més popular al camp.

$$T_c(A,B) = \frac{c}{a+b-c} \tag{2.4}$$

on, a i b són el nombre de motius presents en els compostos A i B, respectivament, i c és el nombre de motius compartits entre A i B.

Per avaluar la similitud i ordenar els candidats en funció de la seva analogia, es projecten els diferents camps de força en una malla 3D (eq. 2.5) [13]. La superposició de malles 3D permet obtenir diferents coeficients de *Tanimoto*, que conformaran la funció de similitud final.

$$Pr(j) = \sum_{i=0}^{n} w_i e^{-ar_i^2} \tag{2.5}$$

On w_i és el valor de hidrofobicitat de la molècula (j), a és el valor d'atenuació, i r_i és la distància entre l'àtom i el punt de la malla 3D.

2.1.2 Algoritme

El sistema té dues entrades: una llibreria de compostos i una o diverses molècules de referència. Independentment del tipus d'entrada es computen els diferents descriptors molècula a molècula. En el següent pas, es procedeix a l'alineament molecular (punt 2.1.1), un cop superposades es projecten els camps de força corresponents als descriptors sobre una malla de punts. La superposició de la malla punt a punt genera l'índex de similitud. La sortida del sistema, serà llavors, el llistat de les molècules d'entrada ordenades en base a la seva similitud respecte a les molècules de referència.

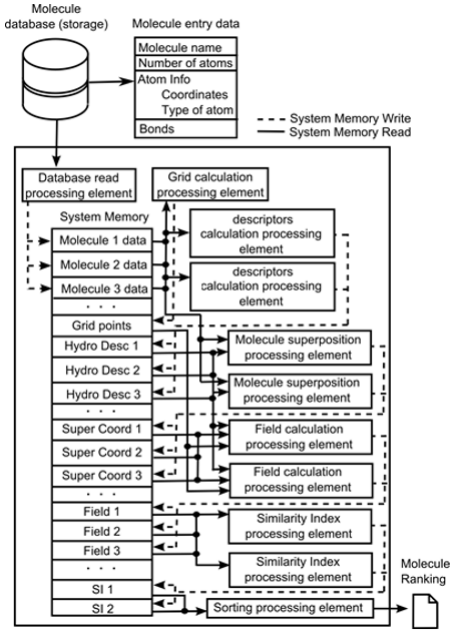

Figura 2.1: Esquema algoritme PharmScreen.

2.2 Mopac

Mopac (Molecular Orbital Package) és un *software* de química computacional amb potents aplicacions de química quàntica per a la predicció de propietats químiques, càlculs de molècules químiques i modelatge de reaccions químiques. *Mopac* és un *software* fiable en la predicció de propietats químiques i físiques tals com les energies lliures de *Gibbs*, energies d'activació, moments dipolars, propietats òptiques no lineals i espectres infrarojos. *Mopac* implementa paràmetres semi empírics hamiltonians com *MNDO, AM1, PM3, i Mindo/3, PM5* i combina càlculs d'espectres de vibració, característiques termodinàmiques, efectes de substitució isotòpica, efectes deprenents del temps i constants de força en un software altament integrat.

Des de *Pharmacelera* hem integrat *Mopac* juntament amb *PharmScreen* per tal de poder calcular diferents propietats químiques de les molècules que es processen en el software. Per poder fer-ho hem compilat *Mopac* i hem creat unes llibreries estàtiques que ens permeten fer ús del software.

Actualment la versió de *Mopac* de la que disposem està implementada amb el llenguatge de programació *Fortran* i fent ús d'una API externa realitzem una portabilitat a llenguatge C que ens permet integrar-ho amb el nostre software.

Tot i no ser el principal objectiu d'aquest treball, es presenten alguns apartats que se'n parla, ja que els càlculs que es fan amb *Mopac* gasten una quantitat de temps molt elevada que fa que els temps d'execució de *PharmScreen* s'eleven de manera dràstica quan es fa ús de les seves característiques. Concretament el que s'ha fet és analitzar el codi de *Mopac* per detectar punts crítics on aplicar tècniques d'optimització.

2.3 Cuda

Cuda és una plataforma de programació paral·lela de caràcter general dissenyada per habilitar a les GPU's a treballar de manera conjunta amb la CPU i proporcionar més poder de còmput. Cuda va ser introduït al novembre de l'any 2006 per *NVIDIA* per poder aprofita el motor de càlcul paral·lel de les seves GPUs per resoldre problemes computacionals complexos d'una manera més eficient que en una CPU. CUDA ve amb un entorn de programari que permet als desenvolupadors utilitzar C com a llenguatge de programació d'alt nivell.

2.3.1 Processament paral·lel de caràcter general

Impulsada per la demanda insaciable del mercat, les unitats de processament gràfic o GPUs han evolucionat fins a convertir-se en processadors altament paral·lels, multiprocés i multicore amb una capacitat computacional i un ample de banda de memòria molt alt, tal com s'il·lustra en les figures 2.2 i 2.3.

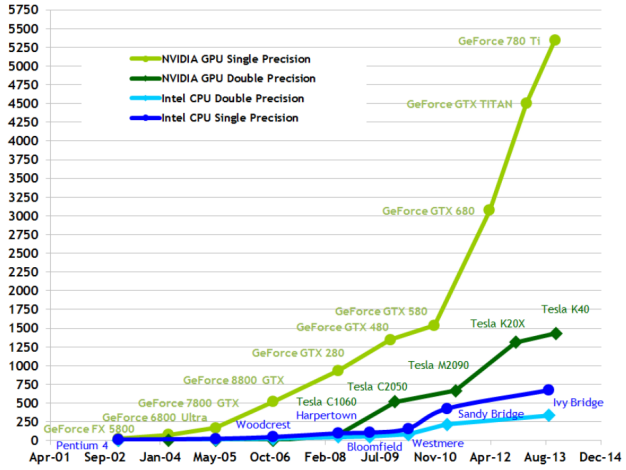

Figura 2.2: Operacions per segon en coma flotant de CPU i GPU. GLOPS/s teòrics [32].

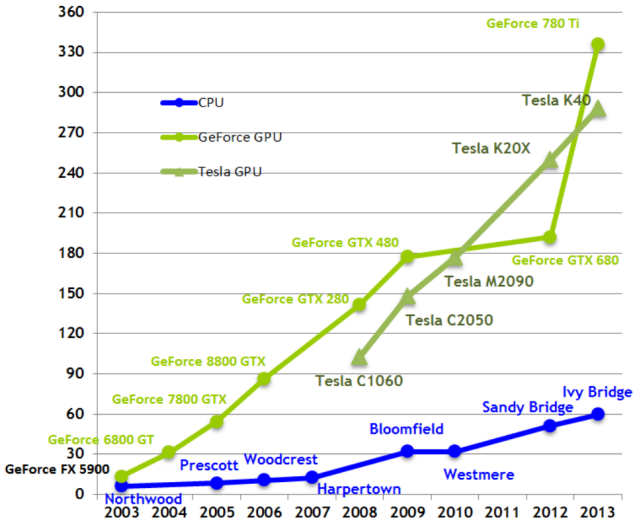

Figura 2.3: Ample de memòria de CPU i GPU. GB/s teòrics [32].

La raó darrera la discrepància en punt flotant entre la CPU i la GPU és que la GPU està especialitzada en computació intensiva, càlcul altament paral·lel i per tant dissenyada de tal manera que té més transistors dedicats al processament de dades en lloc d'emmagatzemar-les en memòria cau i controlar el flux. Això s'il·lustra en la figura 2.4.

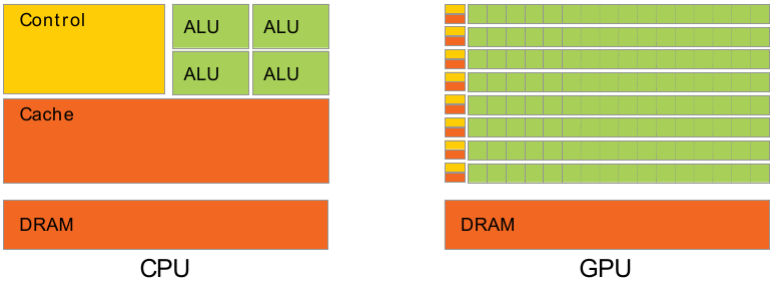

Figura 2.4: Transistors dedicats al processament de dades [32].

Més específicament, la GPU és especialment adequada per abordar els problemes que es poden expressar com a càlculs de dades en paral·lel -el mateix programa s'executa en molts elements de dades en paral·lel- amb alta intensitat aritmètica -la relació de les operacions aritmètiques per les operacions de memòria. A causa de que el mateix programa s'executa per a cada element de dades, no hi ha la necessitat de disposar d'un control de flux sofisticat, i pel fet d'executar-se en molts elements de dades i tenir una intensitat aritmètica alta, la latència d'accés a memòria es pot amagar amb els càlculs en lloc de disposar de molta memòria cau.

El processament de dades paral·lel consisteix en repartir els elements de dades entre fils d'execució paral·lels. Moltes aplicacions que processen grans conjunts de dades poden usar el model de programació de dades en paral·lel per augmentar de manera dràstica la seva velocitat d'execució. Un exemple és la representació d'imatges 3D, on grans conjunts de píxels i vèrtexs són repartits entre fils paral·lels. De manera similar, aplicacions de processament d'imatge, de codificació/descodificació de vídeo, d'escalatge d'imatges o de reconeixement de patrons es poden processar amb programació de dades en paral·lel. De fet, molts algoritmes fora del camp de representació i processament d'imatges són accelerats d'aquesta manera, des del processament de senyals o simulacions físiques fins a algoritmes en biologia o química computacional.

2.3.2 Model de programació escalable

L'aparició de les CPU's i GPU's multicore indica que els processadors d'avui en dia són sistemes paral·leles en si mateixos. Un punt a tenir en compte és que el seu paral·lelisme continua escalant segons la llei de *Moore's*. El repte en si consisteix en desenvolupar software que adapti transparentment el seu paral·lelisme de manera que pugui aprofitar l'increment en el nombre de cores de processament.

El model de programació de *Cuda* paral·lel està dissenyat per a suportar aquest repte mentre manté una corba d'aprenentatge baixa per als programadors familiaritzats amb ell llenguatge de programació C/C++. En general, *Cuda* té tres punts claus -jerarquia de grups de *threads*, memòria compartida, i barreres de sincronització- que són exposats al programador com un conjunt mínim d'extensions del llenguatge.

Aquestes punts proporcionen paral·lelisme de dades i paral·lelisme amb *threads*, de forma que si s'usen conjuntament proporcionen al programador la possibilitat de dividir el problema en subproblemes que es poden resoldre independentment en paral·lel per blocs de *threads*, i on cada subproblema es divideix en trossos més petits per poder ser resolt de forma conjunta entre

tots els *threads* d'un block.

Aquesta descomposició preserva l'expressivitat del llenguatge permeten als *threads* cooperar quan solucionen cada subproblema, i al mateix temps habilita la escalabilitat automàtica. De fet, cada bloc de *threads* pot ser assignat a qualsevol dels multiprocessadors disponibles dins d'una GPU, en qualsevol ordre, simultàniament o seqüencialment, de forma que un programa *Cuda* es pot executar en qualsevol multiprocessador tal com s'il·lustra a la figura 2.5, sent totalment transparent al programador i controlat per la API en temps d'execució.

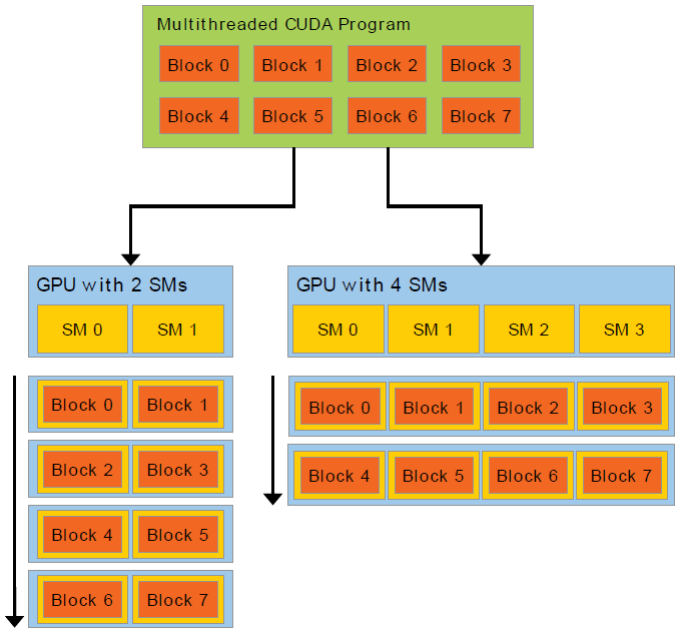

Figura 2.5: Multiprocessadors en una GPU [32].

Una GPU està construïda per un vector de *Streaming Multiprocessors (SM)*. Un programa multiprocés es divideix en blocs de *threads* que s'executen independentment entre si, de manera que una GPU amb més multiprocessadors executarà automàticament el programa en menys temps que un GPU amb menys multiprocessadors.

2.3.3 Model de programació

El model de programació en *CUDA* es basa en quatre punts:

- *Kernels*
- Jerarquia de *threads*
- Jerarquia de memòria
- Programació heterogènia

2.3.3.1 Kernels

CUDA C és una extensió de *C* que permet al programador definir funcions *C*, anomenades *kernels*, que quan es criden són executades N vegades en paral·lel per N *threads* diferents.

Un *kernel* es defineix usant la declaració *__global__* i el nombre de *threads* que l'executen s'indica

amb la declaració <<< ... >>> en el moment de la crida. Cada *thread* que executa el *kernel* té associat un únic *thread ID* que és accessible des de dins del kernel a través de la variable **threadIdx**.

Com a il·lustració, la figura 2.6 mostra un codi d'exemple on es fa la suma de dos vectors A i B de mida N i es guarda el resultat en un vector C:

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Figura 2.6: Exemple de suma de dos vectors amb CUDA [32].

En l'exemple, cadascun tels N *threads* que executen el *kernel* **VecAdd(..)** realitza la suma d'un parell de nombres i ho guarda en la posició corresponent en el vector C.

2.3.3.2 Jerarquia de threads

Per definició, la variable **threadIdx** és un vector amb tres components, de forma que els *threads* es poden identificar amb un *thread index* de una, dues o tres dimensions, formant així un bloc de *threads* d'una, de dues o tres dimensions anomenat *thread block*. Aquesta manera de representar els *threads* dóna lloc a una forma natural per invocar execucions sobre vectors, matrius i volums de dades.

L'index d'un *thread* i el seu *thread ID* es poden relacionar de manera fàcil: per un bloc d'una dimensió són els mateixos; per a un bloc de dues dimensions de mida (D_x, D_y), el *thread ID* d'un *thread d'índex* (x, y) és $(x + yD_x)$; per a un bloc de tres dimensions de mida (D_x, D_y, D_z), el *thread ID* d'un *thread d'índex* (x, y, z) és $(x + yD_x + zD_xD_y)$.

La figura 2.7 mostra un exemple de la suma de dues matrius A i B de mida NxN i guarda el resultat en una matriu C.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Figura 2.7: Exemple de suma de dos matrius amb un bloc [32].

Hi ha un limit de *threads* per bloc, ja que tots els *threads* d'un mateix bloc han de residir en el mateix processador per poder compartir els recursos de memòria. De la mateixa manera hi ha un limit de *threads* per processador. En les GPU actuals el límit de *threads* per bloc és de 1024 i el límit per processador és de 2048. No obstant això, un *kernel* es pot executar en diversos blocs i en diversos processadors a la vegada, de forma que el nombre total de *threads* per a l'execució del *kernel* és el nombre de *threads* per bloc per el nombre totals de blocs.

Els blocs s'organitzen en una malla de blocs d'una, de dues o de tres dimensions anomenada *grid* (Fig. 2.8). El nombre de blocs en un *grid* usualment és dictat per la grandària de les dades que s'estan processant o pel nombre de processadors en el sistema.

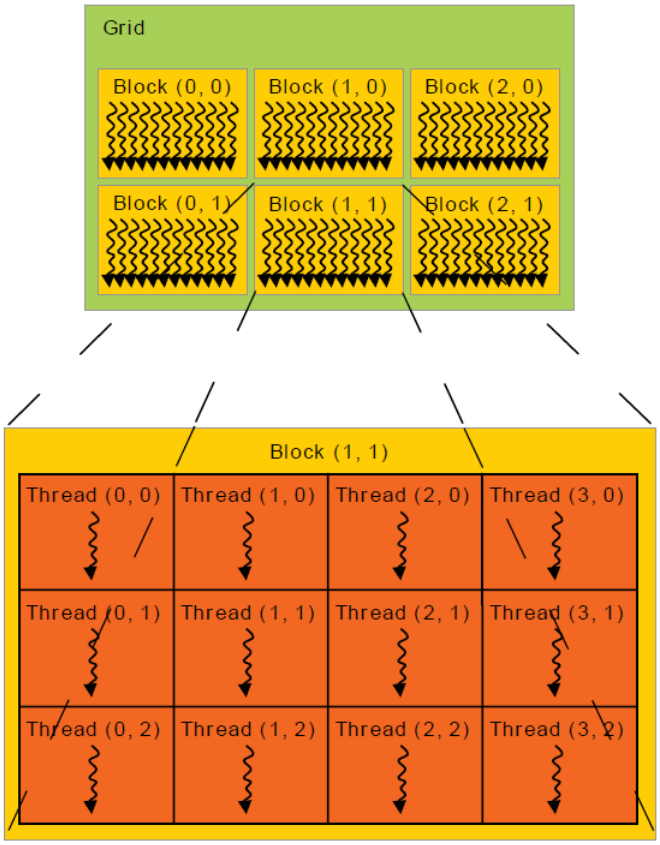

Figura 2.8: Grid de blocs [32].

El nombre de *threads* per bloc i el nombre de blocs per grid que s'especifiquen amb la sintaxi <<< ... >>> han de ser de tipus **int** o **dim3**. Els blocs o *grids* de dues dimensions es poden especificar com en l'exemple de la figura 2.7.

Cada bloc dins d'uns *grid* pot ser identificat per un index d'una, dues o tres dimensions accessible des de dins dels *kernels* mitjançant la variable **blockIdx**. La dimensió d'un bloc és accessible des dels *kernels* a través de la variable **blockDim**.

Si ampliem l'exemple de la figura 2.7 per a fer ús de multiples blocs, el codi queda com es veu en la figura 2.9.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Figura 2.9: Exemple de suma de dos matrius amb varis blocs [32].

En l'exemple es pot veure una mida de bloc de 16x16 (256 *threads*). El *grid* està dimensionat amb suficients blocs per tal de tenir un *thread* per cada element de la matriu. Per simplicitat, es suposa que en aquest exemple el nombre de *threads* per *grid* en cada dimensió es divisible pel nombre de *threads* per bloc en cada dimensió.

En aquest model de programació els blocs s'executen de manera independent: es poden executa en qualsevol ordre, en paral·lel o en serie. Aquesta independència permet que els blocs es reparteixin entre tots els multiprocessadors disponibles (Fig. 2.5), habilitant al programador la possibilitat d'escriure codi que escali automàticament amb el nombre de multiprocessadors.

Els *threads* dins d'un mateix bloc poden cooperar compartint dades a través de la memòria compartida (*shared memory*) i mitjançant la sincronització de les execucions per tal de coordinar els accessos a la *shared memory*. Més específicament, es poden indicar punts de sincronització en un *kernel* mitjançant la crida **__syncthreads()**, la qual actua com una barrera fins que tots els *threads* d'un mateix bloc hagin arribat al punt de sincronització.

2.3.3.3 Jerarquia de memòria

Els *threads* en *Cuda* poden accedir a les dades en diferents espais durant la seva execució, tal com ilustra la figura 2.10. Cada *thread* té un espai de memòria privat. Cada bloc té un espai de memòria compartida per tots els *threads* del mateix bloc. Tots els *threads* tenen accés a un mateix espai de memòria global.

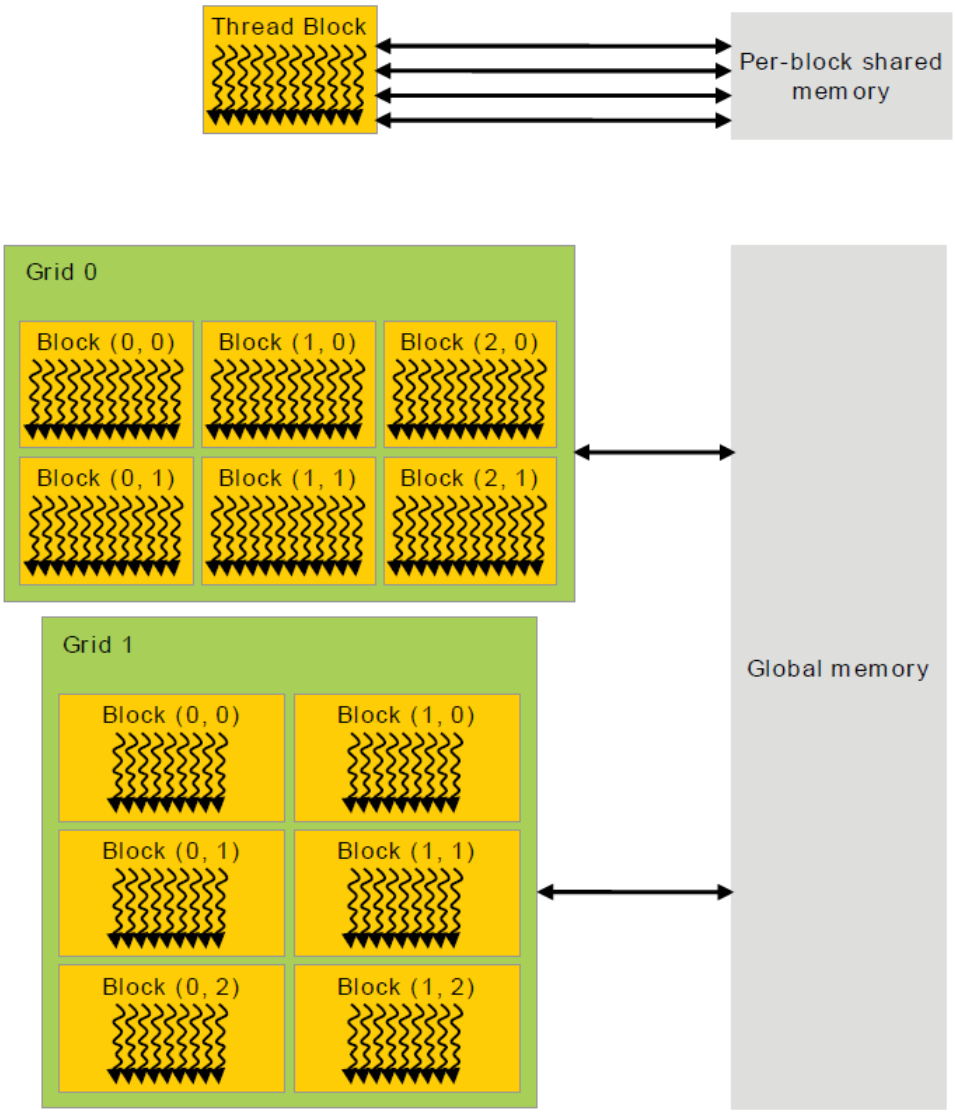

Figura 2.10: Jerarquia de memòria en cuda [32].

2.3.3.4 Programació heterogènia

En general el model de programació amb *CUDA* assumeix que els *threads* s'executen en un dispositiu (*device*) físic separat que opera com a coprocessador del *host* que està executant el programa en C (Fig. 2.11). Aquest és el cas, per exemple, quan els *kernels* s'executen en una GPU i la resta del programa C s'executa en la CPU.

