Avaluació de la tecnologia de les GPUs

Memòria del projecte

d'Enginyeria Tècnica en

Informàtica de sistemes

realitzat per

Jordi Mitjana Trullàs

i dirigit per

Gonzalo Vera Rodríguez

Escola d'Enginyeria

Sabadell, *juny* de *2011*

El sotasignat, ***Gonzalo Vera Rodríguez***

professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció per

Jordi Mitjana Trullàs

I per a que consti firma la present.
Sabadell, ***juny*** de ***2011***

Signat: ***Gonzalo Vera Rodríguez***

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: Avaluació de la tecnologia de les GPUs.	
Autor: Jordi Mitjana Trullàs	**Data:** ***juny 2011***
Tutor: Gonzalo Vera Rodríguez	
Titulació: Enginyeria informàtica de sistemes	

Paraules clau

- GPU.
- GPGPU.
- CUDA.

Resum del projecte

- Català: En el projecte s'ha dut a terme un estudi sobre la tecnologia que aporten les targetes gràfiques (GPU) dins l'àmbit de programació d'aplicacions que tradicionalment eren executades en la CPU o altrament conegut com a GPGPU.

  S'ha fet un anàlisi profund del marc tecnològic actual explicant part del maquinari de les targetes gràfiques i de què tracta el GPGPU. També s'han estudiat les diferents opcions que existeixen per poder realitzar els test de rendiment que permetran avaluar el programari, quin programari està dissenyat per ser executat amb aquesta tecnologia i quin és el procediment a seguir per poder utilitzar-los.

  S'han efectuat diverses proves per avaluar el rendiment de programari dissenyat o compatible d'executar en la GPU, realitzant taules comparatives amb els temps de còmput.

  Un cop finalitzades les diferents proves del programari, es pot concloure que no tota aplicació processada en la GPU aporta un benefici. Per poder veure millores és necessari que l'aplicació reuneixi una sèrie de requisits com que disposi d'un elevat nombre d'operacions que es puguin realitzar en paral·lel, que no existeixin condicionants per a l'execució de les operacions i que sigui un procés amb càlcul aritmètic intensiu.

- Castellà: En el proyecto se ha llevado a cabo un estudio sobre la tecnología que aportan las tarjetas gráficas (GPU) dentro del ámbito de programación de aplicaciones que tradicionalmente eran ejecutadas en la CPU o también conocido como GPGPU.

  Se ha realizado un análisis profundo del marco tecnológico actual explicando parte del hardware de les tarjetas gráficas y de que trata el GPGPU. También se han estudiado las diferentes opciones que existen para poder realizar las pruebas de rendimiento que servirán para evaluar el software, que software está diseñado para ser ejecutado con ésta tecnología y cuál es el procedimiento a seguir para poder utilizarlos. Se han efectuado diferentes pruebas para poder evaluar el rendimiento del software diseñado o compatible de ejecutar en la GPU, realizando tablas comparativas con los tiempos de cómputo.

  Una vez realizadas las pruebas de software, se concluye que no todas las aplicaciones procesadas en la GPU aportan beneficios. Para poder ver las mejoras es necesario que la aplicación reúna una serie de requisitos como disponer de un elevado número de operaciones que se puedan realizar en paralelo, que no existan condicionantes para la ejecución de las operaciones y que sea un proceso de cálculo intensivo.

- Anglès: This project has been carried out a study about the technology that bring the graphics cards (GPU) within the scope of programming applications which traditionally were executed in the CPU or also known as GPGPU.

  It has made a thorough analysis of the current technological framework explaining the hardware side of the graphics cards and what GPGPU is. It has also studied the different options available to performance testing which will help to evaluate the software, decide which software is designed to run with this technology and what is the procedure to use it.

  It has been carried out various tests to evaluate performance or compatible software designed to run on the GPU, making comparative computation times tables.

  Once concluded the software testing, it is concluded that not all

applications processed on the GPU provide benefits. To see the improvements it is necessary that the program satisfies certain requirements such as having a high number of operations which can be performed in parallel, there are no conditions for execution operations and the process has to be a compute-intensive process.

Agraïments

A la meva xicota Anna pels seus ànims i el seu ajut.

Als meus pares pel seu suport durant tota la carrera

Al meu tutor Gonzalo Vera Rodríguez per haver-me guiat durant tota l'elaboració del projecte.

ÍNDEX

ÍNDEX DE FIGURES

CAPÍTOL I: INTRODUCCIÓ

En els últims anys el maquinari i el programari han evolucionat molt ràpidament, permetent crear i executar aplicacions que temps enrere eren impensables. Cada cop s'estan utilitzant amb més freqüència programes dedicats a estudis científics, alguns d'ells realment complexos i amb una gran càrrega de còmput. Molts investigadors que executen aquest tipus de programes busquen la manera de reduir el temps d'execució i com poder fer anar l'aplicació més ràpid. En aquest projecte veurem una possible solució.

1. Motivació

La motivació de dur a terme aquest projecte es va iniciar des del CRAG (Centre de Recerca en Agrigenòmica). Un grup d'investigadors del centre utilitzen, en el seu àmbit laboral diari, un programa dedicat al càlcul de simulacions coalescents de poblacions estructurades anomenat *msms*, el qual dedica la major part de la seva execució a fer iteracions amb els valors de les dades d'entrada per a poder generar, finalment com a valor de sortida, un estadístic que el nostres investigadors faran servir per continuar amb el seu treball.

Es defineix la coalescència com les relacions d'herència entre dos al·lels, on cada un és cadascuna de les variants en que es pot trobar un gen dins d'un cromosoma. Aquest model de genètica fa un seguiment de tots els al·lels d'un gen compartit per tots els membres d'una població, a una còpia ancestral, provocant així un gran cost computacional per poder realitzar simulacions. Conseqüentment l'execució del programa per realitzar simulacions a gran escala desemboca a temps d'execució molt elevats, per tant existeix una necessitat de reduir aquest temps de càlcul i és aquí on apareix per primer cop el terme de GPU, un processador gràfic que pot ser utilitzat per intentar reduir el temps de còmput en càlculs massius.

La GPU (Graphics Processing Unit / Unitat de Processament Gràfic) és un processador dedicat al processament de gràfics i operacions en coma flotant, usat per alleujar la càrrega de treball del processador central en aplicacions com videojocs o aplicacions 3D. Algunes de les seves

característiques són el gran paral·lelisme que proporcionen les múltiples unitats de còmput disponibles i l'optimització de càlculs en coma flotant, les quals es consideren molt atractives per el seu ús en aplicacions fora de l'àmbit gràfic, especialment en el àmbit científic i de simulació.

Dins de l'àmbit de les GPU han aparegut recentment noves solucions com OpenCL, ATI Stream o CUDA, que són infraestructures compostes de biblioteques, compiladors i llenguatges que permeten pensar en la programació de propòsit general (no només en tema de gràfics) fent servir les GPU en lloc de la CPU.

Fins no fa molt temps, abans de l'aparició d'aquestes solucions, l'objectiu de les GPUs era facilitar la programació de petits blocs de codi que apliquen un determinat procés als vèrtex de la geometria (vèrtex shaders) i als píxels resultants (píxel shaders). Amb aquestes noves eines es pot aprofitar tota la potència de la GPU ja que són capaces d'explotar el gran nivell de paral·lelismes de què disposen els dispositius.

2. Context

Els microprocessadors basats en una única unitat de procés (CPU) han incrementat el rendiment de les aplicacions durant dècades fins l'actualitat. Aquests microprocessadors assolien varis GFLOPS (giga flops, FLoating point OPerations per Second) de còmput en els equips domèstics i centenars de GFLOPS en servidors en clúster.

Aquest incessant augment del rendiment ha permès que les aplicacions desenvolupin moltes més funcionalitats i, al mateix temps, ha augmentat la demanda de noves millores a mesura que els usuaris s'han anat acostumant a les últimes tecnologies.

Durant aquesta etapa les millores de rendiment s'obtenien bàsicament augmentant la rapidesa en que el maquinari podia executar el programari, així cada nova generació de microprocessadors executava el mateix programari més ràpid que el seu antecessor. No obstant, aquest model d'evolució del maquinari es va veure frenat a principis del segle XXI ja que, per diferents motius, la freqüència del rellotge de la CPU no es va poder

continuar augmentant i com a conseqüència es va acabar en sec aquest model d'evolució.

Aquest fet va motivar un canvi profund en el sistema del disseny dels microprocessadors que tenien els fabricants, cosa que va causar l'aparició de les CPU multi-core, les quals disposen de varies unitats de processament dins del mateix xip.

Abans de l'aparició dels multi-core totes les aplicacions desenvolupades pels programadors es basaven en un model seqüencial. El programa era executat seqüencialment en un únic core i en cada nova generació de processadors que apareixia al mercat, el programa s'executava més ràpidament.

A partir de l'aparició dels multi-core els desenvolupadors de programari van experimentar grans canvis en la programació, prioritzant així la paralel·lització del codi intentat que diferents fils d'execució col·laborin conjuntament per accelerar el programa. Tot això ha desembocat en el massiu desenvolupament de programes en paral·lel, el que certes fonts anomenen com la revolució del paral·lelisme.

