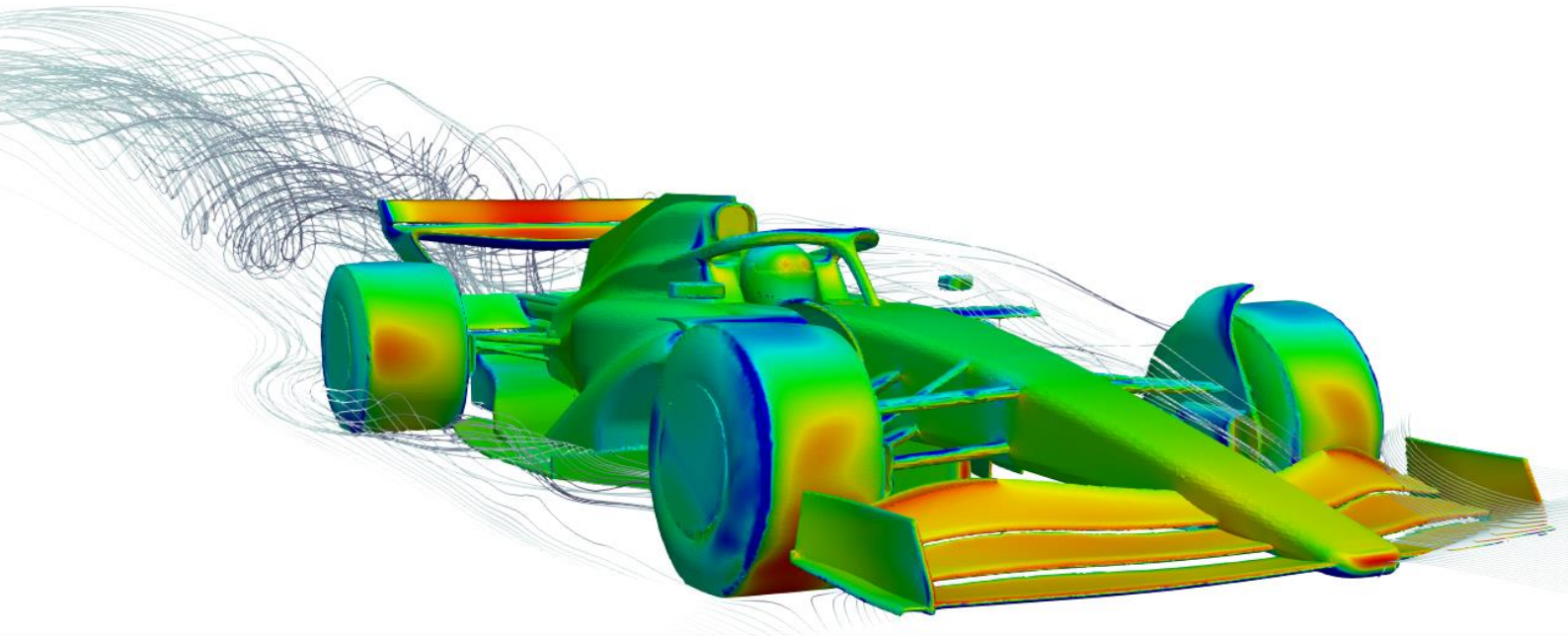
Simulació de fluids en la supercomputació binaria

Treball de recerca de Batxillerat

Pau Esquerda Masip



Tutor: Joan Gatius Lendínez
Centre: IES Joan Oró Lleida
Curs: 2n Batxillerat 2023

Foto portada:
(Font: https://maxtayloraero.com/)

Abstract (English)

It is well known that new technologies have made our daily lives easier and will continue to do so for a long time. One of the areas that has improved the most over time is fluid simulations. These simulations allow us to understand how the world and the fluids that compose it work. From the gases that come out of a vehicle's exhaust to the most remote waterfalls, they all follow the same principle.

Thanks to supercomputing simulations, we can understand these behaviours and analyse them to develop more functional and useful products. These simulations are based on fluid mechanics. The Navier-Stokes equations are used to understand these behaviours, and when combined with other equations, such as the Boltzmann equation, they can be implemented in binary computing.

By following certain laws and principles, we can closely approximate these simulations to obtain values and results that are faithful to reality. Thanks to these results, we can save thousands of resources that would be spent on real wind tunnel tests.

These simulations allow us to know how a solid will behave when in contact with a specific fluid even before we have the solid physically available. They allow us to validate the feasibility of projects that would cost significant amounts of money to validate if we couldn't test them first on a simple computer.

For this work, many languages have been proposed for creating fluid simulations for the practical part, but Python has been chosen to assemble the lines of code and finally create a simulator that runs using the Lattice Boltzmann method.

Some of the problems that have been identified and could be improved in this field of simulation are the difficulty in creating solids using text-based environments (without a graphical interface) and the large amount of computational resources required to run these types of simulations.

This research project concludes with the idea that these simulations will evolve towards the optimization of computational resources so that they are accessible to everyone and do not require such excessive computing power.

Resumen (Español)

Es bien sabido que las nuevas tecnologías nos han facilitado la vida diaria y lo seguirán haciendo mucho tiempo más. Uno de los campos donde más se ha mejorado con el tiempo han sido las simulaciones de fluidos. Estas, nos permiten entender cómo funciona el mundo y los fluidos que lo componen. Desde los gases que salen por el escape de un vehículo hasta las cataratas más recónditas, todos siguen el mismo principio.

Gracias a las simulaciones por supercomputación, podemos entender estos comportamientos y analizarlos para desarrollar productos más funcionales y útiles. Estas simulaciones se basan en la mecánica de fluidos. Se usan las ecuaciones de Navier Stokes para poder entender estos comportamientos, y combinadas con otras ecuaciones, como la de Boltzmann, las podemos implementar en la computación binaria.

Siguiendo ciertas leyes y principios, podemos asemejar al máximo dichas simulaciones para obtener valores y resultados fieles a la realidad. Gracias a estos resultados, podemos ahorrar miles de recursos que tendríamos que gastar haciendo pruebas reales en túneles de viento.

Estas simulaciones nos permiten saber cómo se va a comportar un sólido en contacto con un determinado fluido incluso antes de disponer de dicho sólido físicamente. Nos permiten validar la viabilidad de proyectos que costarían grandes cantidades de dinero si no los pudiésemos probar primero en una simple computadora.

Para este trabajo, son muchos los lenguajes que se han propuesto para crear la simulación de fluidos para la parte práctica, pero ha sido Python el elegido para ensamblar las líneas de código y conseguir finalmente un simulador que se ejecuta usando el método de Lattice Boltzmann.

Algunos de los problemas que se han detectado y que se podrían mejorar en este campo de la simulación son la dificultad para crear sólidos usando entornos de texto (sin interfaz gráfica), y la gran cantidad de recursos computacionales necesarios para ejecutar este tipo de simulaciones.

Se concluye este trabajo con la idea de que estas simulaciones evolucionaran hacia la optimización de recursos computacionales para que estén al alcance de todos, y no sea necesaria esta exagerada potencia computacional.

Agraïments

M’agradaria agrair a multitud de persones que m’han ajudat en la creació i elaboració d’aquest treball de recerca, però m’hauré de limitar a unes poques, ja que sinó, afegiríem pàgines de més a aquest TDR.

La primera persona que vull mencionar és el meu tutor Joan Gatius Lendínez. M’ha donat molts consells de valor en cada una de les sessions de seguiment que hem tingut, i la comunicació ha estat extremadament senzilla degut al seus amplis coneixements en el món de la programació i la computació.

En segon lloc, vull agrair als meus pares els ànims que m’han donat en els moments complicats del desenvolupament d’aquest projecte. Els vull donar gràcies per haver proporcionat els seus punts de vista per a poder millorar aquest treball i dur-lo al següent nivell.

Finalment vull agrair als investigadors del BSC i a la fundació La Pedrera per brindar-me l’oportunitat d’accedir al Supercomputador MareNostrum 4 i poder fer-ne ús. Totes les sessions han estat de gran ajuda per a poder executar aquest Treball de Recerca de manera òptima i obtenir un resultat de qualitat amb informació de valor.

Índex

1 Motivació del treball, procés i dificultats

Des de ben petit, sempre m'han fascinat tots els temes relacionats amb els ordinadors. Sempre he invertit el meu temps lliure en aprendre i investigar sobre hardware i programació.

Durant el meu procés d'aprenentatge, vaig tenir la sort de poder participar en el programa «bojos per la supercomputació». Aquest programa, organitzat per la Fundació La Pedrera, et permet accedit a les instal·lacions del Barcelona Supercomputing Center (un dels centres de supercomputació més importants a nivell Europeu). Es va realitzar un procés de selecció on, a partir d'unes cartes de presentació i els resultats acadèmics d'anys anteriors, es van triar 25 alumnes que podien ser aptes per entrar en aquest programa. Afortunadament, vaig poder participar-hi i extreure gran part de la informació d'aquest treball.

Des de feia temps, volia indagar i investigar més sobre la simulació d'elements de la realitat utilitzant màquines de computació binaria, és a dir, que utilitzen uns i zeros per a tractar tota la informació donada. Volia descobrir quines eren les seves limitacions a l'hora de recrear elements i fenòmens reals.

Una de les sessions d'aquest programa va tractar sobre la simulació de fluids, i em vaig adonar de les moltes de possibilitats que podia tenir aquesta branca de la supercomputació. Des de simulacions d'aerodinàmica per a cotxes d'alt rendiment, fins a simulacions sobre com es distribueixen els medicaments en el torrent sanguini.

Finalment, vaig triar la simulació de fluids utilitzant diferents mètodes, ja que és una de les parts de la supercomputació més complicades de simular fidelment a la realitat, i una de les àrees que requereix més poder de computació.

Per a poder entendre com podem utilitzar aquestes simulacions, ens hem d'imaginar una xemeneia des don surt una columna de fum. Aquest fum, sembla desordenat, dispers i sense cap tipus de sentit, però en realitat en té, i segueix un patró determinat. Això ho podem observar amb una simulació de fluids, i aplicant les formules i els algoritmes adients podem arribar a entendre perquè aquest fum es mou d'una determinada manera i com podem fer aquesta xemeneia més eficient. Aquest és només un dels milers d'exemples per als que podem utilitzar una simulació de fluids.

Aquest treball dedica un important apartat als diferents mètodes de simulació de fluids existents avui en dia en la supercomputació. També se li dona especial atenció al mètode Lattice Boltzmann o Malla de Boltzmann, el qual s'utilitzarà per a fer la part pràctica del treball.

Una part important del mètode tecnològic en un treball on es toca la programació, és triar quin llenguatge s'utilitzarà per a la part pràctica del treball. Per a fer aquest simulador s'ha utilitzat Python, ja que té una gran senzillesa i versatilitat. S'ha escollit aquest llenguatge perquè la major part de les simulacions estudiades i provades al BSC (Barcelona Supercomputing Center) estan programades en Python. La següent etapa del mètode tecnològic és la de recerca d'informació, així que el treball s'ha centrat en documentar i extreure els conceptes més importants en aquest àmbit. L'objectiu d'aquesta etapa és familiaritzar-me amb el vocabulari i les idees més importants per a poder entendre com funciona una simulació d'aquesta magnitud i com l'ordinador, tracta la informació proporcionada per a obtenir un escenari final realista i amb sentit, que es correspongui a la realitat.

Seguint amb el mètode tecnològic es descarrega una simulació ja feta per tal de comprendre els conceptes i sistemes d'iteració que aquesta utilitza. Quan tot està clar, s'inicia l'extrapolació d'aquests conceptes al programa propi. Es van afegint implementacions que permeten una millor visualització de la simulació i una execució més senzilla. Es busquen els valors adequats per a les diferents constants de la simulació, com ara els factors externs que condicionen la vorticitat del fluid, fins a trobar un punt d'equilibri. Tot i això, aquesta informació es detallarà més endavant, al punt on s'explica el codi.

També cal trobar un equilibri en l'escala i la resolució de la simulació. Tot i que no es tracta d'una simulació particularment exigent, implica 2.400 milions d'iteracions en total per a l'ordinador. Una iteració és una sèrie de càlculs que es realitzen de manera repetida. Aquest no és un procés senzill i, per tant, és necessari trobar la qualitat òptima perquè la simulació sigui ben apreciada i, al mateix temps, s'executi de manera fluïda.

Amb la introducció feta, podem passar al marc teòric.

SUPERCOMPUTADORS: QUÈ SÓN, COM FUNCIONEN I COM ESTAN CONSTRUÏTS (marc teòric)

2 Què és i com funciona un supercomputador?

2.1 Potència i poder computacional

Un supercomputador, és una màquina que funciona amb tecnologia binaria i ens ajuda a resoldre problemes i enigmes massa complexos per a una persona humana o un ordenador convencional. Consten d'un gran poder computacional matemàtic, capaç de fer bilions d'operacions flotants (decimals) per segon. Aquesta potència és mesurada amb una unitat denominada «flop». Un flop és l'equivalent a una operació decimal per segon. Aquest aparell de grans dimensions és capaç d'extreure fins a 13,19 petaflops de potència, és a dir, fins a (13,19 x 10^15) operacions flotants per segon.

El concepte de superordenador o supercomputador, dona a entendre que és una màquina extremadament complexa i complicada d'entendre, i no està equivocat. Un superordenador consta de milions de components importants per al seu correcte funcionament, que treballen tots alhora per donar el màxim rendiment, però si ho observem des d'una perspectiva més ampla, funciona exactament igual que l'ordenador que estic utilitzant per a escriure aquest document, simplement que a major escala.

Un supercomputador està format per diversos components que anirem mencionant i agrupant per a crear-ne de més grans.

2.2 Components del supercomputador

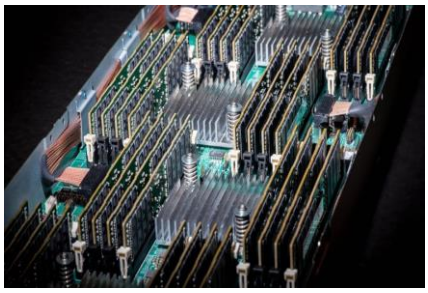
El component principal del supercomputador i que fa tots els càlculs necessaris és el xip. Parlaríem en singular si es tractés d'un ordenador de sobretaula convencional. En el cas d'un supercomputador, hi ha milers de xips treballant conjuntament. Aquests xips, en el seu interior, contenen un nombre de processadors cadascun, i a la vegada aquests xips s'agrupen en nodes per a crear l'uniat bàsica de computació.

Fiquem com a exemple el MareNostrum 4. Aquest poderós supercomputador ubicat a Barcelona, conté 3.456 nodes. Cada node conté dos xips Intel Xeon Platinum, un dels xips més potents que es poden utilitzar en la supercomputació/servidors. Cada processador Intel Xeon conté 24 processadors, és a dir, que en total el MareNostrum 4 compta amb un total de 165.888 processadors treballant a la vegada. Són uns quants, tenint en compte que l'ordenador que estic fent servir ara en té 6.

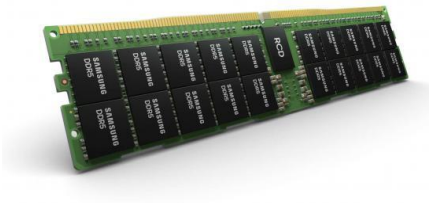
Component 2: Node del MareMnostrum4

Component 1: Xip Intel Xeon Platinum

Però els processadors i els xips no són l'única cosa que permet al supercomputador fer la seva feina. També hem de tenir en compte altres components com la memòria RAM. El MareNostrum 4 compta amb 390 Terabytes de memòria RAM, és a dir, 390.000 Gigabytes, en comparació amb els 16 que té el meu ordenador personal. Sens dubte és un número increïble.



Component 7: Clúster de memòria RAM (Font: muycomputerpro.com)



Component 6: Stick de memòria RAM (Font: amazon.es)

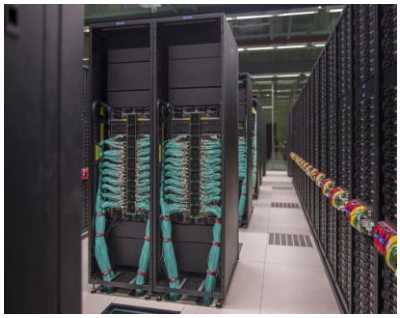
Component 3: Clúster de discs HDD (Font: dreamstime.com)

Component 4: Disc HDD d'alt rendiment (Font: amazon.es)

Component 5: Disc SSD d'alt rendiment (Font: amazon.es)

Si seguim parlant de números, tenim un emmagatzematge intern de 14 Petabytes sumat als 24,6 Petabytes del centre de Big Data del BSC. Estem parlant d'un emmagatzematge total de 38,6 milions de Gigabytes d'espai per a poder emmagatzemar totes les simulacions i estudis que s'hi fan.

Tornant al tema dels xips i nodes, aquests s'emmagatzemen en «racks». Un rack de nodes és un armari ple d'aquests nodes. I tots aquests racks estan connectats entre si per un sistema de cablejat d'alta velocitat per a assegurar que la informació arriba de manera ràpida i precisa a tots i cada un dels diferents xips.