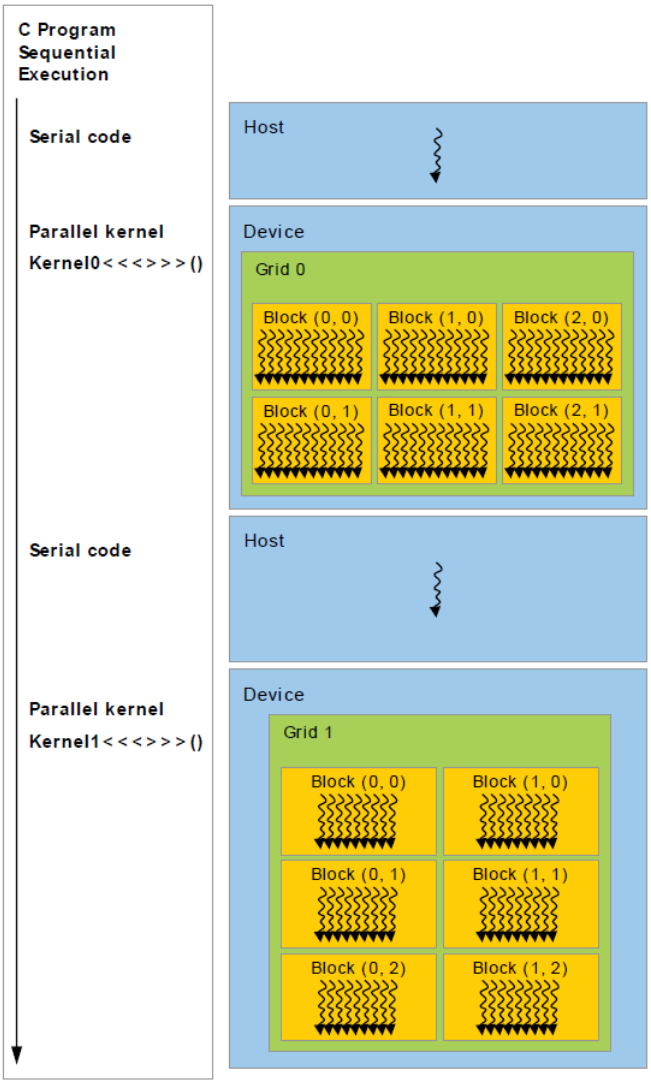

Figura 2.11: Execució de codi serie en CPU i codi paral·lel en GPU [32].

2.4 OpenMP

OpenMP consisteix en una interfície de programació que estén llenguatges de programació, com ara C/C++ i Fortran, mitjançant l'ús de directives o *pragmes*. *OpenMP* és l'estàndard actual per a la programació de sistemes de memòria compartida, com ara sistemes multinucli i computadors d'altres prestacions basats en multiprocessadors amb memòria virtual compartida. Típicament OpenMP s'utilitza en sistemes amb un nombre no massa gran de processadors, tot i que hi ha implementacions de sistemes de memòria compartida amb centenars de nuclis. També hi ha versions d'OpenMP per a altres tipus de sistemes, com ara clústers i acceleradors, però en aquests casos el rendiment dels programes *OpenMP* pot ser inferior al dels que utilitzen entorns de programació específics per a aquests sistemes.

El model d'execució d'*OpenMP* és *fork-join*, com il·lustra la figura 2.12. Això vol dir que inicialment el programa treballa amb un únic flux fins que arriba a una regió paral·lela, és a dir, quan arriba al constructor *#pragma omp parallel* que posa en marxa un nombre de fluxos esclaus (part *fork* del model). El flux que treballa inicialment és el flux mestre d'aquest conjunt d'esclaus. Els fluxos estan enumerats des de 0 fins al nombre de fluxos -1. El mestre i els esclaus treballen en paral·lel en el bloc que apareix a continuació del constructor. En acabar la regió paral·lela hi ha una sincronització de tots els fluxos, els esclaus moren i únicament queda el flux mestre (part *join* del model). A partir d'aquest moment el mestre continua treballant seqüencialment tret que comenci una altra regió paral·lela. Tal com il·lustra la figura, també cal tenir en compte que hi pot haver més d'un nivell de regions paral·leles (regió paral·lela imbricada).

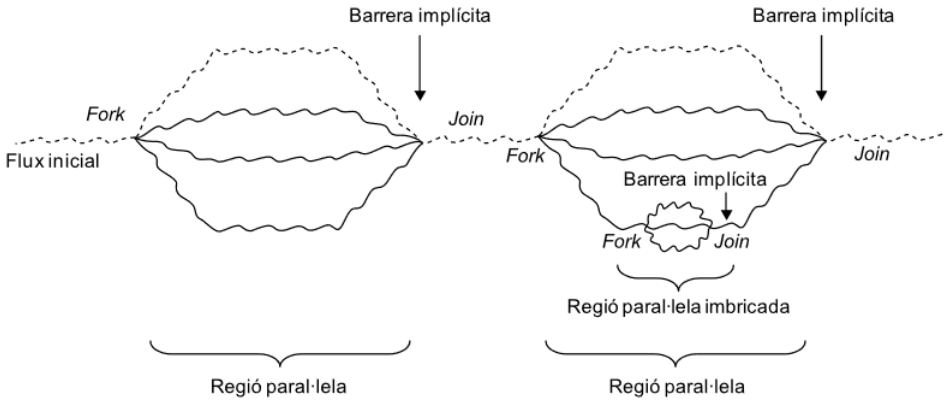

Figura 2.12: Model fork-join d'OpenMP [33].

El codi següent mostra un exemple de programa en *OpenMP* en què es poden observar alguns elements necessaris en programes OpenMP, com ara la inclusió del fitxer de capçaleres *omp.h*, en què es defineixen les funcions d'OpenMP, la utilització de directives (comencen per *#pragma omp*) per a indicar al compilador la manera en què s'ha de distribuir el treball o la utilització de clàusules per a indicar operacions concretes com per exemple la reducció. En concret, el codi retorna la suma de la multiplicació dels elements de dos vectors.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
   int   i, n;
```

```
    float a[100], b[100], sum;

    /* Algunes inicialitzacions - regió seqüencial */
    n = 100;
    for (i=0; i < n; i++)
       a[i] = b[i] = i * 1.0;
    sum = 0.0;

    #pragma omp parallel for reduction(+:sum)   /* Regió paral·lela */
    for (i=0; i < n; i++)
       sum = sum + (a[i] * b[i]);

    printf("Suma = %f\n",sum);   /* Regió seqüencial */
 }
```

2.4.1 Directives OpenMP

Les directives OpenMP segueixen les convencions dels estàndards per a directives de compilació en C/C++, diferencien entre majúscules i minúscules, només es pot especificar un nom de directiva per directiva, i cada directiva s'aplica almenys a la sentència que segueix, que pot ser un bloc estructurat. El format general és el següent:

```
#pragma omp nom_directiva {[clàusules, ...]
```

en què

- *#pragma omp* es demana a totes les directives OpenMP per a C/C++.
- *nom_directiva* és un nom vàlid de directiva, i ha d'aparèixer després del *pragma* i abans de qualsevol clàusula.
- *[clàusules, ...]* són opcionals. Les clàusules poden anar en qualsevol ordre i repetir.se quan sigui necessària, a menys que hi hagi alguna restricció.

2.4.2 Creació de fluxos

Per crear fluxos d'execució s'usa la directiva *parallel* i s'utilitza de la següent manera:

```
#pragma omp parallel [clàusules]
   bloc
```

Amb aquest *pragma*:

- Es crea un grup de fluxos i el flux que els posa en marxa pren el rol de flux mestre.
- Hi ha una barrera implícita al final de la regió, de manera que el flux mestre espera que acabin tots els esclaus per a continuar amb l'execució seqüencial.
- Quan dins d'una regió hi ha un altre constructor *parallel*, cada esclau crearia un altre grup de fluxos esclaus dels quals seria el mestre. Això s'anomena paral·lelisme imbricat i, tot i que es poden programar d'aquesta manera, en algunes implementacions d'*OpenMP* la creació de grups d'esclaus dins d'una regió paral·lela no està suportada.

2.5 Entorn de treball

Tots els experiments i proves realitzades s'han executat en diferents entorns de treball per a tenir una visió més amplia del rendiment del *software*.

La taula 2.1 mostra les característiques tècniques de cada entorn de treball.

	Pharmacelera	**Rutgers**	**Amazon**
Cpu	Intel Core i7-4790	2 x Intel Xeon E5-2680 v3	Intel Xeon E5-2666 v3
Cpu Clock	3.60GHz	2.50GHz	2.90GHz
# of cores	4	2x12	8
#of threads	8	2x24	16
Extensions	SSE4.1/4.2, AVX 2.0	AVX 2.0	SSE4.1/4.2, AVX 2.0
SSD	SI	SI	SI
Memory	16	256	30
SSO	Ubuntu 16.04	CentOS Linux 7	Ubuntu 16.04
GPU	Quadro M4000	Tesla K40	NO

Taula 2.1: Característiques tècniques del maquinari.

A nivell de *software* s'ha fet serivir:

- IDE de programació *Netbeans*.
- *LibreOffice Calc* per a crear taules i gràfiques.
- *PyMOL* ver visualitzar y crear imatges de molècules.
- *Intel Vtune* per fer profiling del codi.
- *Intel Advisor* per buscar possibles bucles per vectoritzar.
- *Nvidia Visual Profiling* per estudiar els *kernels* de Cuda.
- *GanttProject* per crear el diagrama de gantt i portar un seguiment del treball.
- *Ubuntu 16.04* com a sistema operatiu.
- *LaTeX* per redactar la memòria.

Capítol 3

Benchmarks

En computació, un *benchmark* és l'acte d'executar un programa informàtic, un conjunt de programes o altres operacions, amb la finalitat d'avaluar el rendiment relatiu del software, normalment executant un conjunt de proves estàndard. El terme *benchmark* també està utilitzat en l'elaboració de programes de *benchmarking*. El *benchmarking* es sol associar amb l'avaluació de les característiques de rendiment del hardware en una computadora, com per exemple, el rendiment de l'operació en coma flotant d'una CPU però, també és possible aplicar-ho en el software.

Per tal d'avaluar de manera correcta el funcionament del software s'ha definit un conjunt de benchamarks que permeten valorar el rendiment a mesura que es fan canvis, s'apliquen optimitzacions i s'afegeixen noves funcionalitats.

3.1 Pharmscreen

Per tal de verificar el mètode científic usat en el software s'està fent servir la base de dades de molècules *DUD_LIB_VS_1.0*[i] que inclou una gran varietat de conjunts de molècules amb diferent nombre d'àtoms i diferents característiques cadascun. De tota la base de dades s'han usat 4 conjunts per definir els benchmarks:

- **tk** (Fig. 3.1): *timidina quinasa*[ii], conjunt on les molècules tenen una mitja de 30 àtoms.
- **hsp90** (Fig. 3.2): *proteïna de shock tèrmic 90Kda*[iii], en aquest conjunt les molècules tenen una mitja de 43 àtoms.
- **vegfr2** (Fig. 3.3): *receptor 2 del factor de creixement del endoteli vascular*[iv], en aquest conjunt les molècules tenen una mitja de 44.

[i] DUD (de l'anglès *directory of useful decoys*) és una base de dades de molècules dissenyada per ajudar a testejar algoritmes de *virtual screening* i *docking*. Consta d'un total 2950 compostos actius amb un total de 40 *targets*.

[ii] Timidina quinasa és una enzim que pertany al grup de les *fosfotransferases*. Es troben en la major part de les cèl·lules vives i en alguns virus. La timidina quinasa té una funció molt important en la síntesis de l'ADN i la divisió cel·lular, fent possible la introducció de desoxitimidina en la cadena de l'ADN.

[iii] La Hsp90 o proteïna de xoc tèrmic de 90 kDa (heat shock protein 90), és una proteïna *xaperona* (proteïnes presents en totes les cèl·lules que tenen com a funció ajudar al plegament d'altres proteïnes recent formades en la síntesis de proteïnes) que ajuda a altres proteïnes a plegar adequadament, estabilitza proteïnes contra situacions d'hipertèrmia i dóna suport a la degradació de proteïnes

[iv] El factor de creixement endotelial vascular (VEGF, per Vascular endothelial Growth Factor) és una proteïna senyalitzadora implicada en la vasculogènesi (formació de novo del sistema circulatori embrionari) i en l'angiogènesi (creixement de vasos sanguinis provinents de vasos preexistents).

- **fgfr1** (Fig. 3.4): *receptor 1 de creixement fibroblàstic*[v], en aquest conjunt les molècules tenen una mitja de 50 àtoms.

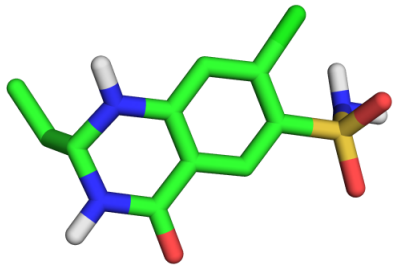

Figura 3.1: Exemple d'una molècula del conjunt TK.

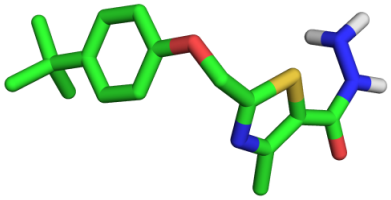

Figura 3.2: Exemple d'una molècula del conjunt HSP90.

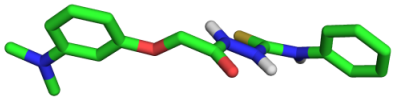

Figura 3.3: Exemple d'una molècula del conjunt VEGFR2.

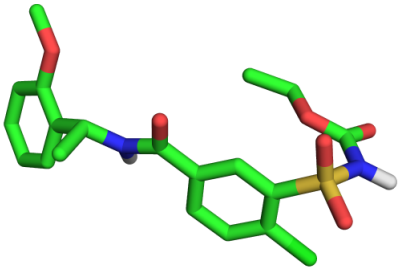

Figura 3.4: Exemple d'una molècula del conjunt FGFR1.

[v]Un factor de creixement de fibroblasts o FCF (també s'usa l'abreviatura FGF de l'anglès per fibroblast growth factor) és un factor de creixement que augmenta l'índex d'activitat mitòtica i síntesi d'ADN facilitant la proliferació de diverses cèl·lules precursores, com el condroblast, colagenoblast, osteoblast, etc ... que formen el teixit fibrós, d'unió i suport del cos.

D'aquests 4 conjunts s'han definit 3 mides amb diferent nombre de molècules en cada cas:

- **tk**
  - *Petit*: 20000 molècules
  - *Mitjà*: 130000 molècules
  - *Gran*: 182200 molècules
- **hsp90**
  - *Petit*: 14000 molècules
  - *Mitjà*: 84000 molècules
  - *Gran*: 199200 molècules
- **vegfr2**
  - *Petit*: 10500 molècules
  - *Mitjà*: 60000 molècules
  - *Gran*: 165000 molècules
- **fgfr1**
  - *Petit*: 8000 molècules
  - *Mitjà*: 50000 molècules
  - *Gran*: 160000 molècules

3.2 Mopac

Com en el cas anterior, s'han definit un conjunt de *benchmarks* per a poder provar i avaluar l'evolució del software *Mopac*. En aquest cas s'han fet ús de 4 conjunts diferents que també pertanyen a la biblioteca de molècules *DUD* anomenada anteriorment.

- **ace**: *acetilcolinesterasa.*[vi].
- **cdk2**: *quinasa dependent de ciclina.*[vii].
- **fxa**: *factor anticuagulació a*[viii].
- **hivrt**: *virus immunodeficiència humana trasnos e inversa*[ix].

En aquest cas però, els conjunts que es fan servir són tots de *20 molècules*, ja que, com s'ha mencionat en el punt 2.2, els temps de càlcul al fer ús de les característiques de *Mopac* són molt elevats i fa que els temps d'execució de *Pharmscreen* cresquin de manera dràstica. També és necessari indicar que en aquest cas el nombre de d'àtoms de les molècules no és tant important com en el punt 3.1, sinó que els temps d'execució de *Mopac* es basen amb els tipus d'àtoms que es processen (carboni, hidrogen,...) i la distancia entre ells.

[vi] La acetilcolinesterasa o colinesterasa en glòbuls vermells (ACE) és un enzim humana de la família de colinesterases que es troba en els teixits nerviosos i els glòbuls vermells, la funció principal és hidrolitzar al neurotransmissor acetilcolina. En els glòbuls vermells constitueix un antigen cel·lular anomenat *Yt*

[vii] La proteïna Cdk2 pertany a la família de les quinases dependents de ciclines amb activitat serina/treonina quinasa. És una subunitat catalítica del complex format amb la quinasa dependent de ciclina, on la seva activitat es restringeix a la fase G1/S del cicle cel·lular i és essencial per a la transició entre les dues fases.

[viii] És un tipus de coagulant que interfereix en la coagulació de la sang creant un estat antitrombòtic o prohemorrágico.

[ix] És HIVRT és el que es coneix com a virus del sida.

Capítol 4

Profiling

En enginyeria del software l'**anàlisi de rendiment**, comunament anomenat **profiling**, és la investigació del comportament d'un programa d'ordinador fent ús d'informació reunida des de l'anàlisi dinàmic del mateix. L'objectiu és esbrinar el temps dedicat a l'execució de diferents parts del programa, així com detectar els punt problemàtics i les àrees on sigui possible portar a terme una optimització del rendiment (ja sigui en velocitat o en el consum de recursos). Usualment el *profiling* es usat durant el desenvolupament del software com a mètode per a la depuració i optimització dels algoritmes. Amb l'objectiu de trobar els punt problemàtics en el nostre software, hem fet servir el conjunt d'eines d'Intel per analitzar el nostre codi, en concret hem usat: *VTunes*[i].

4.1 Profiling de PharmScreen

Per veure els punts crítics de PharmaScreen hem fet servir l'eina *VTunes* d'Intel. L'eina disposa de diferents tipus d'anàlisis que permeten obtenir diferents nivells de precisió, no obstant, nosaltres hem fet servir l'anàlisi bàsic ***Basic Hotspots***.

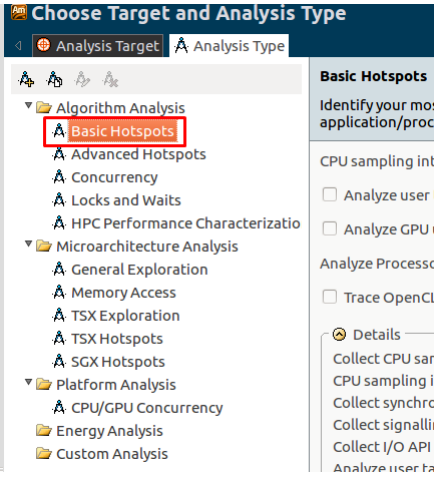

Figura 4.1: Tipus d'anàlisi amb VTtunes.

[i]VTune de Intel és una potent eina d'anàlisi que permet obtenir, a més d'informació sobre el rendiment d'aplicacions, mecanismes d'optimització a través de la identificació de colls d'ampolla.

La figura 4.2 mostra els punts on es gasta més temps de comput després de realitzar l'anàlisi.

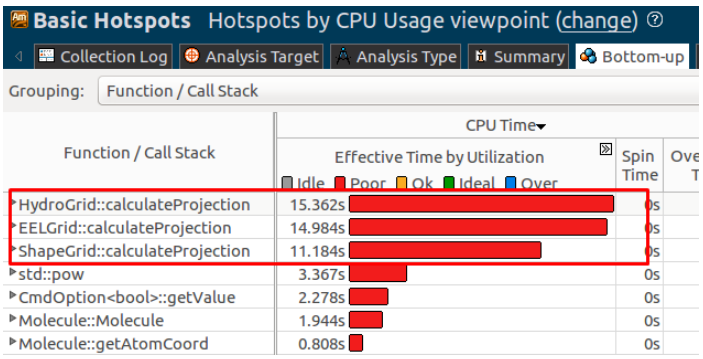

Figura 4.2: Hotspot functions

De l'anàlisi realitzat es dedueix que els punts on es consumeix més temps de càlcul són les **funcions de projecció** del software. D'aquesta manera, gran part del treball consisteix en accelera aquestes funcions amb la tecnologia *Cuda*. L'estudi realitzat es presenta en els capítols 5 i 6.

4.2 Profiling de Mopac

Per trobar els punts crítics de *Mopac* hem fet servir l'eina de *profiling VTune*. En aquest cas però, també hem usat l'eina ***gprof***[ii] per complementar l'anàlisi. La decisió de fer servir una segona eina ve donada pels resultats que hem obtingut en l'acceleració de les funcions de projecció amb *Cuda*. Com es veurà ens els capítols 5 i 6 del treball, en el moment de realitzar el *profiling* vam cometre l'error de no veure que l'eina *VTunes* indica la quantitat de temps gastat en cada funció **durant tota l'execució**, però no indica el **nombre de vegades que es crida**, de manera que el temps total pot ser elevat però el temps de calcul de la funció petit.

En general, els temps de calcul en l'execució de *Mopac* depenen molt del tipus d'àtoms de la molècula (hydrogen, carboni, ...), de la posició dels àtoms en l'espai i de la distància entre ells. En una execució amb *Mopac* els temps d'execució es gasten en una funció o en un altra depenen de la molècula que s'estigui processant. Per aquest motiu, en el moment de fer el *profiling* s'han fet servir **4 molècules diferents** per tal de tenir una gama més amplia de resultats.

En figures 4.3, 4.4, 4.5 i 4.6 es mostren les funcions de *Mopac* que gasten més temps quan s'executa amb les diferents molècules.

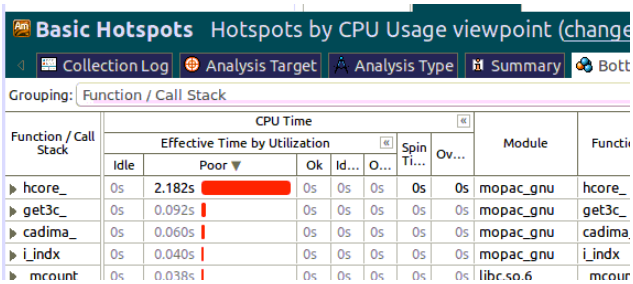

Figura 4.3: Anàlisi primera molècula amb VTunes.

[ii] Gprof és una eina de *profiling* que permet realitzar diferents anàlisis per identificar problemes de rendiment en el software.

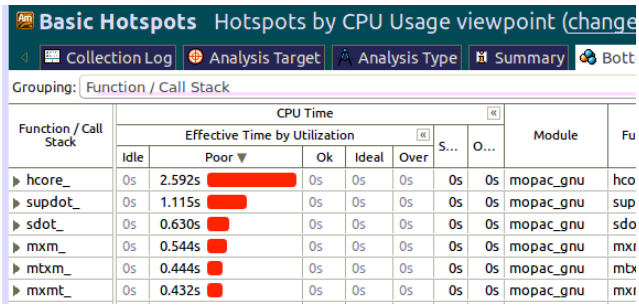

Figura 4.4: Anàlisi segona molècula amb VTunes.

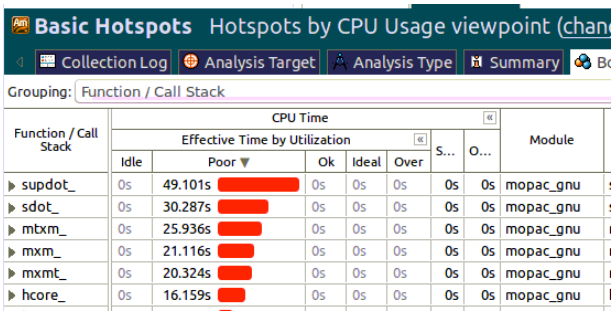

Figura 4.5: Anàlisi tercera molècula amb VTunes.

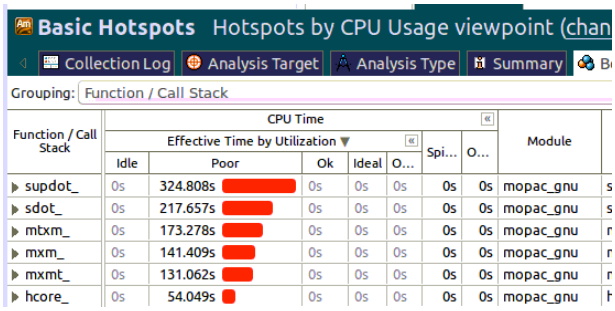

Figura 4.6: Anàlisi quarta molècula amb VTunes.

Si s'analitzen els resultats que dóna *VTunes* es veu clarament que les funcions que gasten més temps són ***hcore_***, ***supdot_***, ***sdot_***, ***mxm_***, ***mtxm_*** i ***mxmt_***. D'aquesta manera el punt de partida per accelera *Mopac* és analitzar questes funcions i buscar oportunitats on aplicar *Cuda*. Abans però, s'ha realitzar el segon anàlisi amb l'eina *gprof* per tal de no cometre el mateix error que en el cas de *PharmScreen*.

La taula 4.1 mostra les funcions que consumeixen més temps en l'execució de *Mopac* per cada molècula. En la taula es mostra la següent informació:

- % **of time**: el percentatge de temps gastat en relació amb el temps total de simulació.
- **spend seconds**: el nombre de segons total gastats en la funció.
- **calls**: el nombre de vegades que es crida la funció.

- **self s/call**: la mitja de segons gastats en cada crida.
- **function**: el nom de la funció.
- **total time**: el temps total de simulació.

	% of time	spend seconds	calls	self s/call	function	total time
Molècula 1	86.51	2.18	27	0.08	hcore_	2.52
	1.19	0.03	1749888	0.00	get3c_	
	1.19	0.03	1	0.03	cadima_	
Molècula 2	38.30	2.62	34	0.08	hcore_	6.84
	13.45	0.92	177727	0.00	supdot_	
	10.53	0.72	4896930	0.00	sdot_	
	8.19	0.56	34233	0.00	mxm_	
	7.16	0.49	21864	0.00	mtxm_	
	5.99	0.41	18120	0.00	mxmt_	
Molècula 3	37.07	49.59	1897434	0.00	supdot_	6.84
	16.62	30.42	111063645	0.00	sdot_	
	14.00	25.62	112784	0.00	mtxm_	
	11.63	21.28	174464	0.00	mxm_	
	11.08	20.28	91580	0.00	mxmt_	
	8.87	16.24	93	0.17	hcore_	
Molècula 3	27.98	318.15	7940947	0.00	supdot_	6.84
	18.52	210.57	617062750	0.00	sdot_	
	15.58	177.14	382429	0.00	mtxm_	
	13.05	148.36	600123	0.00	mxm_	
	11.94	135.74	318484	0.00	mxmt_	
	4.76	64.12	222	0.17	hcore_	

Taula 4.1: Temps d'execució molècules de prova amb Mopac.

Analitzant la taula es pot veure que en les funcions es gasta un percentatge elevat de temps però, tal com indiquen les columnes *call* i *self s/call* les funcions s'executen moltes vegades i el temps propi d'execució de la funció no arriba a 1 segon. Arribats aquest punt i amb els resultats obtinguts ens vam preguntar si valia la pena intentar accelerar *Mopac* amb *Cuda* i la resposta era obvia: tal com tenim el codi no ho podem accelerar *Cuda* perquè es tracta de funcions que de manera individual consumeixen una quantitat mol reduïda de temps.

Donat que quan es fan servir els càlculs de *Mopac* l'execució total de *PharmScreen* augmenta de manera dràstica, des de *Pharmacelera* ens interessava fer alguna cosa per a millorar els temps d'execució, i ja que no podíem usar *Cuda* vam decidir explorar la possibilitat de fer-ho amb *OpenMP* o d'aprofitar les característiques de vectorització de que disposen els processadors amb les tecnologies *SSE/AVX*. Finalment vam decidir utilitzar l'eina *Advisor d'Intel* per adaptar el codi i beneficiar-nos de l'autovectorització (capítol 8)..

Capítol 5

Acceleració amb cuda

5.1 Descripció de les funcions de projecció

Les funcions de projecció són les encarregades de calcular la contribució de cada àtom de la molècula en cada punt de l'espai. A nivell conceptual el que es vol és saber de quina manera afecten els àtoms d'una molècula a l'espai, de manera que s'obtenen uns paràmetres fisicoquímics que indiquen amb quin grau els punts de l'espai es veuen afectats per la molècula. Segons la manera de realitzar els càlculs s'identifiquen diferents funcions de projecció:

- **Electrostàtic projection**: aquesta projecció fa ús de les càrregues dels àtoms per realitzar els càlculs.
- **Shape projection**: aquesta projecció fa ús del volum dels àtoms per realitzar els càlculs.
- **Hydro projection**: aquesta projecció fa ús de les diferents característiques d'hidrofobicitat[i] dels àtoms per realitzar els càlculs.

L'esquelet de la funció de projecció és el mateix en tots els casos i l'únic que canvia és la manera en que es calculen els paramètrics fisicoquímics que afecten als punt de l'espai. En aquest treball però, per motius de confidencialitat, sols es fa referència a la projecció electrostàtica.

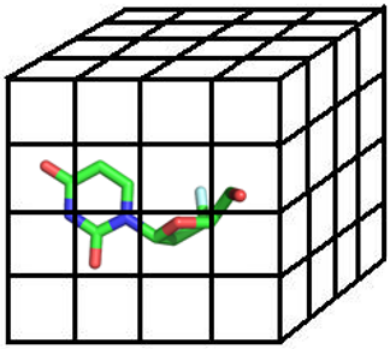

Figura 5.1: Molècula en l'espai

[i] L'efecte hidròfob representa la tendència de l'aigua a excloure molècules no polars. L'efecte s'origina a partir de la ruptura dels enllaços d'hidrogen altament dinàmics entre molècules d'aigua líquida

Matemàticament el que fa la funció projecció electrostàtica és resumeix amb la següent equació:

$$\sum_{i=0}^{M} \frac{q_i}{d_{12}} \tag{5.1}$$

on M és el número d'àtoms de la molècula, q_i és la càrrega que té cada àtom de la molècula i $d12$ és la distancia entre el punt on es troba l'àtom i cada punt de l'espai. Aquests calcul es realitza per cada punt *P(X, Y, Z)* de l'espai.

Com a resultat d'executar els càlculs de la funció de projecció i aplicar l'equació 5.1 s'obté una malla tridimensional que guarda, per a cada punt *P(X, Y, Z)*, la contribució de tots els àtoms de la molècula en el punt.

A l'hora de realitzar els càlculs hi ha el problema que els punts en l'espai són infinits, per tant, és necessari crea un grid tridimensional on posicionar els àtoms. Per dimensionar el grid es fa servir un centre origen, per exemple el centre geomètric de la molècula. Una vegada s'ha posicionat la molècula en el centre origen és fàcil calcular les dimensions del grid a partir de la mida de la molècula.

Un altre aspecte a tenir en compte és el número de punts que ha de tenir el grid, és a dir, s'ha de definir una distancia entre els punts que permeti realitzar els càlculs amb un temps i una precisió d'acord amb el que es vol calcular. Per fer-ho es fa servir la unitat de mesura **Ångström**[ii].

5.2 Implementació de la funció de projecció

Després d'una breu explicació del funcionament es presenten dues versions de l'algorisme en pseudocodi:

- **Versió seqüencial**: és la versió original de la funció.
- **Versió Cuda**: és la nova versió de l'algorisme fent ús de la tecnologia Cuda. Aquesta versió consta de 5 implementacions diferents que intenten explotar la capacitat de comput de la GPU.
  - La primera implementació consisteix en definir un grid i un kernel on cada thread computa un punt de l'espai, és a dir, cada thread calcula la contribució de tots els àtom en un sol punt.
  - La segona implementació consisteix en definir un grid i un kernel en que cada thread computa la contribució d'un sol àtom en N punts de la coordenada Z.
  - La tercera implementació consisteix en definir un grid i executar M (número d'àtoms) kernels de manera concurrent. En aquesta implementació cada thread executa els càlculs per tots els punts de la coordenada Z.
  - La quarta implementació consisteix en definir un grid i executar Z/N kernels de manera concurrent.
  - La quinta implementació és com la primera versió però, explotant la memòria compartida de la GPU.

[ii] El Ångström és una unitat de longitud usada principalment per expressar longituds d'ona i distancies moleculars i atòmiques. Es representa amb la lletra sueca Å i es tracta d'una unitat de mesura que equival a una deu mil milionèsima part d'un metre.

5.2.1 Versió seqüencial

L'esquelet de les funcions de projecció amb pseudocodi és el següent:

```
Projection_function
{
    for each x {
        for each y {
            for each z {
                for each atom {
                    //some calculations
                }
            }
        }
    }
}
```

Figura 5.2: Pseudocodi de la funció de projecció (seqüencial).

Les funcions tenen com a paràmetres d'entrada:

- El número d'àtoms de la molècula.
- Una llista amb les coordenades X, Y i Z de cada àtom.
- Una llista amb les càrregues de cada àtom.

El que fa la funció es iterar per tots els punts de l'espai, i per cada punt es calcula la contribució de tots els àtoms de la molècula segons el tipus de projecció que s'estigui usant. En el cas de la projecció electrostàtica els càlculs estan definits per l'equació 5.1.

5.2.2 Versió CUDA

Les següents versions del codi implementen la paral·lelització de la funció de projecció amb *Cuda*. A l'hora d'implementar-les s'ha seguit el mateix estil de programació que en el codi original, i l'estructura dels fitxers i el nom de les variables s'ha modificat poc.

Per tal de donar suport als diferents tipus de projecció i les diferents implementacions de la funció amb *Cuda*, s'han fet les següents modificacions en el software:

- S'han definit un conjunt de noves classes que es fan servir com a interfície per enllaçar el software amb les diferents implementacions de les projeccions amb Cuda (Fig. 5.4).
- S'ha definit un nou fitxer *.h* que conté la implementació dels diferents kernels (Fig. 5.4).
- S'ha modificat la classe que realitza la lectura dels paràmetres de la línia de comandes per que accepti nous paràmetres i detecti l'ús de CUDA.
- S'ha modificat la classe que executa les funcions de projecció per poder executar la versió seqüencial o la versió amb CUDA segons els paràmetres d'entrada de la línia de comandes.

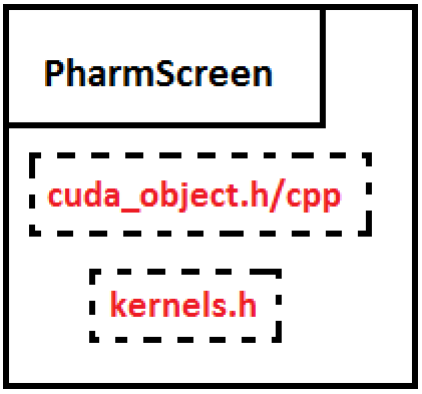

Figura 5.3: Afegit 'cuda_object.h/cpp' i 'kernel.h'.

El fitxer *kernel.h* està definit de manera que es puguin usar les 5 implementacions i amb cadascuna d'elles es pugui executar el codi corresponent a la projecció desitjada. El fitxer disposa de 5 kernels, un per cada implementació, que executen una funció en la GPU que realitza els càlculs segons el tipus de projecció seleccionat.

```
#ifndef KERNELS_H
#define KERNELS_H

electrostatic_function(...) { ... }
shape_function(...) { ... }
hydro_function(...) { ... }

fisrt_kernel(...)
{
   ...
   switch(projection)
   {
       case electrostatic:
           selectrostatic_function(...)
       case electrostatic:
           shape_function(...)
       case electrostatic:
           hydro_function(...)
   }
   ...
}

...

fifth_kernel(...)
{
   ...
   switch(projection)
   {
       case electrostatic:
           selectrostatic_function(...)
       case electrostatic:
           shape_function(...)
```

```
        case electrostatic:
            hydro_function(...)
    }
    ...
}

#endif /* KENELS_H */
```

Els fitxers *cuda_object.h/cpp* defineixen i implementen un conjunt de classes que s'usen com a interfície per a enllaçar el software amb CUDA.

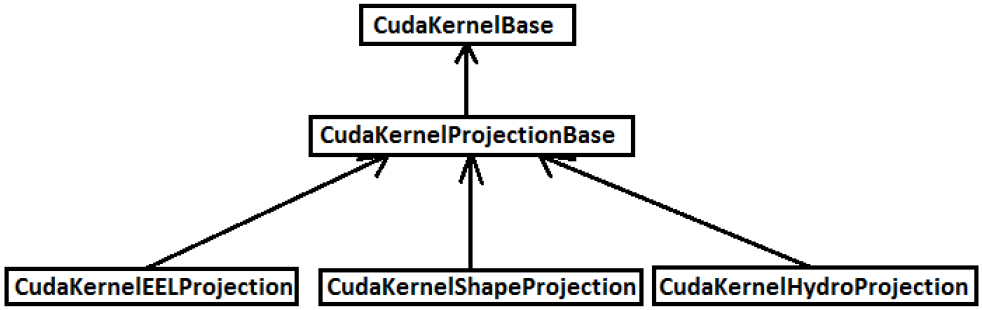

Figura 5.4: Diagrama classes CUDA.

Es té una classe *CudaKernelBase* que guarda tota la informació relacionada amb la GPU (nom del dispositiu, memòria del dispositiu, memòria compartida per block, número màxim de threads per multiprocessador, etc.). Es té la classe *CudaKernelProjectionBase* que hereta de la classe *CudaKernelBase* i guarda la informació relacionada amb el kernel que es va a executar (block size, grid size, tipus de projecció). Per últim es tenen les classes *CudaKernelEELProjection*, *CudaKernelShapeProjection* i *CudaKernelHydroProjection* que hereten de la classe *CudaKernelProjectionBase* i són les encarregades de realitzar les crides al kernel segons la projecció que es vulgui executar.

5.2.2.1 Primera implementació

La primera implementació consisteix en paral·lelitzar els tres bucles de les coordenades X, Y i Z. La idea darrere d'aquesta implementació és executar un kernel que realitza els càlculs de les contribucions de tots els àtoms de la molècula en un únic punt de l'espai.

CPU:

```
Projection_function
{
    //Launch Kernel
}
```

Figura 5.5: Pseudocodi de la funció de projecció (primera implementació).

GPU:

```
kernel
{
    ...
    for each atom {
       // some calculations
    }
     ...
}
```

Figura 5.6: Pseudocodi del kernel (primera implementació).

Per aquesta implementació es té una configuració del grid en la que hi han $X * Y * Z$ threads,

- Blocksize = {8, 8, 1}
- Gridsize = {X/Blocksize.x, Y/Blocksize.y, Z}

i que es representa en l'espai de la següent manera:

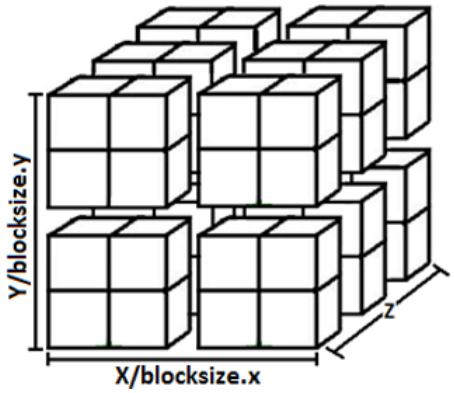

Figura 5.7: Configuració del grid (primera implementació).

5.2.2.2 Segona implementació

En la segona implementació es paral·lelitzen els bucles de les coordenades X i Y i parcialment el bucle de la coordenada Z. La idea darrera d'aquesta implementació és que cada thread del kernel realitzi el calcul de la contribució d'un àtom de la molècula en N punts de la coordenada Z. L'algorisme està dissenyat perquè des de la part del host es llenci seqüencialment un kernel per cada àtom.

CPU:

```
Projection_function
{
    for each atom {
        //Launch Kernel
    }
}
```

Figura 5.8: Pseudocodi de la funció de projecció (segona implementació).

GPU:

```
kernel
{
    ...
    for i=0 to N {
       // some calculations
    }
     ...
}
```

Figura 5.9: Pseudocodi de la funció de projecció (segona implementació).

Per aquesta implementació tenim una configuració del grid en la que hi han $X * Y * Z/N$ threads,

- Blocksize = {8, 8, 1}
- Gridsize = {X/Blocksize.x, Y/Blocksize.y, Z/N}

i que es representa en l'espai de la següent manera:

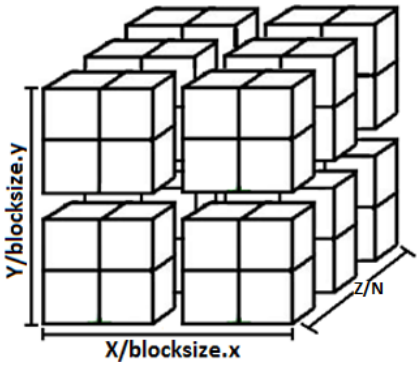

Figura 5.10: Configuració del grid (segona implementació).

5.2.2.3 Tercera implementació

En aquesta implementació es paral·lelitzen els bucles de les coordenades X i Y i el bucle que itera sobre el nombre d'àtoms de la molècula. La idea és llençar un kernel per cada àtom de la molècula de manera concurrent en diferents streams[iii], on cada thread realitza el calcul de la contribució d'un àtom en cada punt de la coordenada Z.

CPU:

```
Projection_function
{
    cudaStream_t streams[numAtoms];
    for each atom {
         cudaStreamCreate(&streams[i])
        //Launch Kernel in streams[i]
    }
}
```

Figura 5.11: Pseudocodi de la funció de projecció (tercera implementació).

GPU:

```
kernel
{
    ...
    for i=0 to Z {
       // some calculations
    }
     ...
}
```

Figura 5.12: Pseudocodi de la funció de projecció (tercera implementació).

En aquesta implementació es configura un grid de $X * Y$ threads,

- Blocksize = {8, 8, 1}
- Gridsize = {X/Blocksize.x, Y/Blocksize.y, 1}

que es representa en l'espai de la següent manera:

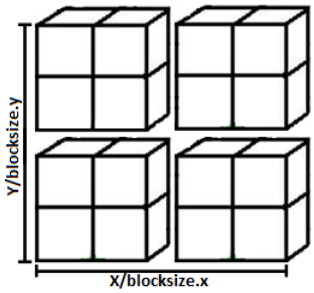

Figura 5.13: Configuració del grid (tercera implementació).

[iii] En cuda, les aplicacions gestionen operacions de manera simultània mitjançant streams. Un stream és una seqüencia de comandes que s'executa en ordre. Amb diferents streams es poden executar diferents comandes a la vegada, com per exemple diferents kernels.

5.2.2.4 Quarta implementació

Aquesta implementació paral·lelitza els bucles de les coordenades X i Y i el bucle de la coordenada Z de manera parcial. La idea es llençar Z/N kernels concurrents en diferents streams on cada thread realitza el calcul de la contribució de tots els àtoms en N punts de la coordenada Z.

CPU:

```
Projection_function
{
    cudaStream_t streams[Z/N];
    for i=0 to Z/N {
         cudaStreamCreate(&streams[i])
        //Launch Kernel in streams[i]
    }
}
```

Figura 5.14: Pseudocodi de la funció de projecció (quarta implementació).

GPU:

```
kernel
{
    ...
    for i=0 to N {
        for each atom {
           // some calculations
        }
    }
     ...
}
```

Figura 5.15: Pseudocodi de la funció de projecció (quarta implementació).

En aquesta implementació es configura un grid de X*Y threads,

- Blocksize = {8, 8, 1}
- Gridsize = {X/Blocksize.x, Y/Blocksize.y, 1}

que es representa en l'espai de la següent manera:

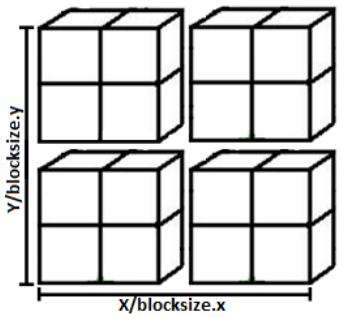

Figura 5.16: Configuració del grid (quarta implementació).

5.2.2.5 Quinta implementació

La quinta implementació és semblant a la primera, és a dir, es paral·lelitzen els tres bucles de les coordenades X, Y i Z i s'executa un kernel que realitza els càlculs de les contribucions de tots els àtoms de la molècula en una únic punt de l'espai. La diferència però, recau en que es fa ús de la memòria compartida de la GPU per a intentar aprofitar la velocitat en la localitat de les dades.

CPU:

```
Projection_function
{
    //Launch Kernel
}
```

Figura 5.17: Pseudocodi de la funció de projecció (quinta implementació).

GPU:

```
kernel
{
    ...
    //Copy data in Shared memory
    ...
    for each atom {
       // some calculations
    }
     ...
}
```

Figura 5.18: Pseudocodi del kernel (quinta implementació).

En la configuració del grid que hi han $X * Y * Z$ threads,

- Blocksize = {8, 8, 1}
- Gridsize = {X/Blocksize.x, Y/Blocksize.y, Z}

i es representen en l'espai de la següent manera:

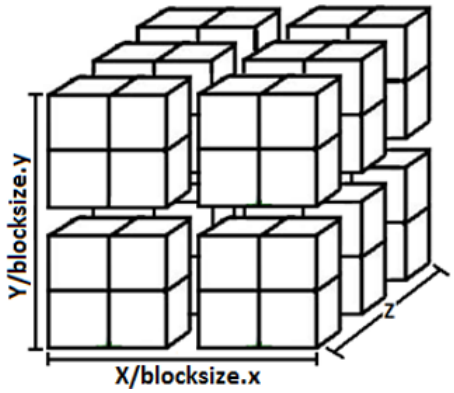

Figura 5.19: Configuració del grid (quinta implementació).

5.3 Resultats

En aquest apartat es presenten els resultats obtinguts en l'execució de la versió seqüencial i les versions CUDA implementades. El capítol es divideix amb tres subcapítols, un per cada mida dels benchmarks, on es mostren tres gràfiques que indiquen els temps de calcul, la mitjana de molècules per segon i els speedups obtinguts. Per tal d'obtenir uns resultats significatius s'ha seguit el següent criteri en el moment d'agafar els resultats:

- S'ha fet 5 execucions de cada versió de l'algorisme.
- S'ha eliminat el mínim i el màxim resultats obtinguts.
- S'ha fet la mitjana aritmètica dels tres resultats.

Els resultats que es presenten s'han obtingut d'un binari compilat amb el flag *O3* i fent ús de la targeta gràfica *Nvidia Quadro M4000* (C.1).

5.3.1 Petit

La figura 5.20 mostra una gràfica de barres amb els temps d'execució dels benchmarks de mida *petita*. Les dades usades per representar la gràfica es mostren en la taula A.1 de l'apèndix. Si s'analitza la gràfica es poden observar varies coses:

- Si mirem el set *tk* es pot veure que els millors temps s'obtenen amb la versió seqüencial.
- Mirant els sets *hsp90*, *vegfr2* i *fgfr1* es pot veure que la primera, tercera i quinta implementació amb *cuda* són més ràpides que la versió seqüencial.
- Per a tots els sets es pot veure que la primera, tercera i quinta implementació amb *cuda* tenen uns temps molt semblants.
- Per a tots els sets es pot veure que la primera, tercera i quinta implementació amb *cuda* són més rapides que les implementacions dos i quatre.

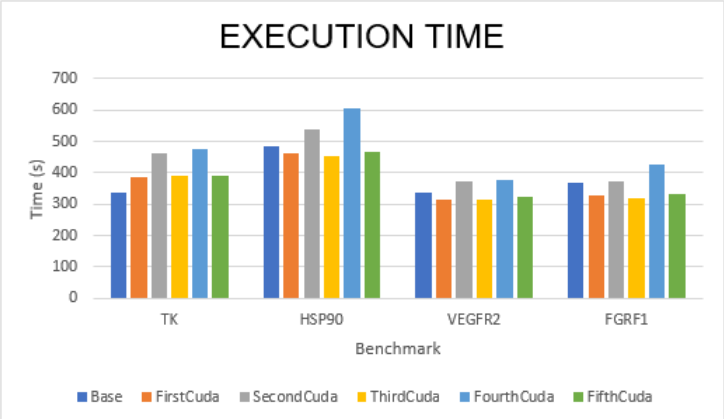

Figura 5.20: Temps d'execució (small benchmark).

La figura 5.21 mostra la mitjana de molècules computades per segon. La mitjana que es mostra és la mitjana global, és a dir, la mitjana de molècules que es capaç de computa cada implementació amb tot els sets. En aquest cas, el que ens interessa des de *Pharmacelera* és saber quantes molècules per segon és capaç de computar el *software* en termes generals. Les dades usades per representar la gràfica es mostren en la taula A.2 de l'apèndix.

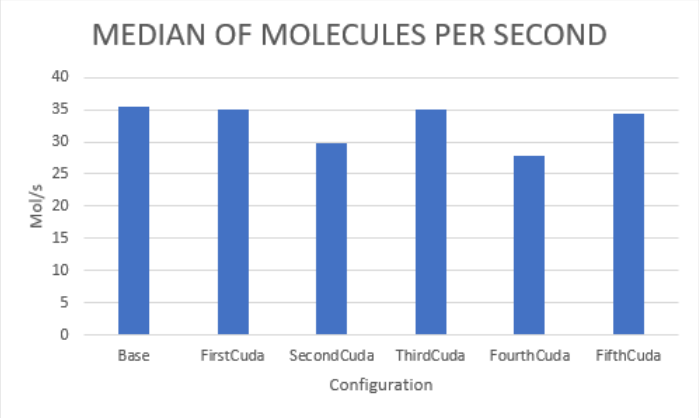

Figura 5.21: Mitjana de molècules per segon (small benchmark).

Analitzant la gràfica es pot observar que:

- La implementació més ràpida és la versió seqüencial (base) amb 35 molècules per segon.
- Les implementacions primera, tercera i quinta de *cuda* processen quasi el mateix nombre de molècules que la versió seqüencial (base).
- Les versions dos i quatre amb *cuda* processen menys molècules que la versió seqüencial (base).

La figura 5.22 mostra els speedups que s'obtenen en les implementacions amb *cuda* per cada set de proba. Les dades usades per representar la gràfica es mostren en la taula A.3 de l'apèndix.

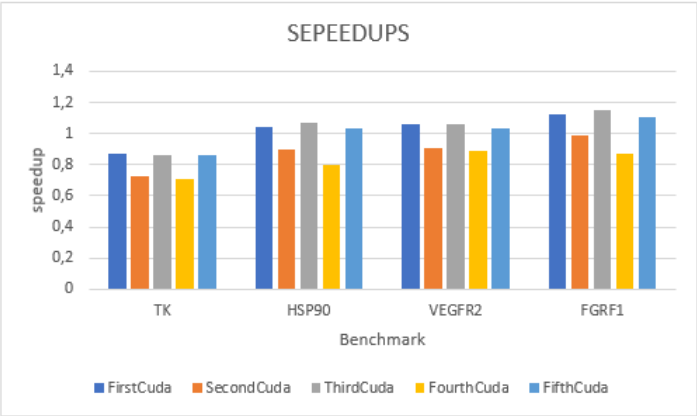

Figura 5.22: Speedups (small benchmark).

Mirant la gràfica s'observa que:

- En el cas del set *tk* cap de les implementacions amb cuda és capaç de millorar el rendiment de la versió base, sinó que l'empitjoren.
- Per als sets *hsp90*, *vegfr2* i *fgfr1* s'observa que les implementacions primera, tercera i quinta són capaces de millorar la versió base però, amb un speedups insignificants que apenes arriba al **1,15x**.

5.3.2 Mitjà

Per al sets de proves mitjans els resultats que s'obtenen són molt semblants als que s'han obtingut amb els sets petits. Per aquest motiu, i per tal de no repetir el mateix que en el punt 5.3.1 del document, s'ha decidit no explicar en detall els resultats, sinó que sols es mostren els punts que poden tenir més rellevància si es comparen amb els resultats obtinguts en el punt 5.3.1. No obstant, es poden veure les gràfiques i les taules de resultats en els punts A.1, A.2, A.3, A.4, A.5 i A.6 de l'apèndix.

Analitzant les figures A.1 i A.3 es veu que el comportament del software és el mateix que amb el set de proves petit, és a dir, els temps i els speedups obtinguts no presenten cap variació significant en quan a rendiment.

D'altra banda, si s'analitza la figura A.2, es pot observar una mínima variació entre les molècules per segon que es computen. En el cas de la figura 5.20 del punt 5.3.1 es tenia que la versió seqüencial era la que computava més molècules per segon, en canvi, per al set mitjà es pot veure que això a canviat i és la tercera implementació amb cuda la que computa més molècules.

5.3.3 Gran

Com en el cas del set de proves mitjà, en aquest cas tampoc es detallen els resultats. Si es desitja es poden veure les gràfiques i les taules en els punts A.4, A.5, A.6, A.7, A.8 i A.9 de l'apèndix.

Si s'analitzen els resultats de manera global es pot veure que són molt semblants que en els dos casos anterior, és a dir, els temps d'execució, speedups i les molècules per segon computades són pràcticament iguals i no es veuen signes de millora.

5.4 Comentaris sobre els resultats

Analitzats els resultats es pot veure d'una manera clara que fent ús de la tecnologia *Cuda* **no ha sigut possible millorar el rendiment** de la funció de projecció, fins arriba al punt en que en alguns casos empitjora. Arribats aquí la pregunta obvia que des de *Pharmacelera* ens hem fet és: *per què no millorar el rendiment?*

La resposta a aquesta pregunta es troba en la mida del problema, és a dir, el nombre de punts del grid. El que fa la funció de projecció és iterar pels punts d'un espai tridimensional definit, de manera que és raonable pensar que el rendiment amb *Cuda* està directament lligat amb el nombre de punts de l'espai. Així, doncs, en el següent bloc del treball s'analitza de manera detallada com afecta el nombre de punts de l'espai amb l'ús dels recursos de la GPU i els speedups que se n'obtenen.

Capítol 6

Estudi del nombre de punts del grid

La funció de projecció itera sobre tots els punts de l'espai definit, de manera que contra més punts tingui l'espai més temps de comput es necessita per realitzar el càlculs. Partint d'aquest fet és fàcil arribar a la conclusió que les implementacions amb *Cuda* tindran un rendiment major si el nombre de punts de l'espai és elevat.

6.1 Nvidia visual profiler

Per validar aquesta teoria s'ha fet servir *Nvidia visual profiler* per realitzar un estudi on es veu com afecta el nombre de punts de l'espai en el rendiment i en l'ús dels recursos de la GPU. S'han fer servir dos mides de l'espai:

- Espai petit amb **1000 punts**:
  - Coordenada X: *10 punts*
  - Coordenada Y: *10 punts*
  - Coordenada Z: *10 punts*
- Espai gran amb **1000000 de punts**:
  - Coordenada X: *100 punts*
  - Coordenada Y: *100 punts*
  - Coordenada Z: *100 punts*

En la taula 6.1 es mostren els limits d'ocupabilitat per SM, nombre de warps[i], registres i memòria compartida dels que disposa el dispositiu "*GPU Quadro M4000*". Aquest informació ens servira per comparar l'ús dels recursos utilitzats en cadascuna de les mides del grid.

[i] Un *warp* és un grup de 32 threads que s'executen en paral·lel. Els threads individuals que componen un *warp* comencen junts en la mateixa adreça del programa però, disposen del seu propi contador de direccions d'instrucció i estat de registre i, per tant, són lliures de divergir i executar-se independentment.

Occupancy per SM

Active blocks	32
Active warps	64
Active threads	2048

Warps

Threads/block	1024
Warps/block	32
Block limit	32

Registers

Registers/thread	65536
Register/block	65536
Block limit	32

Shared memory

Shared memory/block	98304
Block limit	32

Taula 6.1: GPU Quadro M4000 device limit.

6.1.1 Espai petit

La taula 6.2 mostra els recursos usats en la GPU quan s'executen les proves amb l'espai petit.

Profiling information (general)

	First	**Second**	**Third**	**Fourth**	**Fifth**
Grid size	[2,2,10]	[2,2,2]	[2,2,1]	[2,2,2]	[2,2,10]
Block size	[8,8,1]	[8,8,1]	[8,8,1]	[8,8,1]	[8,8,1]
Threads per block	[64]	[64]	[64]	[64]	[64]
Number of blocks	40	8	4	8	40

Profiling information (per SM)

	First	**Second**	**Third**	**Fourth**	**Fifth**
Occupancy	7,20%	2,40%	2,40%	2,40%	7,30%
Active Wraps	4,61	1,54	1,54	1,51	4,65
Active threads	64-192	0-64	0-64	0-64	192-256
Registers/Thread	26	20	26	32	32
Blocks	1-3	0-1	0-1	0-1	3-4

Execution time (ms)

	First	**Second**	**Third**	**Fourth**	**Fifth**
Sequential	0,101995				
Cuda	0,719094	1,49825	0,820506	1,05389	0,713408

Speedups

Cuda	0,141838202	0,068076089	0,12430744	0,09677955	0,14296868

Taula 6.2: Taula nvidia visual profiling (Grid petit).

En la taula es poden observar 4 blocs: informació general, informació per SM [ii], temps d'execució i speedups. El primer bloc mostra informació general sobre com està dimensionat el problema: mida del grid, mida del bloc, threads per bloc i número de blocs. La informació d'aquest apartat no és rellevant i bàsicament indica quanta feina hi ha.

El segon bloc ens indica la quantitat de recursos que s'estan fent servir en cada SM. Aquesta informació és de bastant interès, ja que ens diu quin ús dels recursos de la GPU s'estan utilitzant per realitzar els càlculs. Si analitzem les dades podem veure que el percentatge d'ocupació apenes arriba al **7,5%** per a les implementacions primera i quinta i al **2,5%** per a les implementacions segona, tercera i quarta. Si mirem els wraps es pot veure que en totes les implementacions es té menys de **5 wraps** actius d'un total de *64* que i pot haver-hi. També és interessant veure que el número de threads actius en el SM no supera els **256** threads d'un total de *2048*.

El tercer i quart blocs indiquen els temps de calcul i els speedups aconseguits en l'execució de la prova. Com es pot observar, els temps d'execució de les implementacions amb *cuda* està molt per sobre als temps de la versió seqüencial, fet que provoca l'obtenció d'uns speedups que apenes arriben a **0.15x**.

Amb aquests resultats es pot deduir de manera clara que quan el nombre de punts en l'espai és petit els resultats que s'obtenen són molt pitjors que en la versió seqüencial. Per una banda s'obtenen uns speedups de l'ordre de **10-15x** més lents, i per l'altra es pot veure que en dures feines s'arriba a utilitzar un **8%** dels recursos de la GPU.

6.1.2 Espai gran

La taula 6.3 mostra els recursos usats en la GPU quan s'executen les proves amb l'espai gran.

Profiling information (general)

	First	Second	Third	Fourth	Fifth
Grid size	[13,13,100]	[13,13,13]	[13,13,1]	[13,13,13]	[13,13,100]
Block size	[8,8,1]	[8,8,1]	[8,8,1]	[8,8,1]	[8,8,1]
Threads per block	[64]	[64]	[64]	[64]	[64]
Number of blocks	16900	2197	169	2197	16900

Profiling information (per SM)

	First	Second	Third	Fourth	Fifth
Occupancy	73,70%	93,26%	38,50%	91,20%	72,70%
Active Wraps	47,15	59,88	24,64	58,34	46,53
Active threads	1536-2048	1920-2048	768-832	1856-1920	1472-1536
Registers/Thread	26	20	26	32	30
Blocks	24-32	30-32	12-13	29-30	23-24

Execution time (ms)

	First	Second	Third	Fourth	Fifth
Sequential	102,655				
Cuda	16,4519	70,4858	70,6646	192,388	17,3046

[ii] Un SM o Streaming multiprocessor és la part de la GPU on s'executen els kernels. La GPU està formada per un array de SMs on cadascun d'ells conté: centenars de registres, memòria compartida, memòria constant, warp schedulers, cores d'execució, etc.

Speedups					
Cuda	6,239704837	1,456392635	1,452707579	0,533583176	5,932237671

Taula 6.3: Taula nvidia visual profiling (Grid gran).

En el primer bloc es pot veure com està dimensionat el problema. Ràpidament es pot observar que la mida del grid és més gran que en el primer cas i que la quantitat de feina ha realitzar arriba fins a **16900 blocs** a computar envers els **40 blocs** del primer cas.

En el segon bloc es pot veure que la quantitat de recursos que s'utilitzen en cada SM és elevat, fent ús fins a un **72-73%** en la primera i quinta implementació i fins a un **91-93%** en la segona i quarta. Mirant els warps es veu un ús de **58-59 warps** en les implementacions segona i quarta i de **46-47 wraps** en la primera i quinta. Mirant el nombre de threads actrius es pot veure, també, que aquest és elevat en tots els casos. En general es pot observar que quan el nombre de punts en l'espai és elevat l'ús de recursos de la GPU també ho és, per tant, és d'esperar que el rendiment millori.

El tercer i quart bloc ens serveixen per veure i validar que quan el nombre de punts en l'espai és gran el rendiment augmenta. Per una banda, si mirem els temps d'execució es pot veure que en la primera, segona, tercera i quinta implementació el temps d'execució és menor que el seqüencial. D'altra banda, es poden veure uns speedups de fins a **6x** en la primera i quina implementació i de **1,5x** per les implementacions segona i tercera. Aquí però, cal destacar les implementacions tres i quatre, ja que mirant les taules es veu que la implementació tres amb un **38%** d'ocupació es capaç de treure més rendiment que la implementació quatre amb un **91%** d'ocupació (veure punt 6.3).

6.2 Comparativa d'speedups

La figura 6.1 mostra una comparativa entre els speedups aconseguits en les proves realitzades amb un espai petit i un de gran.

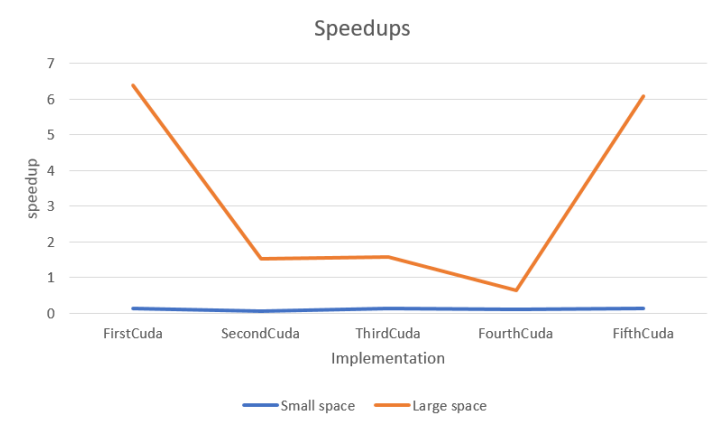

Figura 6.1: Comparativa de speedups entre un espai petit i un de gran.

En la figura es pot veure que en els cas de tenir un nombre petit de punts en l'espai el rendiment que s'obté és negatiu, mentre que quan es té un nombre elevant de punts en l'espai el rendiment augmenta entre **1-2x** en el cas de la segona i tercera implementació i fins a **6-7x** en les implementacions primera i quinta.

6.3 Observacions

És necessari indicar que en les proves realitzades la quantitat d'ocupació de la GPU està directament relacionada amb el número de blocs, el número de *threads* per bloc i el número de registres usats per cada *thread*. La manera amb que s'assignen els blocs a un SM es resumeix en el següents punts:

- Els blocs es guarden en una cua i es van assignant als diferents SMs a mesura que hi ha recursos disponibles.
- Cada SM pot tenir un màxim de blocs assignats a la vegada. En el cas de la *GPU Quadro M4000* és de 32.
- El nombre de blocs que s'assignen a un SM depèn directament del nombre de threads que tingui cada bloc i el nombre de registres que faci servir cada thread.
- Un bloc pot tenir un màxim de *1024* threads.
- Un SMs pot tenir un màxim de *2048* threads.

En el cas de la prova amb l'espai petit s'obté una ocupació molt baixa perquè es poden assignar tots els blocs a la GPU. El motiu és que hi ha més recursos que blocs. En aquest cas el baix rendiment pot ser degut al *overhead* entre la transferència de dades entre el host i la GPU, per tant, tot sembla indicar que **no és recomanable** l'ús de la GPU en espais que tinguin pocs punts.

Per al cas de l'espai gran l'ocupació és elevada perquè hi ha més blocs que recursos en la GPU, per tant, els SMs estaran tota l'estona amb una ocupació del *70-90%*. Aquí però, cal destacar els casos de la primera, tercera i quinta implementació que tenen una ocupació de *73%*, *38%* i *72%*. Teòricament, si hi ha més blocs que recursos, la intuïció ens pot fer creure, de manera errònia, que els SM haurien de tenir una ocupació del *90-100%* però, tal com s'ha dit l'ocupació depèn del nombre de blocs per SM, del nombre de threads per bloc i del nombre de registres que utilitza cada thread. El que pot succeir es que el percentatge que falta per arribar a utilitzar completament el SM no es pugui aconseguir perquè afegir un bloc més al SM significaria superar el nombre de blocs per SM, el nombre màxim de threads que pot tenir el SM o el màxim nombre de registres per SM.

Un altre punt a comentar és el cas de la tercera i quarta implementació amb l'*espai gran*. En aquests dos casos es pot veure que la implementació tres té una ocupació d'un *38,50%* i la implementació quatre té una ocupació del *91,20%*. A primera vista tot sembla indicar que la implementació quatre, al tenir més ocupació, hauria de tenir un rendiment millor però, si mirem els speedups es pot veure que la implementació tres és més ràpida.

Es pot concloure, doncs, que el percentatge d'ocupació pot ser un indicatiu del rendiment però, també és important veure com es reparteix la feina. A vegades pot ser interessant tenir menys blocs que facin més feina (tercera implementació amb l'espai gran) que tenir molts blocs que facin poca feina (quarta implementació amb l'espai gran). La idea que se'n pot treure és que a l'hora de desenvolupar i configurar un kernel s'ha d'intentar aconseguir un equilibri entre el número de blocs i la quantitat de feina que han de fer els threads de cada bloc per tal d'intentar aconseguir la maximitzar ocupació dels SMs.

6.4 Nombre de punts òptim

La figura 6.2 mostra una gràfica d'speedups segons el nombre de punts usats. Aquesta gràfica surt d'un estudi que té com a intenció esbrinar a partir de quants punts seria recomanable l'ús de la GPU per a obtenir una millora en el rendiment del calcul de les projeccions. Les dades associades a la gràfica es poden veure en la taula A.10 de l'apèndix.

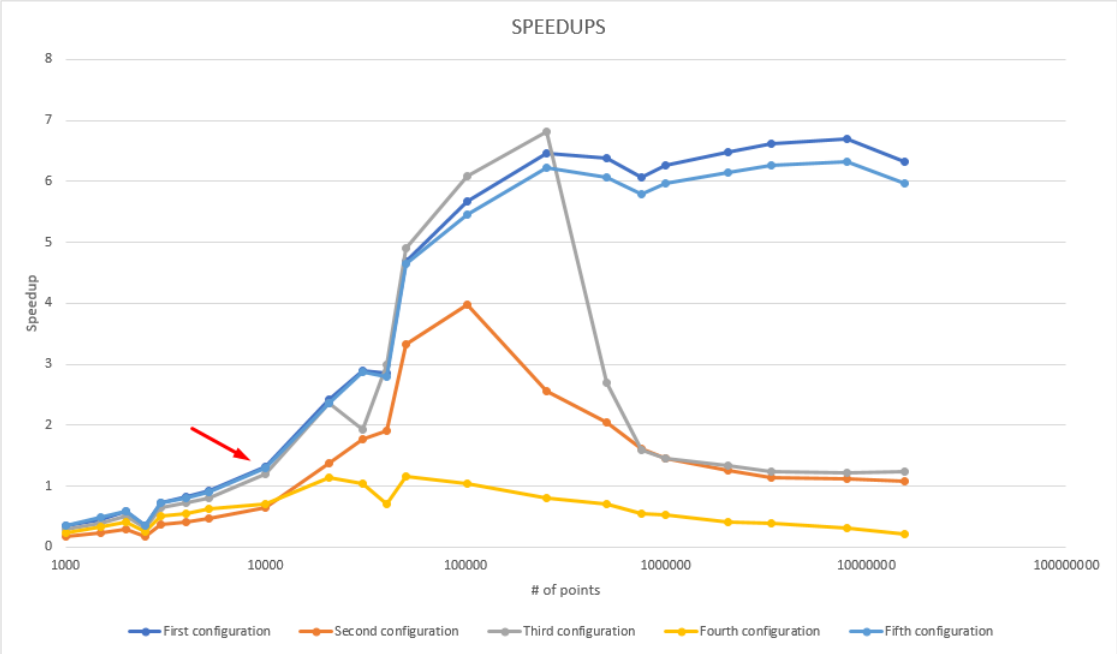

Figura 6.2: Speedups segons el nombre de punts de l'espai.

Analitzant la gràfica podem veure que aproximadament a partir de **10000 punts** es comença a obtenir una millora en el rendiment. Podem veure que les millors implementacions són la primera i la quinta que arriben fins a un speddups de **6-7x** i es mantenen estables. Cal destacar la implementació tres que escala molt be fins aproximadament els *500000 punts* i després cau en picat.

6.5 Mitjana de punts en els benchamarks

Quan es fa una simulació amb *PharmScreen* s'executa dues vegades la funció de projecció. La primera vegada es fa un prealineament de molècules que consisteix en processar les molècules de manera simple per descartar les pitjors i. La segona vegada es fa el processament de la resta de molècules. La idea és fer un prefiltratge amb una distància entre punts més gran (2 Å) per tal de descartar les molècules més dolentes i finalment realitzar els càlculs més precisos amb una distancia entre punts més petita (0.5 Å).

La figura 6.3 mostra el nombre mitja de punts que hi ha en el proces de prealineament (pre align) i el proces final (final align).

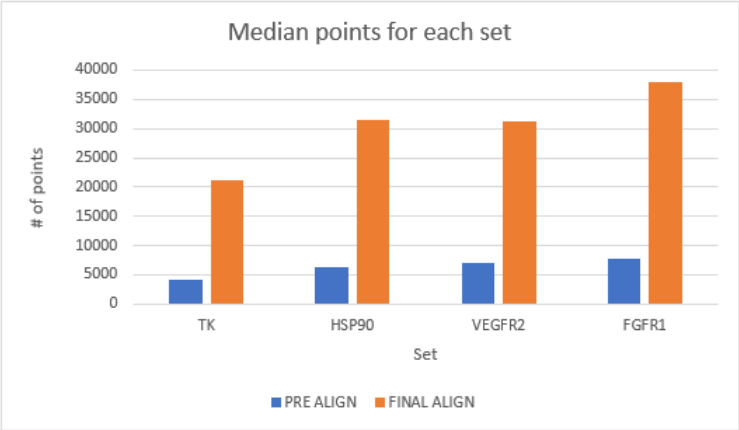

Figura 6.3: Mitjana de punts en els benchmarks.

Analitzant la gràfica es pot veure que en el cas del prealineament el nombre de punts mitjà que es fa servir està entre **4000-8000**. Pel que fa al calcul de l'alineament final es pot veure que hi ha entre **20000-37000** punts. Si es relacionen aquests resultats amb els resultats que es mostren en la figura 6.2 podem veure que per al cas del prealineamnet es perd rendiment amb un speedup aproximat de **0,5-0,8x**. Pel que fa al cas dels alineaments finals es pot veure una millora d'entre **1,5-3x** aproximadament.

6.6 Mitjana de projeccions en els benchamarks

Un altre aspecte a tenir en compte és el nombre de vegades que es crida la funció de projecció en el prealineament i l'alineament final. La figura 6.3 mostra el nombre mitja de vegades que s'executa la funció en cada cas.

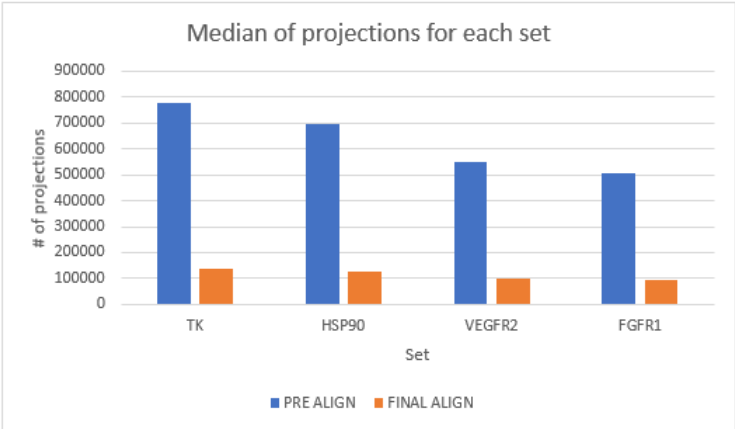

Figura 6.4: Mitjana de projeccions en els benchmarks.

Analitzant la gràfica podem veure que la funció es crida entre **500000-800000** vegades en el prealineament, mentre que en l'alineament final apenes arriba a **100000**. Aquest fet és d'**important rellevància**, ja que si es relacionen aquest resultats amb els resultats de les figures 6.2 i 6.3 es pot dir que el rendiment que es guanya en l'alineament final queda anul·lat pel rendiment que es perd en el prealineament.

6.7 Comentaris dels resultats

Amb l'estudi realitzat s'ha pogut observar que la qüestió important en les funcions de projecció és el nombre de punts en el grid. S'ha pogut veure que quan es fan els càlculs en un grid amb un nombre de punts petit el rendiment que se'n treu és negatiu, mentre que si els càlculs es fan amb un nombre elevat de punts s'aconsegueixen millores significatives.

D'altra banda, s'han analitzat els conjunts de molècules definits en els benchmarks i s'ha pogut observar que el nombre de punts varia entre 4000-8000 en el cas del prealineament i entre 20000-37000 en el cas de l'alineament final. Amb aquests resultats s'ha pogut observar que en el prealineament es perd rendiment, mentre que en l'alineament final se'n guanya.

Per últim, s'ha analitzat el nombre de vegades que es fan les operacions de prealineament i d'alineament final i s'ha observat que en el primer cas el nombre és molt elevat respecte del segon. Si es relacionen aquests últims resultats amb els resultats anteriors es pot concloure que la millora de rendiment que s'aconsegueix en l'alineament final queda anul·lat per la pèrdua que hi ha en el prealineament.

La conclusió que se'n pot treure d'aquest estudi està en sintonia amb els resultats que s'han obtingut en el capítol 5, és a dir, per al cas que s'està tractant **no s'aconsella l'ús de la tecnologia *Cuda***.

Capítol 7

Acceleració amb OpenMP

Després de veure que amb la tecnologia *Cuda* no és possible accelerar *PharmaScreeen*, vam decidir explorar la possibilitat de fer-ho amb OpenMP. Com en el cas de *Cuda*, s'ha fet ús dels benchmarks definits en el capítol 3 per veure l'impacte dels canvis en el rendiment.

En aquest cas però, en el moment d'accelerar amb *OpenMP* no s'ha fet ús dels resultats obtinguts en el *profiling* del punt 4, sinó que s'ha canviat de plantejament i s'ha fet des d'un punt de vista general.

Com en el punt 5.2, es presenten dues versions de l'algoritme:

- **Versió seqüencial**: és la versió original del software *PharmScreen*.
- **Versió OpenMP**: és la nova versió del software *PharmScreen* amb l'ús de la tecnologia *OpenMP*.

7.1 Versió seqüencial

De manera general el que fa *PharmScreen* és processar els conjunts de molècules que es volen provar contra N molècules de referència (veure punt 2.1.2). Per cada molècula de referència el *software* executa les funcions de prealineament i alineament final(veure punt 6.5).

L'esquelet amb pseudocodi de *PharmScreen* és el següent:

```
Main_function
{
    ...
    ref_molecules = N
    for i=0 to ref_molecules {
        ...
        preAling(...); //es fa el prefiltratge
        ...
        finalAlign(...) //es l'alineament final
        ...
    }
    ....
}
```

Figura 7.1: Pseudocodi PharmScreen.

El que fan les funcions *preAlign* i *finalAlgin* és superposar les molècules dels conjunts amb les molècules de referència. Un cop superposades s'alineen i es calcula el coeficient de *Tanimoto* per veure com de semblants són. En aquest procés és on es fa ús de les funcions de projecció explicades en el punt 5.1. La diferència entre les dues funcions és la precisió amb que es realitzen els càlculs. En el cas de *preAlign* es fan amb una distància entre els punts de l'espai de *2* Å i per al *finalAlign* es fan amb una distància de *0.5* Å.

La figura 7.2 mostra un exemple d'alineament fet amb *PharmScreen*.

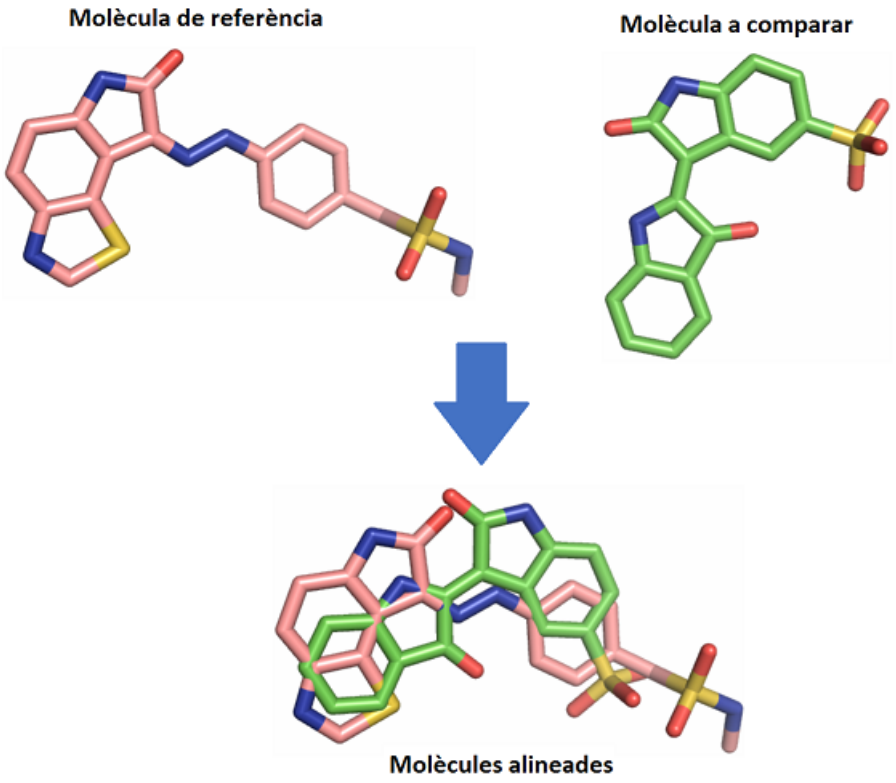

Figura 7.2: Alineament de molècules amb PharmScreen.

En la figura es pot veure com quedaria l'alineament entre una molècula de referència i una molècula del conjunt. El procés consisteix en agafar els centres de cada molècula, per exemple el centre geomètric, i superposar les molècules una a sobre de l'altra. Un cop superposades s'alineen i es calcula el coeficient de *Tanimoto*.

En la figura 7.3 es pot veure el pseudocodi de les funcions d'alineament *preAlign* i *finalAlign*.

```
Align_function(ref_molecule
)
{
    ...
    molecules = M
    for i=0 to molècules {
        ...
        readMolecule(...)
        //càlculs
        ...
    }
    ...
}
```

Figura 7.3: Pseudocodi funcions d'alineament.

Les funcions d'alineament reben com a paràmetre d'entrada la molècula de referència contra la que es volen comparar totes les molècules del conjunt de proves. La idea de la funció és simple, per a cada molècula del conjunt d'entrada es realitzen diferents operacions: superposició de les molècules, alineament de les molècules, calcul de coeficient de *Tanimoto*. Un cop realitzat l'alineament final s'obté com a resultat un fitxer amb un ranking de les millors molècules del conjunt -les més semblants a la molècula de referència- ordenades pel coeficient de *Tanimoto*.

7.2 Versió amb OpenMP

En el moment d'accelerar el software s'ha realitzat un anàlisis general de l'algoritme per tal d'identificar els millors punts on aplicar paral·lelisme. Concretament es va identificar una àrea potencialment candita: les funcions d'alineament.

Si es revisa el punt 7.1 es pot veure la figura 7.3 on s'explica com funcionen les funcions d'alineament. El que es fa és llegir una a una les molècules de cada conjunt i es realitzen els càlculs adients per tal d'obtenir els coeficients de *Tanimoto*. Analitzant l'algoritme ens vam adonar que en els càlculs no hi ha cap dependència, de manera que les molècules es poden processar paral·lelament i independentment entre elles.

La versió en *OpenMP* consisteix en adaptar les funcions d'alineament per realitzar els càlculs concurrentment. Això es plasma en el psudocodi de la figura 7.4.

```
Align_function(ref_molecule)
{
    ...
    molecules = M
    #pragma omp parallel {
        for i=0 to molècules {
            ...
            readMolecule(...)
            //càlculs
            ...
        }
    }
    ...
}
```

Figura 7.4: Pseudocodi funcions d'alineament amb OpenMP.

La idea consisteix en paral·lelitzar el bucle que itera per les molècules del conjunt. Al final s'obté una execució seqüencial fins arribar al bloc paral·lel on es bifurcara amb N *threads* que realitzen els càlculs concurrentment.

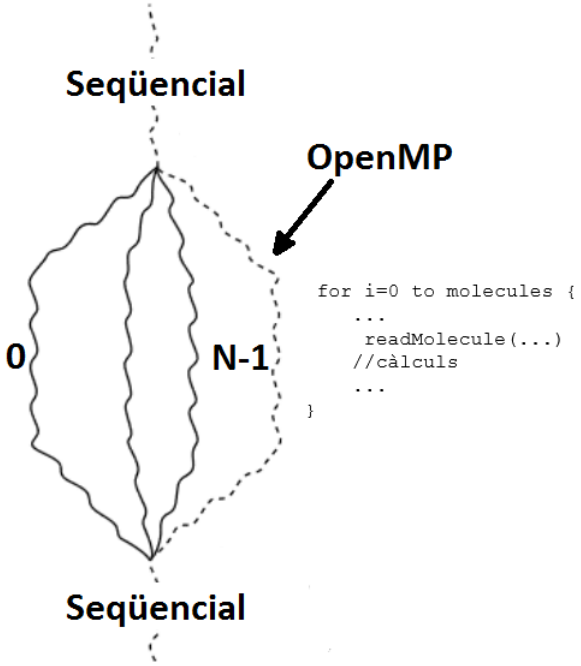

Figura 7.5: Execució amb OpenMP bucle d'alineament.

7.3 Resultats

En aquesta secció es presenten els speedups i les molècules per segon que s'obtenen amb l'ús de *OpenMP*. La secció està composta per tres subseccions corresponents a tres workstations: Pharmacelera, Amazon Web Service i Rutgers. Les característiques del maquinari es poden veure en l'apartat 2.5. Per obtenir un resultats més significatius s'ha seguit el següent criteri a l'hora d'agafar els resultats:

- S'ha fet 5 execucions.
- S'ha eliminat el mínim i el màxim resultats obtinguts.
- S'ha fet la mitjana aritmètica dels tres resultats.

7.3.1 Workstation Pharmacelera

Les figures 7.6 i 7.7 mostren els speedups i les molècules per segon que s'obtenen en la workstation de Pharmacelera després d'executar els benchamarks. Els resultats que es mostren són la mitjana obtinguda en les execucions de les tres mides, és a dir, és la mitjana obtinguda en les mides petita, mitjana i gran dels benchmarks. En l'apèndix B es mostren les figures B.1, B.3, B.5, B.7 i les taules B.1, B.3, B.5 i B.7 corresponents a les dades dels speedups per totes les mides dels conjunts, i les figures B.2, B.4, B.6 i B.8 i taules B.2, B.4, B.6 i B.8 corresponents a les dades de les molècules per segon per totes les mides dels conjunts.

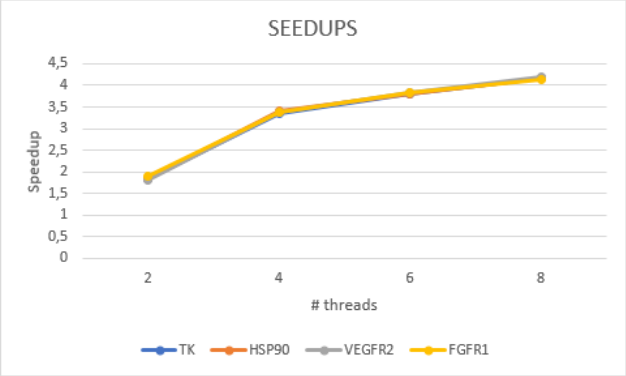

Figura 7.6: Speedups OpenMP workstation Pharmacelera.

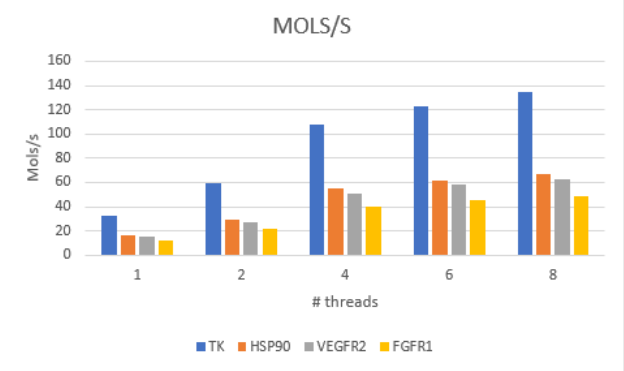

Figura 7.7: Molècules per segon OpenMp workstation Pharmacelera.

Analitzant les gràfiques es pot observar el següent:

- Es veu que els speedups que s'obtenen són pràcticament els mateixos per a tots els conjunts de dades.
- S'observa que amb dos i quatre *threads* l'escalat és quasi lineal al nombre de *threads*, mentre que per a sis i vuit *threads* no és així.
- Es pot observar que per al conjunt *tk* el software és capaç de processar entre 30-130 molècules per segon a mesura que s'augmenta el nombre de *threads*, mentre que per als altres conjunts el nombre es redueix entre 15-60 molècules per segon.

7.3.2 Workstation Amazon Web Services

La figures 7.8 i 7.9 mostren els speedups i les molècules per segon que s'obtenen en la workstation de Amazon Web Services. Els resultats que es mostren són la mitjana obtinguda en les execucions de les tres mides dels benchmarks. En l'apèndix B es mostren les figures B.9, B.11, B.13, B.15 i les taules B.9, B.11, B.13 i B.15 corresponents a les dades dels speedups per totes les mides dels conjunts, i les figures B.10, B.12, B.14 i B.16 i taules B.10, B.12, B.14 i B.16 corresponents a les dades de les molècules per segon per totes les mides dels conjunts.

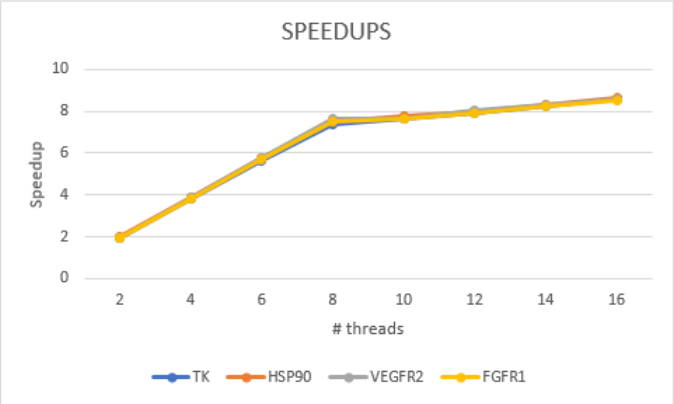

Figura 7.8: Speedups OpenMP workstation Amazon Web Services.

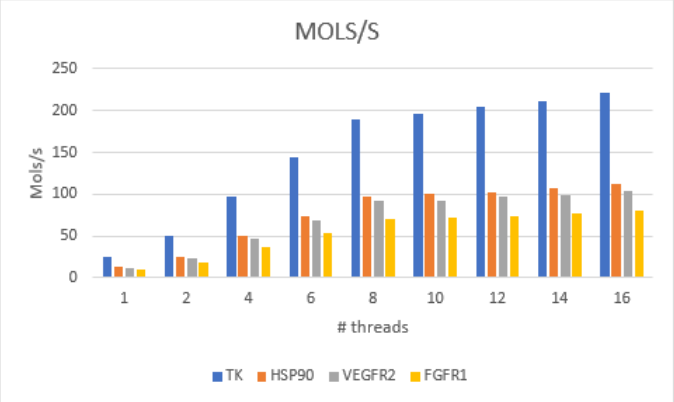

Figura 7.9: Molecules per segon OpenMP workstation Amazon Web Services.

Analitzant les gràfiques es pot observar un comportament molt semblant que amb la workstation de *Pharmacelera*:

- Es veu que els speedups que s'obtenen són pràcticament els mateixos per a tots els conjunts de dades.
- En aquest cas l'escalat es quasi lineal entre dos i vuit *threads*, mentre que entre vuit i setge no és així.
- En aquest cas també s'observa un processat d'entre 30-190 molècules per segon el en conjunt *tk*, mentre que per als altres conjunts el nombre es redueix entre 15-100 molècules per segon.

7.3.3 Workstation Rutgers

La figures 7.10 i 7.11 mostren els speedups i les molècules per segon que s'obtenen en la workstation de Rutgers. Els resultats que es mostren són la mitjana obtinguda en les execucions de les tres mides dels benchmarks. En l'apèndix B es mostren les figures B.17, B.19, B.21, B.23 i les taules B.17, B.19, B.21 i B.23 corresponents a les dades dels speedups per totes les mides dels conjunts, i les figures B.18, B.20, B.22 i B.24 i taules B.18, B.20, B.22 i B.24 corresponents a les dades de les molècules per segon per totes les mides dels conjunts.

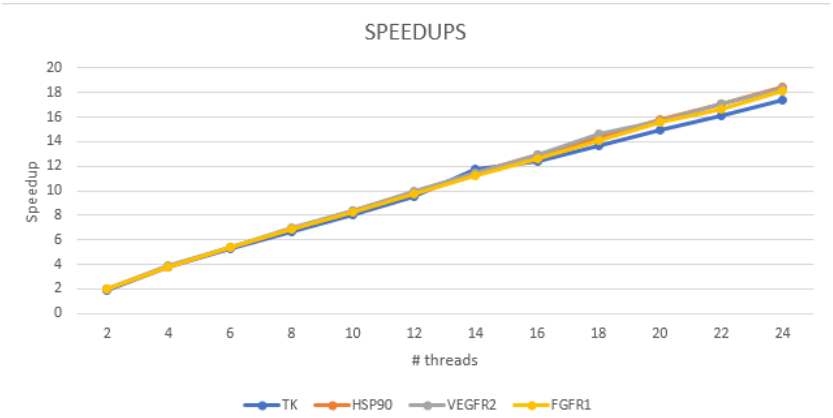

Figura 7.10: Speedups OpenMP workstation Rutgers.

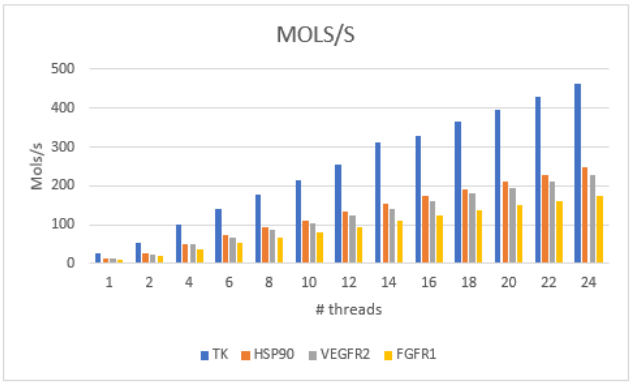

Figura 7.11: Molècules per segon OpenMP workstation Rutgers.

Observant les gràfiques es pot observar un comportament com el de les màquines de Pharmacelera i Amazon Web Services:

- Es veu que els speedups que s'obtenen són pràcticament els mateixos per a tots els conjunts de dades.
- En aquest cas però, l'escalat és lineal en tots el nombres de threads.
- Es pot observar el mateix comportament per al nombre de molècules per segon en el conjunt *tk*, és a dir, amb aquest conjunt es poden processar moltes més molècules que en els altres tres.

7.4 Comparativa dels resultats

La figures 7.12 i 7.13 mostren una comparativa entre els speedups i les molècules per segon obtinguts en les tres workstations en que s'han fet les proves. Els speedups i les molècules per segon que es mostren fan referència a la mitjana global que s'aconsegueix en cada màquina, és a dir, per cada conjunt de dades es calcula la mitjana entre les tres mides i un cop es tenen es calcula la mitjana dels 4 conjunts.

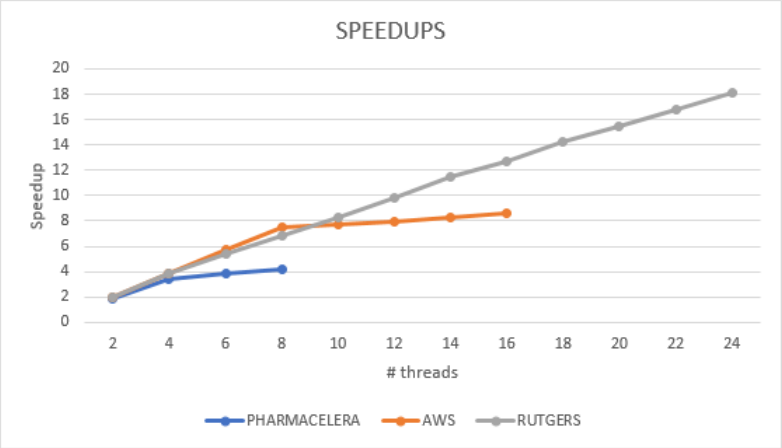

Figura 7.12: Comparativa speedups entre workstations.

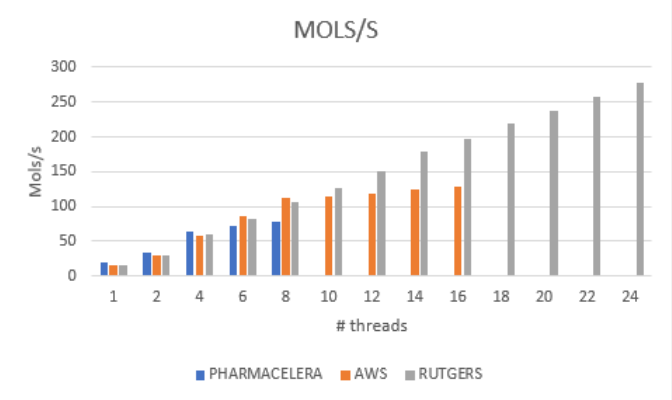

Figura 7.13: Comparativa molècules per segon entre workstations.

Analitzant la gràfica d'speedups es pot veure que el software escala diferent en cada màquina. Per exemple, si es miren els speedups amb 6 i 8 threads, es pot veure que per a les workstations de Amazon i Rutgers l'escalat és quasi lineal al nombre de *threads*, mentre que per a la màquina de Pharmacelera no és així. Passa el mateix si es miren els speedups amb 10, 12, 14 i 16 *threads* entre les màquines de Amazon i Rutgers.

El comportament és semblant si es mira la gràfica de molècules per segon. Es pot veure que amb 6 i 8 *threads* el nombre de molècules per segon és millor en les màquines de Amazon i Rutgers i que entre 10-16 *threads* Rutgers és millor que Amazon. Un punt a destacar és el cas de 4 *threads*, si es miren les dues gràfiques es pot veure que la màquina de Pharmacelera té un escalat menor que Amazon i Rutgers però en canvi es capaç de processar més molècules per segon. Bàsicament això es degut a la velocitat de les *CPU* en cada màquina.

A primera vista el comportament que es veu en les gràfiques pot sembla estrany, ja que en principi es pot esperar que amb el mateix binari l'escalat i el nombre de molècules processades hauria de ser molt semblant però, tal com es veu en les figures no és així. No obstant, si s'analitzen en detall les característiques de cada màquina en el punt 2.5 es pot veure el següent:

- La màquina de Pharmacelera disposa d'una CPU amb 4 cores *multithread* capaç d'executar 8 *threads*.
- La màquina de Amazon disposa d'una CPU amb 8 cores *multithread* capaç d'executar 16 *threads*.
- La màquina de Rutgers disposa de dues CPUs amb 12 cores *multithread* capaç d'executar un total de 48 *threads*.

El que esta passant doncs, és que quan s'executen les simulacions en les màquines de Pharmacelera i Amazon amb un nombre de *threads* més gran que el nombre de cores, **en cada core hi ha dos *threads*** que han de compartir recursos. D'altra banda, en el cas de Rutgers **cada *thread* s'executa en un core diferent** sense la necessitat de compartir els recursos.

7.5 Comentaris dels resultats

Després d'accelerar *PharmScreen* amb *OpenMP* s'han pogut observar uns resultats molt bons arribant a obtenir uns speedups de fins a **4x**, **8x** i **18x** en les *workstations* de Pharmacelera, Amazon i Rutgers respectivament. Per a ser sincer, els resultats obtinguts són molt millors del que ens esperaven i alhora són molt esperançadors per continuar investigant en aquesta línia de treball i intentar treure una mica més de rendiment en el *software*. Un *learning* important que se'n pot treure de l'estudi és la capacitat d'escalatge del software que **escala linealment** al nombre de cores amb que s'executa.

Pel que fa al nombre de molècules que es processen, és d'especial importància destacar el cas del conjunt de molècules *TK*. Es pot veure que en totes tres màquines el software és capaç de processar quasi el doble de molècules que en els altres tres conjunts. Aquest comportament és degut al nombre d'àtoms de la molècula i a la distribució que tenen en l'espai. Si es revisen les figures 3.1, 3.2, 3.3 i 3.4 del punt 3.1 es pot veure que les molècules del conjunt *TK* són més compactes, és a dir, els àtoms estan més junts entre ells, de manera que l'espai tridimensional que es computa és més petit i conté menys punts que en els altres casos on les molècules són més estirades i els àtoms estan més separats.

Capítol 8

Vectorització Mopac

Quan es parla de vectorització es refereix al desenrotllat (*unrolling* en anglès) d'un bucle combinat amb la generació de paquets d'instruccions *SIMD*[i] pel compilador. Com que les instruccions *SIMD* operen sobre un conjunt de dades, el bucle es pot executar de manera més eficient. Normalment el que es coneix com autovectorització és quan el compilador és capaç d'identificar i optimitzar els bucles automàticament sense la intervenció del programador.

Avui en dia les CPUs són processadors altament paral·lels amb diferents nivells de paral·lelisme. Es pot trobar el paral·lelisme en qualsevol lloc, des de les les unitats d'execució paral·leles amb els cores de la CPU i el conjunt d'instruccions *SIMD* fins a l'execució paral·lela en múltiples *threads*. L'ús del conjunt d'instruccions *SSE/AVX* és el que es coneix com a vectorització. En ciències de la computació el proces de convertir un algoritme des de la seva implementació escalar, que fa una operació d'un sol parell d'operands a la vegada, a un procés de vector on una sola instrucció pot referir-se a un vector (sèrie de valors adjacents) es diu vectorització. Les instruccions *SIMD* operen sobre un conjunt d'elements en una sola instrucció i fan ús de registres *SIMD* de 128/256 bits en coma flotant del processador.

Per a entendre aquesta idea ens podem imaginar la suma de dos vectors, tal com es veu en la figura 8.1.