Actualment, la gran majoria de microprocessadors es basen en arquitectures paral·leles i les targetes gràfiques no en són una excepció. El processador d'aquestes és anomenat GPU (Graphic Processing Unit) i sempre ha estat lligat a les operacions de coma flotant. Justament les GPUs van ser creades com a instrument per alliberar la CPU del renderitzat (procés de complex càlcul que permet generar una imatge des d'un model), però avui en dia ja s'estan utilitzant com a un coprocessador paral·lel per a la CPU.

Una situació similar es va donar temps enrere amb el coprocessador matemàtic, que tenia la funció d'alliberar al microprocessador de les operacions matemàtiques. A principis de la dècada dels 80 els microprocessadors de la família 8088 comptaven amb aquest coprocessador com a un circuit de suport, però no va ser fins el 1989 amb l'aparició del 486 que no es va integrar el coprocessador dins del microprocessador.

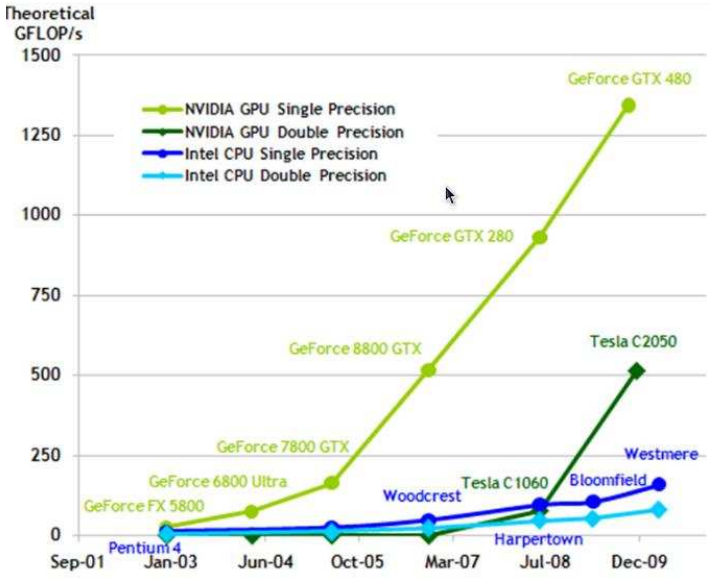

Figura 1. Comparativa GFLOPS entre GPU i CPU.

Com es pot apreciar en la figura 1, existeix una gran diferència de rendiment entre les GPUs i les CPUs a l'hora de fer càlculs en coma flotant. La diferència neix en que les GPUs estan dissenyades per a la computació en paral·lel i per la computació intensiva, tema principal del renderitzat de gràfics, a part de dedicar els transistors que componen el processador al processament de dades en lloc de l'emmagatzematge de dades i el control de flux que duen a terme les CPUs.

Les targetes gràfiques d'última generació s'han convertit en autèntics supercomputadors a un preu relativament baix comparat amb una CPU. Aprofitant els avantatges que ens ofereixen aquestes targetes, diversos fabricants de processadors gràfics han optat per dissenyar alguns mètodes per a poder programar sobre el seu maquinari. És el cas de nVidia, que ha creat un conjunt d'eines de desenvolupament i una arquitectura basada en el llenguatge C, anomenada CUDA (Compute Unified Device Architecture) amb el propòsit de facilitar la programació de caràcter paral·lel per les seves GPUs.

Aquestes tècniques de programar sobre la GPU aplicacions que fins ara es destinaven a la CPU són anomenades GPGPU (General Purpose computation on the GPU) i es basen en solucions com les que proporciona nVidia amb la seva eina CUDA, que permeten transferir el codi que ha de ser executat al processador principal cap a la GPU (figura 2).

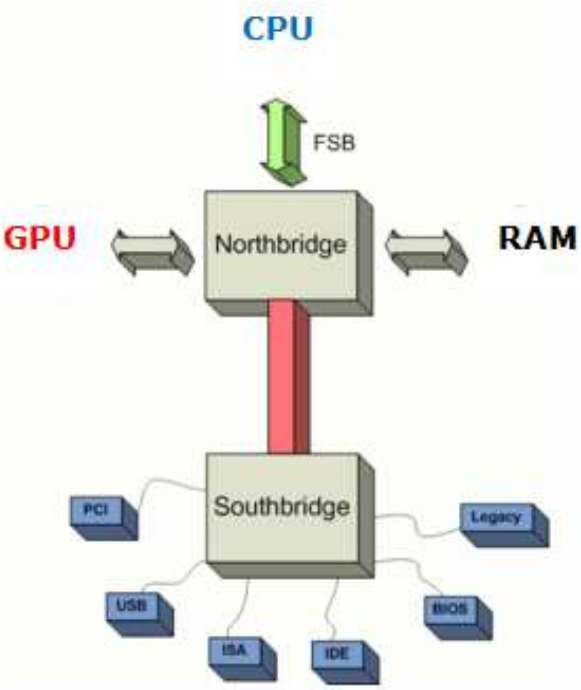

Figura 2. Esquema bàsic de l'arquitectura d'un computador.

Tot això desemboca a la necessitat de re-inventar la forma de generar codi i de re-interpretar el que ja està escrit, ja que ara s'ha de pensar en la manera de paral·lelitzar l'execució per tal de que el codi convencional esdevingui un codi en paral·lel i per a una execució concurrent.

3. Situació actual

Un dels programes que utilitzen els investigadors per a poder assolir els seu objectius és el *msms*, un programa per a calcular estadístics a partir d'unes certes dades introduïdes, que depenent del volum de dades introduïdes augmenta exponencialment el seu temps d'execució.

El desig de qualsevol usuari final d'una aplicació és poder usar l'aplicació fàcilment, que tingui una interfície amigable i que al demanar-li un resultat, tardi el mínim temps possible. És evident que per fer un càlcul d'un estadístic menor, el temps per assolir un resultat no és gaire elevat, però la situació real és que per a poder fer un estadístic que s'aproximi a la realitat, són necessàries moltes iteracions, per tant el temps en assolir el resultat esperat es pot allargar molt.

4. Objectius del projecte

Per a la realització d'aquest projecte ens hem fixat un objectiu general per indicar la finalitat i els punts que es pretenen assolir. El principal objectiu del projecte és poder avaluar la recent tecnologia de les GPUs en l'àmbit científic. D'aquesta manera podrem, un cop finalitzat el projecte, diferenciar en quins casos podem aplicar aquesta tecnologia i ens quins no. També podrem intuir el grau d'adaptació que podrien tenir diferents tipus d'aplicacions i mostrar els beneficis d'utilitzar GPUs dins de l'àmbit científic.

Per poder assolir aquest objectiu hem seleccionat un conjunt d'eines de desenvolupament que ens permetran codificar algoritmes en las GPUs del fabricant *nVidia*. Aquest conjunt d'eines s'anomena CUDA (Compute Unified Device Architecture) i intenta explotar tots els avantatges que tenen les GPUs davant les CPUs utilitzant el paral·lelisme que ofereixen els múltiples nuclis dels dispositius de *nVidia*.

Per altra banda, un altre objectiu de caire més particular és poder reduir el temps d'execució de l'aplicació que fa servir el nostre investigador (*msms*) utilitzant una GPU de nVidia i el conjunt de eines CUDA.

5. Proposta de solució

En el context actual tenim tot un ventall nou de possibilitats que ens aporten les noves solucions per a les GPUs, les quals mitjançant diverses eines permeten reduir el temps de còmput d'una execució enviant els càlculs a la targeta gràfica en comptes del processador. Mitjançant aquestes eines pretenem alliberar el processador de còmput, millorar en la mesura del que sigui possible la velocitat d'execució dels programes basats en càlculs en coma flotant i, a la vegada, poder assolir un nivell més alt de paral·lelisme.

S'ha de tenir en compte que és molt costós generar una solució que aporti beneficis generals a qualsevol aplicació. Algunes aplicacions estaran escrites amb diferents llenguatges, cada aplicació tindrà moltes funcions diferents i no totes les funcions són aptes per a ser executades en una GPU.

Per generar una solució necessitem un cas més concret, necessitem trobar una solució no tan general que ens permeti avaluar la tecnologia en moments particulars, on aquesta desenvolupi les seves virtuts i tota la seva potència i així poder extrapolar les nostres conclusions en casos similars.

Per això el nostre cas particular serà avaluar les noves tecnologies que aporten les GPUs en el món de la ciència utilitzant el *msms*, ja que va ser l'aplicació que va originar als investigadors la necessitat de buscar una solució al seu problema.

CAPÍTOL II: ESTUDI DE VIABILITAT

L'estudi de viabilitat consta de les descripcions del cas general i el cas específic que es tractarà els objectius del projecte, els requisits del sistema, les seves restriccions i les diferents alternatives per donar solucions als casos proposats. També trobarem la planificació del projecte i, finalment, unes petites conclusions. L'objectiu d'estudiar aquests aspectes és per tant determinar la viabilitat dels objectius descrits en el capítol de la introducció.

1. Introducció

Per una banda, i com a cas més general, volem avaluar la tecnologia de càlcul i de paral·lelisme basada en GPUs dins de l'àmbit científic, quins beneficis aporta, si és aplicable en qualsevol àmbit i quines diferències presenta la GPU de la CPU.