Component 9: Racks MareNostrum 4 (Font: bsc.es)

Component 8: Cablejat Ethernet d'alta velocitat (Font: bsc.es)

2.3 El viatge de la informació fins al xip

La informació, es carrega a la memòria interna del supercomputador, és a dir, als discs durs tant mecànics com sòlids. Aquí surt el primer problema, i és que aquest tipus d'emmagatzematge és massa lent a l'hora d'entregar la informació als xips. Així que s'utilitza un intermediari, la memòria RAM. Això no només es cosa de superordinadors sinó que també funciona així als ordenadors convencionals.

La memòria RAM, es un tipus de memòria molt ràpida però volàtil, és a dir, quan s'acaba de fer el procés del tractament de les dades i ja no es necessiten més, s'esborren de la memòria RAM i tornen a l'emmagatzematge intern.

I com es coordinen tots els xips per a tractar la informació?

Es ben senzill, simplement es divideix la prova o la simulació en petits trossos i cada xip s'encarrega d'executar, tractar i analitzar els resultats d'un d'aquests bocins.

Utilitzar tots els processadors del supercomputador per a fer una sola tasca, no seria gents eficient, ja que es llençarien recursos innecessàriament, així que el supercomputador es divideix en parts per a que pugui ser utilitzat per molts usuaris a la vegada, ja que normalment és molt difícil executar una tasca que ocupi tot el potencial de l'aparell.

2.4 Connexió usuari-màquina

Els diferents usuaris es connecten remotament al supercomputador utilitzant el protocol SSH (Secure Shell), el qual ens permet establir una connexió segura entre el nostre ordenador i el supercomputador. Aquest últim utilitza un entorn Linux per a funcionar, ja que a l'hora d'introduir comandaments té moltes menys limitacions que qualsevol altre sistema operatiu. El supercomputador no té interfície gràfica, tot es fa utilitzant comandes de Linux. Per tant, s'han de tenir uns mínims coneixements de com funciona un entorn d'aquest estil i com moure's amb facilitat pels diferents directoris.

2.5 Refrigeració del sistema

Un aparell d'aquestes dimensions consumeix molts recursos, tant d'electricitat com d'aigua. La refrigeració és imprescindible per al correcte funcionament de la màquina. Com es obvi, el procés computacional i tractament de dades fan que els xips arribin a temperatures extremadament altes, i això baixa exponencialment el rendiment del supercomputador.

Quan es tracta d'un ordenador convencional, només és necessari un simple dissipador per extreure la calor del xip o, com a molt, en els models de més alta gamma, s'utilitza un sistema de refrigeració liquida, però es tracta d'un sistema a petita escala.

Un superordenador utilitza el mateix principi que un ordenador convencional, simplement augmenta el sistema de refrigeració a unes dimensions que facin

justícia al poder computacional de l'aparell. En el cas del MareNostrum 4, tenim una sèrie de turbines de grans dimensions instal·lades al sostre del BSC que garanteixen el flux d'aire fresc per alimentar al supercomputador. A la vegada, s'aprofita també el poder de la refrigeració líquida. El sistema de refrigeració, té connexió directa a la xarxa d'aigües pública per poder mantenir els racks en una temperatura òptima per al seu correcte funcionament. Aquesta aigua és filtrada i tornada a reutilitzar per a poder assegurar un flux continu cap a les instal·lacions de la màquina.

En termes de consum, el MareNostrum 4, necessita 1,3 MW d'electricitat a l'any. Tot i que és 10 vegades més potent que el seu antecessor, el MareNostrum 3, només consumeix un 30% més que aquest. Això és degut a la increïble optimització del sistema de xips. Els nous Intel Xeon Platinum són extremadament eficients en relació al seu rendiment.

En la meva visita al BSC, vaig observar que el sistema elèctric era molt diferent del que podem tenir qualsevol de nosaltres a les nostres llars. La línia d'alta tensió arriba directament a l'edifici sense passar per transformadors externs. Un cop a dins, el BSC utilitza els seus propis transformadors per a assegurar-se que es subministra el voltatge òptim a l'aparell.

2.6 Seguretat de les instal·lacions i extinció d'incendis

Les instal·lacions d'un supercomputador han de ser extremadament segures en l'àmbit dels incendis. Hi ha milions i milions d'elements que per un mal funcionament poden crear un incendi, ja que la quantitat d'energia elèctrica que s'utilitza és molt gran. Òbviament, si el supercomputador s'incendia no s'hi poden tirar líquids conductors com l'aigua, ja que es podria malmetre el material valorat en milions d'euros. Així que disposen d'un curiós sistema per a apagar el foc.

En primer lloc, el superordenador es preserva en un lloc sec i fresc amb una concentració d'oxigen més baixa del normal. És el primer que es nota a l'accedir dins de l'habitacle on es troba l'aparell.

En el cas que hi hagués foc, s'allibera un gas no conductor que ocupa el lloc de l'oxigen de l'estança i les flames per si soles s'acaben consumint sense oxigen. És un enginyós sistema d'extinció, encara que s'ha de tenir en compte que no hi poden haver persones quan el gas és alliberat, ja que podrien patir ofegaments degut a la falta d'oxigen.

3 Superordinadors Catalans[1]

A Catalunya, tenim la sort de comptar amb un centre de supercomputació dedicat a la investigació i a prestar serveis a les empreses més grans del món a l'hora de fer simulacions o provar els seus nous productes, ja que com he dit abans, amb un supercomputador es poden simular coses molt diverses.

Estem parlant del BSC, Barcelona Supercomputing Center. En aquest centre, treballen enginyers i investigadors de tot el món amb l'ajut del supercomputador construït el 2017, el MareNostrum 4.

Aquest superordenador destaca sobre els demès per una raó, i és el lloc on està col·locat. Està situat dins d'una capella, i és així com va guanyar el premi al supercomputador més bonic del món. No està situat allí per cap raó en concret, simplement l'espai estava buit i les dimensions de la sala eren les adequades per a emmagatzemar els 48 racks.

Però com us podeu imaginar, el MareNostrum 4 no és la primera versió d'aquest potent supercomputador. Des de l'any 2004 s'han anat desenvolupant diferents versions, cada cop més potents i més sofisticades, començant pel MareNostrum 1, també ubicat en la mateixa capella al igual que totes les versions posteriors.

3.1 MareNostrum 1

Aquest supercomputador va ocupar un dels primers llocs a la llista dels 500 supercomputadors més potents del món, i no es d'estranyar, ja que pel seu temps era una màquina revolucionaria.

Comptava amb 163 racks i cada rack contenia 14 nodes. Amb un total de 4812 processadors IBM PowerPC 970FX que funcionaven a una freqüència pic de 2,2GHz desenvolupava una potència computacional de 42,3 TeraFlops. Ara pot semblar una potència bastant limitada, ja que aquesta potència la podríem tenir perfectament a casa nostra connectant uns quants ordenadors i fent-los treballar en paral·lel, però al 2004 aquest era un poder computacional extremadament alt i molt difícil d'assolir per a qualsevol supercomputador perquè com ja he mencionat, el MareNostrum 1 es trobava entre les màquines més potents del planeta.

També comptava amb 9.6 TeraBytes de memòria RAM i amb 236 Terabytes de emmagatzematge intern. El punt fort d'aquest supercomputador eren les connexions entre els racks, ja que utilitzaven una tecnologia de baixa latència anomenada Myrinet, actualment patentada per la companyia Myricom.

Tenia un consum elèctric total de 650 KW, i tenint en compte la relació potència-consum, és infinitament menys eficient que l'actual MareNostrum.

[1]Informació elaborada a partir de dades de la web del BSC (bsc.es)

També tenia un sistema de connexió usuari-màquina molt diferent a l'actual, ja que s'accedia físicament al propi supercomputador i no remotament com es fa ara utilitzant Secure Shell (SSH).

Secure Shell es un protocol que s'inicia normalment al port 22 de la màquina i permet establir una connexió segura i donar ordres a la màquina a distància.

Comptava amb un rack d'operacions el qual estava conformat per un monitor, dos nodes, tres nodes remots, un xassís i quatre switches Cisco per a poder garantir la connexió per cable utilitzant el sistema Myrinet amb tots i cada un dels racks computacionals.

A part de la tecnologia Myrinet, també s'utilitzava la tecnologia actualment coneguda com a Gigabit. La màquina comptava amb un rack únicament dedicat a gestionar i interconnectar mitjançant switches Ethernet tots els altres racks computacionals i d'operacions.

Resumint, el supercomputador comptava amb quatre tipus de racks diferents, els racks amb tecnologia Myrinet, els racks amb tecnologia Gigabit, els racks de tipus «server» els quals servien per a emmagatzemar informació i dades i, finalment, el rack d'operacions, el qual permetia al usuari interactuar amb la màquina.

En termes de refrigeració, aquesta màquina no necessitava refrigeració addicional com sí que la necessita la nova versió del MareNostrum. Això és degut a la seva «limitada» potència, ja que aquesta no era suficient per a sobre escalfar els components imprescindibles per al funcionament.

3.2 MareNostrum 2

Dos anys després, el Juny del 2006, el MareNostrum 1, seria ampliat i se li afegiria més capacitat, tant computacional com d'emmagatzematge. És així com naixeria el MareNostrum 2.

Aquest, es basava en la mateixa tecnologia Myrinet que el MareNostrum 1, comptava amb el doble de processadors IBM PowerPC, aquest cop funcionant a 2,3GHz. És així com va assolir una potència pic de 94,21 TeraFlops, més del doble que el seu antecessor, el MareNostrum 1.

La memòria RAM també va ser ampliada a 20 TeraBytes i la memòria interna va ser ampliada fins a 480 TeraBytes. És així com aconseguirien tractar amb projectes de majors dimensions i complexitat.

Com ja s'ha mencionat abans, seguia utilitzant les dues connexions de baixa latència principals que es van utilitzar dos anys abans en la primera versió del computador. Estem parlant de la xarxa Myrinet i la xarxa Gigabit Ethernet, dues tecnologies extremadament eficients i ràpides a l'hora de intercanviar informació.

El número de racks utilitzats es va disminuir considerablement perquè la nova màquina era molt més eficient que l'anterior. Van passar d'utilitzar 163 racks a

només utilitzar-ne 44, quatre dels quals estaven dedicats a les connexions Myrinet i un a les connexions Gigabit.

La màquina també comptava amb 20 servidors d'emmagatzematge, els quals ocupaven 7 dels 44 racks disponibles i afegien 280 TeraBytes extra al sistema global de fitxers Paral·lels. Aquesta tecnologia de fitxers permetia una visió global del sistema de dades i un accés paral·lel, és a dir, permetia accedir a les dades des de diferents nodes a la vegada. Això va accelerar la velocitat de computació a nivells descomunals.

A l'igual que la primera versió del MareNostrum, aquest també comptava amb un rack d'operacions des del qual es podia accedir al sistema de fitxers de la màquina i interactuar amb aquesta.

Aquest rack no havia canviat respecte a la versió anterior i seguia disposant d'un monitor, 4 switches Cisco per a garantir la connexió amb la resta de la màquina, 2 nodes físics i 3 nodes remots.

3.3 MareNostrum 3

Sis anys després, a l'agost del 2012, el MareNostrum 3 entrava en funcionament. Aquesta versió havia augmentat exponencialment les prestacions que oferia el seu antecessor, el MareNostrum 2.

I no estem parlant de que la potència s'hagués augmentat dos vegades, sinó que comptava amb una potència més d'onze vegades superior, arribant a un pic de 1,1 PetaFlops, és a dir, 1100 TeraFlops.

Aquesta potencia era desenvolupada gràcies als nous processadors Xeon Phi i SandyBridge, aquest cop proporcionats pel gegant de la computació i la fabricació de hardware Intel.

Cada un dels 3056 nodes heterogenis repartits en els 52 racks del computador, comptava amb 2 processadors Xeon Phi i dos processadors SandyBridge funcionant a una freqüència de 2,6 GHz. Això donava un total de 15280 xips treballant alhora.

A tota aquesta potència computacional se li afegien 115,5 TeraBytes de memòria RAM i 3 Petabytes (3000 TeraBytes) de memòria d'emmagatzematge, per així assegurar que cap byte d'informació es perdia o no s'emmagatzemava bé.

Aquest cop la xarxa Myrinet, ja obsoleta, es va substituir per una altra tecnologia de comunicació de baixa latència anomenada Infiniband FDR10. Aquesta nova tecnologia permetia enviar i rebre les dades de cada un dels nodes de manera ràpida i precisa. Mentrestant, la xarxa Gigabit Ethernet seguia donant servei igual que el primer dia ja que és una tecnologia que es segueix utilitzant avui en la gran majoria de sistemes d'intercomunicació.

Aquest cop ja no s'utilitzava un rack d'operacions físic, i qualsevol interacció amb la màquina es feia a través del protocol Secure Shell, establint una connexió segura amb la consola de comandaments Linux allotjada en el supercomputador.

Tota la informació que l'usuari enviava a través d'aquest protocol, arribava amb èxit als nodes utilitzant 4 dels 52 racks disponibles. Aquest 4 racks estaven dedicats a la connexió de tipus Infiniband, la qual connectava tots i cadascun dels nodes disponibles per al càlcul.

El consum total de l'aparell era de 1,458 MW. Pot semblar molt elevat, ja que es el consum d'unes 1000 llars aproximadament, però tenint en compte l'augment del poder de computació, es pot observar que cada cop el sistema és més eficient en relació a la potència que ofereix.

3.4 MareNostrum 4

Aquesta es la màquina que hi ha instal·lada a dia d'avui. Va entrar en funcionament a finals de juny del 2017.

Ha patit canvis diversos respecte al MareNostrum 3, ja que a banda d'augmentar la potència computacional, també s'han afegit nous blocs dedicats a altres àmbits com, per exemple, les tecnologies emergents.

En aquesta versió del MareNostrum, tenim un augment de potència que el propulsa fins als 13,19 PetaFlops de poder computacional en el seu pic de potència. Tenint en compte que aquesta versió es 10 vegades més potent que el seu antecessor, el MareNostrum 3, podem observar que l'evolució en la potència i el desenvolupament dels supercomputadors no és lineal, sinó que cada cop es desenvolupen de forma més ràpida i eficient.

Això s'ha aconseguit implementant un hardware molt específic que permet a l'última versió del MareNostrum assolir aquestes xifres tant impressionants.

En primer lloc, comptem amb 3456 nodes repartits en 48 racks. Una vegada més, s'utilitzen els xips del arxiconegut fabricant Intel. Aquest cop, cada node compta amb 2 Intel Xeon Platinum amb 24 cores (processadors) cada xip. Això dona un total de 165.888 processadors a la disponibilitat de l'usuari. Però la màquina no només disposa de xips Intel sinó que també utilitza altres tecnologies com el ARM.

La memòria RAM també ha estat augmentada fins a 390 TeraBytes, és a dir, més de 3 vegades la capacita de memòria volàtil del MareNostrum 3.

La capacitat d'emmagatzematge també es superior, ja que compta amb 14 PetaBytes d'emmagatzematge intern del propi computador sumats als 24,6 PetaBytes del centre de Big Data del BSC. Això dona un total de 38,6 PetaBytes disponibles per a fer recerca i emmagatzemar informació que pot ser d'utilitat a l'hora de realitzar simulacions.

El MareNostrum 4, com ja he dit abans, està dividit en diferents blocs, ja que no tots els estudis i simulacions requereixen del mateix hardware per a ser executats. És així com el MareNostrum 4 està dividit en 2 blocs principals.

En primer lloc, tenim el bloc de propòsit general. Aquest és el bloc que disposa de més poder computacional i és capaç d'executar les tasques més exigents i costoses. Aquest bloc, està format pel conjunt de xips Intel Xeon Platinum. Encara que la seva potencia és 10 vegades superior a la de la versió anterior del computador, el consum energètic només ha augmentat un 30%.

En segon lloc, disposem del bloc de tecnologies emergents, que a la vegada està format per 3 sub-blocs amb tecnologies diferents cada un.

El primer sub-bloc es el MN4 CTE-Power. Aquest clúster està format per processadors IBM POWER9. No s'utilitzaven processadors IBM des del MareNostrum 2. Aquest clúster també compta amb una sèrie de processadors gràfics del fabricant de hardware Nvidia. En concret compta amb targetes gràfiques Nvidia Volta. Aquest hardware és el mateix que utilitza el supercomputador més potent del món, el *Summit* a EEUU. En total aquest bloc del MareNostrum 4 té una potència superior a 1,5 PetaFlops.

El segon sub-bloc s'anomena MN4 CTE-AMD, i com el propi nom indica, està format per xips de la marca AMD, que és la competència directa de Intel. En concret compta amb xips AMD Rome, acompanyats d'una secció de processadors gràfics AMD Radeon Instinct. Aquest sub-bloc de la màquina té una potència d'uns 520 TeraFlops és a dir, 0,52 PetaFlops.