```
for (i=0; i<=MAX; i++)
    c[i] = a[i] + b[i];
```

Figura 8.1: Suma de dos vectors [37].

Si la vectorització no està activada, en el moment de compilar el compilador pot fer una cosa semblant al de la figura 8.2.

[i]En computació, SIMD (Single Instruction, Multiple Data) és una tècnica usada per aconseguir paral·lelisme a nivell de dades. Les instruccions SIMD consisteixen en instruccions que apliquen una mateixa operació sobre un conjunt de dades.

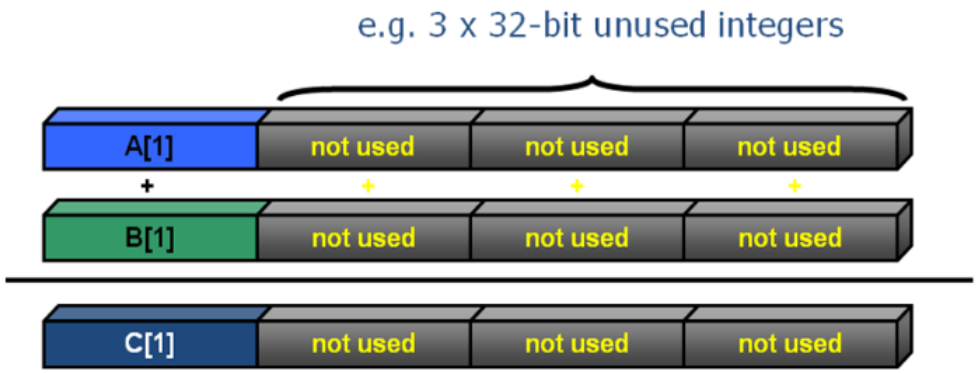

Figura 8.2: Registres usats sense vectorització [37].

La figura ens indica que si no fem ús de la vectorització estem desaprofitant tres registres *SIMD* i el bucle s'executara seqüencialment sumant els elements del vector d'un en un. En canvi, si s'usa la vectorització, en el moment de compilar el compilador usara els tres registres addicionals i s'obtindrà un codi que realitzara la suma de quatre elements en una sola instrucció (figura 8.3).

Figura 8.3: Registres usats amb vectorització [37].

8.1 Vectorització vs OpenMP

Una pregunta obvia que ens podem fer és: per què la vectorització i no OpenMP? La resposta aquesta pregunta be donada, principalment, pel fet que ara mateix el software *PharmScreen*, tal com em vist en el capítol 7, ja fa ús de *OpenMP*, de manera que si si executem *PharmScreen* amb varis *threads* això implicaria tenir un doble nivell de paral·lelisme que creiem que podria afectar negativament.

Quan s'executa *Mopac*, el que es fa és executar un proces fill mitjançant la instrucció *fork*. El problema que veiem és que quan executem *PharmScreen* amb N *threads*, aquest N *threads* fan un *fork* i es creen N processos intendents entre ells. D'aquesta forma, creiem que l'ús de *OpenMP* en aquesta situació no seria recomanable ja que, per una banda no es disposen de *threads* lliures per poder executar-ho, i per l'altra aplicar un segon nivell de paral·lelisme faria que el rendiment baixes molt.

8.2 Anàlisi amb Intel Advisor

Per tal d'identificar possibles punts on aplicar millores i aprofitar l'autovectorització s'ha fet servir l'eina *Advisor* d'Intel. Aquesta eina és molt semblant a l'eina de *VTunes*, però en aquest cas el que fa és buscar bucles que siguin susceptibles de millora per usar vectorització.

Les figures 8.4, 8.5, 8.6 i 8.7 mostren els anàlisis obtinguts executar cadascuna de les quatre molècules de prova que s'han fet servir en el profiling del punt 4.2.

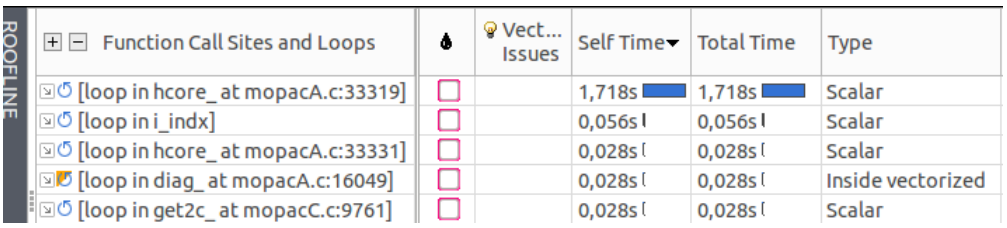

Figura 8.4: Anàlisi primera molècula amb Intel Advisor.

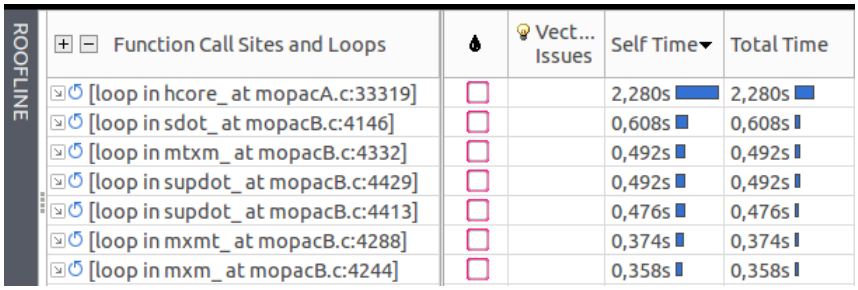

Figura 8.5: Anàlisi segona molècula amb Intel Advisor.

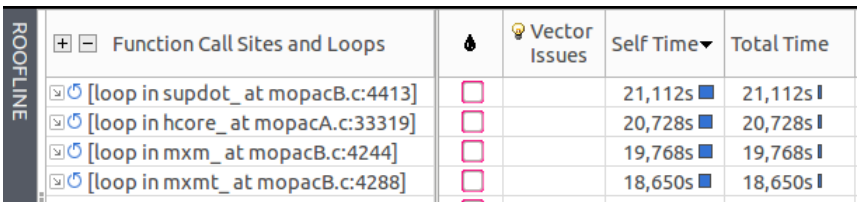

Figura 8.6: Anàlisi tercera molècula amb Intel Advisor.

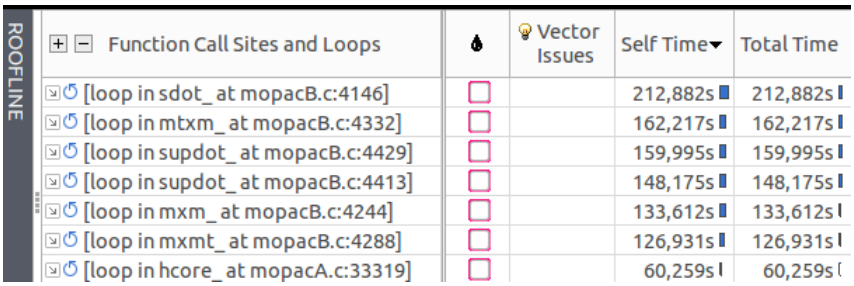

Figura 8.7: Anàlisi quarta molècula amb Intel Advisor.

L'anàlisi fet amb *Intel Advisor* ens indica que en les funcions *hcore_*, *sdot_*, *supdot_* , *mtxm_*, *mxm_* i *mxmt_* hi han bucles que gasten una gran quantitat de temps i que es podrien modificar per tal d'aprofitar l'autovectorització. Si comparem els resultats d'aquest anàlisi amb el de l'eina *VTunes* (punt 4.2) podem veure que en els dos casos apareixen les mateixes funcions.

L'eina *Advisor* també indica quin és el principal motiu del mal rendiment dels bucles i les possibles millores que es poden aplicar. En aquest cas el principal problema que hi ha és que l'accés a les dades en memòria no és continu.

8.3 Solució i resultats

Un cop identificat el problema, hem analitzat les funcions i em vist que la major part operen sobre matrius de dades fent sumes, restes i productes. Podem imaginar que tenim el codi de la figura 8.8 que fa una suma de dues matrius.

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}
```

Figura 8.8: Codi suma de dues matrius.

En aquest codi, per cada fila es sumen els elements de cada columna de les matrius A i B i es guarda en la matriu C. La idea, doncs, és aprofitar la vectorització per que el bucle intern es realitzi amb instruccions *SIMD* i es millori el rendiment. Des d'un punt de vista de memòria això és possible perquè les dades d'una matriu es guarden per files. Per exemple, una matriu *NxN*

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{pmatrix}$$

queda guardada en memòria com s'indica en la figura 8.9.

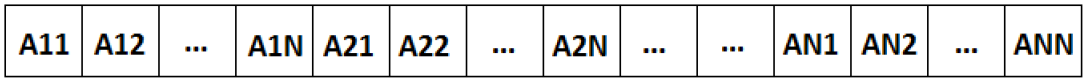

Figura 8.9: Matriu en memòria.

Si tenim dues matrius A i B, fent ús de la vectorització la suma es podria fer amb instruccions *SIMD* i quedaria de la següent manera:

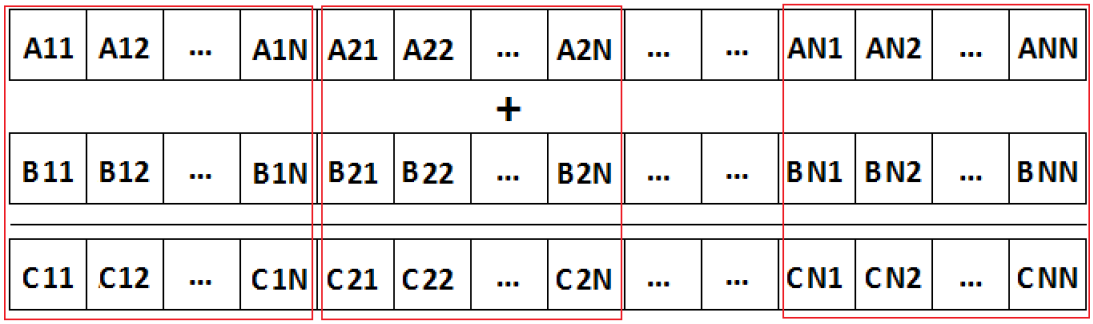

Figura 8.10: Suma matrius representació en memòria.

En la figura els quadres vermells representen cada fila de la matriu iterada fent ús dels registres SIMD. El nombre d'iteracions per processar tota una fila dependrà del tipus de dades que s'esta fent servir, *int* o *double*, i del nombre de bits reservats per als registres en la CPU. Per exemple, en una CPU que permet instruccions *AVX2* hi ha 256 bits reservats, de manera que es pot processar la suma de 8 *int* a la vegada -un int equival a 32 bits- o la suma de 4 doubles a la vegada -un double equival a 64 bits.

En el cas dels bucles de les funcions de *Mopac* però, les dades de les matrius no estan guardades per files, sinó per columnes (figura 8.11).

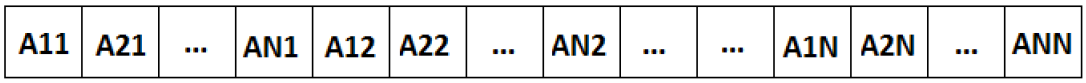

Figura 8.11: Dades en memòria per columnes.

El problema és que en el codi s'intenta accedir a les dades per files, però en memòria estan guardades per columnes, de manera que l'accés no és continuo i la vectorització no es pot dur a terme. En un primer moment no sabíem quin era el motiu pel qual s'accedia d'aquesta manera a les dades, però investigant ens vam adonar que en el llenguatge *Fortran* les dades es guarden per columnes, de manera que al convertir el codi a C l'accés queda així.

Aquest fet ens ha portat molts problemes i fins al moment ens ha impedit arreglar la major part dels bucles per aprofitar la vectorització. Primer vam intentar canviar l'ordre dels bucles per veure si el problema es solucionava, però al fer-ho els resultats de les simulacions eren incorrectes perquè si es modifica el patró de memòria en una funció hi ha el problema que les següents funcions que fan ús de les dades les obtenen en un ordre incorrecte.

No obstant això, encara vam poder trobar un bucle en que si es podia invertir l'ordre d'accés i aprofitar la vectorització. El bucle en qüestió es troba en la funció *hcore_* i el que fa és inicialitzar una matriu a zero en que no importa amb quin ordre es fa.

La figura 8.12 mostra els speedups que s'aconsegueixen quan s'inverteix l'ordre del bucle de la funció *hcore_* i s'aprofita la vectorització amb *AVX2*.

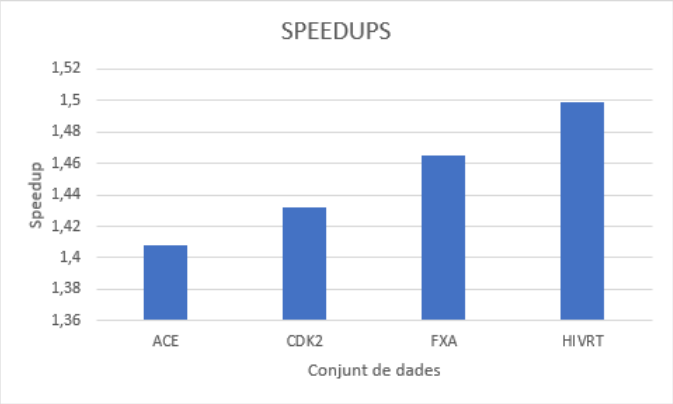

Figura 8.12: Speedups mopac.

Si s'analitza es pot veure que amb aquest canvi s'aconsegueix entre **1.4x** i **1.5x** de millora quan s'executen els benchamarks definits en el punt 3.2. En principi aquest speedups poden semblar poca cosa però en el moment de fer simulacions aquest **40% de millora** en el rendiment es nota molt.

8.4 Comentaris dels resultats

De l'estudi realitzat s'han pogut detectar diferents bucles en el codi de *Mopac* que són potencialment candidats a vectoritzar. El problema però, és el patró d'accés a les dades amb que es realitzen. Tal com s'ha indicat, el software de *Mopac* que fem servir es una traducció del software original amb Fortran a C i amb aquest llenguatge les dades de les matrius es guarden en memòria per columnes, fet que ens ha impedit la vectorització de la major part dels bucles. No obstant això, encara vam poder vectoritzar un bucle i obtenir un rendiment d'entre 1.4x-1.5x.

Pel que fa als altres bucles s'està treballant amb una solució que ens permeti aprofitar la vectorització. La idea és modificar les part del codi de *Mopac* que facin ús de les matrius per que tots els càlculs es facin amb l'ordre d'accés a dades correcte però, aquesta és una tasca que requereix de molt de temps i que no té cabuda en aquest treball.

Capítol 9

Anàlisis Econòmic

9.1 Valoració econòmica del projecte

Es presenta un anàlisis de costos dividit en tres blocs:

- Costos de programari
- Costos de hardware
- Costos de desenvolupament

9.1.1 Costos de programari

Quan es parlar de costos de programari es fa referència ales llicencies de programari que s'han pagat per a dur a terme el desenvolupament del treball.

El software usat és:

Software	**Unitats**	**Euros**
Latex	1	0
LibreOffice Calc	1	0
Netbeans	1	0
GanttProject	1	0
Nvidia visual profiling	1	0
Ubuntu Server 14.04.3 LTS	1	0
Llicencia Intel VTune	1	899
Llicencia Intel Advisor	1	1600
Llicencia PyMol	1	0
Total	-	**2500**

Taula 9.1: Taula de costos de programari.

Els costos de programari són de **2500 euros**.

9.1.2 Costos de hardware

Quan es parla de costos de hardware es fa referència a tot el hardware necessari per al desenvolupament del treball. La taula 9.2 mostra les despeses fetes en *Pharmacelera* per a disposar

del hardware necessari per poder desenvolupar el treball.

Component	**Unitats**	**Preu Euros**
Workstation Pharmacelera	1	700
Nvidia Quadro M4000	1	1000
Amazon Web Services5	500h	300
Total	-	**2000 euros**

Taula 9.2: Taula de costos del hardware.

El preu aproximat de hardware és de **2000 euros**.

9.1.3 Costos de desenvolupament

Quan es parla de costos de desenvolupament es fa referència als costos humans que es fan servir per portar endavant un projecte. En el desenvolupament d'un projecte intervenen diferents rols: project managers, analistes, programadors seniors. Cadascun d'aquests rols té un preu brut a l'hora. Una taula aproximada del preu d'hora segons el rol pot ser:

Rol	**Preu Euros/h**
Project Manager(P.M)	50
Analista(A)	40
Programador Senior(P.S)	30

Taula 9.3: Taula dels preus de mà d'obra segons el rol

La següent taula mostra un desglòs dels preus de les tasques realitzades segons el rol necessari per portar-les a terme.

Concepte	**A.**	**P.S.**	**P.M.**	**Euros**
Anàlisis i disseny global del TFM	15	0	15	1350
Preparació de l'entorn de treball	0	15	0	450
Preparació informe Previ	0	0	25	1250
Benchmarks	5	15	0	650
Profiling	5	15	0	650
Acceleració amb cuda	50	150	0	6500
Estudi del nombre de punts del grid	25	70	0	3100
Acceleració amb OpenMP	30	100	0	4200
Vectorització Mopac	20	50	0	2300
Documentació memòria	20	50	60	5300
Preparar defensa	0	0	20	1000
Total	-	-	-	**26750**

Taula 9.4: Preu de les tasques segons el rol.

Aproximadament el preu en els recursos humans el projecte ascendeix a uns **26750 euros**.

9.1.4 Preu total aproximat

El preu total del treball realitzat és la suma dels preus de software, hardware i de desenvolupament. Aproximadament, el preu total del treball ascendeix a uns **31250 euros**.

Capítol 10

Valoracions i treballs futurs

10.1 Valoració personal

En general estic content amb la feina realitzada, crec que tot el que s'ha fet encaixa perfectament en els estudis de màster que s'avaluen. Durant el transcurs del treball s'han estudiat i s'han fet ús de diferents tecnologies en l'àmbit de les ciències de la computació: sistemes paral·lels en GPU, sistemes paral·lels de memòria compartida i sistemes *SIMD*.

Un punt a tenir en compte és el marc general de treball en el camp del *I+D* com és la investigació en el descobriment de noves molècules per al desenvolupament de nous medicaments. Crec que que és d'especial rellevància remarcar aquest fet i crec que és important subratllar la tasca realitzada en l'estudi -a nivell bàsic- del comportament de les molècules.

En quan als resultats obtinguts, val a dir que la part de *Cuda* m'ha decepcionat una mica. Quan vaig arribar a *Pharmacelera* ho vaig fer amb moltes ganes i pensava que podríem obtenir uns grans resultats en aquest punt però, tal com s'ha vist no ha sigut així. No obstant, no tot són males noticies, ja que en l'estudi fet amb OpenMP s'han obtingut uns resultats molt bons, d'una banda els speedups arriben fins a **4x**, **8x** i **18x**, i de l'altra hem pogut esbrinar que el software escala linealment al nombre de cores amb que s'executa.

Pel que fa a la part de vectorització de *Mopac* s'ha pogut obtenir un speedeup de **1.4x**. En general estic content amb el resultat, ja que és la primera vegada que he treballat amb aquesta tecnologia i obtenir resultats positius que ens ajuden ens la feina del dia a dia sempre és reconfortant.

En quant als objectius que es buscaven en el treball, crec que s'han complert: s'ha aconseguit millorar el rendiment de *PharmScreen* amb la tecnologia *OpenMP*, he millorar molt els meus coneixements en investigació i en les tecnologies usades i el més important, a traves de l'experiència he pogut veure amb claredat que no totes les tecnologies funcionen be a l'hora d'accelerar algun software, tal i com s'ha vist en els capítols 5 i 6 d'acceleració amb *Cuda*.

10.2 Futures línies de treball

PharmScreen és un eina que encara té un gran recorregut pel davant i que té un gran potencial de creixement. Si tenim en compte que l'eina té aproximadament dos anys de vida, es pot dir de manera clara que està creixent i evolucionant molt ràpid. Actualment ja fa ús de la implementació d'*OpenMP* que s'ha desenvolupat i s'esta fent ús dels estudis realitzats amb GPU i vectorització per intentar millorar més.

Les següents passes a seguir estan emmarcades en el continu estudi i investigació de nous dissenys que ens permetin realitzar càlculs de CPU i GPU al mateix temps de manera eficient, la cerca d'alternatives per augmentar l'efectivitat de l'ús de la tecnologia *OpenMP*, veure el comportament del software amb l'última versió de *Xeon Phi KN*L que permet un nivell de paral·lelisme elevat i ofereix una vectorització amb 512 bits, millorar l'eficiència dels bucles fent ús d'instruccions *SSE/AVX*, aprofitar el desplegament de l'eina en *Amazon Web Services* per poder implementar mecanismes de paral·lelisme com *MPI* o *Hadoop* i la possibilitat d'investigar amb sistemes *FPGA*.

Com es pot veure, l'eina ofereix moltíssimes possibilitats per al futur, i des de *Pharmacelera* hem agafat tot la feina realitzada en aquest treball com a punt de partida per continuar evolucionant.

Apèndix

Apèndix A

Execució amb Cuda

Aquest apèndix conté totes les figures i taules de dades relacionades amb les execucions en Cuda que no tenen cabuda en el contingut del treball, ja sigui perquè no són rellevants o per tal de no repetir informació.

A.1 Figures

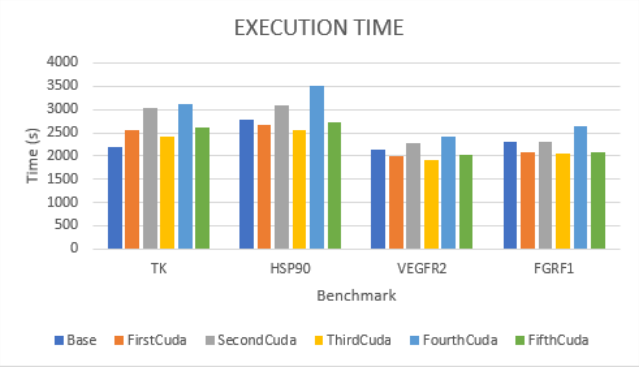

Figura A.1: Temps d'execució (medium benchmark).

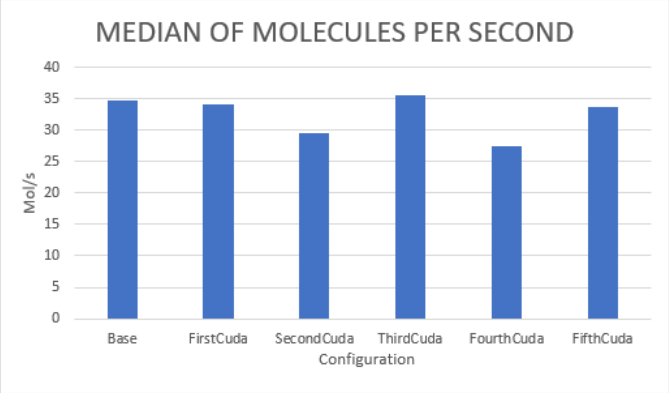

Figura A.2: Mitjana de molècules per segon (medium benchmark).

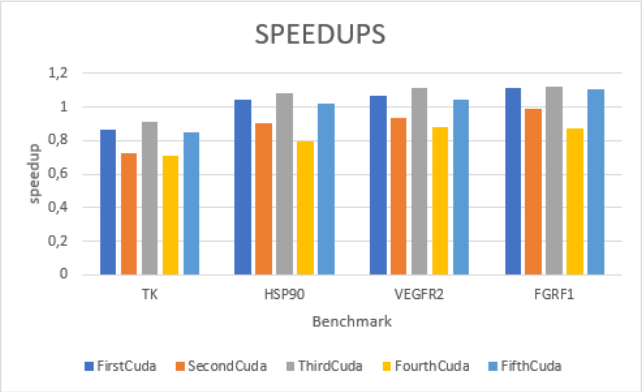

Figura A.3: Speedups (medium benchmark).

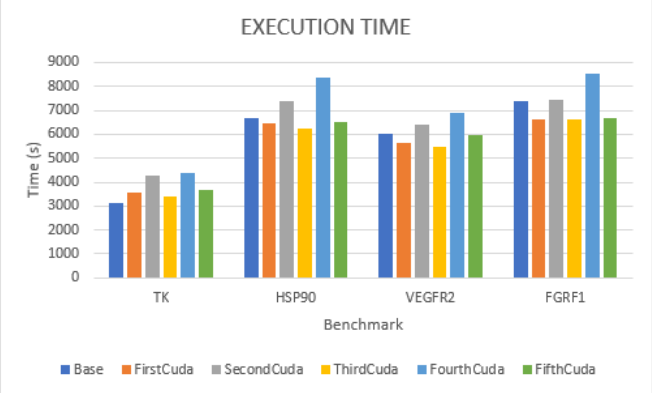

Figura A.4: Temps d'execució (large benchmark).