Per l'altre banda, i com a cas particular del projecte, es vol aplicar aquesta tecnologia en un programa en concret, el *msms*, modificant el codi font i fent les operacions que siguin necessàries per tal de que la part del programa on hi hagi el major còmput de operacions, sigui executat per la targeta gràfica en comptes de la CPU.

Per a poder entendre el que es vol dur a terme en aquest projecte, primer recordarem i ampliarem què és el msms.

El msms és un programa de simulació coalescent. És capaç de modelar l'estructura de la població i la demografia, i pot ser utilitzat per estudiar la coalescència existent entre poblacions estructurades o adaptacions locals a l'empremta genètica. Aquest procés el porta a terme construint un arbre genealògic a partir de les fulles (o sigui construeix l'arbre al revés) i mitjançant una sèrie de processos n'obté un estadístic.

2. Tipologia i paraules clau

En la part principal del projecte es durà a terme un estudi sobre diferents aplicacions que utilitzen les GPUs com a processador per realitzar la majoria de còmput.

La part restant del projecte consisteix en adaptar un programa (el msms) per a que aquest s'executi en la targeta gràfica, per tant el projecte té una part de desenvolupament necessari per avaluar els sistemes. Per altra banda, un cop el *msms* estigui adaptat per a aquesta nova funcionalitat, es mostrarà la part de comparacions, comprovacions, i proves on s'hauran d'avaluar els resultats obtinguts i l'eficàcia dels nous canvis. Aquesta part del projecte serà de comparació.

Paraules clau escollides per a definir aquest projecte:
msms, GPU, CPU, CUDA, GPGPU.

Totes aquestes paraules estan explicades en l'apartat de definicions, acrònims i abreviacions (annex I).

3. Descripció del cas general

La tècnica de fer servir les GPUs per a fer computació d'aplicacions tradicionalment tractades per la CPU (GPGPU) ha anat en augment durant els darrers anys.

La tecnologia GPGPU està prenent una gran importància en el món de la programació degut a que incrementarà de manera significativa diversos tipus de tasques que els usuaris utilitzen en el seu dia a dia, com per exemple les conversions de àudio i vídeo, aplicacions gràfiques intensives o en els jocs.

Encara que sigui un concepte relativament recent, ens centrarem en estudiar les capacitats, avantatges i inconvenients que ofereixen les GPUs.

4. Descripció del cas particular

El *msms* és un programa dedicat al càlcul a les simulacions coalescents de poblacions estructurades. La funció que li donem en aquest projecte és la de calcular un estadístic a partir d'unes mostres de DNA extretes de *n* individus d'una mateixa espècie. La part d'investigació que duen a terme els investigadors del CRAG es focalitza majoritàriament en l'obtenció d'estadístics, per tant les mostres de DNA necessàries com a entrada de dades del *msms* es creen automàticament. D'aquesta manera si es necessita fer un estadístic de 1000 individus no farà falta tenir les mostres d'aquests.

Mitjançant una sèrie d'iteracions es calcularà un estadístic sobre les possibilitats que existeixen de que dos individus tinguin algun tipus de llinatge.Com més aproximat a la realitat es vol aconseguir que sigui l'estadístic, més iteracions requereix el msms per a poder calcular-lo.

Així doncs els investigadors per realitzar la seva feina han de fer el següent cicle:

-Generar les mostres automàticament de DNA

-Esperar resultats

-Obtenir l'estadístic

Podem veure més gràficament en la següent figura 3 on cada color representa una etapa d'aquest cicle respecte el temps.

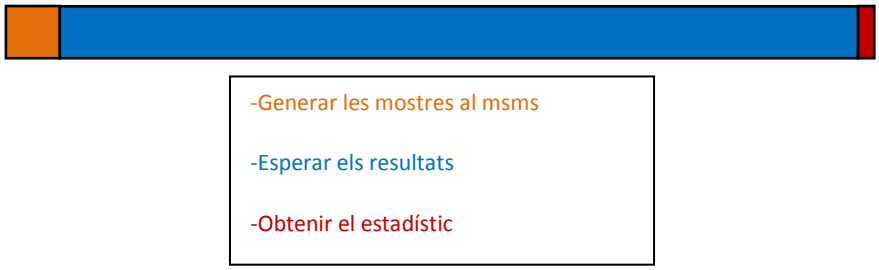

Figura 3. Cicle respecte el temps.

El problema principal que presenta és que per a poder fer un estadístic que s'aproximi al màxim a la realitat és necessari fer moltes repeticions amb les dades inicials de DNA que s'han generat automàticament. Si el nombre de mostres és relativament gran i volem aproximar els resultats al màxim a la realitat, el programa he de fer milions de iteracions i per tant, el temps d'execució es dispara.

En definitiva, el problema que volem resoldre en aquest cas particular del projecte és el de reduir el temps d'execució del programa msms quan hi ha un gran nombre de iteracions a calcular.

5. Objectius del projecte

El que pretenem principalment amb aquest projecte és avaluar la tecnologia que aporten les GPUs dins de l'àmbit científic mitjançant unes proves de rendiment, les quals ens permetran valorar els beneficis, la dificultat, la utilitat i ens quins casos ens surt a compte aplicar aquesta tecnologia. Dins d'aquest objectiu, hi trobem un cas més específic, que es basarà en traspassar l'execució del programa a la targeta gràfica utilitzant aquesta tecnologia i al mateix temps minimitzant els costos per tal de que els usuaris del msms vegin reduït el temps d'espera.

Tenim diferents alternatives per poder dur a terme aquesta tasca.

5.1 Objectius Generals

1. Estudiar la tecnologia GPGPU.
2. Estudiar les diferents possibilitats que existeixen per aplicar GPGPU.
3. Escollir una de les possibles opcions per realitzar les proves de rendiment.
4. Realitzar les proves.
5. Avaluar els beneficis i inconvenients que apareixen.

5.2 Objectius Específics

6. Familiarització amb el msms.
7. Calcular el temps que tarda el msms en fer x càlculs.
8. Estudiar com fer executar el programa en la targeta gràfica.
9. Familiarització amb la llibreria escollida.
10. Fer funcionar la llibreria seleccionada amb el msms.
11. Recalcular els temps d'execució del msms per x càlculs amb la llibreria escollida i veure la millora.
12. Verificar que els resultats obtinguts són els esperats.

5.3 Prioritats

Objectius Crítics: 1,2,3,4
Objectius Prioritaris: 5,6,7,8
Objectius Secundaris: 9,10,11,12

6. Parts interessades

6.1 Investigadors del CRAG

Els investigadors són les persones que utilitzen el msms en el seu lloc de treball.

6.2 Stakeholder

Jordi Mitjana: responsable del projecte. Supervisa la feina de l'analista, el programador, el tècnic de proves i el director de projecte, a part de marcar les pautes a seguir.

6.3 Supervisor

Gonzalo Vera: supervisarà el treball del responsable de projecte.

7. Referències

Aquest projecte queda sota la normativa de projectes finals de carrera de la UAB, la normativa de privacitat de dades.

1. Normativa de projectes d'enginyeria tècnica:
http://www.uab.cat/Document/639/153/normativaProjectesEEsabadell.pdf

2. LOPD:
https://www.agpd.es/portalweb/canaldocumentacion/legislacion/estatal/index-ides-idphp.php

*3.*nVidia:Normativa sobre les llicències de nVidia
http://www.nvidia.com/object/nv_swlicense.html

*4.*CUDA: Normativa sobre la llibreria i eines de CUDA
http://developer.download.nvidia.com/compute/cuda/2_2/toolkit/docs/cudaprof_eula.pdf

8. Producte i documentació del projecte

El producte final serà la mateixa aplicació optimitzada de tal manera que, el procés es calcularà en la GPU en comptes d'executar-se en la CPU.

Caldrà entregar una memòria on s'observi el canvi que s'ha produït en el temps total de una mateixa execució, de tal manera que es puguin treure unes conclusions que demostrin que el programa és més ràpid en la targeta gràfica que en el processador.

9. Estudi del cas específic

El nostre investigador disposa d'un programari lliure (msms) que utilitza en el seu àmbit professional diàriament. Aquest programari està dissenyat en java i pot ser executat en un PC en qualsevol sistema operatiu que disposi de una màquina virtual de java.

9.1 msms

EL *msms* calcula estadístics que més tard l'usuari farà servir per continuar amb el seu treball. Com més precisió i més aproximació a la realitat busca l'usuari, més iteracions ha d'executar el programa i per tant serà necessari més temps per donar els resultats (figura 4).

L'usuari busca reduir aquest temps d'espera de resultats, i al mateix temps espera que la solució aportada no comporti una despesa econòmica molt gran.

Aquest temps d'espera és degut a que el programa crea un arbre genealògic a partir de les mostres generades de DNA. Aquestes mostres són el punt de partida per començar a desenvolupar l'arbre, però no en són la base, sinó les fulles.