L'últim sub-bloc es el MN4 CTE-ARM. Està format per un conjunt de processadors de tecnologia ARM, en concret utilitza processadors del fabricant SnapDragon. Els processadors ARM són els que podem trobar en el nostre telèfon mòbil o en la nostra tauleta. Són processadors extremadament eficients, ja que suposen un consum mínim per al computador, i en total desenvolupen una potència de 650 TeraFlops o el que es el mateix, 0,65 PetaFlops.

Totes aquestes tecnologies estan interconnectades gracies a la tecnologia d'alta velocitat i baixa latència Omnipath. Aquesta és una tecnologia d'intercomunicació desenvolupada per Intel i que permet una transferència d'informació a altíssimes velocitats. A part també segueixen utilitzant la tecnologia Gigabit Ethernet per a transferències que no requereixen de tant poca latència.

A l'igual que les versions anteriors del supercomputador, aquest està connectat amb els centres d'investigació i universitats Europees a través de les xarxes RedIris i Geant.

3.5 MareNostrum 5

Aquesta màquina encara no està funcionant, però ja està dissenyada i en construcció. En la meva visita al Supercomputing Center vaig poder veure les instal·lacions d'emmagatzematge i de subministrament elèctric.

Per desgràcia, aquest supercomputador no podrà ocupar l'espai de la capella com els seus antecessors, ja que ocupa un espai bastant més gran que els anteriors. S'ha construït una sala especialitzada per a mantenir el superordenador a una temperatura òptima per al seu funcionament.

També vaig poder visitar el sistema de refrigeració nou que s'està construint. Es tracta d'un sistema de refrigeració liquida complet format per filtres que mantenen el líquid refrigerant lliure d'impureses que puguin embossar qualsevol tub i un sistema de distribució i refredament d'aquest líquid per a portar-lo cap a cada un dels xips.

S'espera que el MareNostrum 5, quan estigui totalment instal·lat, tingui una potència pic de 314 PetaFops, és a dir, més de 23 vegades la potència de l'actual supercomputador.

La instal·lació d'aquest supercomputador tindrà un cost de 220 milions d'euros. D'aquests, 115 seran invertits en la compra i instal·lació dels components de la màquina i aproximadament 70 seran dedicats als costos d'operació fins al 2028, any que es preveu substituir el MareNostrum 5 per una màquina encara més avançada.

SIMULACIONS DE FLUIDS: LLEIS, EQUACIONS, MOVIMENTS, I VARIABLES (marc teòric)

4 Introducció a les simulacions utilitzant la supercomputació

Un supercomputador s'utilitza principalment per a executar complexes simulacions que serien impossibles d'executar amb un ordenador normal i corrent, ja que en termes generals, estan formades per grups d'iteracions (repeticions en termes de programació) molt complexes i que requereixen molts recursos perquè creixen exponencialment i necessiten una gran capacitat per a suportar aquest creixement de les dades.

Aquestes simulacions ens ajuden a anticipar-nos a un problema i buscar-hi una solució abans de que aquest problema esdevingui més gran i més complex. Gràcies a que la humanitat ha estat molt temps estudiant i recollint dades sobre el comportament de grans fenòmens naturals, avui en dia podem aplicar aquests coneixement a les noves tecnologies que tenim disponibles.

Un bon exemple serien els desastres naturals. Gràcies a entendre com funciona un terratrèmol, podem anticipar-nos per a poder evacuar a la població abans de que el sisme arribi a qualsevol localitat poblada.

En la meva visita al centre de supercomputació, mentre tractàvem el tema de la dinàmica de fluids, ens van explicar com havien aplicat aquest tipus de simulació en un cas real. Ens remuntem a l'any 2021, quan una terrible erupció volcànica de magnitud 3 va assetjar la illa de La Palma. Abans de que els gasos nocius arribessin a les zones poblades, es va ficar en marxa un protocol al centre de Supercomputació per a poder simular la erupció d'una manera rapida i precisa utilitzant el MareNostrum 4. D'aquesta manera, coneixent les corrents d'aire de la zona i el comportament dels gasos del volcà van poder predir amb exactitud quines zones serien les afectades i es va poder desallotjar aquesta part de la població de manera ràpida i segura abans de que qualsevol d'aquests esdeveniments passés a la vida real. Tot gracies a la supercomputació i a la simulació de fluids.

Però no només podem simular catàstrofes naturals, sinó que podem aplicar la supercomputació en camps com la meteorologia, l'astronomia o la biologia.

Gràcies a la supercomputació, podem veure el pronòstic del temps cada dia al nostre canal de televisió preferit, ja que les dades que capten els satèl·lits són tractades per un supercomputador.

També podem predir alineacions d'astres o col·lisions de meteorits gràcies a un superordenador. Segur que alguna vegada haureu sentit a les notícies «d'aquí uns dies passarà un asteroide relativament a prop de la Terra», doncs ha estat un supercomputador el que ha simulat la trajectòria amb anterioritat.

Però no ens cal anar a escales tan grans per a utilitzar un supercomputador sinó que ho podem fer amb coses tant petites com una seqüència d'ADN. Gràcies a la supercomputació, recentment s'estan trobant pertorbacions genètiques que

causen malalties greus a la població. Utilitzant aquesta informació, podem simular el resultat d'un tractament sense utilitzar un esser viu real i veure si tindria un impacte positiu o pel contrari negatiu per al subjecte estudiat.

També les empreses privades utilitzen el MareNostrum per al desenvolupament dels seus productes. Aquest és el cas de Seat, la qual ha utilitzat les instal·lacions del MareNostrum 4 per a millorar l'aerodinàmica dels seus vehicles i, per tant, fer-los més eficients energèticament. Gràcies a models de càlcul que ens permeten simular com es comporta un fluid a través d'un sòlid, Seat ha pogut disminuir les turbulències que creen els retrovisors dels seus cotxes al travessar un fluid d'aire a gran velocitat.

5 Definició d'una simulació de fluids

Una simulació de fluids aplicada a la supercomputació, és un mètode que aprofita el gran poder computacional del que disposa una d'aquestes màquines i que ens permet veure i analitzar el comportament de qualsevol fluid que puguem trobar a la realitat de forma macroscòpica.

Quan ens referim a fluids, no només estem parlant d'aerodinàmica, sinó qualsevol grup de partícules que s'adaptin al seu entorn mantenint el volum. Podríem simular perfectament la corrent d'un riu, el sistema d'una presa d'aigua, el funcionament d'una turbina o la eficiència d'un perfil alar.

Gràcies a aquestes simulacions aerodinàmiques, ens estalviem hores i hores de túnel de vent i estudis fets amb elements reals.

Aquestes simulacions tenen en compte la majoria de variables de la realitat, i segueixen les mateixes lleis físiques que seguiria un fluid real.

5.1 Variables a tenir en compte a l'hora d'executar una simulació de fluids

5.1.1 Densitat

La densitat d'un fluid és la quantitat de massa per unitat de volum, és a dir, quanta massa hi cap en un cert espai. Com podem observar en la vida real, la mel d'abella, és un fluid més dens que l'aigua, ja que en un centímetre cúbic de mel, hi ha més massa que en un centímetre cúbic d'aigua.

Com major és la densitat del fluid, més compactes estaran les partícules perquè n'hi ha d'haver més en un espai més reduït. S'expressa en kg/m^3.

5.1.2 Pressió

La pressió és de gran importància en la simulació de fluids, ja que és una de les variables principals que determinaran el moviment d'aquest.

Les partícules que estan sotmeses a menys pressió, tendeixen a anar cap a les zones sotmeses a més pressió, i les zones de baixa pressió tendeixen a atreure

les zones d'alta pressió. Es així com funciona un perfil alar d'un avió o un aleró d'un cotxe de competició.

Aquesta variable és la causant del efecte «Venturi».

5.1.2.1 Efecte Venturi

L'efecte Venturi, és un fenomen recurrent en les simulacions de fluids i les seves dinàmiques, ja que és el responsable de que la velocitat d'un flux o corrent de qualsevol fluid augmenti.

Aquest efecte es produeix quant el fluid ha de traspassar una zona amb menys cavitat o volum que la resta de l'entorn. Podem ficar com a exemple una canonada més estreta que la resta del sistema de canonades. Com ha de passar el mateix volum de fluid per una cavitat més petita, la pressió augmenta i com a conseqüència, la velocitat del fluid és superior.

Amb aquest principi funcionen les turbines d'un avió. Absorbeixen una quantitat d'aire exageradament gran per una entrada del motor amb una obertura de grans dimensions i la forcen a passar per un petit conducte a l'altre extrem de la turbina a gran pressió. Això crea un flux d'aire concentrat a una pressió molt elevada i per tant a una gran velocitat, això crea la propulsió que necessita l'aparell per a moure's amb el principi d'acció reacció. És per això, que les simulacions aerodinàmiques en l'àmbit aeroespacial són tant útils.

5.1.3 Temperatura

Aquesta és una de les principals variables, ja que afecta directament a la viscositat i densitat del fluid. Per tant, també afecta directament al seu comportament. La temperatura és d'especial importància en les simulacions de dinàmica de fluids enfocades a la meteorologia, ja que d'aquesta manera, gràcies a les lectures dels termòmetres i amb l'ajut d'un supercomputador es pot predir de forma precisa el moviment de les grans masses d'aire.

5.1.4 Velocitat

Aquesta variable va lligada a la pressió, donat que aquesta és creada gràcies a la diferencia de velocitats. Si fiquem com a exemple un perfil alar, podem observar que entre la part superior de l'ala i la part inferior hi ha una gran diferencia de velocitat de l'aire. Aquesta diferencia de velocitat és la causant de la diferencia de pressió que fa que l'ala creï sustentació. Aquesta variable també està relacionada amb l'efecte «Venturi» anteriorment mencionat, ja que és aquest efecte un dels responsables de la diferencia de velocitats en la part superior i inferior. Si fiquem una vegada més l'exemple d'un perfil alar, la inclinació de l'ala força al fluid a passar enganxat a la superfície alar inferior. Això, en conjunt amb l'efecte Venturi, crea un augment de la velocitat de l'aire. En canvi, degut a la inclinació d'aquesta, l'aire és incapaç d'enganxar-se a la part superior de l'ala. Per tant, la velocitat i la pressió disminueixen creant una gran diferència de velocitats i generant sustentació.

6 Equacions, lleis i condicions a tenir en compte

Per a poder calcular els nous resultats i moviments de les partícules del fluid en cada iteració de la simulació, hem de fer ús d'algunes equacions i fórmules. D'aquesta manera, podem obtenir les dades necessàries i els resultats òptims per a poder distribuir les partícules, creant així un moviment conjunt del fluid.

6.1 Equació de Navier-Stokes

Aquesta, és l'equació principal que descriu el moviment i el comportament d'un conjunt de partícules com a fluid i es divideix en dos parts:

6.1.1 Equació de continuïtat

$\nabla \cdot V = 0$ (on ∇ és l'operador nabla i V es la velocitat del flux de fluid)

Una altra manera de veure aquesta equació es la següent:

$(\partial V_x/\partial x) + (\partial V_y/\partial y) + (\partial V_z/\partial z) = 0$

$(\partial V_x/\partial x)$ representa la derivada parcial (en una funció amb diferents variables, la derivada parcial és la derivada de cadascuna d'aquestes variables tenint en compte les altres com a constants), de la component X de la velocitat respecte a la coordenada X.

$(\partial V_y/\partial y)$ representa la derivada parcial de la component Y de la velocitat respecte a la coordenada Y.

$(\partial V_z/\partial z)$ representa la derivada parcial de la component Z de la velocitat respecte a la coordenada Z. (Aquesta última només s'aplica en el cas de que es tracti d'una simulació en 3 dimensions).

6.1.2 Equació de momentum

$\rho (\partial V/\partial t + (V \cdot \nabla)V) = -\nabla P + \mu \nabla^2 V + \rho g$

On «ρ» és la densitat del fluid, «$\partial V/\partial t$» és la derivada parcial en el temps, «P» representa la pressió del fluid, «μ» és la viscositat del fluid i «g» és la acceleració causada per la gravetat.

6.2 Llei de conservació de la massa

Aquesta llei estableix que la massa total d'un sistema fluid no canvia amb el temps i es manté constant a mesura que les iteracions de la simulació van augmentant.

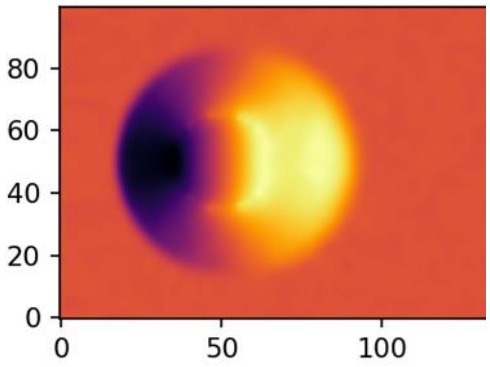
Aquesta llei s'expressa com $\partial \rho/\partial t + \nabla \cdot (\rho V) = 0$.

6.3 Condicions de frontera o límits

Aquesta condició especifica com han d'interactuar les partícules del fluid quan col·lideixen amb els límits o superfícies sòlides de la simulació.

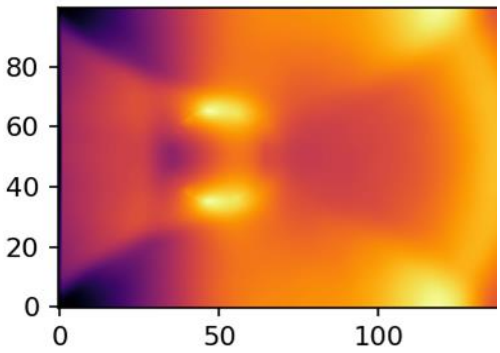
Generalment, en el codi de la simulació es dediquen unes quantes línies de codi per a definir el moviment del fluid en col·lidir amb un sòlid. Normalment, aquest comportament es limita a exercir la força que aplica el fluid sobre el sòlid però en direcció contraria, creant un moviment de rebot i així impedint que el fluid travessi el sòlid.

Aquí es pot veure un clar exemple d'aquesta condició en una simulació programada per mi utilitzant el mètode de Lattice Boltzmann:



Simulació LBM Part Pràctica 2

Les ones de partícules que conformen el fluid surten des d'un punt conformant una ona circular i enviant partícules en totes les direccions.



Simulació LBM Part Pràctica 1

Al col·lidir amb els límits de la simulació, s'aplica la mateixa força en direcció contraria i per tant les partícules reboten.

6.4 Condicions de mobilitat de les partícules d'un fluid

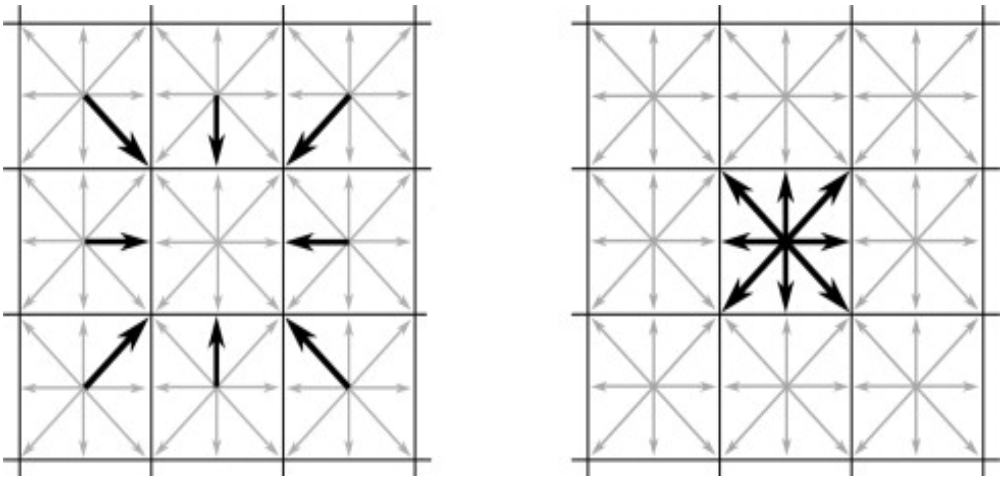
Aquesta condició és essencial per a què una simulació de fluids sigui realista i segueixi les principals lleis i les físiques de la realitat.

Aquesta, diu que a l'hora de moure les partícules de la simulació per a crear el moviment general del fluid, no poden deixar cap espai buit, és a dir, un grup de partícules concentrades en un punt, no poden moure's totes a les posicions que tenen al voltant ja que al centre quedaria un espai buit i, per tant, s'hauria de crear massa del no res.

Això no es pot fer, ja que incompleix la llei de conservació de la massa. Com ja s'ha explicat abans, aquesta llei dictamina que la massa total d'un fluid al llarg d'una simulació no canvia i es manté constant durant tot el procés.

Simplement aplicant una mica de sentit comú, podem deduir que no es pot crear massa del no res, ja que no es físicament possible.

Això també aplica en sentit contrari, és a dir, un grup de partícules no es pot moure al mateix punt perquè es crearia una superposició de dues o més partícules ocupant un sol espai i això (en la física convencional) tampoc és possible.