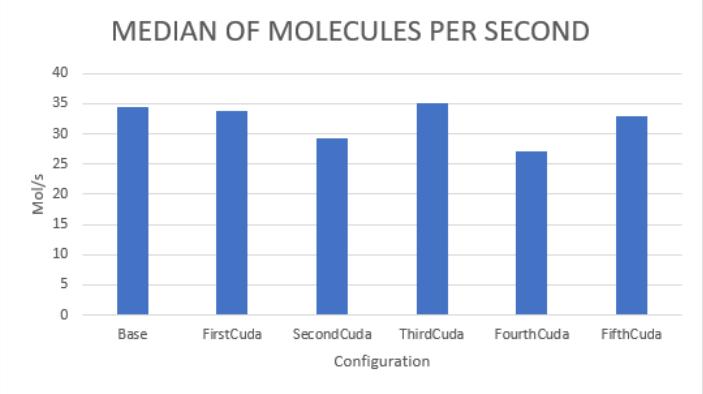

Figura A.5: Mitjana de molècules per segon (large benchmark).

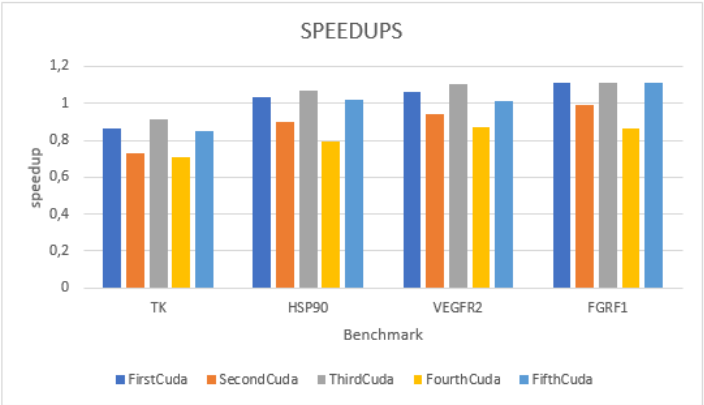

Figura A.6: Speedups (large benchmark).

A.2 Taules

	TK	HSP90	VEGFR2	FGFR1
Base	336,43	483	335,7	368,84
FirstCuda	385,61	462,63	315,83	328,8
SecondCuda	462,12	537,96	371,29	374,33
ThirdCuda	390,99	451,91	315,8	320,17
FourthCuda	475,49	607,7	378,95	425,28
FifthCuda	390,09	468,48	325,13	334,3

Taula A.1: Temps d'execució per a mida petita.

	TK	HSP90	VEGFR2	FGFR1	MEDIAN
Base	59,44773058	28,98550725	31,27792672	21,68962152	35,35019652
FirstCuda	51,86587485	30,26176426	33,24573346	24,33090024	34,9260682
SecondCuda	43,27880204	26,02423972	28,2797813	21,37151711	29,73858504
ThirdCuda	51,15220338	30,97961984	33,2488917	24,9867258	35,09186018
FourthCuda	42,06187302	23,03768307	27,70814092	18,81113619	27,9047083
FifthCuda	51,27021969	29,88387978	32,2947744	23,93060126	34,34486878

Taula A.2: Molècules per segon per a mida petita.

	TK	HSP90	VEGFR2	FGFR1
FirstCuda	0,872461814	1,044030867	1,062913593	1,121776156
SecondCuda	0,728014369	0,89783627	0,904145008	0,985333796
ThirdCuda	0,860456789	1,068796884	1,063014566	1,152012993
FourthCuda	0,707543797	0,794800066	0,885868848	0,867287434
FifthCuda	0,862442001	1,030993852	1,032510073	1,103320371

Taula A.3: Speedups per a mida petita.

	TK	HSP90	VEGFR2	FGFR1
Base	2203	2782,2	2125,6	2300,6
FirstCuda	2545	2672,5	1990,7	2069,7
SecondCuda	3023,6	3081	2272,3	2318
ThirdCuda	2411,6	2568,2	1908,6	2043,4
FourthCuda	3116,5	3496,7	2409,9	2644,3
FifthCuda	2601,8	2716	2032,2	2076,6

Taula A.4: Temps d'execució per a mida mitjana.

	TK	HSP90	VEGFR2	FGFR1	MEDIAN
Base	59,01044031	30,19193444	28,22732405	21,73346084	34,79078991
FirstCuda	51,0805501	31,43124415	30,14015171	24,15809054	34,20250913
SecondCuda	42,99510517	27,26387537	26,40496413	21,57031924	29,55856598
ThirdCuda	53,90612042	32,70773304	31,43665514	24,46902222	35,6298827
FourthCuda	41,71346061	24,02264993	24,89729864	18,90859585	27,38550126
FifthCuda	49,96540856	30,92783505	29,52465309	24,07781951	33,62392905

Taula A.5: Molècules per segon per a mida mitjana.

	TK	HSP90	VEGFR2	FGFR1
FirstCuda	0,865618861	1,041047708	1,067765108	1,111562062
SecondCuda	0,728601667	0,9030185	0,935439863	0,992493529
ThirdCuda	0,91350141	1,083326844	1,113695903	1,12586865
FourthCuda	0,706882721	0,795664484	0,8820283	0,870022312
FifthCuda	0,8467215	1,02437408	1,045960043	1,107868631

Taula A.6: Speedups per a mida mitjana.

	TK	HSP90	VEGFR2	FGFR1
Base	3108,9	6653,2	6028,2	7370,7
FirstCuda	3587,2	6458	5659,5	6613,2
SecondCuda	4258,6	7371,3	6417,4	7429
ThirdCuda	3393,1	6222,4	5461,8	6619,7
FourthCuda	4389,6	8373,3	6913,4	8515,5
FifthCuda	3668,7	6540,45	5966,1	6652,3

Taula A.7: Temps d'execució per a mida gran.

	TK	HSP90	VEGFR2	FGFR1	MEDIAN
Base	58,60593779	29,94047977	27,37135463	21,70757187	34,40633602
FirstCuda	50,79170384	30,84546299	29,15451895	24,19403617	33,74643049
SecondCuda	42,78401353	27,02372716	25,71134727	21,53721901	29,26407674
ThirdCuda	53,69720904	32,01337105	30,20982094	24,17027962	35,02267016
FourthCuda	41,50719883	23,78990362	23,86669367	18,78926663	26,98826569
FifthCuda	49,6633685	30,45661996	27,65625786	24,0518317	32,9570195

Taula A.8: Molècules per segon per a mida gran.

	TK	HSP90	VEGFR2	FGFR1
FirstCuda	0,866664808	1,030226076	1,065147098	1,11454364
SecondCuda	0,730028648	0,902581634	0,939352386	0,992152376
ThirdCuda	0,916241785	1,069233736	1,103702076	1,11344925
FourthCuda	0,708242209	0,794573227	0,871958805	0,865562797
FifthCuda	0,84741189	1,017238875	1,01040881	1,107992724

Taula A.9: Speedups per a mida gran.

		Speedups				
		First	**Second**	**Third**	**Fourth**	**Fifth**
Points	**1000**	0,341437	0,165	0,291622	0,239839	0,354577
	1500	0,457043	0,226196	0,39644	0,323914	0,480115
	2000	0,592093	0,289778	0,501571	0,407441	0,581662
	2500	0,351294	0,172399	0,306947	0,246131	0,349462
	3000	0,733312	0,368813	0,644663	0,508781	0,730659
	4000	0,81469	0,407664	0,717121	0,545186	0,804821
	5200	0,925746	0,464538	0,797231	0,630622	0,907075
	10000	1,31326	0,655478	1,20696	0,708327	1,28977
	20700	2,42293	1,38477	2,35414	1,14032	2,36687
	30600	2,89032	1,7642	1,93178	1,0351	2,87209
	40425	2,85666	1,91327	2,98446	0,709545	2,79583
	50400	4,68654	3,32555	4,89863	1,14955	4,64665
	101250	5,6801	3,97169	6,0896	1,04903	5,45768
	253500	6,46033	2,55359	6,82166	0,809587	6,22106
	505600	6,38905	2,05161	2,69377	0,701639	6,06323
	753300	6,07426	1,611	1,58515	0,55226	5,79017
	1000000	6,26802	1,45784	1,4536	0,537335	5,97775
	2044900	6,49049	1,26566	1,3429	0,4165	6,1475
	3375000	6,61438	1,13017	1,23503	0,381563	6,25668
	8000000	6,69845	1,12311	1,22402	0,303723	6,33313
	15625000	6,32433	1,08839	1,23011	0,221033	5,96289

Taula A.10: Speedups segons el nombre de punts.

Apèndix B

Execució amb OpenMP

L'apèndix conté totes les figures i taules de dades relacionades amb les execucions en OpenMP que no tenen cabuda en el contingut del treball, ja sigui perquè no són rellevants o per tal de no repetir informació.

B.1 Figures

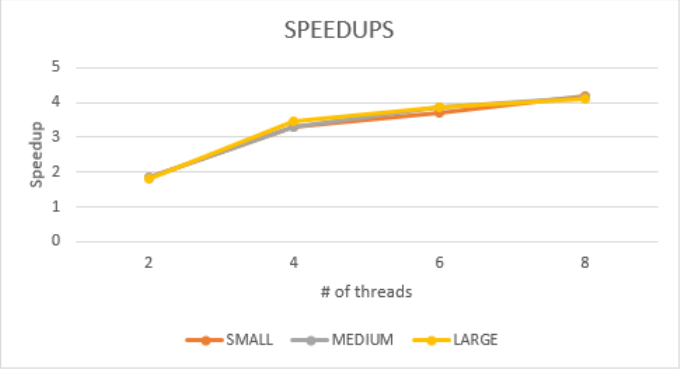

Figura B.1: Speedups TK en Melanina.

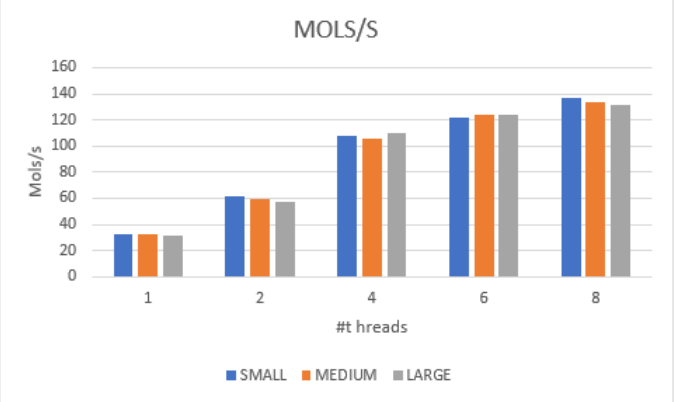

Figura B.2: Molècules per segon TK en Melanina.

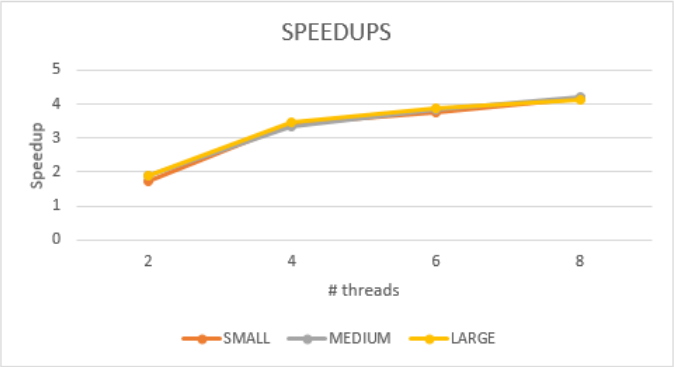

Figura B.3: Speedups HSP90 en Melanina.

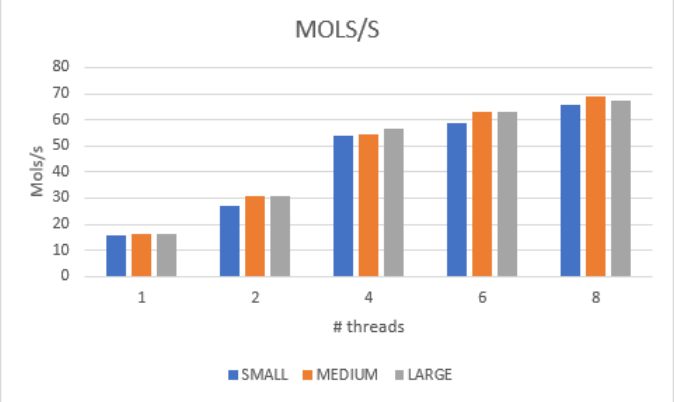

Figura B.4: Molècules per segon HSP90 en Melanina.

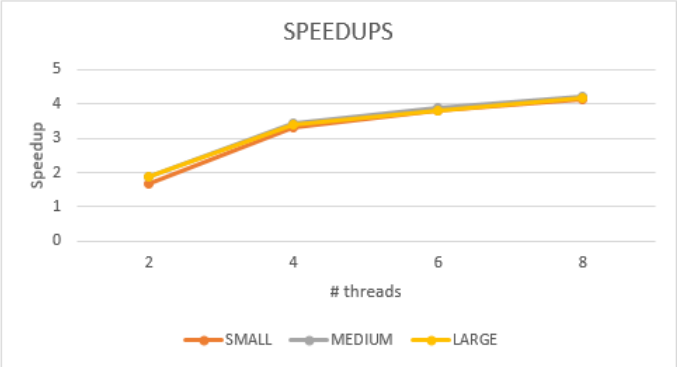

Figura B.5: Speedups VEGFR2 en Melanina.

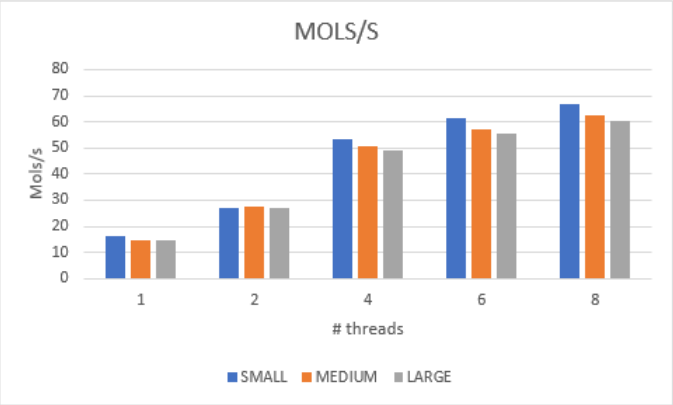

Figura B.6: Molècules per segon VEGFR2 en Melanina.

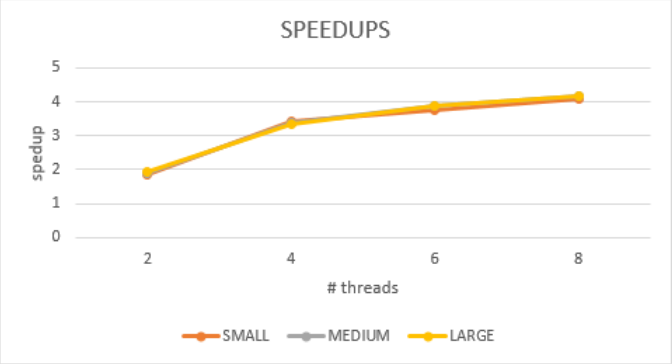

Figura B.7: Speedups FGFR1 en Melanina.

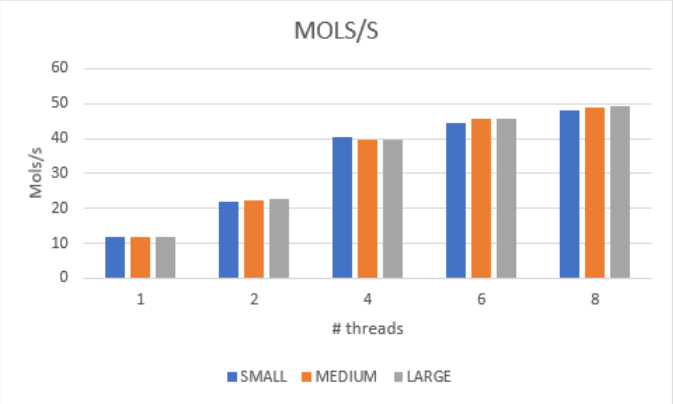

Figura B.8: Molècules per FGFR1 en Melanina.

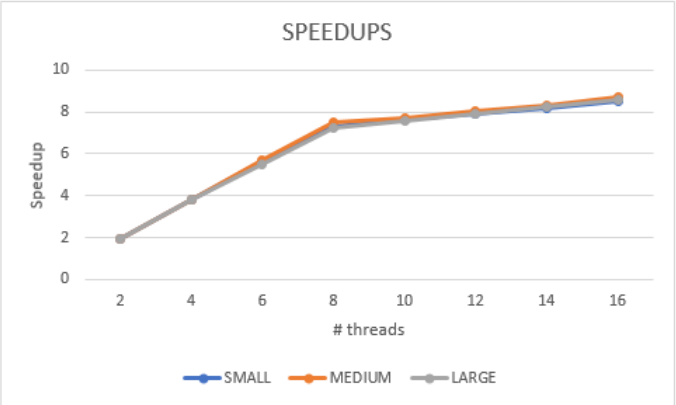

Figura B.9: Speedups TK en Amazon Web Services.

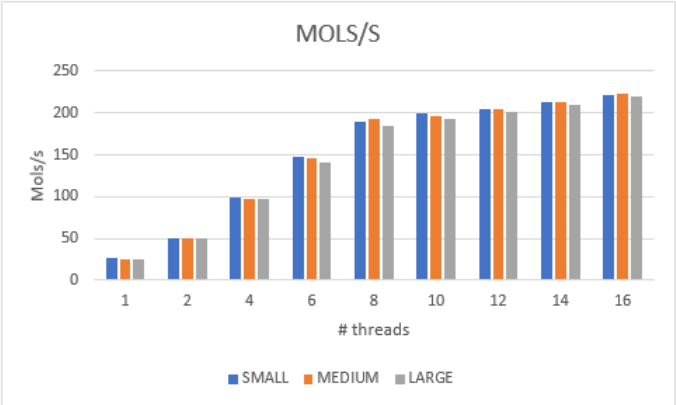

Figura B.10: Molècules per segon TK en Amazon Web Services.

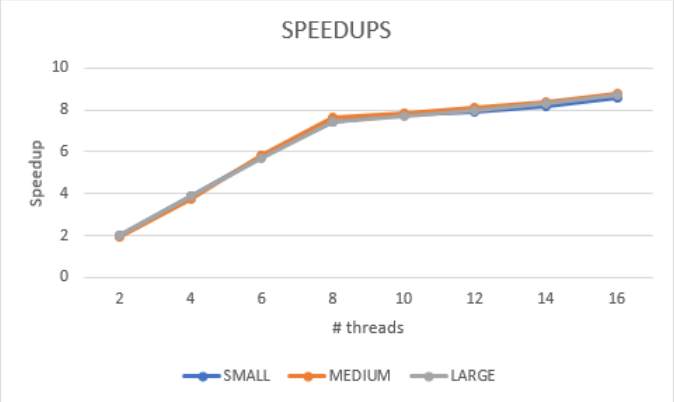

Figura B.11: Speedups HSP90 en Amazon Web Services.

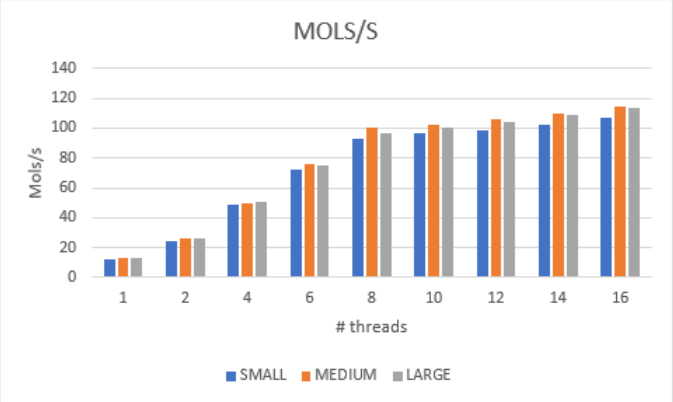

Figura B.12: Molècules per segon HSP90 en Amazon Web Services.

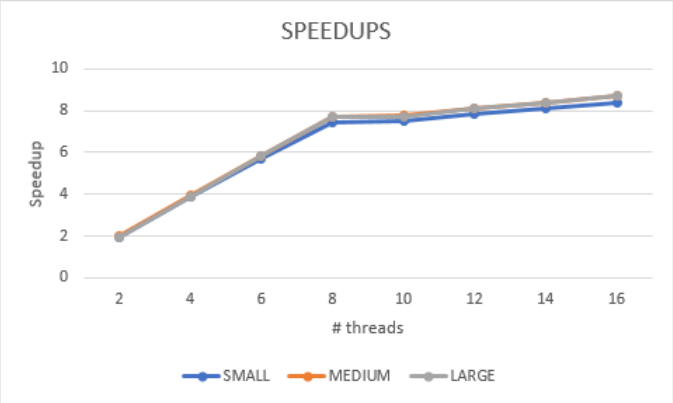

Figura B.13: Speedups VEGFR2 en Amazon Web Services.

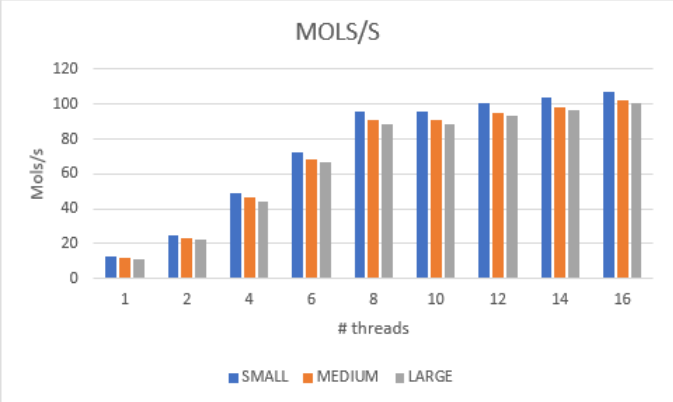

Figura B.14: Molècules per segon VEGFR2 en Amazon Web Services.

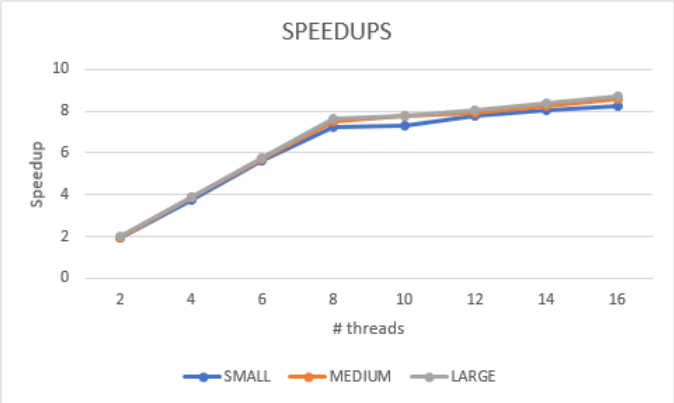

Figura B.15: Speedups FGFR1 en Amazon Web Services.

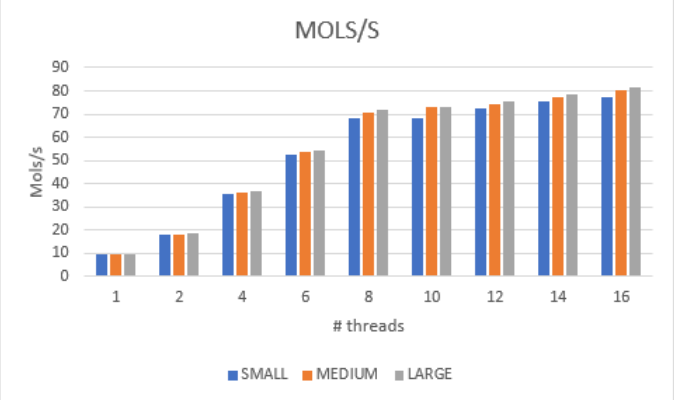

Figura B.16: Molècules per segon FGFR1 en Amazon Web Services.