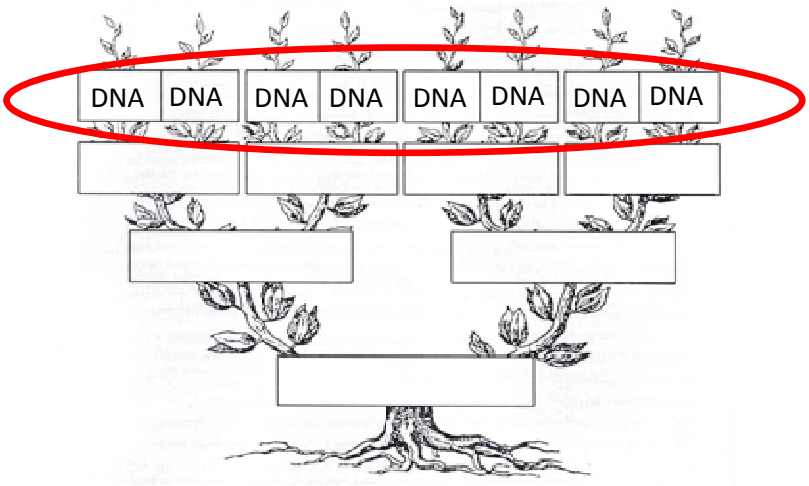

Figura 4. Representació de l'arbre creat pel msms.

Així doncs podem resumir que el *msms* crea un arbre genealògic a partir de les mostres generades de DNA que seran les fulles d'aquest, i amb una sèrie de iteracions el programa va recorrent generacions de l'arbre establint llinatges, migracions i fins i tot mutacions entre els individus.

9.2 Informació

Sabem que una GPU es capaç de realitzar càlculs sobre gràfics d'una forma molt més ràpida que una CPU tot i tindre una menor freqüència de rellotge, ja que està optimitzada per realitzar càlculs de valors en coma flotant (usuals en entorns 3D) i tenen un alt nivell de paral·lelisme. Podem aplicar aquestes funcionalitats de les GPUs per optimitzar la execució del msms.

9.3 Restriccions

L'aspecte econòmic és una de les principals restriccions. Busquem la millor forma d'optimitzar el temps d'execució, però sense treure l'ull de sobre als costos de la solució.

Una altre gran restricció és la disponibilitat dels recursos de maquinari de que es disposin.

L'utilització d'alguna de les possibles llibreries (desenvolupades generalment en llenguatge C) amb el programa msms (desenvolupada en llenguatge Java) és una altre restricció important, on s'haurà de prestar atenció en la comunicació que s'estableixi entre les dues parts.

10. Lògica del sistema

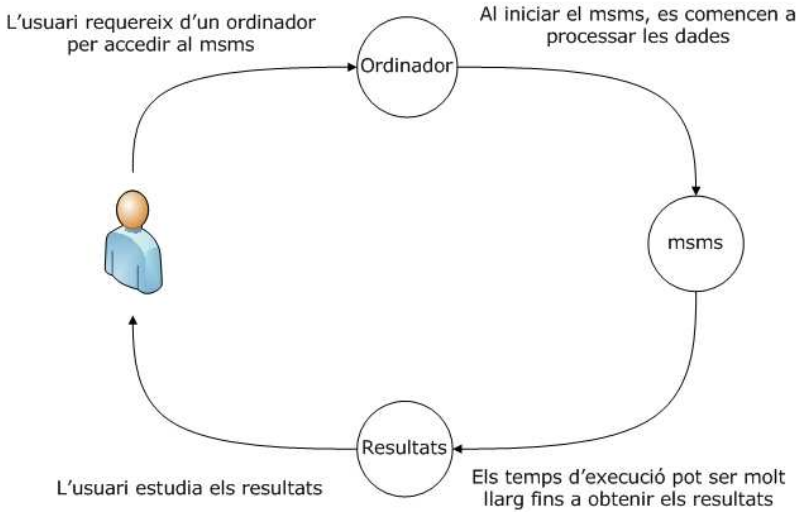

Figura 5. Representació de la lògica del sistema

11. Descripció física

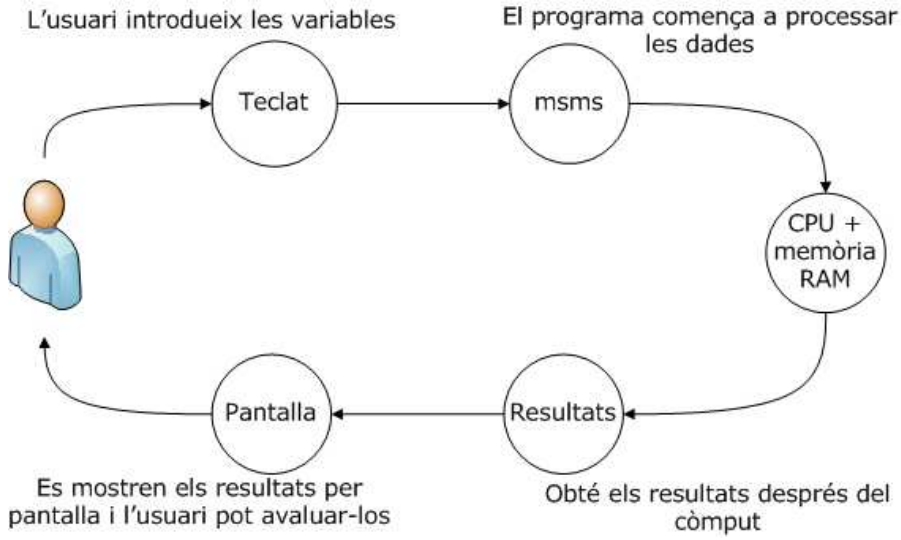

Figura 6. Descripció física del procés.

12. Usuaris i/o personal del sistema

El programa l'utilitzen els investigadors del CRAG, per tant són el usuari del sistema, però podríem pensar que qualsevol persona que utilitzi aquest programari podria ser un possible usuari.

El nostre usuari s'encarrega de la execució del programa, escollint totes les variables inicials que necessiti per poder cercar el estadístic que li permetrà continuar el desenvolupament del seu treball.

13. Diagnòstic del sistema

13.1 Deficiències

El principal inconvenient que presenta el msms, és el gran temps d'execució del programa en el moment de cercar un estadístic que s'aproximi a la realitat.

13.2 Millores

Intentar reduir el temps que l'usuari ha d'estar esperant els resultats.

14. Requisits del sistema

14.1 Requisits funcionals

En aquest apartat llistarem els requisits que defineixen el comportament del programari (càlculs, detall tècnics, manipulació de dades, i altres funcionalitats que ens mostren com els casos d'ús es portaran a terme).

- Programació/adaptació del msms: S'haurà de buscar la manera d'adaptar el msms a la tecnologia de les GPUs.

- Comunicació entre Java i C: Haurem de prestar atenció en el procés de comunicació que s'establirà entre el programa msms (dissenyat en java) i la llibreria decidida (dissenyada en C).

14.2 Requisits no funcionals

Els requisits no funcionals són tots els requisits que no descriuen les funcions a realitzar (hi inclourem els requisits del maquinari).
- Necessitat d'un PC.
- Necessitat de que el PC disposi d'una GPU compatible amb alguna de les llibreries que es poden utilitzar.
- Programari tant per l'avaluació de la tecnologia, com el de programació. Més específicament aplicacions per a usuaris finals, llibreries i compiladors.

15. Restriccions del sistema

El nostre sistema ha d'estar adaptat a les següents restriccions:

- Els programes han de ser compatibles amb el sistema operatiu en què es realitzaran les proves.

- La llibreria que utilitzarem està dissenyada en C, per tant s'haurà de buscar la millor manera per poder establir una connexió entre el programa i la targeta gràfica.

- Per executar l'aplicació modificada i poder utilitzar la llibreria en C escollida, necessitarem una targeta gràfica que sigui totalment compatible amb aquesta.

16. Catalogació i priorització dels requisits

Requisits	*Funcional*	*Essencial*	*Condicional*	*Opcional*
Adaptació del msms	*Si*		*x*	
Comunicació entre Java i C	*Si*	*X*		
CPU amb targeta gràfica	*No*	*X*		
Targeta gràfica compatible amb la llibreria	*No*	*X*		
Programes compatibles SO	*No*	*X*		

17. Alternatives al cas general i selecció de la solució

17.1 Alternativa 1

Una de les opcions per avaluar les GPUs per aplicacions d'àmbit general és OpenCL, un estàndard obert i lliure per a la programació paral·lela de CPUs i GPUs. Va ser creat inicialment per Apple qui encara posseeix els drets, però actualment ho desenvolupa el grup Khronos.

17.2 Alternativa 2

CUDA és una altre possibilitat, està dissenyada per el fabricant nVidia per donar un rendiment més òptim al seus productes.

17.3 Alternativa 3

El fabricant ATI (actualment fusionat amb AMD) també disposa de una llibreria anomenada ATI Stream (o AMD Stream com es vol començar a dir a partir d'ara), que també millora el rendiment de les seves GPUs notablement.

17.4 Solució proposada

La solució que es farà servir en aquest projecte, es durà a terme amb CUDA a causa d'un requisit no funcional.

Es disposa de una targeta gràfica de la casa nVidia, concretament del model GeForce 8400 GS totalment compatible amb CUDA, així doncs per una restricció econòmica, s'utilitzarà el programari que proporciona nVidia per les seves GPUs.

18. Alternatives al cas específic i selecció de la solució

18.1 Alternativa 1

Comprar ordinador amb més potencia. Aquesta solució és la més ràpida en quant a temps per a obtenir una solució, però la més cara econòmicament. Tenir una màquina el doble de potent, que ens costarà el doble o més que la màquina de que disposem, tampoc ens dóna cap garantia de poder executar el programa el doble de ràpid.