Sistema 1: Condicions de mobilitat (Font: medium.com)

7 Espirals i divergències[2]

Aquests dos tipus de moviments poden descriure a la perfecció el comportament real d'un fluid.

Les espirals i les divergències combinades són capaces de crear vòrtex tant laminars com turbulents, i crear a nivell macroscòpic moviments i sistemes de fluids complexos.

7.1 Espirals

Podem definir com a «espirals» tots aquells moviments que provoquen que les partícules d'un fluid, girin en un mateix sentit en qualsevol de les 3 dimensions.

Aquestes espirals són les principals causants dels fluids turbulents als quals els dedicarem un apartat més endavant.

Aquí podem observar una simple representació d'un moviment de tipus espiral. Les partícules giren de manera ordenada seguint un únic centre per a totes elles.

Il • lustració 1: Moviment espiral (Font: But How DO fluid Simulation works? – youtube.es)

7.2 Divergències

Les divergències es defineixen com a tot aquell moviment que provoca que les partícules es distanciïn o s'apropin d'un mateix centre.

Aquest tipus de moviment és el responsable de que a mesura que el fluid es va distanciant del centre es torni menys dens, i a mesura que les partícules es concentren en un mateix centre, la densitat sigui més alta.

Aquesta, a l'igual que la de dalt, és una simple representació d'unes partícules en divergència

Il • lustració 2: Moviment divergència (Font: But How DO fluid Simulation works? – youtube.es)

[2]Informació extreta de «But How DO fluid Simulation works?» https://www.youtube.com/watch?v=qsYE1wMEMPA&t=470s

7.3 Combinació d'espirals i divergències per a crear moviments complexos

Aquests dos moviments per si sols no tenen la capacitat de crear moviments canviants que puguin descriure la realitat en una simulació. Per tant, és combinen per a poder representar vòrtex complexos.

Podríem comparar-los amb l'eix X i l'eix Y. Per si sols només poden crear vectors en una sola direcció però si els combinem podem obtenir qualsevol vector en 2 dimensions.

Un computador tracta aquests dos comportaments d'una manera peculiar. Primer, són combinats en un camp vectorial. En aquest camp, les partícules reaccionen a les forces aplicades per aquests dos moviments. Una vegada ja aplicat aquest procediment, per a poder visualitzar la simulació final, tenim que extreure la divergència de les espirals ja que si no, es crearia o es destruiria matèria al concentrar-se totes les partícules en un punt o separar-se de cop. Estaríem incomplint les condicions de mobilitat de les partícules explicades anteriorment.

Aquí ens topem amb el primer problema, un computador no és capaç de separar la divergència de les espirals simplement extraient-ne una i després l'altra, sinó que tenim que seguir un procés una mica més complex.

Aquest procés s'anomena «Descomposició d'Helmholtz».

7.4 Descomposició d'Helmholtz

D'acord amb el teorema de Helmholtz o també conegut com el teorema fonamental de càlcul de vectors, qualsevol sistema vectorial es pot representar utilitzant dos subsistemes.

El primer subsistema és un sistema que no té moviment de tipus espiral, és a dir, només està compost per divergència.

El segon, en canvi, només està compost per moviment en espiral, o sigui que no té divergència.

La combinació d'aquests dos subsistemes dona lloc a un sistema de vectors més complex. Com ja hem dit abans, no hi ha manera de computar el subsistema de vectors lliure de divergència així que el procés de descomposició es tracta de restar el subsistema de moviment divergent al sistema total, obtenint així el camp vectorial nomes amb moviment de tipus espiral.

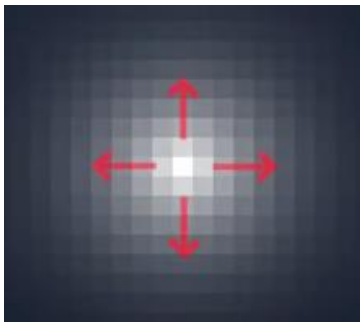
8 Difusió[3]

A l'hora de simular el comportament d'un fluid, és essencial aplicar de manera correcta la difusió d'aquest. Però, què és la difusió d'un fluid?

La difusió és el terme pel qual ens referim a la dispersió d'un fluid i, per tant, a la pèrdua de pressió a canvi de la ocupació de més volum.

Per poder dispersar un fluid correctament, tenim que resoldre un sistema d'equacions lineals per a poder convertir gradualment els valors que envolten una partícula a un límit que tendeix a 0.

Òbviament, no pot ser 0, ja que faríem desaparèixer massa i, per tant, estaríem violant la llei de conservació de massa que a l'hora de simular el comportament d'un fluid és de les més importants.



Sistema 2: Difusió (Font: But How DO fluid Simulation works? – youtube.es)

8.1 Equacions per a dispersar fluids

En primer lloc, disposem d'aquesta equació on el nou valor de la partícula (dc) es calcula restant la diferència de valors entre la partícula actual (Sn) i les partícules que envolten aquesta (dn), multiplicades per la quantitat de canvi. Aquesta és una variable escalar que indica la velocitat de la difusió. Com més alta sigui, més ràpid es dispersarà el fluid. Aquesta fórmula però, té un problema, i és que en algun moment acabarà sent 0 i, per tant, desobeirà la llei de conservació de la massa. Això es soluciona amb la següent equació:

$$d_c = d_n - ks_n + kd_n$$

Equació 1

Aquesta equació, en comptes d'utilitzar el valor actual per a calcular el següent valor, calcula el següent valor i el compara amb el valor actual. Bàsicament, reverteix la dispersió del fluid creant una concentració. Aquesta equació, a part, no pot donar 0 i, per tant, ja s'està respectant la llei de conservació de la massa.

$$d_n = \frac{d_c + ks_n}{1 + k}$$

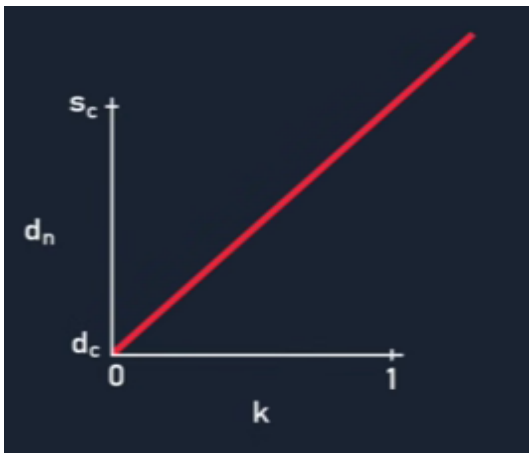
$$d_n(x,y) = \frac{d_c(x,y) + k\left(\frac{d_n(x+1,y)+d_n(x-1,y)+d_n(x,y+1)+d_n(x,y-1)}{4}\right)}{1+k}$$

Equació 2

Aquesta és la equació final on Sn (el valor actual) s'ha substituït per la mitjana de les 4 partícules que l'envolten, Es així com es calculen primer els valors futurs per a poder determinar el valor present. Per tant, es reverteix en el temps la simulació deduint a partir de valors futurs els valors presents.

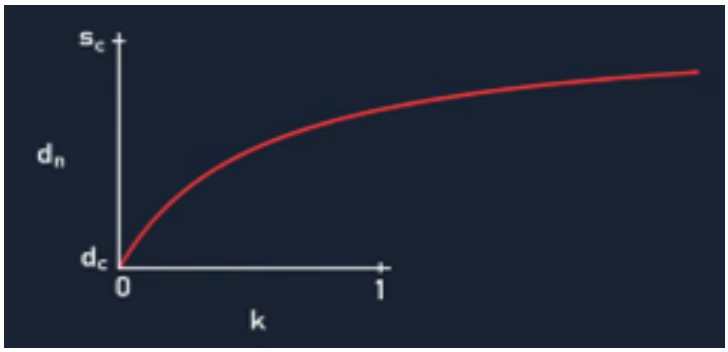
[3]Informació extreta de «How fluid simulation works» https://www.youtube.com/watch?v=qsYE1wMEMPA&t=470s

El computador no té la capacitat de resoldre sistemes lineals com ho fem els humans, així que utilitza mètodes numèrics com l'aproximació. Bàsicament, agafa valors aleatoris i els va substituint a les equacions milers i milers de vegades fins que aconsegueix un valor que s'aproxima a la solució.



Gràfic 1: Augment de difusió lineal (Font: But How DO fluid Simulation works? – youtube.es)

En la primera equació no es respecta la conservació de la massa, ja que l'augment de difusió és lineal i, per tant, aquesta en algun moment desapareixerà.



Gràfic 2: Augment de difusió exponencial (Font: But How DO fluid Simulation works? – youtube.es)

La segona equació crea un límit on l'augment de difusió és exponencial i, per tant, crea un límit el qual no es podrà sobrepassar mai.

9 Fluids laminars i fluids turbulents

Existeixen dos tipus de difusions de moviments. Una difusió en termes de dinàmica de fluids és quan les partícules parteixen des d'un punt 0 i a mesura que la variable del temps («timesteps» en termes de programació) va avançant, aquestes partícules es van separant, reduint així la concentració del fluid i dispersant-lo per l'espai que aquest ocupa.

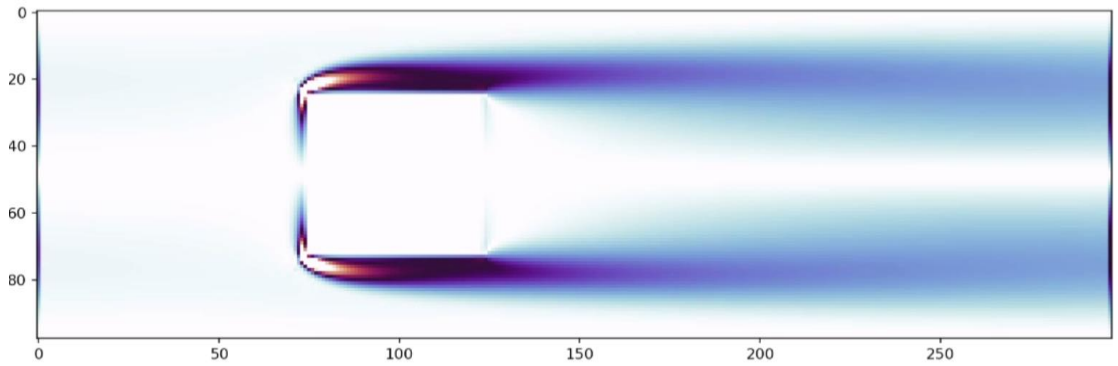
Un clar exemple es el tub d'escapament d'un cotxe. Els gasos procedents de la combustió, surten pel tub i ràpidament es dispersen per l'espai que aquest fluid ocupa.

Hi ha dos tipus de fluids a l'hora de tenir en compte aquest factor:

9.1 Fluids laminars

Aquest tipus de fluids, són aquells que a l'hora de dispersar-se ho fan de manera ordenada i, com el propi nom diu, en forma de làmines. En aquest tipus de dispersió podem observar clarament les diferents capes del fluid per separat i sense barrejar-se entre si.

Aquest tipus de fluid és molt més estable que un fluid turbulent, ja que no crea quasi alteracions en la pressió ni en altres variables que podrien afectar a l'estabilitat del propi fluid.



Simulació LBM Part Pràctica 3: Fluid laminar

Aquí podem observar de forma clara un fluid laminar que jo mateix he simulat en la meva part pràctica. Com podem observar, la difusió d'aquest és clara i no té pertorbacions de cap tipus.

Els fluids laminars són bastant difícils de trobar en el medi natural, ja que no han d'estar condicionats per quasi cap variable externa que pugui provocar una turbulència no desitjada.

Podem trobar aquest tipus de fluid, per exemple, en la sang que passa pels vasos sanguinis del nostre cos. Aquesta, ha de ser un fluid laminar ja que si no al cor li costaria molt esforç bombejar aquest fluid.

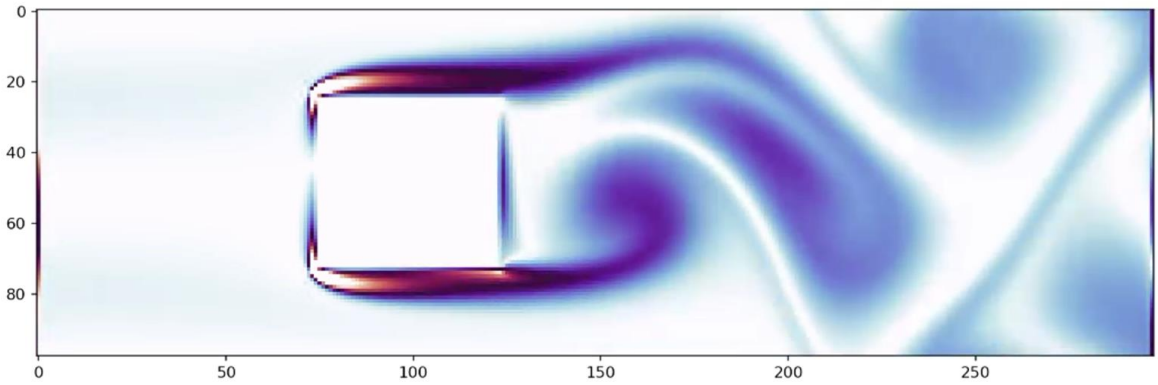
Aquesta, al no portar una velocitat extremadament alta, pot mantenir un ritme igual per a totes les partícules i, per tant, no crea turbulències en el seu curs.

9.2 Fluids turbulents

Aquest tipus de fluids són totalment diferents als que hem explicat anteriorment, ja que es caracteritzen per tenir un caràcter poc controlable i més aviat inestable.

Ja sigui perquè duen una gran velocitat o perquè estan condicionats per elements externs, tendeixen a crear vòrtex complexos i a barrejar les capes del fluid fins al punt en que és impossible identificar-les clarament.

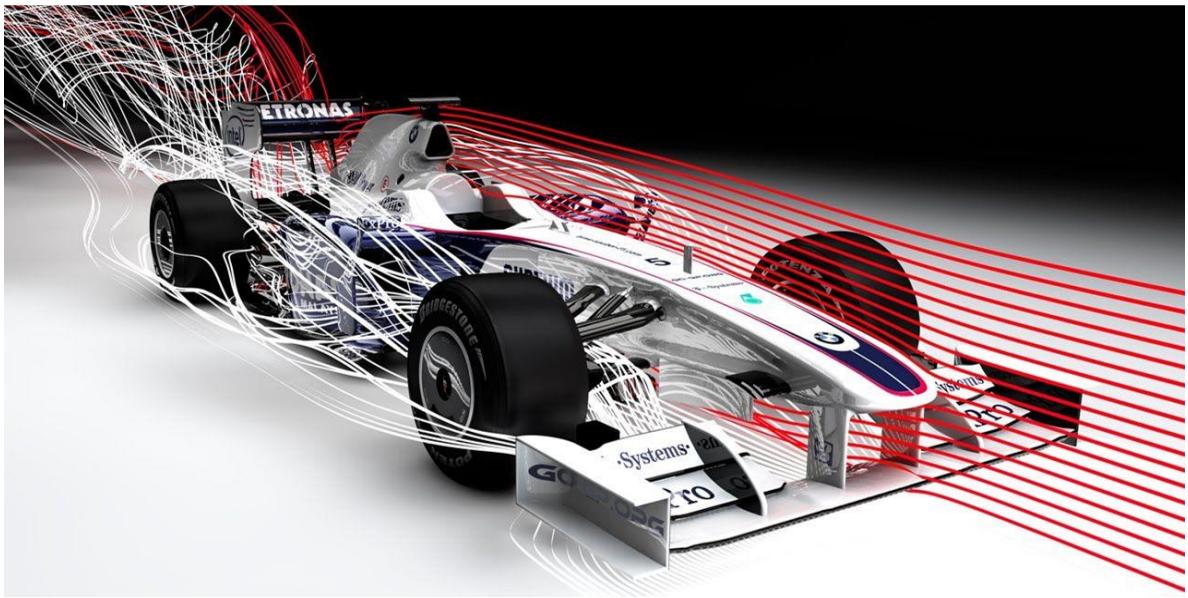
Aquests fluids son els que en major part podem trobar a la natura, ja sigui en el vent, en la corrent d'un riu o en el fum que produeix un incendi.



Simulació LBM Part Pràctica 4: Fluid turbulent

Aquí podem observar un clar exemple d'un fluid turbulent influenciat per elements externs que han estat introduïts a la meva simulació de la part pràctica mitjançant el que en termes de simulació es coneix com «soroll», és a dir, una errada numèrica que s'introdueix en el sistema de partícules expressament per a simular condicions externes a la pròpia simulació.

Que un fluid sigui turbulent, no vol dir que no serveixi o no es pugi aprofitar, tot el contrari, hi ha un munt d'invencions del esser humà que funcionen gràcies als fluids turbulents. Un exemple és el sistema aerodinàmic d'un cotxe de competició. Aquest, va equipat amb unes petites aletes que creen expressament vòrtex per a generar més adherència al terra, i així poder agafar les corbes amb més velocitat.