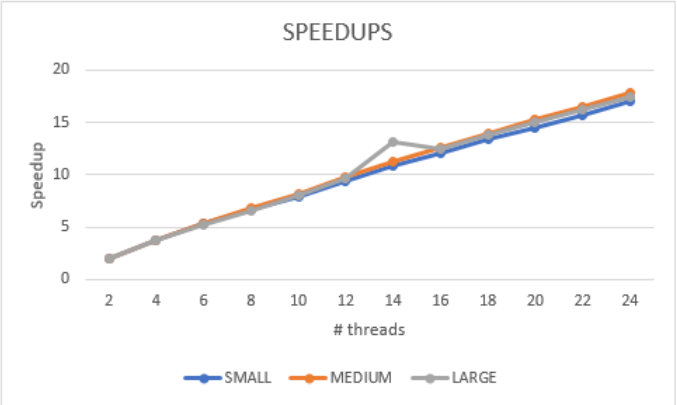

Figura B.17: Speedups TK en Rutgers.

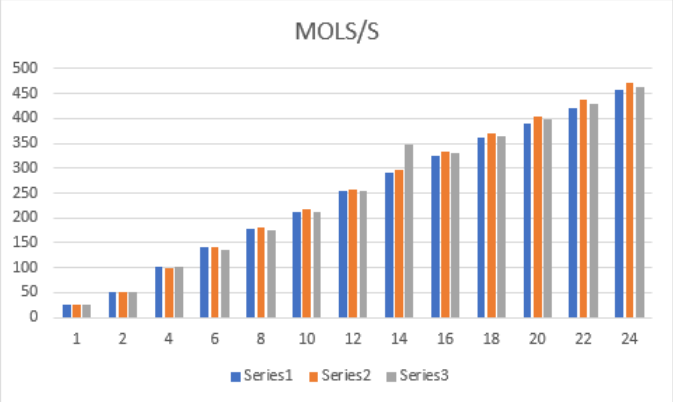

Figura B.18: Molècules per segon TK en Rutgers.

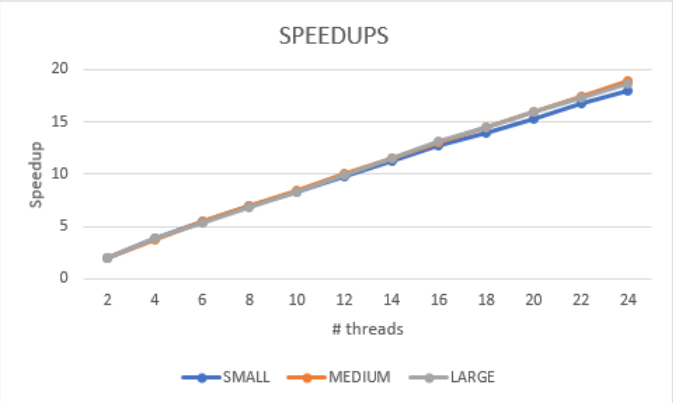

Figura B.19: Speedups HSP90 en Rutgers.

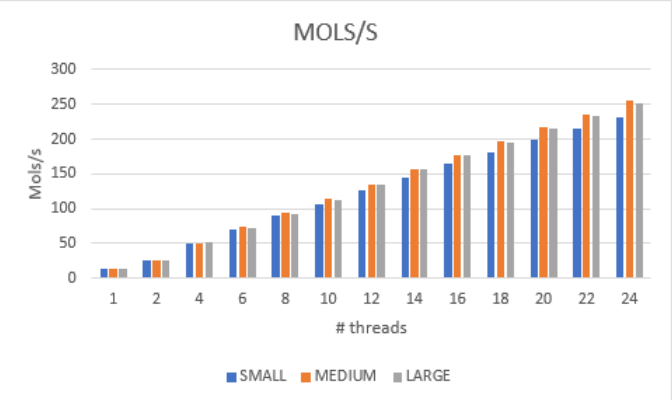

Figura B.20: Molècules per segon HSP90 en Rutgers.

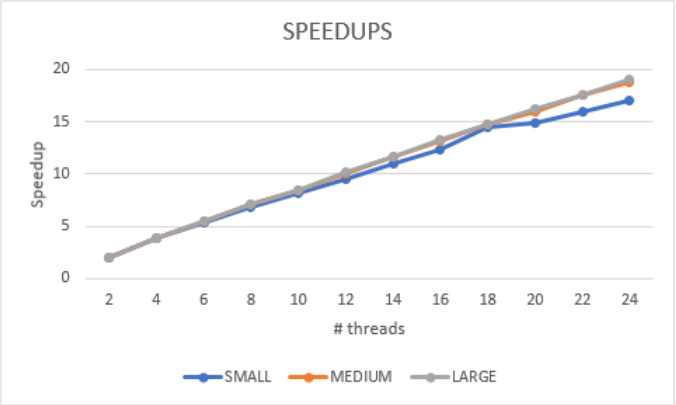

Figura B.21: Speedups VEGFR2 en Rutgers.

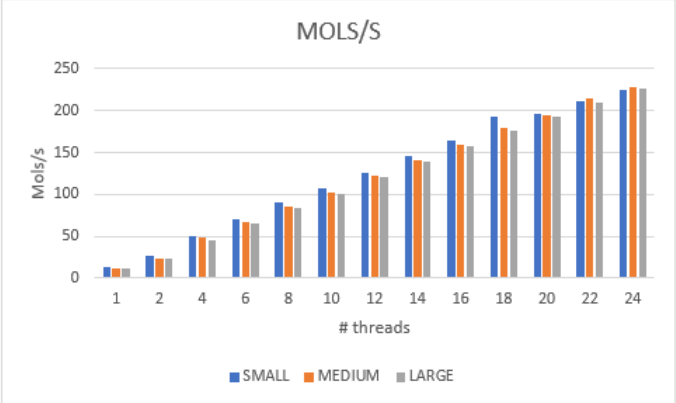

Figura B.22: Molècules per segon VEGFR2 en Rutgers.

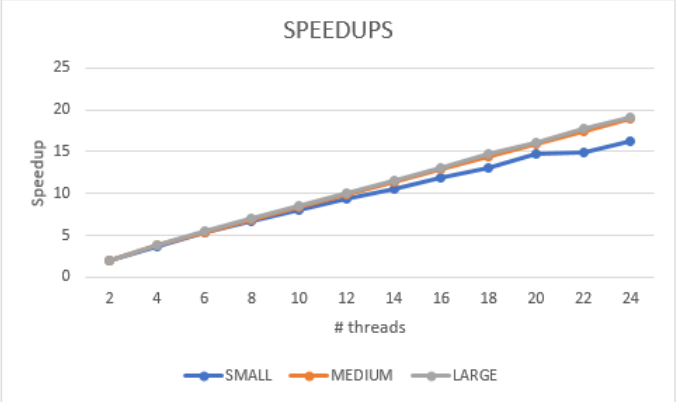

Figura B.23: Speedups FGFR1 en Rutgers.

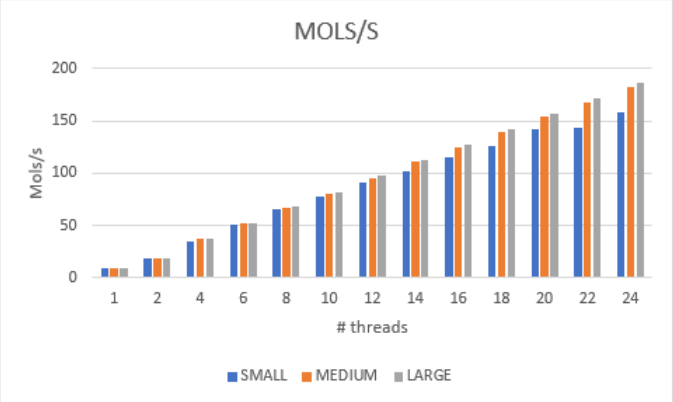

Figura B.24: Molècules per segon FGFR1 en Rutgers.

B.2 Taules

THREADS	SMALL	MEDIUM	LARGE
2	1,864531116	1,860033183	1,813086482
4	3,287169262	3,284423828	3,451538926
6	3,708657588	3,853255681	3,876109793
8	4,187834684	4,164585698	4,116605485

Taula B.1: Speedups TK en Melanina.

THREADS	SMALL	MEDIUM	LARGE
1	32,78688525	32,21090711	31,85760246
2	61,13216775	59,91335607	57,76058838
4	107,7760414	105,7942708	109,957755
6	121,5953307	124,1168608	123,4835649
8	137,3060552	134,1450831	131,145181

Taula B.2: Molècules per segon TK en Melanina.

THREADS	SMALL	MEDIUM	LARGE
2	1,734215648	1,880836648	1,879967283
4	3,438317577	3,323096942	3,476995091
6	3,750641916	3,834534535	3,867301587
8	4,178382181	4,206209339	4,128371967

Taula B.3: Speedups HSP90 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
1	15,71197702	16,44608035	16,35199475
2	27,2479564	30,93239063	30,74121514
4	54,02276674	54,65191932	56,85580546
6	58,92999958	63,06306306	63,23809524
8	65,65064478	69,17565676	67,50711671

Taula B.4: Molècules per segon HSP90 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
2	1,674234058	1,889223638	1,883130802
4	3,330081218	3,424922275	3,378394368
6	3,825431287	3,890426649	3,826771654
8	4,13976378	4,236345684	4,173731911

Taula B.5: Speedups VEGFR2 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
1	16,10602365	14,72031403	14,44707118
2	26,96525334	27,80996524	27,20572474
4	53,63436686	50,41593143	48,80790392
6	61,6124868	57,26830199	55,28564249
8	66,67513335	62,36033882	60,29820202

Taula B.6: Molècules per segon VEGFR2 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
2	1,867818467	1,897117318	1,916792802
4	3,431283219	3,380694489	3,34650531
6	3,752075722	3,861717613	3,862709883
8	4,08318776	4,150581793	4,183389935

Taula B.7: Speedups FGFR1 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
1	11,8020211	11,77911798	11,80811808
2	22,04403296	22,34636872	22,63371575
4	40,49607694	39,82159924	39,51592986
6	44,28207683	45,48762737	45,61133442
8	48,18986808	48,89019263	49,39796233

Taula B.8: Molècules per segon FGFR1 en Melanina.

THREADS	SMALL	MEDIUM	LARGE
2	1,950946076	1,928636329	1,938642652
4	3,82087328	3,783926317	3,810717142
6	5,659480539	5,695964757	5,502348141
8	7,31576945	7,527150224	7,218171168
10	7,676447106	7,700551735	7,574986751
12	7,882881036	8,029474348	7,924469725
14	8,171204572	8,324019608	8,240326523
16	8,536011541	8,713567323	8,597481023

Taula B.9: Speedups TK en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
1	26,00171611	25,518717	25,49321394
2	50,72794603	49,21632468	49,42223187
4	99,34926233	96,56094481	97,14742735
6	147,1562063	145,3537127	140,2725383
8	190,2225604	192,0832164	184,0143818
10	199,6007984	196,5082004	193,1107578
12	204,9684349	204,9018835	202,020202
14	212,4653416	212,4183007	210,072407
16	221,9509488	222,3590586	219,177423

Taula B.10: Molècules per segon TK en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
2	1,970104634	1,964564161	1,972206063
4	3,889255338	3,767958717	3,892233901
6	5,755458515	5,804426378	5,726511768
8	7,454255107	7,674621375	7,420955346
10	7,748115361	7,815959129	7,722601873
12	7,88943662	8,098178839	7,976576388
14	8,194718748	8,3706782	8,327056383
16	8,583358872	8,765432099	8,691391785

Taula B.11: Speedups HSP90 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
1	12,49665268	13,07291261	13,05715784
2	24,61971336	25,6825756	25,75140586
4	48,60267315	49,25819504	50,8215124
6	71,92396609	75,88075881	74,77196802
8	93,15323707	100,3296546	96,89658527

10	96,8255066	102,1773507	100,8352316
12	98,5915493	105,8667843	104,1514169
14	102,406554	109,4291446	108,7276895
16	107,2632547	114,5897278	113,4848744

Taula B.12: Molècules per segon HSP90 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
2	1,95260765	1,978072214	1,951815737
4	3,850316827	3,935712634	3,865415131
6	5,674529607	5,829204247	5,828061638
8	7,463470112	7,72505825	7,695025968
10	7,49073144	7,765120983	7,731870024
12	7,872456814	8,086602575	8,096901408
14	8,09942733	8,38693781	8,394859813
16	8,384884291	8,721005705	8,741560732

Taula B.13: Speedups VEGFR2 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
1	12,80003901	11,75134161	11,48065683
2	24,9934541	23,24500232	22,40812668
4	49,28420559	46,24990365	44,37750464
6	72,63420033	68,50097043	66,90997567
8	95,53270858	90,77979847	88,34395245
10	95,88165464	91,25058933	88,76694642
12	100,7677543	95,02842934	92,95774648
14	103,6729858	98,55777128	96,37850467
16	107,326846	102,4835172	100,358859

Taula B.14: Molècules per segon VEGFR2 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
2	1,948889092	1,954176122	1,97826945
4	3,777413495	3,85473973	3,901556777
6	5,607734807	5,729861991	5,788919461
8	7,25864124	7,542734439	7,662769784
10	7,279091608	7,786495552	7,775344466
12	7,747387551	7,915202728	8,039816955
14	8,073099138	8,253516772	8,362530055
16	8,253630203	8,591796341	8,708671879

Taula B.15: Speedups FGFR1 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
1	9,38306357	9,364698831	9,388569417
2	18,28655024	18,30027084	18,57312005
4	35,44371096	36,09847664	36,63003663
6	52,61773218	53,65843189	54,3496722
8	68,10829218	70,63543639	71,94244604
10	68,30017929	72,9181858	72,99936126
12	72,6942299	74,12348973	75,48237958
14	75,75040242	77,29169887	78,51219393
16	77,44433688	80,45958515	81,76197046

Taula B.16: Molècules per segon FGFR1 en Amazon Web Services.

THREADS	SMALL	MEDIUM	LARGE
2	1,941079769	1,930938359	1,935313457
4	3,779301936	3,766927083	3,798786542
6	5,294142766	5,329944838	5,175232943
8	6,644158628	6,866361377	6,604526275
10	7,918755788	8,222872429	8,04983812
12	9,445615461	9,717260729	9,582057989
14	10,85623386	11,19531072	13,10749086
16	12,0608979	12,59243138	12,48133382
18	13,42259112	13,98538361	13,77412452
20	14,49210988	15,2527602	15,07705779
22	15,65766486	16,52032247	16,18765571
24	16,97503537	17,80694449	17,48064672

Taula B.17: Speedups TK en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
1	26,88605689	26,43297208	26,45487281
2	52,18798111	51,04043973	51,19847135
4	101,6105269	99,57107843	100,4964148
6	142,3386236	140,8862831	136,9101292
8	178,6352269	181,4983386	174,7219026
10	212,9041186	217,3549574	212,9574436
12	253,9553546	256,8560816	253,4921253
14	291,8813211	295,9253358	346,7570037
16	324,2699872	332,8553871	330,1920986
18	360,8805485	369,6752545	364,3927121
20	389,6356906	403,1757846	398,8616462
22	420,9728683	436,6812227	428,242373
24	456,3917667	470,6904667	462,4482855

Taula B.18: Molècules per segon TK en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
2	1,964629059	1,96134151	1,972809142
4	3,878603402	3,673712256	3,887921941
6	5,421192252	5,452939626	5,397993556
8	6,941613792	6,961004285	6,795888264
10	8,286916603	8,411142741	8,35818357
12	9,777035566	9,998710863	9,939324479
14	11,24504246	11,53091375	11,58585462
16	12,70647583	13,06844987	13,09673981
18	13,95387787	14,50420757	14,47520864
20	15,3333239	16,02877735	15,97637625
22	16,72018278	17,41726316	17,24246819
24	17,9434081	18,87996349	18,66273403

Taula B.19: Speedups HSP90 en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
1	12,92586095	13,53768796	13,51149698
2	25,39452204	26,55202933	26,65560477
4	50,13428827	49,73357016	52,53164557
6	70,07357726	73,8201951	72,93497364
8	89,72633468	94,23590388	91,82262377
10	107,1155318	113,8674258	112,9315721
12	126,3766023	135,3594276	134,2951527
14	145,3518553	156,1019122	156,5422397
16	164,2421398	176,9165965	176,9565604
18	180,3658851	196,3534362	195,5817378
20	198,1964126	216,9925861	215,8647594
22	216,1227577	235,7894737	232,9715569
24	231,9339982	255,5910543	252,1614745

Taula B.20: Molècules per segon HSP90 en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
2	1,951118761	1,973621103	1,95123665
4	3,833800367	3,928400955	3,85394693
6	5,301088917	5,49956008	5,489878222
8	6,790347424	7,056503473	7,055744702
10	8,157575508	8,449113681	8,434064265
12	9,545301448	10,06030478	10,11804999
14	10,96627971	11,63360505	11,71828846
16	12,37921405	13,11971943	13,25536993
18	14,5493216	14,70430588	14,7657787
20	14,82605284	16,00700185	16,17148647

22	15,97487569	17,6407545	17,60826834
24	17,04258929	18,83582545	19,03097588

Taula B.21: Speedups VEGFR2 en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
1	13,23218066	12,15066829	11,88332733
2	25,81755594	23,98081535	23,18718381
4	50,72953909	47,7326969	45,79771289
6	70,14496626	66,82333025	65,23801993
8	89,85110388	85,74123296	83,84572387
10	107,9425129	102,6623777	100,2247464
12	126,3051533	122,2394263	120,2361
14	145,1077944	141,356076	139,2522576
16	163,8039968	159,4133588	157,5178998
18	192,5192519	178,6671431	175,4665816
20	196,1810097	194,4957697	192,1710672
22	211,3824412	214,3469563	209,2448164
24	225,5106205	228,867867	226,1513158

Taula B.22: Molècules per segon VEGFR2 en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
2	1,945824194	1,954511079	1,975943165
4	3,628650598	3,852500559	3,896291047
6	5,285137732	5,418361357	5,42729277
8	6,739176605	6,910201735	7,015312633
10	8,023828049	8,29053523	8,454480213
12	9,410187973	9,830917874	10,03529054
14	10,5659435	11,43688461	11,61070046
16	11,86774458	12,92063492	13,12509947
18	13,04487382	14,42738717	14,677405
20	14,67310497	15,89794851	16,16168545
22	14,85166517	17,36278132	17,69825089
24	16,31647647	18,86941919	19,15830313

Taula B.23: Speedups FGFR1 en Rutgers.

THREADS	SMALL	MEDIUM	LARGE
1	9,696852159	9,67323802	9,701085309
2	18,86836954	18,90645088	19,1687932
4	35,18648839	37,26615488	37,79825183
6	51,24919923	52,41309908	52,65063016

8	65,34879922	66,84402615	68,05614632
10	77,80587434	80,19632059	82,01763379
12	91,24920157	95,09680855	97,35320961
14	102,456392	110,631707	112,6363956
16	115,0797647	124,984377	127,3277097
18	126,4942129	139,5595501	142,386758
20	142,2829296	153,78464	156,7858893
22	144,0144014	167,9543164	171,6922417
24	158,2184601	182,5283832	185,8563331

Taula B.24: Molècules per segon FGFR1 en Rutgers.

Apèndix C

Altres

Aquest apèndix conté figures i taules que no tenen cabuda en el contingut principal del treball i que tampoc encaixen amb la resta d'apèndixs.

C.1 Figures

```
Device 0: "Quadro M4000"
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    5.2
  Total amount of global memory:                 8119 MBytes (8513126400 bytes)
  (13) Multiprocessors, (128) CUDA Cores/MP:     1664 CUDA Cores
  GPU Max Clock rate:                            773 MHz (0.77 GHz)
  Memory Clock rate:                             3005 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

Figura C.1: Característiques tècniques targeta Nvidia Quadro M4000.

Bibliografia

[1] *Geppert, H.; Vogt, M.; Bajorath, J.; J. Chem. Inf. Model. 2009, 50, 205.*

[2] *Curutchet, C.; Orozco, M.; Luque, F. J. J. Comput. Chem. 2001, 11, 1180.*

[3] *Soteras, I.; Curutchet, C.; Bidon-Chanal, A.; Orozco, M.; Luque, F. J. J. Mol. Struct. THEOCHEM 2005, 727, 29.*

[4] *Curutchet, C.; Salichs, A.; Barril, X.; Orozco, M.; Luque, F. J. J. Comput. Chem. 2003, 24, 32.*

[5] *Morreale, A.; Gelpí, J. L.; Luque, F. J.; Orozco, M. J. Comput. Chem. 2003, 24, 1610.*

[6] *Soteras, I.; Forti, F.; Orozco, M.; Luque, F. J. J. Phys. Chem. B 2009, 113, 9330.*

[7] *Soteras, I.; Orozco, M.; Luque, F. J. J. Comput.-Aided Mol. Design 2010, 24, 281.*

[8] *Luque, F. J.; Barril, X.; Orozco, M. J. Comput.-Aided Mol. Design 1999, 13, 139.*

[9] *Barril, X.; Muñoz, J.; Luque, F. J.; Orozco, M. Phys. Chem. Chem. Phys. 2000, 2, 4897.*

[10] *Muñoz, J.; Barril, X.; Hernández, B.; Orozco, M.; Luque, F. J. J. Comput. Chem. 2002, 23, 554.*

[11] *Muñoz-Muriedas, J.; Perspicace, S.; Bech, N.; Guccione, S.; Orozco, M.; Luque, F. J. J. Comput.-Aided Mol. Design 2005, 19, 401.*

[12] *B. D. Silverman; Daniel. E. Platt. J. Med. Chem. 1996, 39, 2129 – 2140.*

[13] *Ginex, L; Muñoz-Muriedas, J; Herrero, E; Gibert, E; Cozzini, P; Luque, F. J. J. Comput. Chem 2016, 37, 1147–1162.*

[14] *Título: CALCULATING MOLECULAR SIMILARITY; Autores: Vázquez, J; Herrero, E; Gibert, E: Luque, F,J; Numero de solicitud: PCT/EP2016/082850; Oficina de depósito: EPO; 29 de Diciembre de 2016.*

[15] *Rabal, O; Ibrahim Amr, F; Oyarzabal, J. J. Chem. Inf. Model. 2015, 55, 1-18.* PG-02829-001_v8.0

[16] *Pipeline Pilot, version 8.5; Accelrys, Inc.: San Diego, CA, 2011.*

[17] *ROCS, version 3.1.1; OpenEye Scientfic Software: Santa Fe, NM, 2014.*

[18] *Huang, N.; Shoichet, B. K.; Irwin, J. J. Benchmarking Sets for Molecular Docking. J. Med. Chem. 2006, 49, 6789.*

[19] *LigPrep, version 2.3; Schrödinger, LLC: New York, 2009.*

[20] *OMEGA, version 2.5.1.4; OpenEye Scientific Software.: Santa Fe, NM, 2014.*

[21] *Wei, D. G; Yang, G. F; Wan, J; Zhan, C. G; J. Agric. Food Chem. 2005, 53, 1604.*

[22] *Prasanna, S; Daga, P. R; Xie, A; Doerksen, R.J; J. Comput. Aided Mol. 2009, 23, 113.*

[23] *Méndez-Lucio, O; Pérez-Villanueva, J; Romo-Mancillas, A; Castillo, R ; Med. Chem. Commun. 2011, 2, 1058*

[24] *Forti, F; Barril, X; Luque, F.J; Orozco, M; J Comput Chem 29: 578–587, 2008*

[25] *Nvidia Cuda Installation Guide for Linux, versió DU-05347-001_v8.0; Nvidia; September 2016*

[26] *Cuda C Best Practices Guide, version DG-05603-001_v8.0; Nvidia; September 2016*

[27] *Cuda Driver API, version TRM-06703-001_v8.0; Nvidia, February 2016*

[28] *Cuda Compiler Driver Nvcc, version TRM-06721-001_v8.0; Nvidia, September 2016*

[29] *Cuda Debugger API, version TRM-06710-001_v8.0; Nvidia, February 2016*

[30] *Cuda-Memcheck, version DU-05355-001_v8.0; Nvidia, September 2016*

[31] *Cuda Runtime API, version v8.0; Nvidia, February 2016*

[32] *Cuda C Programming Guide, version PG-02829-001_v8.0; Nvidia; September 2016*

[33] *Computació d'altes prestacions, versió PID_00213061; Ivan Rodero Castro, Francesc GUIM Bernat; Setembre 2014*

[34] *OpenMP - Application Programming Interface, version 4.5; OpenMP Architecture Review Board; November 2015*

[35] *Introduction to Intel Advanced Vector Extensions; Chris Lomont; June 2011;* https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

[36] *SIMD: CilkPlus and OpenMP; Kent Milfeld, Georg Zitzlsberger, Michael Klemm, Carlos Rosales; June 2015;* https://www.nersc.gov/assets/Uploads/SIMD12-Milfeld.pdf

[37] *A Guide To Vectoritzation with Intel C++ Compilers; Intel Corporation; 2010;* https://software.intel.com/sites/default/files/m/4/8/8/2/a/31848-CompilerAutovectorizationGuide.pdf