18.2 Alternativa 2

Modificar el codi per millorar els colls d'ampolla i intentar paral·lelitzar. Aquesta solució és la més econòmica en quan a comprar maquinari nou es refereix, però al mateix temps és la més costosa en temps ja que s'hauria d'estudiar tot el codi per tal de trobar els colls d'ampolla i intentar eliminar-ne tots els possibles. En el cas de tenir disponible un processador amb doble nucli, s'hauria d'intentar paral·lelitzar totes les tasques que no tinguin dependència entre elles.

18.3 Alternativa 3

Implantar una llibreria en C per tal de que l'execució del programa es faci en la GPU. Aquesta solució no té un alt cost econòmic, però te un cost moderat de programació. Afegint la llibreria CUDA podrem tenir un processador de operacions en coma flotant amb molta més disponibilitat

que la CPU per a fer els càlculs i poder paral·lelitzar tasques. La diferència existent entre aquesta proposta i la anterior és el retorn de la inversió, degut a que si arriba a funcionar la millora de rendiment podria ser molt elevada.

19. Solució proposada

La solució més adient per el nostre cas és l'alternativa 3, implementar la llibreria en C CUDA per executar el programa des de la GPU.

Aquesta és la solució escollida perquè a part del temps d'execució, hem de tenir en compte el cost de la solució, i en aquest cas, comprant només una targeta gràfica compatible amb la llibreria CUDA n'hi hauria prou.

En quant al temps, les GPUs estan dedicades als càlculs amb valors en coma flotant, i al mateix temps, permeten una gran paral·lelització, cosa que podria augmentar notablement el rendiment, més inclús que optimitzant el codi.

Aquestes dues funcionalitats ens fan pensar que és la millor solució per a resoldre el problema plantejat.

20. Planificació

En la planificació veurem els recursos humans i materials assignats a cada tasca, les fases del projecte.

20.1 Recursos del projecte

Recursos humans	Remuneració
Director de projecte	100 €/h
Cap de projecte	60 €/h
Analista	45 €/h
Programador	30 €/h

Recursos materials	Cost
Ordinador	1000 €
msms	0 €
Targeta gràfica nVidia	50 €

El valor del programa msms és 0 € perquè és programari lliure, però encara que en el apartat econòmic no faci variar el resultat és un recurs principal del projecte i cal ser esmentat.

20.2 Fases i activitats del projecte

Fases	Descripció
Iniciació	Fase d'iniciació del projecte. Inclou les activitats de definició del projecte, assignació i matriculació.
Planificació	Inclou l'estudi de viabilitat i el pla de projecte.
Anàlisis i disseny	Anàlisis dels requisits funcionals i no funcionals.
Desenvolupament	Fase de desenvolupament de l'aplicació.
Test i proves	Fase de proves del sistema. Inclou test unitaris i d'integració del programa en el nou entorn de treball.
Implantació	L'aplicació modificada s'instal·la en el entorn de treball.
Generació de documents	Fase de documentació del projecte. Inclou manuals i memòria del projecte.
Tancament del projecte	El director del projecte signa l'acceptació i tancament del projecte.
Defensa del projecte	Defensa del projecte davant la comissió.

20.3 Calendari de recursos

En el calendari de recursos distribuïm els recursos humans per a poder realitzar totes les fases del projecte.

- Cap de projecte: Planificació, generació de documents, tancament i defensa. Punts de control.

- Analista: Anàlisi i disseny, implantació i punts de control d'anàlisi, disseny i desenvolupament.
- Programador: Disseny, desenvolupament i test. Parcialment en l'implantació.
- Tècnic de proves: Fase de test.

Els recursos materials s'utilitzaran principalment en les fases de desenvolupament, test i implantació.

20.4 Calendari del projecte:

La data d'inici del projecte es va establir per el dia 1 de febrer del 2011. Tots els recursos humans hi treballaran fins a complir amb totes les tasques, i està previst que s'acabi el dia 28 juny de 2011 amb un total de unes 190 hores invertides aproximadament.

20.5 Quadre de tasques del projecte

Tasques	Durada	Inici	Fi	Predec.	Cost	Treball
Projecte final de carrera	**70,5 días**	**mar 01/02/11**	**mar 28/06/11**		**8.639,62 €**	**190 horas**
Inici del projecte: assignació i matriculació del projecte	0,4 días	mar 01/02/11	mar 01/02/11		128,00 €	2 horas
Planificació	**3,75 días**	**mar 01/02/11**	**mar 15/02/11**	**2**	**980,00 €**	**16 horas**
Estudi de viabilitat	3,75 días	mar 01/02/11	lun 14/02/11		900,00 €	15 horas
Aprovació Estudi Viabilitat (Punt de control)	1 día	lun 14/02/11	mar 15/02/11	4	80,00 €	1 hora
Anàlisi de l'aplicació	**17,6 días**	**mar 15/02/11**	**vie 18/03/11**	**5**	**2.352,00 €**	**51 horas**
Anàlisi de requisits fuincionals i no funcionals	1,67 días	mar 15/02/11	lun 21/02/11		225,00 €	5 horas
Anàlisi de dades	5 días	jue 24/02/11	mié 09/03/11	7	900,00 €	20 horas
Anàlisi de la seguretat i legalitat	2 días	lun 21/02/11	jue 24/02/11	7	360,00 €	8 horas
Documentació de l'anàlisi	3,75 días	mié 09/03/11	vie 18/03/11	9;7;8	675,00 €	15 horas
Aprovació de l'anàlisi (Punt de control)	0,6 días	vie 18/03/11	vie 18/03/11	10	192,00 €	3 horas
Disseny de l'aplicació	**46,45 días**	**vie 18/03/11**	**mié 22/06/11**	**11**	**4.299,62 €**	**107 horas**
Proves programari	25,2 días	vie 18/03/11	lun 30/05/11		1.537,50 €	35 horas
Proves programari amb CUDA	21,48 días	vie 18/03/11	vie 20/05/11		1.317,86 €	30 horas
Programació amb CUDA	11,36 días	vie 18/03/11	mié 06/04/11	11	450,00 €	15 horas
Conclusions	2 días	lun 30/05/11	jue 02/06/11	15;13;14	0,00 €	5 horas
Document del disseny	7,09 días	jue 02/06/11	vie 17/06/11	16	873,91 €	20 horas
Aprovació del disseny (Punt de control)	0,52 días	vie 17/06/11	mié 22/06/11	17	120,35 €	2 horas
Finalitzar projecte	**4,79 días**	**vie 17/06/11**	**mar 28/06/11**	**17**	**880,00 €**	**14 horas**
Generació de documents	2,5 días	vie 17/06/11	mié 22/06/11		600,00 €	10 horas
Tancament del projecte	2 días	mié 22/06/11	lun 27/06/11	20	160,00 €	2 horas
Entrega del projecte	0,5 días	mar 28/06/11	mar 28/06/11	21	120,00 €	2 horas

Figura 7. Descripció de les tasques del projecte a realitzar.

20.6 Dependències

Aquest projecte és gairebé tot lineal, no podem solapar la majoria de tasques perquè moltes depenen de les anteriors com podem comprovar en apartat anterior. No seria lògic començar a desenvolupar el cas específic sense abans haver estudiat els beneficis que aporta CUDA i com és la millor forma d'implementar-los, per exemple. Però si que podem anar avançant per una altra banda mentre es duen a terme les proves de rendiment dels programes. Per tant, en general no podem iniciar una nova fase fins a no tenir assegurada la que estem duent a terme, excepte en determinats moments.

20.7 Calendari temporal

En aquest apartat podem observar el diagrama de Gantt, on podem observar clarament les dependències que hi ha entre cadascuna de les activitats, quin recurs humà té assignat i quin interval de temps necessita per ser desenvolupada.