Il • lustració 3: Turbulència en un BMW Sauber F1.09 (Font: topgear.es)

Aquí podem observar com un fluid laminar, com és l'aire en repòs, al passar per la carrosseria d'aquest cotxe de Fórmula 1, es transforma en un fluid de tipus turbulent, ja que com podem observar, les capes del fluid no són clarament divisibles ni s'aprecien a primera vista.

10 Mètodes de simulació de fluids existents[4]

A l'hora de simular aquests fenòmens, tenim al nostre abast diferents mètodes i sistemes que podem utilitzar per a obtenir resultats precisos i fidels a la realitat.

Alguns d'aquests sistemes requereixen de més poder computacional, i també n'hi ha d'altres que poden funcionar amb un poder més limitat.

En aquest apartat s'explicarà el mètode escollit i perquè va ser triat.

A continuació s'enumeren i s'expliquen els 5 mètodes principalment utilitzats:

10.1 Dinàmica de Fluids Computacional (CFD)

La CFD és un mètode amplament utilitzat en la supercomputació per a simular el comportament de fluids com, per exemple, gasos i líquids aplicats en sistemes complexos.

Es basa en una malla tridimensional dividida en cubs de mida 1x1. Com més gran sigui la malla més resolució tindrà la simulació, però també serà més exigent en requeriments de hardware.

Per exemple, una malla de 300x300 serà menys exigent que una malla de 1000x1000, però a la vegada la resolució de la simulació serà més baixa i esdevindrà més difícil apreciar els petits detalls del sistema de fluids.

Les partícules es mouen a través de la malla seguint les lleis de conservació de la massa, la quantitat de moviment i, finalment, les equacions de Navier Stokes, tant la de momentum com la de continuïtat, les quals hem explicat anteriorment.

Aquest mètode s'utilitza en una gran varietat de problemes de sistemes de fluids, des de la producció de fuselatges per a avions i cotxes, fins a prediccions climàtiques i optimització de processos industrials. Com es pot observar, és un mètode molt general i que té una gran varietat d'usos, cap en específic.

10.2 Mètode d'elements finits per a fluids (FEM)

El mètode FEM es combina normalment amb l'estudi de la dinàmica de fluids per a analitzar problemes en els que es vol estudiar com reaccionarà un fluid en contacte amb una estructura sòlida complexa.

Un clar exemple d'aquest sistema el trobem en la refrigeració líquida de motors de combustió. Aquests sistemes, abans de ser construïts, són provats de forma virtual en simulacions de FEM, ja que s'ha de tenir en compte la transferència de calor del metall del motor al líquid refrigerant, en aquest cas el fluid.

Amb aquest mètode podem tenir constància de la temperatura del fluid i, per tant, és ideal per a fer simulacions on la transferència de calor és important per a resoldre el problema. Es pot aplicar, per exemple, en intercanviadors de calor

[4]Informació extreta del curs de Bojos per la Supercomputació impartit al BSC

per a produir electricitat, o com hem dit abans, en radiadors per a motors de combustió.

10.3 Mètode de Partícules suaus (SPH)

Aquest mètode utilitza un sistema de partícules suaus que col·lideixen entre si i reaccionen, creant moviments realistes i propis de la dinàmica i comportament d'un fluid.

Normalment, s'utilitza per a problemes amb superfícies lliures, com podrien ser les ones del mar o la corrent d'un riu.

Aquest mètode s'ha utilitzat en camps tan diversos com la producció cinematogràfica, la simulació de cossos d'aigua o la investigació en ciències de la terra.

10.4 Mètode d'Elements de vora (BEM)

El BEM és especialment utilitzat quan s'ha de simular la interacció de partícules de fluid amb un element sòlid, normalment submergit. Un exemple podria ser el casc d'un submarí, o una turbina submarina que aprofita les corrents per a produir electricitat.

En comptes d'enfocar-se en tot el camp que té possibilitat de simular-se, només s'enfoca en la superfície de l'estructura amb la que el fluid ha de col·lidir. D'aquesta manera, no consumeix tants recursos i es poden fer simulacions més precises i ràpides.

Com ja s'ha mencionat abans, s'utilitza especialment en el sector naval.

10.5 Malla de Boltzmann (LBM)

Aquest mètode, més conegut com LBM o Lattice Boltzmann Method, és un mètode àmpliament utilitzat per a simular fluids especialment complexos i que requereixen d'una observació a nivell microscòpic, encara que també es pot utilitzar per a simular col·lisions a una escala més gran.

El funcionament d'aquest mètode és senzill. Com el propi nom indica, es distribueixen un seguit de partícules fictícies que des d'un punt de vista general, totes juntes creen un cos amb les mateixes propietats d'un fluid.

Aquestes partícules, seguint les equacions de Boltzmann i jugant amb la probabilitat, es dispersen i col·lideixen entre sí per a crear un moviment conjunt.

Funciona especialment bé per a poder apreciar fluids turbulents, ja que podem potenciar aquesta turbulència únicament canviant el valor d'una variable dins de les equacions que defineixen el moviment d'aquestes partícules dins de la malla.

MÈTODE, LLENGUATGE I SIMULADOR (MARC PRÀCTIC)

11 Quin mètode vaig triar per a realitzar la meva simulació de fluids?

La tria del mètode a utilitzar no va ser una feina difícil, i aviat vaig tenir clar quin mètode utilitzar per a programar la meva simulació de fluids. Volia un mètode que no fos complicat de programar i que tampoc consumís massa recursos, ja que a pesar de tenir el MareNostrum 4 a la meva disposició, també volia provar si es podia executar una simulació de fluids, per petita que fos, en un ordenador de sobretaula com el que podem tenir a casa.

Es així com vaig triar el mètode Lattice Boltzmann (LBM) o Malla de Boltzmann. Aquest mètode no només em permetria executar la simulació des de la comoditat del meu ordenador personal, si no que també la podria passar al MareNostrum 4 via Secure Shell (SSH) quan volgués.

11.1 Com funciona el mètode LBM?

El mètode LBM és un dels més senzills d'entendre. Es basa en la probabilitat i en un sistema vectorial que es comporta en base a les equacions de Boltzmann.

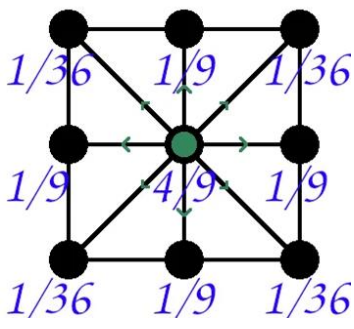
Aquest mètode representa el fluid utilitzant partícules. Cada una d'aquestes partícules és, simplement, un valor que es representa d'un color determinat. Aquestes partícules, com el propi nom indica, es troben repartides en una malla o una quadrícula. En el meu cas, he fet una simulació d'un perfil alar que està col·locat en una quadrícula de 100x200, és a dir, que consta de 20.000 partícules o 20.000 valors. En termes matemàtics, estaríem parlant d'una matriu de 100X200, és a dir, d'una matriu rectangular.

Les partícules, segons la llei de Boltzmann, tenen cert grau de probabilitat de moure's en segons quines direccions. Per a poder exemplificar-ho millor, podem atribuir a la partícula el valor de la peça del rei en els escacs. El rei es pot moure un quadrat en totes direccions i també es pot quedar quiet en el lloc en el que està. Doncs així es mouen les partícules de la simulació dins de la quadrícula o malla.

11.2 Sistema probabilístic

Les partícules tenen un cert nivell de probabilitat per a moure's en cada direcció. Té 4 possibilitats entre 9 de quedar-se al seu lloc, en tenen 1 entre 9 de moure's verticalment o horitzontalment, i en tenen 1 entre 36 de moure's cap als vèrtex.

Aquestes probabilitats han estat estudiades en base a les equacions de Boltzmann, observant i entenent el comportament dels fluids reals.

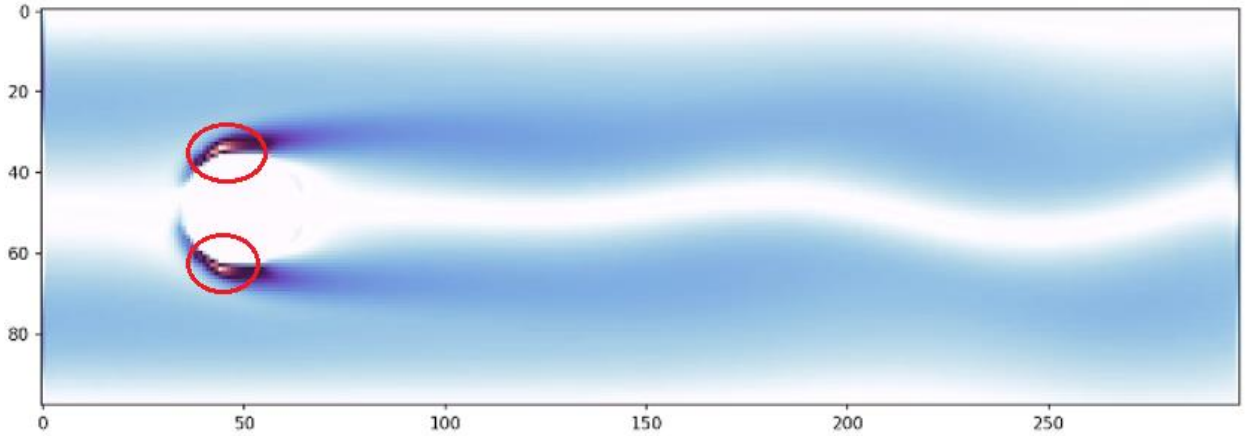


Sistema 3: Sistema probabilístic LBM (Font: medium.com)

11.3 Càlcul de vorticitat

Les simulacions de Lattice Boltzmann són ideals per a poder observar la vorticitat creada per un fluid a l'impactar contra un cos sòlid, líquid o gasós. La vorticitat és el valor que obtenen un grup de partícules de la simulació en funció de la capacitat que tenen per crear vòrtex o remolins, és a dir, la capacitat que tenen d'esdevenir un fluid turbulent.

Les simulacions de Boltzmann representen aquestes zones en funció d'un mapa de colors triat específicament per a aquest tipus de simulacions. En el meu cas, he triat el mapa «twilight» que dona un color o un altre en funció del valor que rep aquella quadrícula de la simulació. Aquest valor és la vorticitat que hem calculat a partir de les equacions de Boltzmann.

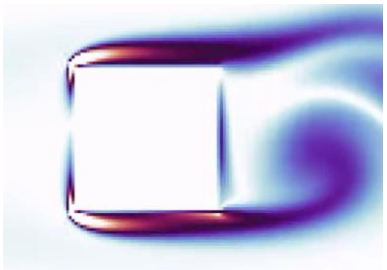


Simulació LBM Part Pràctica 5: Zones de vorticitat

En aquesta simulació de la meva part pràctica es representa un fluid passant al voltant d'un cilindre vist de costat. Es pot observar dos punts que destaquen clarament. Llegint la llegenda del mapa de colors «twilight» es pot observar que la vorticitat en aquells dos punts és major que en la resta del sistema de partícules. Això vol dir que en aquella zona, la pressió és superior i, per tant, es crea la turbulència que es pot veure posteriorment.

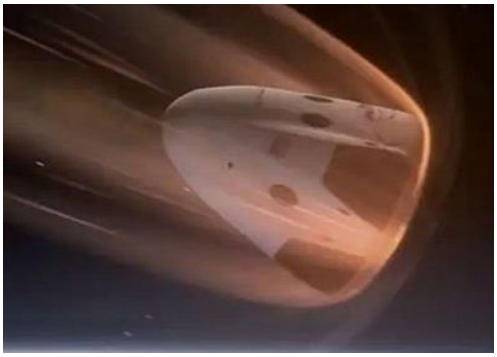
11.4 Diferència de temperatura respecte a la resta del sistema

Aquí en podem veure un altre exemple, on clarament, la vorticitat és major que en l'anterior simulació. Els vèrtex del cub creen més vorticitat que les vores del cilindre, ja que els angles són rectes.



Simulació LBM Part Pràctica 6: Fricció i temperatura

D'aquestes zones no només en podem treure dades de vorticitat i pressió, sinó que també en podem treure dades de temperatura. Aquestes zones, al tenir mes fricció amb l'aire, sofreixen un augment de temperatura.



Il • lustració 4: Dragon Capsule SpaceX (Font: quora.com)

Aquesta informació és molt útil, per exemple, a l'hora de dissenyar una càpsula espacial. Aquesta, en la seva reentrada a l'atmosfera, pateix un increment molt elevat de temperatura degut a la fricció de l'escut tèrmic amb l'aire. Si abans de construir-la podem provar-la en una simulació d'aquest tipus, podrem observar les zones que estaran més afectades per la fricció de l'aire i, per tant, les podrem reforçar i protegir.

12 Llenguatge utilitzat

El primer pas per a començar a programar era triar un llenguatge que em permetés executar totes i cada una de les ordres amb les quals el mètode comptava. En l'actualitat, la majoria de llenguatges coneguts com, per exemple, el Java o el C++, ja compten amb tots els recursos necessaris per a poder programar aquest tipus de simuladors. Tot i això, es va escollir el llenguatge Python per programar la simulació pels següents motius:

12.1 Aplicacions i usos

Aquest, és actualment un dels llenguatges de programació més utilitzats en tot tipus de projectes. La coneguda xarxa social Instagram està programada en Python, però també es poden fer jocs, aplicacions Web i, en aquest cas, simulacions enfocades a la supercomputació.

12.2 Optimització de la sintaxis

Aquest llenguatge té un clar avantatge respecte als altres perquè és relativament nou. Això fa que estigui optimitzat al màxim per a consumir els mínims recursos del computador. També es bastant amigable de cara al usuari, ja que té una sintaxis molt senzilla que utilitza terminologies del anglès com, per exemple, «print», «while», «and» o «or». Aquestes terminologies en altres llenguatges amb una sintaxis de més alt nivell, s'escriurien usant símbols o tecnicismes que el programador ha d'aprendre.

12.3 Biblioteques i paquets

Aquest llenguatge compta també amb una gran quantitat de biblioteques i repositoris. Una «biblioteca», en termes de programació parlant, es un conjunt de codi distribuït en paquets pre-programats que faciliten molt les tasques en tots els camps en els que es pot usar el llenguatge.

Tenim, per exemple, la biblioteca numpy, que ens permet operat amb un munt d'elements matemàtics. Sense aquesta biblioteca no podríem aplicar coses com, per exemple, les matrius essencials en la meva simulació de fluids.

12.4 Comunitat de desenvolupadors

Una altre avantatge d'aquest llenguatge és que consta d'una gran comunitat de desenvolupadors que estan constantment treballant per a millorar l'experiència de l'usuari final.

Si tens un problema a l'hora de programar, també hi ha una gran comunitat de programadors de Python en plataformes com StackOverflow que estan disposats a ajudar-te i a resoldre qualsevol dubte que tinguis.

13 Conceptes i terminologies per a poder comprendre el codi

Per a poder disposar d'una comprensió amplia i sense dubtes, en aquest apartat s'explicaran els termes que anirem utilitzant en el comentari del codi i que no s'hagin mencionat anteriorment.

Variable
Una variable és un contenidor el qual pot emmagatzemar informació com, per exemple, un número sencer (int), un número decimal (float), un text (string), una llista (array), una opció binaria, és a dir, 0 o 1, blanc o negre, o verdader o fals (bool) entre moltes altres.

Iteració
Una simulació d'aquest tipus funciona repetint el mateix procediment constantment i aplicant els resultats anteriors a la següent repetició. Aquest programa es repeteix 10.000 vegades per a poder representar la simulació completa. Per tant, consta de 10.000 iteracions.

Bucle
Aquest és el codi responsable de crear les iteracions que hem explicat abans. Podem especificar quantes iteracions volem, modificant els paràmetres d'aquest bucle.

14 Com vaig programar la meva simulació LBM

El primer pas va ser trobar el llenguatge, i una vegada trobat (com s'explica a l'apartat de dalt), el següent pas va ser trobar exemples de simulacions Laticce Boltzmann fetes en Python. En vaig trobar unes quantes i les vaig estudiar per entendre com s'aplicaven els fonaments del mètode LBM a la programació.

Una vegada vaig saber com fer aquesta conversió, vaig començar a programar el meu simulador.

El codi explicat pas a pas es el següent:

```
import matplotlib.pyplot as plt
import numpy as np
```

Aquí s'importa la biblioteca matplotlib.pyplot, la qual serveix per a crear un entorn gràfic per al nostre resultat, ja que sinó, l'únic que obtindríem com a simulació final seria una matriu extremadament gran i amb números canviants constantment, i no es podrien visualitzar les partícules. Aquesta biblioteca s'importa amb l'abreviació «plt». Així és més fàcil escriure-la i no cal repetir el nom sencer de la biblioteca.

També s'importa la biblioteca numpy amb l'abreviació de «np». Aquesta, l'utilitzarem per a fer els càlculs matricials i per a generar nombres aleatoris, entre d'altres coses.

```
plotRealTime = True
```

Aquesta petita línia de codi definirà cada quant temps s'han de mostrar els resultats per la pantalla. Hem de trobar un equilibri entre el rendiment i la fluïdesa per a que el moviment de la simulació s'apreciï de la millor manera possible.

```
def distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
```

Aquesta funció anomenada «distance» ens calcula la distancia entre la posició de dos partícules. Bàsicament, calcula el mòdul del vector entre les dues partícules, i és així com obté la distancia necessària. S'utilitza el teorema de Pitàgores per a obtenir la distancia entre els dos punts.

```
def main():
    # paràmetres de la simulació
    Nx = 200   # resolució en la direcció X
    Ny = 100   # resolució en la direcció Y
    rho0 = 1.225   # densitat del fluid
    tau = 0.6   # temps de col·lisió
    Nt = 10000   # número d'iteracions
    plotRealTime = True   # traçar la simulació en temps real
```

Aquí comença la funció principal de la simulació «main». Es defineixen les variables més importants com, per exemple, el número de partícules en l'eix X (resolució en X) i el número de partícules en l'eix Y (resolució en Y). És així com obtenim una malla de 300x100.

També definim altres variables com la densitat mitja del fluid, que no té unitats com sí que les té fora de l'àmbit de la computació. No en té perquè és una magnitud de relació, és a dir, s'ha de comparar amb una altra prova de la simulació canviant la variable (per exemple 10) per a mesurar els resultats en funció de la relació que guarda amb l'anterior prova. (En cas de que canviéssim la densitat per 10 seria una relació de 0.5, ja que 10/20=0.5).

També tenim la variable de col·lisions (tau) que es pot ajustar en funció de si volem un sistema més o menys volàtil. Sempre s'ha de trobar un equilibri, perquè si no el sistema pot esdevenir inestable i causar un desbordament de les dades i, per tant, una fallada en la simulació.

Una altra variable que podem configurar es (Nt). Aquesta variable representa el nombre d'iteracions (repeticions) que contindrà la nostra simulació.

La variable «plotRealTime» és una variable de tipus booleana i es pot especificar com «true» (verdader) o «false» (fals). Aquesta, activa o desactiva la simulació a temps real, és a dir, la possibilitat de veure com es desenvolupa. En la meva simulació està activat per a poder veure a través de la pantalla el comportament del fluid.

```
NL = 9
```
Aquesta variable especifica el nombre de partícules del sistema a nivell bàsic, és a dir, es necessiten com a mínim 9 partícules per a crear una malla de 3x3 i, per tant, per a poder fer les permutacions adients i aconseguir el moviment del fluid.

```
cxs = np.array([0, 0, 1, 1, 1, 0, -1, -1, -1])
```
Aquesta matriu especifica els moviments en l'eix X que pot fer la partícula dins de la malla de 3x3. Un 0 vol dir que la partícula de l'eix X es queda en la mateixa posició, és a dir, no es mou. Un 1 vol dir que es mou a una posició cap a la dreta, i un -1 vol dir que es mou a una posició cap a l'esquerra.

```
cys = np.array([0, 1, 1, 0, -1, -1, -1, 0, 1])
```
Per altra banda, aquí tenim la mateixa matriu per l'eix Y. Aquesta matriu defineix si la partícula romandrà quieta o es mourà en alguna de les direccions de l'eix Y. Un 0 vol dir que romandrà en la mateixa posició dins de l'eix vertical, un 1 vol dir que puja una posició cap amunt, i -1 vol dir que baixa una posició dins de l'eix.

Per exemple, si el sistema probabilístic que més endavant explicarem decideix moure la partícula en diagonal cap a baix i cap a la dreta, aquesta agafarà els valors d'1 per a la X i -1 per a la Y.

Aquestes variables, com podem observar, són de tipus array, és a dir, formen una llista o una matriu en aquest cas, i són part de la biblioteca Numpy.

```
weights = np.array([4/9, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36])
```
Aquí podem observar el sistema probabilístic que s'ha mencionat anteriorment. Podem observar els valors explicats en l'apartat "Com funciona una simulació LBM". Cada partícula del sistema té 4 entre 9 possibilitats de romandre quieta, 1 entre 9 possibilitats de moure's només en un dels eixos, tant X com Y (i Z en el cas de que tinguéssim una simulació LBM en tres dimensions), i finalment, 1 entre 36 possibilitats de moure's en diagonal, és a dir, afectant els dos eixos (tres en el cas d'una simulació 3D).

```
F = np.ones((Ny, Nx, NL)) + 0.015*np.random.randn(Ny, Nx, NL)
```
Aquest petit error de càlcul es multiplica per 0,1 i aquest multiplicador es pot canviar en funció de si volem obtenir un flux més o menys turbulent. Si volem un flux completament laminar podem ajustar aquest valor a un nombre extremadament petit, en canvi, si volem obtenir un flux turbulent i dispers, aquest valor es pot ajustar a un nombre més gran per així obtenir un major resultat a l'hora d'obtenir turbulències.

S'ha de trobar un equilibri a l'hora d'ajustar aquest paràmetre, ja que un nombre excessivament gran pot desencadenar un col·lapse de la simulació i un nombre extremadament petit no donaria un resultat realista, perquè seria un fluid en condicions completament ideals i això en el món real es quasi impossible de trobar, per a no dir completament impossible.

```
F[:, :, 3] = 2.4
```

En aquesta petita línia de codi, afegim més valor a les partícules de la dreta per a fer que el fluid es mogui en aquesta direcció i no en una altra. Per a poder veure com el fluid reacciona amb un cos, aquest, ha d'estar en moviment i en fricció amb el sòlid.

```
cube = np.full((Ny, Nx), False)
```

Aquesta línia defineix com a valor fals totes les partícules que no formen part d'un sòlid i que, per tant, són un espai buit. En canvi, les partícules que formen part d'un sòlid, en aquest cas un cub, són definides amb un valor verdader.

Això ens permet, junt amb la llei de frontera i límits, fer que les partícules del fluid no travessin els sòlids, ja que això a la vida real no és possible.

```
start_x = Nx // 4
end_x = Nx // 4 + Ny // 2
start_y = Ny // 4
end_y = Ny // 4 + Ny // 2
cube[start_y:end_y, start_x:end_x] = True
```

Aquestes línies formen part d'una funció que dibuixa un cub en dos dimensions, és a dir, un quadrat. Els números que fan que aquesta funció sigui certa, seran les partícules que formaran part del nostre sòlid. Aquesta no és una manera gaire pràctica de fer sòlids, ja que són funcions que no són senzilles de calcular. En aquest cas, la funció no és massa complicada, però en el cas d'un altra simulació d'un perfil alar que s'ha fet, s'ha necessitat ajuda de la intel·ligència artificial per a poder construir la funció corba del perfil alar. Ja que avui en dia tenim la gran sort de disposar d'aquestes tecnologies, les podem utilitzar i aplicar en entorns i camps com poden ser la simulació de fluids en la supercomputació.

En aquest cas podem veure representats els quatre costats del quadrat emmagatzemats en les variables "start" i "end" per a cadascuna de les dimensions de la simulació.

```
for it in range(Nt):
    print(it)
```

Aquest bucle "for" (una de les maneres d'executar una ordre repetidament en programació) ens permet obrir una finestra per a veure quantes iteracions han estat executades en la nostra simulació. És així com podem tenir en compte el pas del temps i veure els resultats aconseguits depenent del temps que ha transcorregut en la simulació. Cada una d'aquestes iteracions és una actualització dels resultats i, per tant, un càlcul de cada una de les 30.000 partícules de les que consta la nostra simulació.

```
F[:, -1, [6, 7, 8]] = F[:, -2, [6, 7, 8]]
F[:, 0, [2, 3, 4]] = F[:, 1, [2, 3, 4]]
```

En aquesta línia de codi, els valors contraris a la direcció en la que van la majoria de partícules de la simulació, es contraresten a si mateixos per tal de no interferir en el resultat obtingut.

```
for i, cx, cy in zip(range(NL), cxs, cys):
    F[:, :, i] = np.roll(F[:, :, i], cx, axis=1)
    F[:, :, i] = np.roll(F[:, :, i], cy, axis=0)
```

Amb aquest bucle «for», movem les partícules en les direccions obtingudes gràcies al sistema probabilístic que hem implementat anteriorment. Amb la funció np.roll(), fem «rodar» o, més ben dit, movem a la següent posició cada una de les partícules del fluid per tal que la pròpia simulació avanci.

```
bndryF = F[cube, :]
bndryF = bndryF[:, [0, 5, 6, 7, 8, 1, 2, 3, 4]]
```

Gràcies a la variable «bndryF», s'emmagatzemen els punts de la simulació on el sòlid que estem simulant ocupa un espai. Aquest espai no pot ser ocupar per les partícules del fluid perquè es crearia una superposició de dos elements, i això a la realitat no és possible. En aquest primer pas, es calcula la posició d'aquestes partícules.

```
# Calculem la densitat
rho = np.sum(F, 2)
# Momentum
ux = np.sum(F*cxs, 2) / rho
uy = np.sum(F*cys, 2) / rho
```

Seguidament, calcularem l'avenç per cada iteració d'algunes variables que ens faran falta per a poder aplicar correctament les equacions de Boltzmann. Una de les variables és «rho», és a dir, la densitat respecte a la que hem especificat com a paràmetre inicial. També calcularem el moment (quantitat de moviment) en l'eix vertical i horitzontal, sumant els valors de la matriu de moments cxs i cys multiplicats per les interferències, i també utilitzarem, com es pot veure, la densitat en el temps calculada a la línia anterior.

```
F[cube, :] = bndryF
ux[cube] = 0
uy[cube] = 0
```

El següent pas és agafar el vector d'aquestes partícules i revertir-ne la direcció i la velocitat per tal de que rebotin amb el sòlid i no ocupin el seu espai.

```
Feq = np.zeros(F.shape)
for i, cx, cy, w in zip(range(NL), cxs, cys, weights):
    Feq[:, :, i] = rho * w * (
        1 + 3*(cx*ux + cy*uy) + 9*(cx*ux + cy*uy)**2 / 2 - 3*(ux**2 + uy**2)/2)
```

En aquest pas calculem el valor de «Feq». Aquesta, és una variable que tendeix a donar equilibri a la fórmula. Aquesta, es tracta d'una aproximació, on la primera part representa la transmissió de moviment entre partícules i la segona part és l'encarregada d'aproximar les col·lisions.

Amb l'ajut d'un altre bucle «for», podem executar la resolució d'aquesta equació per cadascuna de les iteracions de la simulació.

```
F = F + -(1/tau) * (F-Feq)
```

En la següent línia de codi substituïm el valor de la variable d'equilibri trobada anteriorment per tal d'aconseguir el nou valor que prendran les partícules a la següent iteració, i és així com finalitza un cicle de càlcul i desplaçament de les partícules.

```
if it % plot_every == 0:
    # Identifiquem els punts que creen vòrtex
    dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
    dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
    curl = dfydx - dfydy
```

Aquest condicional «if» ens permet ensenyar el resultat a la pantalla si la condició de que el residu de la divisió d'iteracions entre el període en que es mostra cada una és igual a zero. Això ho fem per a assegurar-nos de que no ens quedin iteracions sense mostrar en el sistema, ja que si el nombre d'iteracions entre el període no fossin divisibles, a l'acabar la simulació quedarien iteracions sense mostrar. Posarem un exemple per a poder entendre-ho millor.

El nombre d'iteracions total es 3.333 i les volem ensenyar de deu en deu. El residu és 3 i, per tant, no arriba a les 10 necessitaries per a poder-les mostrar a la pantalla. Per tant, aquestes iteracions s'executarien però no es mostrarien.

Les següents línies de codi identifiquen les àrees de partícules amb més vorticitat, és a dir, moviments més bruscs i turbulents, i ho emmagatzemen en una variable anomenada «curl» (vorticitat en anglès).

```
magnitude = np.abs(curl)
```

En aquesta petita línia de codi utilitzem la funció abs() de Numpy per a convertir tots els valors negatius que ens ha pogut donar el càlcul de vorticitat a positius, per a poder representar-los de manera correcta i precisa a la nostra malla.

```
vmax = 0.1
vmin = 0.0001
```

Després de fer unes proves i obtenir els valors màxims i mínims de les nostres partícules, els arrodonim i els assignem a dues variables «vmax> i «vmin». Aquestes ens serviran per a representar el mapa de colors d'una manera adequada. Aquest pas és important, ja que al no tenir valors unitaris (que és mesuren amb unitats), necessitem comparar els valors de la simulació amb si mateixos per a obtenir la diferencia de vorticitat, pressió, velocitat, etc.

```
plt.imshow(magnitude, cmap="twilight", vmin=vmin, vmax=vmax)
plt.pause(.01)
plt.cla()
```

A continuació creem la representació visual utilitzant la funció imshow() de Matplotlib. El valor representat és la variable «magnitud». Recordem que aquesta variable emmagatzema la vorticitat de cada una de les partícules. També hem d'utilitzar un mapa de colors per a la nostra simulació.

Cada mapa de colors representa els valors donats d'una manera diferent. En la simulació s'ha utilitzat el mapa «twilight», però també es podem utilitzar altres mapes de colors com poden ser «virdis», «plasma», o «seismic» entre molts altres.

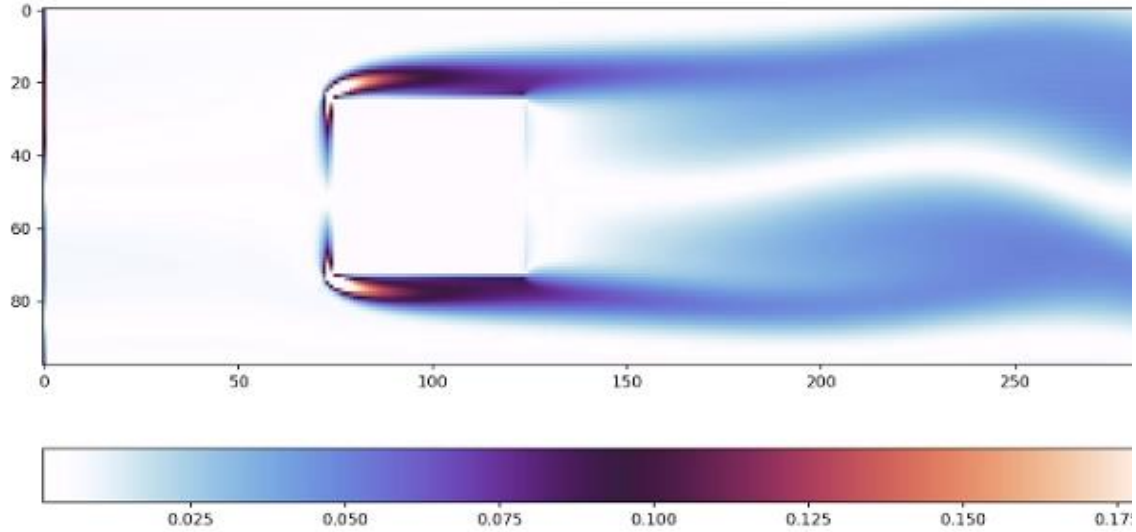
Aquí en tenim uns quants exemples:

Llibreria Python 1: Colormaps matplotlib (Font: matplotlib.org)

A continuació, amb la funció pause(), parem la simulació 0.01 segons per a que no es saturi a l'hora d'ensenyar les iteracions i, finalment, mostrem el resultat amb la funció cla().

```
plt.colorbar(orientation="horizontal")
```

Finalment, aquesta línia de codi ens serveix per a crear una barra que ens ensenya tot el rang de colors que pot adquirir la simulació i, per tant, que puguem saber quins són els valors més alts i més baixos.



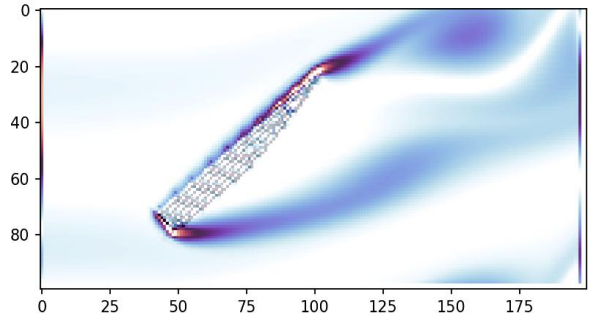
Simulació LBM Part Pràctica 7: Resultat final del codi

Aquest és el resultat final, que podeu veure en funcionament amb el següent codi QR:

Còdi QR 1: https://github.com/pauesquerda/lbm.sim.tdr

15 Exemple de simulació amb un sòlid funcional

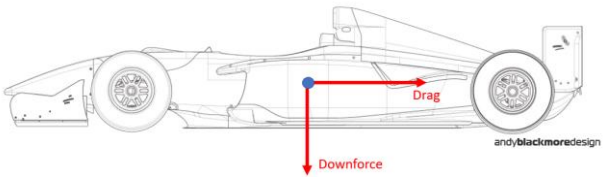
Il·lustració 5: Aleró posterior RedBull RB5 (Font: https://www.youtube.com/watch?v=jYaIXWNOa_A)



Simulació LBM Part Pràctica 8: Aleró en angle d'atac de 40 graus

S'ha pogut observar com funciona aquest mètode de simulació amb cossos simples, com poden ser un cilindre o un cub, però aquests elements no tenen una aplicació directa al món real, així que per a poder comprovar que es podria utilitzar en ambients reals i funcionals, s'ha provat la simulació amb un aleró d'un cotxe dissenyat per a crear carrega aerodinàmica i mantenir la carrosseria enganxada al terra. Aquí es pot veure la comparació entre el flux d'aire creat per l'aleró real d'un cotxe de F1 de l'equip RedBull i el flux creat per l'aleró simulat gràcies al mètode LBM.