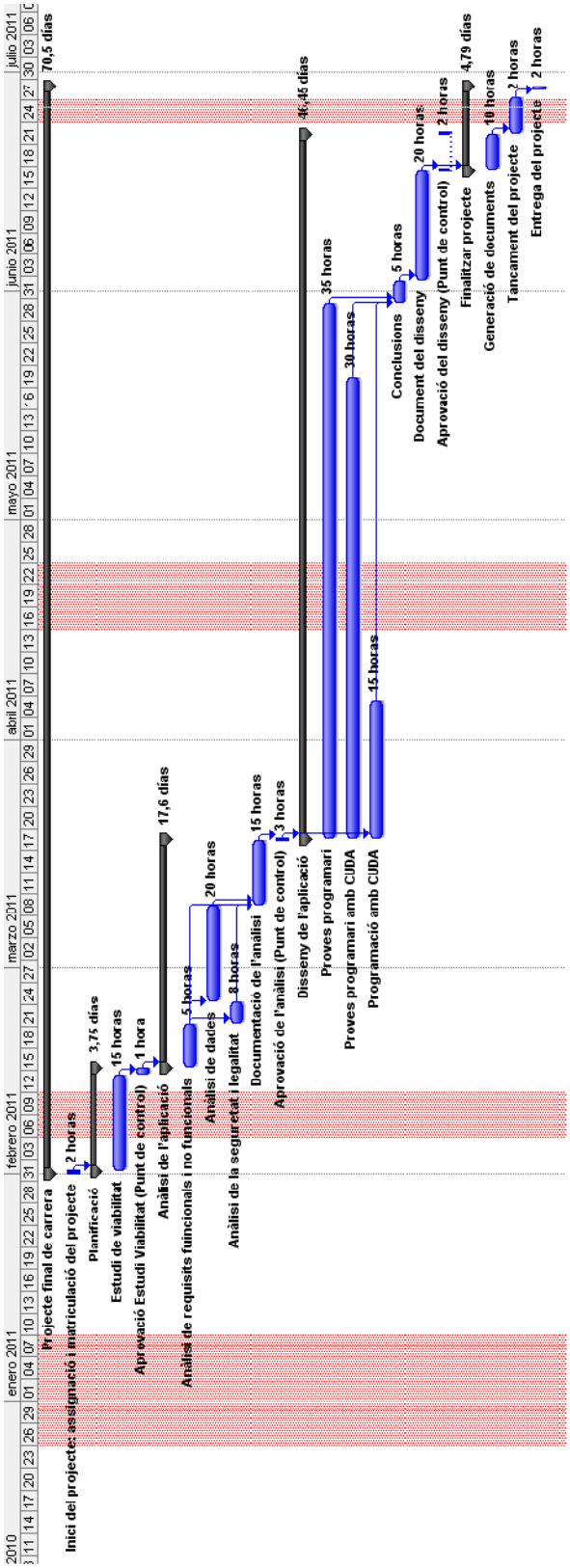

Figura 8. Calendari temporal de les tasques del projecte a realitzar.

20.8 Llista de riscos

1-Planificació temporal optimista: pla de projecte. No s'acaba en la data prevista, augmenten els recursos.

2- Manca alguna tasca necessària: pla de projecte. No es compleixen els objectius del projecte.

3- Canvi de requisits: estudi de viabilitat, anàlisi. Endarreriment en els desenvolupament i resultat.

4- Equip del projecte massa reduït: pla de projecte. Endarreriment en la finalització del projecte, no es compleixen els objectius del projecte.

5- No es fa correctament la fase de test: desenvolupament, implantació. Manca de qualitat, deficiències en l'operativa, insatisfacció usuaris, pèrdua econòmica.

6- Incompliment d'alguna norma, reglament o legislació: en qualsevol fase. No es compleixen els objectius, repercussions legals.

7- Abandonament del projecte abans de la finalització: en qualsevol fase. Pèrdues econòmiques, frustració.

20.9 Catalogació de riscos

	Probabilitat	Impacte
1	Alta	Crític
2	Baixa	Crític
3	Mitjana	Crític
4	Baixa	Crític
5	Mitjana	Crític
6	Baixa	Crític
7	Baixa	Catastròfic

20.10 Pla de contingència

En el pla de contingència establirem les pautes a seguir si per algun motiu ens trobéssim davant d'algun dels riscos anteriors.

	Solució que caldria adaptar davant dels possibles riscos
1	Ajornar alguna funcionalitat, afrontar possibles pèrdues, fer una assegurança.
2	Revisar el Pla de Projecte, modificar la planificació.
3	Renegociar amb el client, ajornar funcionalitat, modificar planificació i pressupost.
4	Demanar un ajornament, negociar amb el client, afrontar pèrdues.
5	Dissenyar els test amb antel·lació, realitzar tests automàtics, negociar contracte de manteniment, donar garanties, afrontar pèrdues econòmiques.
6	Revisar les normes i legislació, consultar un expert, afrontar possibles repercussions penals.
7	No té solució.

20.11 Estimació del cost

Aquí especificarem el cost aproximat del projecte

Recurs humà	Preu
Director de Projecte	322,00 €
Cap de projecte	1.978,00 €
Analista	5.698,00 €
Programador	641,00 €

En total el projecte sumaria doncs uns 8.639 €. Però al tractar-se de un projecte final de carrera tots els recursos humans faran la feina sense cobrar, per tant el cost real del projecte és 0€.

20.12 Resum anàlisi cost – benefici

Un cop el projecte s'hagi realitzat, no farà falta abonar res econòmicament, per tant en l'aspecte econòmic no ens fa falta amortitzar les despeses del projecte.
Per altra banda els possibles beneficis que es poden obtenir poden ser de gran utilitat per els investigadors del CRAG, sobretot si el temps d'execució es veu reduït.

21. Conclusions de l'estudi de viabilitat

Després de decidir les solucions òptimes per al nostre projecte, i avaluar-ne la seva implantació, el seu cost, i les necessitats del usuari, podem assegurar que la proposta que oferim al problema és la més adequada.

Un cop tinguem tot el procés realitzat, serà l'hora d'observar els resultats obtinguts, fixant-nos en els canvis que s'han produït en el temps d'execució, i podrem avaluar d'una forma molt clara la relació que hi haurà entre els temps CUDA i els normals.

D'altra banda l'aspecte econòmic no representarà cap conflicte a l'hora de tirar el projecte endavant, per tant una vegada realitzat el projecte podrem afirmar quin és el benefici que aporten als usuaris les aplicacions GPGPU i ens quins casos val la pena o no traspassar el còmput a la GPU.

Per totes aquestes raons podem concloure que el projecte és **viable**.

CAPÍTOL III: ANÀLISI

1. Marc tecnològic

1.1 El Maquinari

Les targetes gràfiques que existeixen actualment al mercat estan constituïdes per un conjunt de multiprocessadors i una capacitat de memòria amb un accés molt ràpid. Tant el nombre de processadors com la capacitat de memòria canvien depenent de cada model.

Fins fa poc temps l'esquema bàsic de les targetes gràfiques estava constituït d'una memòria DRAM, que utilitzava cada processador (ALU) per a realitzar els seus càlculs com podem veure en la figura 9:

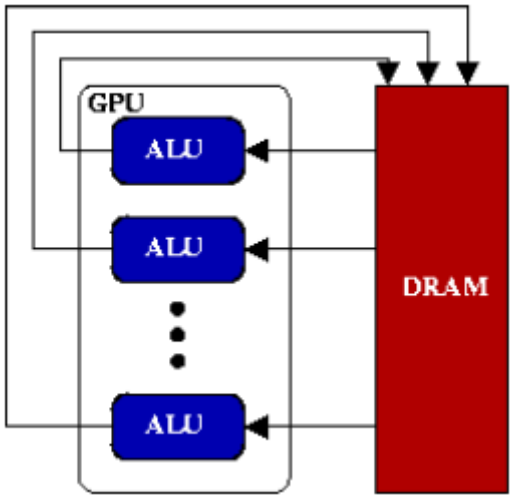

Figura 9. Esquema bàsic d'una targeta gràfica.

L'aparició de les targetes de nova generació combinades amb les eines de CUDA va significar un canvi de maquinari molt important, introduint així una memòria cau dins de la GPU (emulant així una CPU) que permetria compartir dades d'un processador a un altre (figura 10).

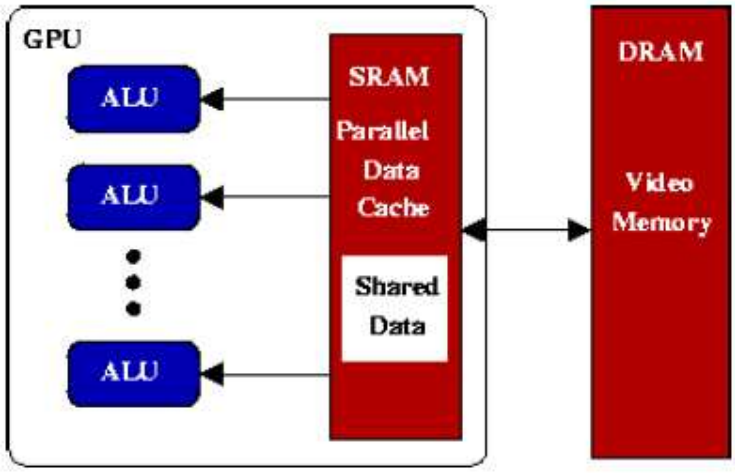

Figura 10. Esquema d'una targeta gràfica actual.

Cada un d'aquets microprocessadors s'encarrega de processar els threads (fils). El disseny intern dels microprocessadors és bastant similar en molts models i són capaços de realitzar operacions de coma flotant com SQRT, LOG o EXP, apart de comptar també amb una unitat de suma i multiplicació (MAD).

La GPU està adequada per executar el mateix programa amb moltes dades en paral·lel, d'aquesta manera el mateix programa s'executa per a cada element i per això requereix un control menor de flux respecte les CPU (figura 11). A més a més, la latència dels accessos a memòria es camufla dins la intensitat dels càlculs d'operacions aritmètiques possibilitant l'eliminació de grans memòries cau pels accessos a memòria.