Es pot observar clarament el funcionament d'aquest aleró, ja que desvia l'aire d'una manera molt similar a un aleró provat en un túnel de vent real. Un aleró utilitza un principi molt simple. Com es pot veure a la segona fotografia (simulació amb mètode LBM), a la part superior de l'aleró es crea una zona d'altes pressions on l'aire es concentra i s'acumula. Això redueix la velocitat d'aquest. Per altra banda, a la part inferior de l'aleró, es crea una zona de baixes pressions que fa que l'aire vagi a una major velocitat. Aquesta diferència de velocitats i pressions crea el que es coneix en el món de la aerodinàmica com a «downfoce» o «càrrega aerodinàmica». En aquest cas, una força que actua en direcció a la part inferior del cotxe i el manté enganxat a la pista per a que no es desviï.



Il·lustració 6: Drag i Downforce (Font: twitter.com)

15.1 Vòrtex creats per l'aleró

L'aleró posterior també serveix per a crear fluids d'aire turbulents, i com hem explicat abans, no sempre és una cosa dolenta, ja que quan parlem d'automoció esportiva, crear vòrtex al voltant de la carrosseria serveix per a desviar l'aire i que no xoqui amb zones més propenses a frenar el vehicle. Els cotxes de Fórmula 1 consten d'un sistema amb una gran complexitat aerodinàmica per a crear vòrtex turbulents al voltant del cotxe i crear la màxima càrrega aerodinàmica, afectant mínimament a la resistència amb l'aire.

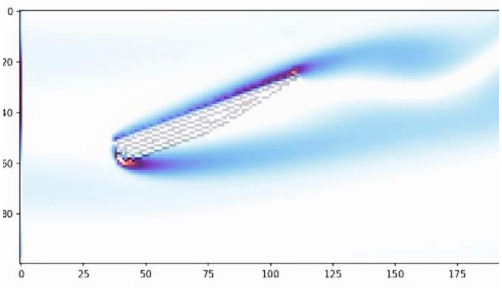
15.2 Ajustament de l'angle d'atac de l'aleró

Un dels factors clau que influeixen en la carrega aerodinàmica d'un aleró és l'angle d'atac. Aquest angle determina com el flux d'aire s'apropa i interacciona amb l'aleró, i pot afectar la carrega ascendent o descendents (lift o downforce) que aquest genera.

Per aconseguir més carrega aerodinàmica es pot augmentar l'angle d'atac. Això fa que l'aleró "ataqui" més directament l'aire que es mou al seu voltant, creant així una diferència de pressió entre la part superior i inferior de l'aleró. Aquesta diferència de pressió genera una força que aixeca l'aleró i pot augmentar la carrega ascendent. Això és fonamental en l'automobilisme i l'aviació, ja que permet augmentar la tracció i la maniobrabilitat del vehicle.

No obstant això, cal tenir en compte que un angle d'atac excessivament alt pot provocar fenòmens com la separació del flux, on l'aire ja no segueix una línia suau al voltant de l'aleró sinó que es torna turbulent. Això pot generar resistència aerodinàmica (drag) i disminuir l'eficiència del sistema. Per tant, l'ajust de l'angle d'atac és una tasca delicada que implica trobar l'equilibri òptim entre la generació de carrega i la minimització del drag.

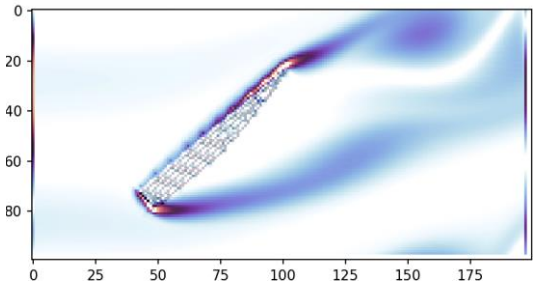
Podem modificar aquest angle d'atac en la nostra simulació canviant la variable dels graus en la funció rotate().



Simulació LBM Part Pràctica 9: Aleró en angle d'atac de 20 graus

```
rotate(wing, 20, reshape=False)
```

En aquest cas, l'angle d'atac de l'aleró ha estat ajustat a 20 graus. Això vol dir que té menys «drag», és a dir, menys resistència a l'aire. Però també té menys càrrega aerodinàmica i, per tant, genera menys adherència.



Simulació LBM Part Pràctica 10: Aleró en angle d'atac de 40 graus

```
rotate(wing, 40, reshape=False)
```

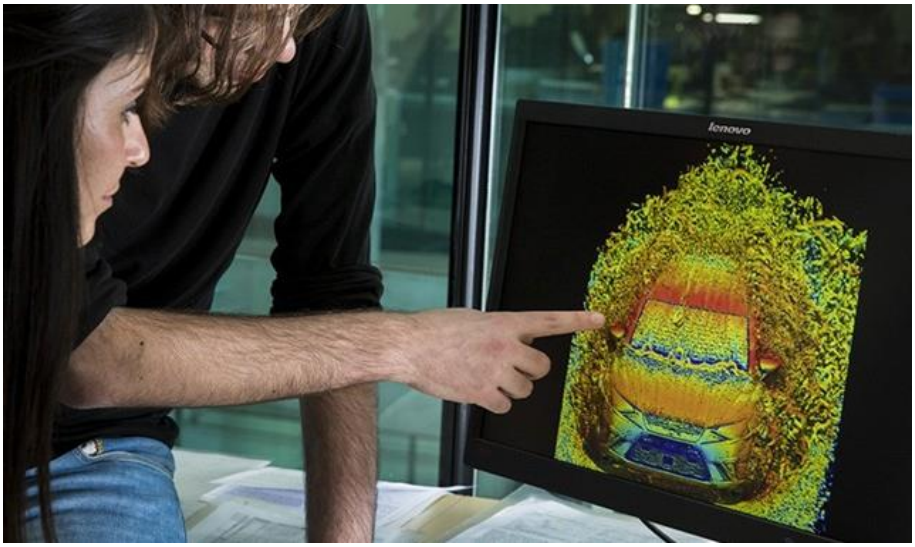
Aquí hem ajustat l'angle d'atac a 40 graus. L'aleró té més drag i, per tant, el vehicle que l'incorpori tindrà menys velocitat punta, però això ho guanyarà en termes d'adherència, ja que la resistència aerodinàmica haurà augmentat. En aquesta simulació, podem veure que la diferència de pressions respecte a la primera es major ja que s'ha augmentat l'angle d'atac i per tant la càrrega aerodinàmica.

16 Cas real d'utilització de la simulació de fluids[5]

Una de les empreses pioneres en tot el món en desenvolupar la aerodinàmica dels seus vehicles utilitzant la supercomputació ha estat Seat. Aquesta empresa espanyola, ha utilitzat el gran poder de l'actual MareNostrum 4 per a poder desenvolupar i estudiar com reduir la fricció i la resistència amb l'aire.

«Treballar amb un túnel de vent es costós, els models d'argila sofreixen més i tenim que fer canvis constantment, a més la potència de càlcul del supercomputador del BSC ens permet incloure més paràmetres i veure com es comporta l'aire dins de les llantes de les rodes en moviment. Es tracta d'apropar la simulació cada vegada més a la realitat» [María García-Navas, enginyera del Departament de Desenvolupament i Aerodinàmica de Seat]

«*En un futur ens agradaria simular-ho tot junt, el fluid, l'estructura del cotxe, la combustió i inclús la persona asseguda a dins. Actualment no podem, però d'aquí 15 anys els supercomputadors seran 1000 vegades més potents, ja ho podem imaginar.» [Investigador del BSC]*



Il • lustració 7: Enginyers de Seat amb una simulació 3D del MareNostrum4

(Font: autocasion.com)

16.1 Petita entrevista amb un investigador del BSC

En una de les meves classes al Barcelona Supercomputing Center vaig tenir la oportunitat de parlar amb un dels encarregats del projecte de Seat.

Em va explicar un munt de tecnicismes i procediments, però una cosa que em va cridar molt l'atenció va ser el motiu pel qual Seat estava intentant reduir les turbulències en els seus cotxes. Un dels motius principals era el soroll que creaven aquestes.

[5]Informació extreta de autocasión.com

L’aire, en passar pels retrovisors del cotxe, causa unes turbulències i uns vòrtex que van xocant amb tota la carrosseria del cotxe i, per tant, creen un desagradable soroll. Aquest soroll es fàcilment reconeixible per qualsevol de nosaltres. Es tracta d’aquella remor de fons que tenim quan anem per la carretera. No és el motor del cotxe com molts pensàvem, i tampoc són els neumàtics rodant per la carretera, sinó que es tracta de l’aire xocant contra el nostre vehicle i creant vòrtex turbulents.

Seat va utilitzar la supercomputació i la simulació de fluids per a convertir aquest flux turbulent creat pels retrovisors en un flux laminar que no creï soroll de fons, per a fer que els passatgers del cotxe viatgin mes còmodament.

17 Àrees de millora i investigació en les simulacions de fluids sense interfície gràfica

Durant el procés d'investigació i programació s'han trobat diverses àrees en les que les simulacions de fluids es podrien millorar i optimitzar. Aquestes petites millores farien que les simulacions de fluids en un entorn cent per cent de codi, es poguessin apropar a un públic més planer i sense grans coneixements sobre supercomputació i programació.

Aquestes millores estan enfocades a entorns sense interfície gràfica. Aquesta consta de botons, desplegables, pantalles i molts altres elements que faciliten molt la navegació dins del programa. La meva simulació únicament té interfície gràfica a l'hora de ser executada, ja que consta d'un entorn 100% de codi. Existeixen programes amb una interfície gràfica que solucionen molts dels problemes que es comentaran a continuació, però seria una aposta interessant millorar aquests sectors en entorns de codi i consola, sense interfície com és el meu cas.

17.1 Creació de sòlids per a que interactuïn amb les partícules

Hem pogut veure com una de les parts més difícils de les simulacions LBM en Python es la creació dels sòlids per a simular. En el cas d'un cub o un cilindre, encara es una funció relativament senzilla, però una de les proves que s'han fet (i que podeu veure a GitHub [qr]) es simula el comportament d'un flap. Aquest flap, exemplifica un aleró d'un cotxe el qual amb una diferència de pressions crea una força que enganxa el vehicle al terra.

```
wing = np.full((Ny, Nx), False)

# crear un perfil alar simple utilitzanr la funció del sinus
for y in range(Ny):
    for x in range(Nx):
        if y > 0.3 * Ny and y < 0.6 * Ny and x > 0.2 * Nx and x < 0.8 * Nx:
            wing[y][x] = y < 0.3 * Ny + 10 * np.sin(np.pi * (x - 0.01 * Nx) / (0.6 * Nx))+1

rotated_wing = rotate(wing, 20, reshape=False)
```

Aquí podem veure el codi per a crear aquest flap. Es pot observar que s'han d'utilitzar biblioteques externes com, per exemple, «scipy.ndimage» per a rotar la figura i donar-li un angle d'atac al flap. A banda, s'han d'utilitzar funcions com el sinus per a crear corbes, i aquestes no són extremadament precises. Aquesta és una clara àrea de millora que es podria solucionar amb l'ajut de la famosa Intel·ligència Artificial. Es podria entrenar un model de llenguatge amb una interfície gràfica que permeti crear una figura tant en dos com en tres dimensions, per a poder-les aplicar directament a les simulacions. Aquest model d'IA podria convertir la creació d'un sòlid creat en entorn CAD a codi pur, per a poder implementar la figura a l'entorn de text.

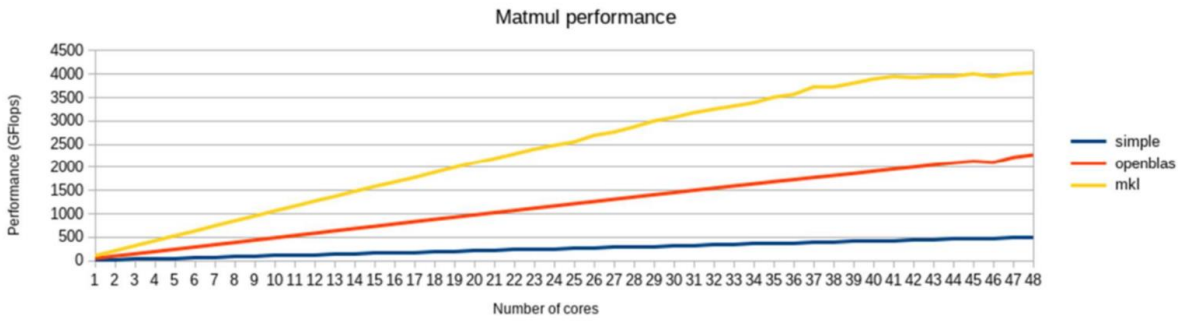
17.2 Exigència de la simulació en termes de hardware

Una simulació en dues dimensions com la que s'ha creat en aquest treball de recerca ja és extremadament exigent, i no es pot executar amb una altíssima resolució. És per això que per fer simulacions més complexes, com poden ser les simulacions en 3D o simulacions d'elements més complexes, s'utilitzen els supercomputadors.

S'hauria de trobar un mètode per a reduir els bucles dins d'altres bucles el màxim possible, ja que aquestes són les parts del codi més exigents i les que multipliquen exponencialment els càlculs per segon que s'han d'executar i que, per tant, forcen més el poder de computació matemàtic.

Paral·lelització

Per a solucionar aquest problema, es va intentar aplicar la paral·lelització de processos en la simulació. Aquest sistema permet distribuir les tasques de càlcul de la simulació entre tots els nuclis del xip. És a dir, aprofitar que el xip d'un ordinador està dissenyat per a multi-tasking (fer múltiples tasques a la vegada) per a dividir la simulació en diferents tasques i donar-ne una a cada nucli.



Gràfic 3: Potència respecte a número de cores (Font: Elaboració pròpia)

Això finalment es va descartar, ja que fent les proves adients no sortia a compte degut a que els recursos que es guanyaven per un costat es perdien per l'altre. Aquesta solució sí que seria efectiva en un ordinador de grans dimensions com el MareNostrum, perquè té molts més processadors entre els que dividir la tasca a fer i, per tant, pot anar molt més ràpid.

Aquí tenim un gràfic d'una prova que vaig fer amb el MareNostrum 4. Es pot observar com sense aplicar paral·lelització (línia blava), la potència computacional és constant i varia molt poc. En canvi, aplicant la paral·lelització (línia groga), a mesura que es van augmentant els «cores» o processadors, la potència computacional també augmenta. Aquesta però, s'estanca a l'arribar als 40 cores actius. Això és el que ha passat a l'aplicar la paral·lelització en un ordenador convencional, ja que no té tant marge de cores com un supercomputador.

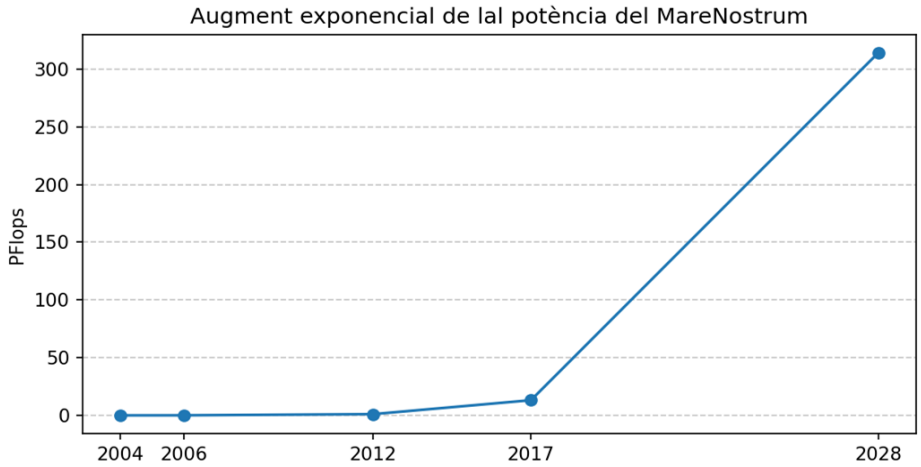
18 Conclusions i idees generals

Els principals objectius plantejats a la introducció s'han complert. S'ha aconseguit no només una, sinó diferents simulacions funcionals en un computador convencional. Utilitzant el mètode Lattice Boltzmann i el llenguatge de programació Python, s'ha aconseguit demostrar el funcionament d'aquest tipus de simulació i s'ha establert els principals usos que pot tenir.