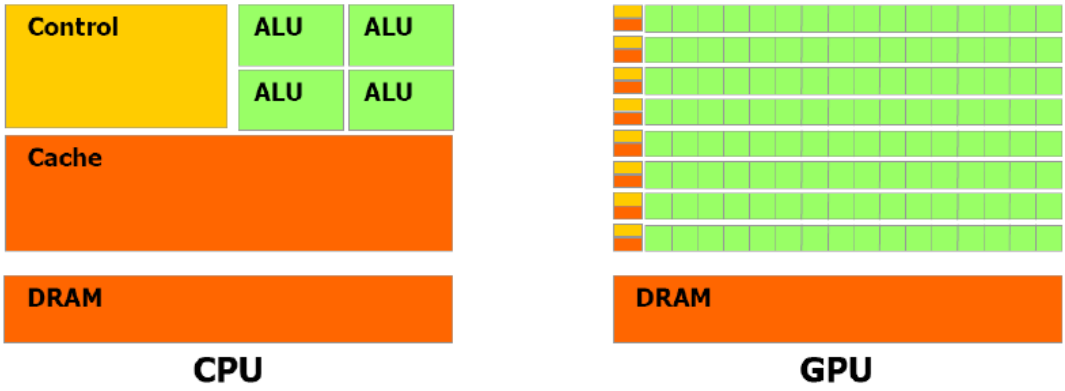

Figura 11. La GPU està formada per moltes més ALUs que una CPU.

El gran canvi de cau que hi ha en les dues arquitectures és degut a que la CPU situa els fitxers amb més nombre d'accessos a la CAU, millorant així el temps d'accés a aquests (la memòria cau és molt més ràpida que la RAM), mentre que les GPUs no requereixen accedir als mateixos arxius un elevat nombre de vegades, ja que les dades no es reutilitzen, i a més el accés a la memòria gràfica és molt ràpid.

Per poder veure amb més detall el flux de les dades i les instruccions sobre aquestes dins de la GPU, podem utilitzar la Taxonomia de Flynn. Michael J. Flynn va desenvolupar una classificació de l'arquitectura dels computadors al 1966 que encara avui ens permet distingir els nous sistemes disponibles. Va definir com a flux d'instruccions al conjunt d'instruccions seqüencials que són executades per un únic processador, i també va definir flux de dades com a flux seqüencial de dades requerits pel flux d'instruccions.

Amb aquesta base va poder establir una classificació dels sistemes segons els flux (figura 12).

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Figura 12. Classificació sistemes segons Flynn.

SISD (Single Instruction Single Data streams): un computador seqüencial que no explota el paral·lelisme ja sigui en les instruccions o en el flux de dades.

SIMD (Single Instruction Multiple Data streams): un computador que explota fluxos de dades múltiples contra un flux simple d'instruccions per realitzar operacions que poden ser paral·lelitzades.

MISD (Multiple Instruction Single Data stream): instruccions múltiples que operen en un flux de dades simple.

MIMD (Multiple Instructions, Multiple Data stream):processador o processadors múltiples que executen simultàniament instruccions diferents amb dades diferents.

Així doncs els SISD (figura 13) es caracteritzen per tenir un únic flux d'instruccions sobre un únic flux de dades, per tant s'executa una instrucció d'un flux darrera una altre d'un altre flux i en qualsevol moment només s'està executant una instrucció.

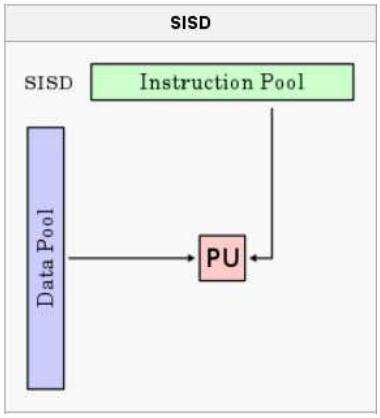

Figura 13. Esquema SISD.

Els sistemes SIMD (figura 14) són els utilitzats en les **GPU** i es basen en un únic flux de instruccions que opera sobre varis flux de dades. El processament és **síncron**, l'execució de les instruccions continua sent sequencial i tots els elements realitzen la mateixa instrucció però sobre un gran quantitat de dades diferents. Aquesta classificació és el origen de les máquines paral·leles.

El funcionament és el següent: La unitat de control envia la mateixa instrucció a totes les unitats de procés (ALU), i cada unitat opera sobre dades diferents utilitzant la mateixa instrucció.

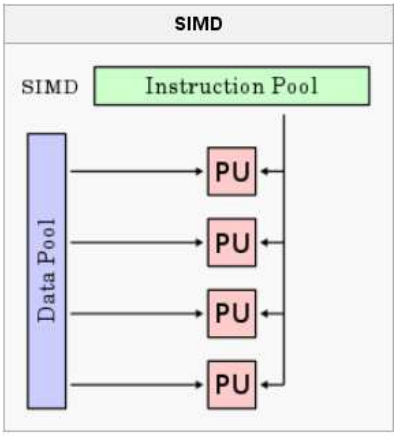

Figura 14. Esquema SIMD.

Els sistemes MISD (figura 15) tenen múltiples instruccions que operen sobre un únic flux de dades, però aquesta arquitectura és poc freqüent.

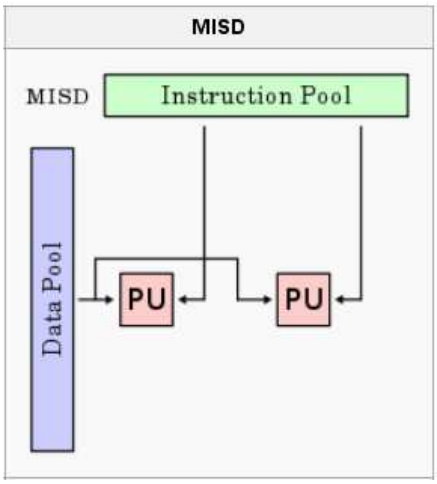

Figura 15. Esquema MISD.

El MIMD (figura 16) és una tècnica per obtenir paral·lelisme. Aquests sistemes permeten intercalar fluxos d'instruccions de memòria asíncrona, per tant, qualsevol processador pot executar diferents instruccions en diferents dades. Aquest sistema és el que utilitzen tots els microprocessadors x86.

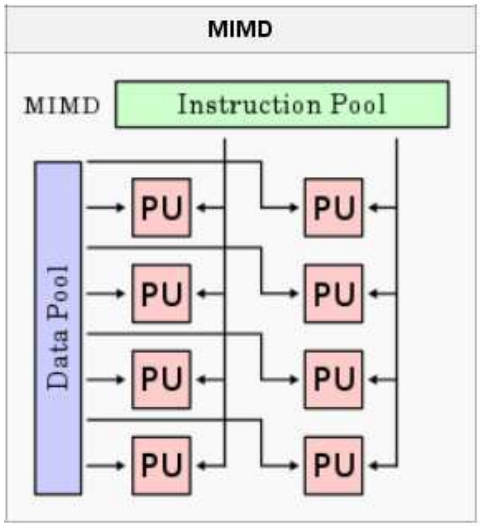

Figura 16. Esquema MIMD.

Per poder situar globalment la GPU podem veure l'esquema genèric de l'arquitectura d'un ordinador que consta d'una targeta gràfica dedicada en la figura 17. La CPU es comunica a través del Northbridge de la placa base amb la memòria principal del sistema (RAM) i amb la ranura de la targeta gràfica (actualment amb la ranura PCI Express).

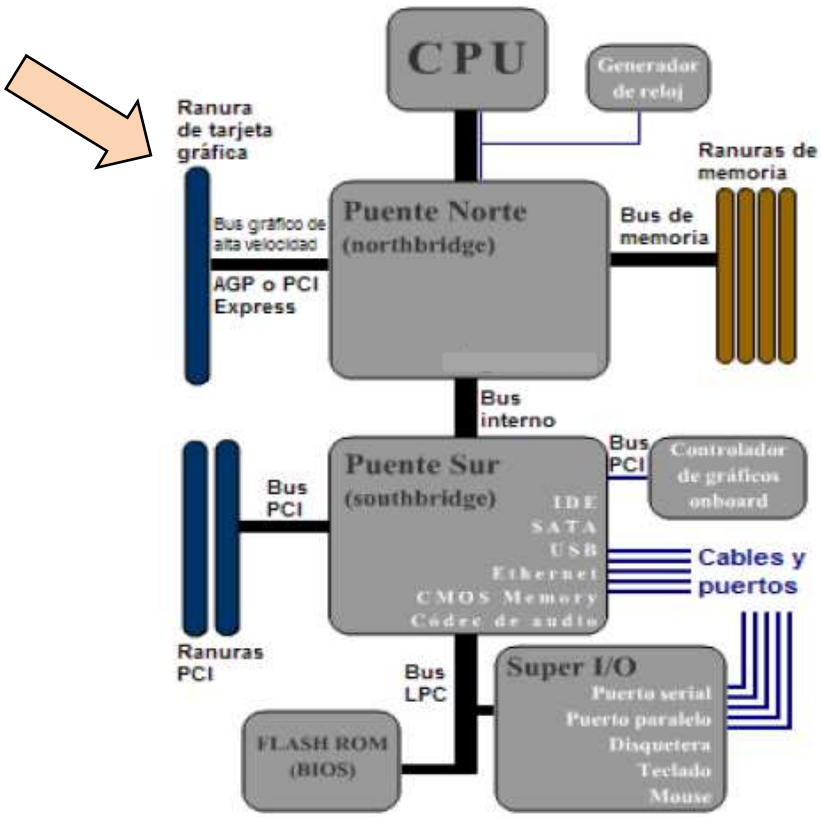

Figura 17. Esquema genèric d'un ordinador. La fletxa indica on s'ubica la targeta gràfica.