La simulació de fluids mitjançant supercomputació seguirà avançant en tots els àmbits i branques on es pugui expandir. No només en potència computacional, sinó també en optimització per a que estigui a l'abast de tothom que vulgui experimentar amb aquest tipus de simulacions.

Es millorarà la capacitat per a simular sòlids més complexes i més detallats, com bé ha mencionat María García-Navas, enginyera del Departament de Desenvolupament i Aerodinàmica de Seat. En un futur es vol arribar a simular fins i tot la persona que anirà asseguda dins del vehicle. Per assolir aquest objectiu, la potència d'aquestes simulacions s'incrementarà exponencialment degut als bucles dins d'altres bucles que incrementaran les iteracions en aquest tipus de simulacions.

Per tant, la potència computacional haurà de créixer exponencialment com ho ha estat fent en els últims anys.



Gràfic 4: Creixement exponencial de la potència en els supercomputadors (Font: Elaboració pròpia)

Aquest gràfic representa el creixement exponencial en la potència del supercomputador MareNostrum al llarg dels anys. Com es pot observar, segons les afirmacions del BSC, en pocs anys podrem executar totes aquelles simulacions que avui en dia no podem dur a terme gràcies als 314 PetaFlops de potència del MareNostrum5, l'any 2028.

La capacitat de les simulacions no només augmentarà en exigències de potència sinó que també s'incrementarà en l'optimització de processos, per tal de que no siguin tant costosos de dur a terme i que, finalment, es puguin executar en ordenadors convencionals com els que tenim a casa.

18.1 ¿Qui en farà us?

L'ús d'aquestes simulacions serà cada vegada més extens i cada cop més sectors implementaran aquestes simulacions en els seus respectius camps, productes i experiments, però hi haurà uns grans beneficiats d'aquest tipus de simulacions.

Els primers beneficiats seran les agències privades de llançaments espacials. Aquestes empreses, com poden ser SpaceX o Blue Origin, estan dedicades a fer llançaments privats a l'espai utilitzant les últimes tecnologies. Actualment, gasten molts i molts diners en fer proves per a poder desenvolupar els seus models de coets el més ràpid possible per a poder avançar-se a la competència. Aquestes simulacions aerodinàmiques els permetran fer aquestes proves d'una manera més rapida i barata que no pas llençant i llençant milions ens combustible i altres recursos per a fer les proves. Aquestes simulacions els permetran, no només provar l'aerodinàmica dels coets i les càpsules, sinó que també els permetrà simular el consum i ús del combustible líquid per part dels motors químics.

El següent beneficiari serà el món de l'automoció, tant la comercial com la de competició. En el món comercial ja s'ha començat a implementar gràcies a l'empresa espanyola Seat, que ha implementat la gran potència del MareNostrum 4 per a poder executar simulacions en tres dimensions i poder així reduir les turbulències creades per components del cotxe, com els retrovisors o les llantes. D'altra banda, el món de l'automoció esportiva també se'n beneficiarà. Per exemple, els equips de Fórmula 1 disposen d'unes certes hores de proves en el túnel de vent. Com més alts estan en la classificació, menys hores de túnel de vent disposen per a millorar l'aerodinàmica dels seus monoplaces. Les simulacions d'aerodinàmica són una gran solució a aquest problema, perquè la FIA (Federació Internacional d'Automobilisme) no regula les hores que els equips passen fent proves amb les simulacions de fluids.

Els últims grans beneficiaris seran els fabricants d'avions comercials com Boeing o Airbus, i també els fabricants d'avions d'us militar com Lockheed Martin o Raytheon Technologies. Els fabricants comercials podran provar els avions abans de construir-los físicament, per a poder comprovar la viabilitat del projecte abans de gastar-se milions i milions en desenvolupaments per portar-los a la realitat. Per altra part, els fabricants d'avions d'ús bèl·lic podran provar els seus aparells sense posar en perill projectes secrets o confidencials, i amb la seguretat de que altres cossos militars no tindran informació privilegiada sobre aquests projectes que podrien ficar en perill a la població.

Com s'ha pogut observar, aquestes simulacions solucionen una gran varietat de problemes, des de reduir la despesa de les empreses en el desenvolupament de nous productes fins a la seguretat civil i la protecció de països sencers. Tampoc es pot oblidar que aquestes simulacions ajuden a desenvolupar la majoria dels petits aparells i artefactes que utilitzem en el nostre dia a dia.

19 Annexos

19.1 Annex 1 (Simulació cub)

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3.
4. plot_every = 10
5. def distance(x1, y1, x2, y2):
6.     return np.sqrt((x2-x1)**2 + (y2-y1)**2)
7.
8. def main():
9.     Nx = 300  # resolució x
10.          Ny = 100  # resolució y
11.          rho0 = 20  # densitat mitja
12.          tau = 0.6  # col·lisions
13.          Nt = 10000  # iteracions
14.          plotRealTime = True
15.
16.          NL = 9
17.
18.          cxs = np.array([0, 0, 1,
19.                          1, 1, 0,
20.                         -1, -1, -1])
21.
22.          cys = np.array([0, 1, 1,
23.                          0, -1, -1,
24.                         -1, 0, 1])
25.          weights = np.ar-
   ray([4/9, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36])
26.
27.
28.          F = np.ones((Ny, Nx, NL)) + 0.1*np.rdom.randn(Ny,
    Nx, NL)
29.
30.          F[:, :, 3] = 2.4
31.
32.          cube = np.full((Ny, Nx), False)
33.
34.          start_x = Nx // 4
35.          end_x = Nx // 4 + Ny // 2
36.          start_y = Ny // 4
37.          end_y = Ny // 4 + Ny // 2
38.          cube[start_y:end_y, start_x:end_x] = True
39.
40.          vmin = 0.001
41.          rho = np.sum(F, 2)
42.          ux = np.sum(F*cxs, 2) / rho
43.          uy = np.sum(F*cys, 2) / rho
44.          # Identifiquem els punts que creen vòrtex
45.          dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
46.          dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
47.          curl = dfydx - dfydy
48.          magnitude = np.abs(curl)
49.          plt.imshow(magnitude, cmap="twilight", vmin=vmin)
```

```
50.         plt.colorbar(orientation="horizontal")
51.
52.         for it in range(Nt):
53.             print(it)
54.
55.             F[:, -1, [6, 7, 8]] = F[:, -2, [6, 7, 8]]
56.             F[:, 0, [2, 3, 4]] = F[:, 1, [2, 3, 4]]
57.
58.             # Movem les partícules
59.             for i, cx, cy in zip(range(NL), cxs, cys):
60.                 F[:, :, i] = np.roll(F[:, :, i], cx, axis
   =1)
61.                 F[:, :, i] = np.roll(F[:, :, i], cy, axis
   =0)
62.
63.             bndryF = F[cube, :]
64.             bndryF = bndryF[:, [0, 5, 6, 7, 8, 1, 2, 3, 4
   ]]
65.
66.             rho = np.sum(F, 2)
67.             # Momentum
68.             ux = np.sum(F*cxs, 2) / rho
69.             uy = np.sum(F*cys, 2) / rho
70.
71.
72.             F[cube, :] = bndryF
73.             ux[cube] = 0
74.             uy[cube] = 0
75.
76.             # Colisió
77.             # Calculem el F equilibrium
78.             Feq = np.zeros(F.shape)
79.             for i, cx, cy, w in zip(range(NL), cxs, cys,
   weights):
80.                 Feq[:, :, i] = rho * w * (
81.                     1 + 3*(cx*ux + cy*uy) + 9*(cx*ux +
   cy*uy)**2 / 2 - 3*(ux**2 + uy**2)/2)
82.
83.             F = F + -(1/tau) * (F-Feq)
84.
85.             if it % plot_every == 0:
86.                 dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
87.                 dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
88.                 curl = dfydx - dfydy
89.
90.                 magnitude = np.abs(curl)
91.                 vmax = 0.1
92.                 vmin = 0.0001
93.
94.                 plt.imshow(magni-
   tude, cmap="twilight", vmin=vmin, vmax=vmax)
95.                 plt.pause(.01)
96.                 plt.cla()
97.
98.     if __name__ == '__main__':
99.         main()
```

19.2 Annex 2 (Simulació cilindre)

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3.
4. plot_every = 10
5. def distance(x1, y1, x2, y2):
6.     return np.sqrt((x2-x1)**2 + (y2-y1)**2)
7.
8. def main():
9.    Nx = 300  # resolució x
10.         Ny = 100  # resolució y
11.         rho0 = 20  # densitat mitja
12.         tau = 0.6  # col·lisions
13.         Nt = 10000  # iteracions
14.         plotRealTime = True
15.
16.         NL = 9
17.         cxs = np.array([0, 0, 1,
18.                         1, 1, 0,
19.                        -1, -1, -1])
20.         cys = np.array([0, 1, 1,
21.                         0, -1, -1,
22.                        -1, 0, 1])
23.         weights = np.ar-
   ray([4/9, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36])
24.
25.         F = np.ones((Ny, Nx, NL)) + 0.1*np.ran-
   dom.randn(Ny, Nx, NL)
26.
27.         F[:, :, 3] = 2.4
28.
29.         cube = np.full((Ny, Nx), False)
30.         start_x = Nx // 4
31.          end_x = Nx // 4 + Ny // 2
32.          start_y = Ny // 4
33.          end_y = Ny // 4 + Ny // 2
34.          cube[start_y:end_y, start_x:end_x] = True
35.
36.         vmin = 0.001
37.          rho = np.sum(F, 2)
38.          ux = np.sum(F*cxs, 2) / rho
39.          uy = np.sum(F*cys, 2) / rho
40.          # Identifiquem els punts que creen vòrtex
41.         dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
42.          dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
43.          curl = dfydx - dfydy
44.          magnitude = np.abs(curl)
45.          plt.imshow(magnitude, cmap="twilight", vmin=vmin)
46.          plt.colorbar(orientation="horizontal")
47.
48.         for it in range(Nt):
49.              print(it)
50.
51.
52.             F[:, -1, [6, 7, 8]] = F[:, -2, [6, 7, 8]]
```

```
53.              F[:, 0, [2, 3, 4]] = F[:, 1, [2, 3, 4]]
54.
55.
56.              for i, cx, cy in zip(range(NL), cxs, cys):
57.                  F[:, :, i] = np.roll(F[:, :, i], cx, axis
   =1)
58.                  F[:, :, i] = np.roll(F[:, :, i], cy, axis
   =0)
59.
60.
61.              bndryF = F[cube, :]
62.              bndryF = bndryF[:, [0, 5, 6, 7, 8, 1, 2, 3, 4
   ]]
63.
64.              rho = np.sum(F, 2)
65.              # Momentum
66.              ux = np.sum(F*cxs, 2) / rho
67.              uy = np.sum(F*cys, 2) / rho
68.
69.
70.              F[cube, :] = bndryF
71.              ux[cube] = 0
72.              uy[cube] = 0
73.
74.              Feq = np.zeros(F.shape)
75.              for i, cx, cy, w in
   zip(range(NL), cxs, cys, weights):
76.                  Feq[:, :, i] = rho * w * (
77.                      1 + 3*(cx*ux + cy*uy) + 9*(cx*ux + cy
   *uy)**2 / 2 - 3*(ux**2 + uy**2)/2)
78.
79.              F = F + -(1/tau) * (F-Feq)
80.
81.              if it % plot_every == 0:
82.
83.                  dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
84.                  dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
85.                  curl = dfydx - dfydy
86.
87.                  magnitude = np.abs(curl)
88.                  vmax = 0.1
89.                  vmin = 0.0001
90.
91.
92.                  plt.imshow(magni-
   tude, cmap="twilight", vmin=vmin, vmax=vmax)
93.                  plt.pause(.01)
94.                  plt.cla()
95.
96.      if __name__ == '__main__':
97.          main()
```

19.3 Annex 3 (Simulació aleró)

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3. from scipy.ndimage import rotate
4.
5. plot_every = 20
6.
7. def distance(x1, y1, x2, y2):
8.     return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
9.
10.     def main():
11.         Nx = 200    # resolució en la direcció X
12.         Ny = 100    # resolució en la direcció Y
13.         rho0 = 1.225    # densitat del fluid
14.         tau = 0.6    # temps de col·lisió
15.         Nt = 10000    # número d'iteracions
16.         plotRealTime = True    # traçar la simulació
17.         NL = 9
18.         cxs = np.array([0, 0, 1, 1, 1, 0, -1, -1, -1])
19.         cys = np.array([0, 1, 1, 0, -1, -1, -1, 0, 1])
20.
21.         weights = np.aray([4/9, 1/9, 1/36, 1/9, 1/36, 1/9,
    1/36, 1/9, 1/36])
22.
23.         F = np.ones((Ny, Nx, NL)) + 0.015*np.ran-
   dom.randn(Ny, Nx, NL)
24.          F[:, :, 3] = 2.4
25.
26.          wing = np.full((Ny, Nx), False)
27.
28.         for y in range(Ny):
29.              for x in range(Nx):
30.                  if y > 0.3 * Ny and y < 0.6 * Ny and
   x > 0.2 * Nx and x < 0.8 * Nx:
31.                      wing[y][x] = y < 0.3 * Ny + 10 * np.s
   in(np.pi * (x - 0.01 * Nx) / (0.6 * Nx))+1
32.
33.          rotated_wing = rotate(wing, 20, reshape=False)
34.
35.         vmin = 0.001
36.          rho = np.sum(F,2)
37.          ux  = np.sum(F*cxs,2) / rho
38.          uy  = np.sum(F*cys,2) / rho
39.              #identifiquem els punts que creen vòrtex
40.         dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
41.          dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
42.          curl = dfydx - dfydy
43.          magnitude = np.abs(curl)
44.          plt.imshow(magnitude, cmap="twilight", vmin=vmin)
45.          plt.colorbar(orientation="horizontal")
46.
47.         for it in range(Nt):
48.              print(it)
49.
50.              F[:, -1, [6, 7, 8]] = F[:, -2, [6, 7, 8]]
```

```
51.             F[:, 0, [2, 3, 4]] = F[:, 1, [2, 3, 4]]
52.
53.              # Mover las partículas
54.             for i, cx, cy in zip(range(NL), cxs, cys):
55.                  F[:, :, i] = np.roll(F[:, :, i], cx, axis
   =1)
56.                  F[:, :, i] = np.roll(F[:, :, i], cy, axis
   =0)
57.
58.             bndryF = F[rotated_wing, :]
59.              bndryF = bndryF[:, [0, 5, 6, 7, 8, 1, 2, 3, 4
   ]]
60.
61.              # calcular densitat
62.             rho = np.sum(F, 2)
63.              # Momentum
64.             ux = np.sum(F*cxs, 2) / rho
65.              uy = np.sum(F*cys, 2) / rho
66.
67.             F[rotated_wing, :] = bndryF
68.              ux[rotated_wing] = 0
69.              uy[rotated_wing] = 0
70.
71.             Feq = np.zeros(F.shape)
72.              for i, cx, cy, w in
   zip(range(NL), cxs, cys, weights):
73.                  Feq[:, :, i] = rho * w * (
74.                       1 + 3*(cx*ux + cy*uy) + 9*(cx*ux + c
   y*uy)**2 / 2 - 3*(ux**2 + uy**2)/2
75.                  )
76.             F = F + -(1/tau) * (F - Feq)
77.
78.              if it % plot_every == 0:
79.                  dfydx = ux[2:, 1:-1] - ux[0:-2, 1:-1]
80.                  dfydy = uy[1:-1, 2:] - uy[1:-1, 0:-2]
81.                  curl = dfydx - dfydy
82.                  #
83.                 magnitude = np.abs(curl)
84.                  vmax = np.max(magnitude)
85.                  vmin = 0.001
86.                  plt.imshow(magni-
   tude, cmap="twilight", vmin=vmin, vmax=0.11)
87.                  plt.imshow(rotated_wing, cmap="bi-
   nary", alpha=0.2)
88.                  plt.pause(0.01)
89.                  plt.cla()
90.
91.      if __name__ == '__main__':
92.          main()
```

20 Taula d'il·lustracions

21 Web grafia i autors

- Maria García-Navas, enginyera del Departament de Desenvolupament i Aerodinàmica de Seat (2019)
- But how DO Fluid simulation Work? https://www.youtube.com/watch?v=qsYE1wMEMPA&t=470s
- Create Your Own Lattice Boltzmann Simulation (With Python) https://medium.com/swlh/create-your-own-lattice-boltzmann-simulation-with-python-8759e8b53b1c
- MareNostrum 4: un superordenador al servicio de Seat https://www.autocasion.com/actualidad/noticias/marenostrum-4-un-superordenador-al-servicio-de-seat
- Alguns components del supercomputador https://amazon.es
- MareNostrum https://www.bsc.es/es/marenostrum/marenostrum
- Fórmula 1 Aerodynamics with Martin Bundle https://www.youtube.com/watch?v=jYaIXWNOa_A