La GPU té la possibilitat de comunicar-se amb la memòria local que té integrada en la mateixa targeta sense haver de compartir la memòria RAM del sistema. Aquest fet és molt important a l'hora de veure les diferències que sorgeixen en l'ample de banda dels càlculs.

Amb tot això podem veure que les GPU estan especialment dissenyades per executar problemes que poden ser expressats com a computació de dades en paral·lel amb un gran nombre d'operacions aritmètiques.

Però no ha estat fins l'aparició de CUDA que no s'ha pogut desenvolupar el terme de GPGPU a un gran nivell a causa de:

- La GPU només podia ser programada a través d'una API (Application Programming Interface) gràfica específica per a cada model, cosa que provocava tenir un grau elevat de coneixements de l'arquitectura utilitzada en cada cas per a poder adaptar la targeta gràfica a una aplicació científica.

- Els programes que estaven en la GPU no tenien el suficient control per poder escriure a la DRAM de la targeta gràfica d'una manera eficient.

- Alguns programes veien influït el seu rendiment a causa dels colls d'ampolla sorgits a causa de l'ample de banda de la memòria DRAM.

Totes aquestes limitacions van poder ser sobrepassades gràcies a la utilització de CUDA i les noves arquitectures emprades en les targetes gràfiques.

1.2 GPGPU

En els últims anys, les vies d'investigació i desenvolupament de les arquitectures de computació han anat canviant. Les velocitats dels rellotges de la CPU van veure frenat el seu increment degut, principalment, als problemes de dissipació de calor i consum.

Aquest fet va desencadenar un canvi en els principals fabricants de processadors que van haver de canviar el sistema d'evolució d'anar augmentant el número de cicles per segon, que havia durat dues dècades, a desenvolupar processadors amb varis nuclis on la velocitat passava a un segon terme i apareixia el paral·lelisme.

Però abans de que aquest canvi es produís a les CPUs, les targetes gràfiques ja contaven amb processadors de varis nuclis, les GPUs.

GPGPU és un mot que fa referència a l'ús de la GPU com a un processador de propòsit general. Tot i que les GPUs van ser dissenyades inicialment com a uns processadors de suport de la CPU per alliberar-la del processament gràfic, poden arribar a ser molt útils per a altres aplicacions, ja que es comporten com a coprocessadors paral·lels, a nivell de dades, per la CPU.

En l'actualitat les GPUs ja no s'utilitzen només per al processament gràfic, i això és degut a que:

- Són uns processadors massius que poden superar àmpliament el rendiment de les CPUs en aplicacions on es requereixin paral·lelismes.

- Suporten dades de 32 i 64 bits.

- A partir de l'aparició de CUDA (en el cas de les GPU nVidia), el model de programació és relativament flexible.

- Disposen d'un gran ample de banda per accedir a la memòria (figura 18).

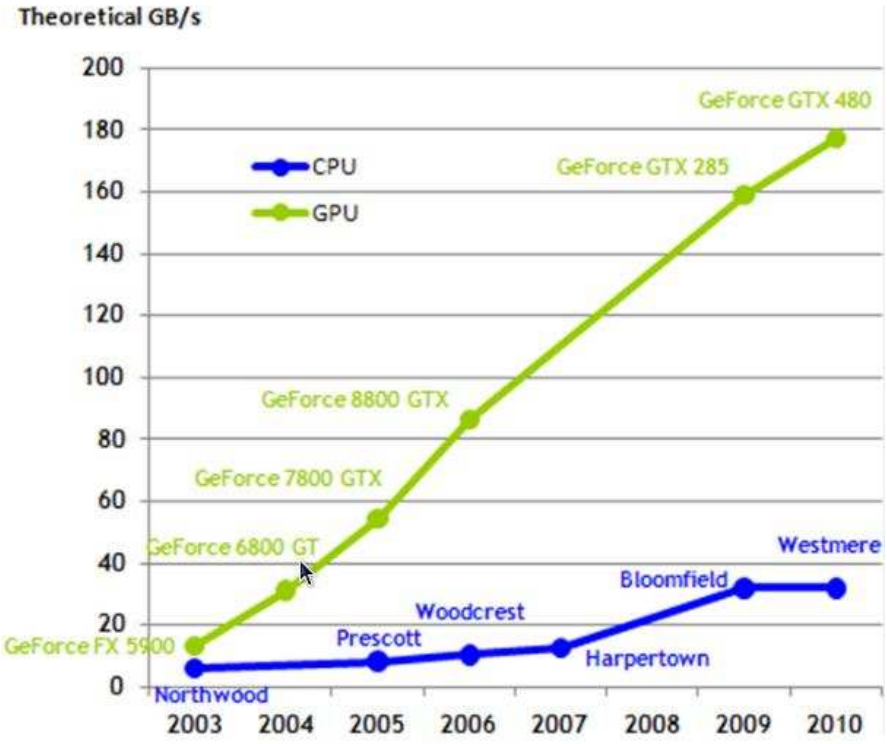

Figura 18. Ample de banda de les GPUs.

2. Models de programació GPGPU

2.1 OpenGL

OpenGL (Open Graphics Library) és una especificació estàndard que defineix un conjunt de funcions i el comportament exacte que aquestes han de tenir per poder escriure aplicacions que produeixin gràfics 2D o 3D, en altres paraules, una API (Application Program Interface) combinada amb un llenguatge propi basat en C99 (una especificació del C). Disposa d'una interfície multi plataforma que consisteix en més de 250 funcions diferents (C/C++) que es poden utilitzar per dibuixar escenes tridimensionals complexes a partir d'unes primitives geomètriques simples, tals com punts, línies o triangles. Es podria considerar que OpenGL té la comesa de transformar les dades d'una aplicació en alguna cosa que pugui ser visualitzat. Aquest procés rep el nom de rendering.

OpenGL té dos propòsits essencials:

- Proporcionar al programador una API única per intentar amagar la complexitat de les interfícies amb les diferents targetes gràfiques.

- Ocultar les diferents capacitats de les diverses plataformes hardware, requerint que totes les implementacions suportin la funcionalitat completa de l'OpenGL.

El funcionament bàsic de l'OpenGL (figura 20) consisteix en acceptar primitives com línies, punts o polígons i convertir-les en píxels. OpenGL és una API basada en procediments de baix nivell que requereix que el programador dicti els passos necessaris per renderitzar una escena. Aquest disseny de baix nivell requereix que els programadors coneguin en profunditat la pipeline gràfica (una màquina d'estats que té com a entrada les primitives, les processa i dóna com a sortida els píxels), però a canvi permet molta llibertat per poder implementar algoritmes gràfics nous (figura 19).

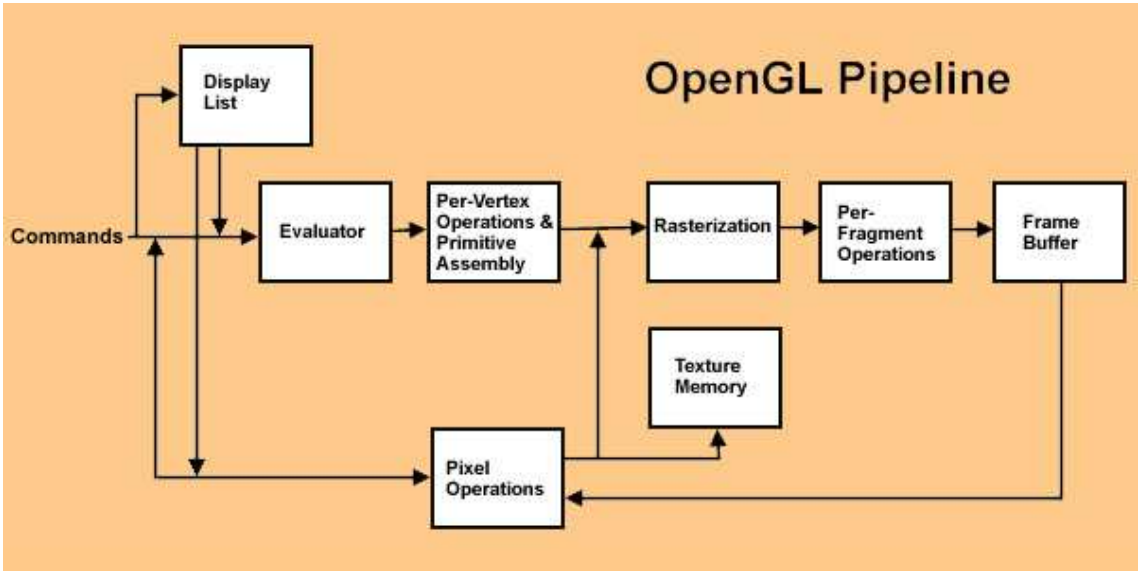

Figura 19. Esquema del pipeline de openGL

Després de tot, el conjunt de funcions estàndard de OpenGL s'ha quedat enrere comparat amb la gran evolució que hi ha hagut en el maquinari gràfic. Per aquest motiu alguns fabricants han optat per desenvolupar les seves pròpies extensions i així poder aprofitar al màxim les prestacions de les seves targetes. (figura 20)

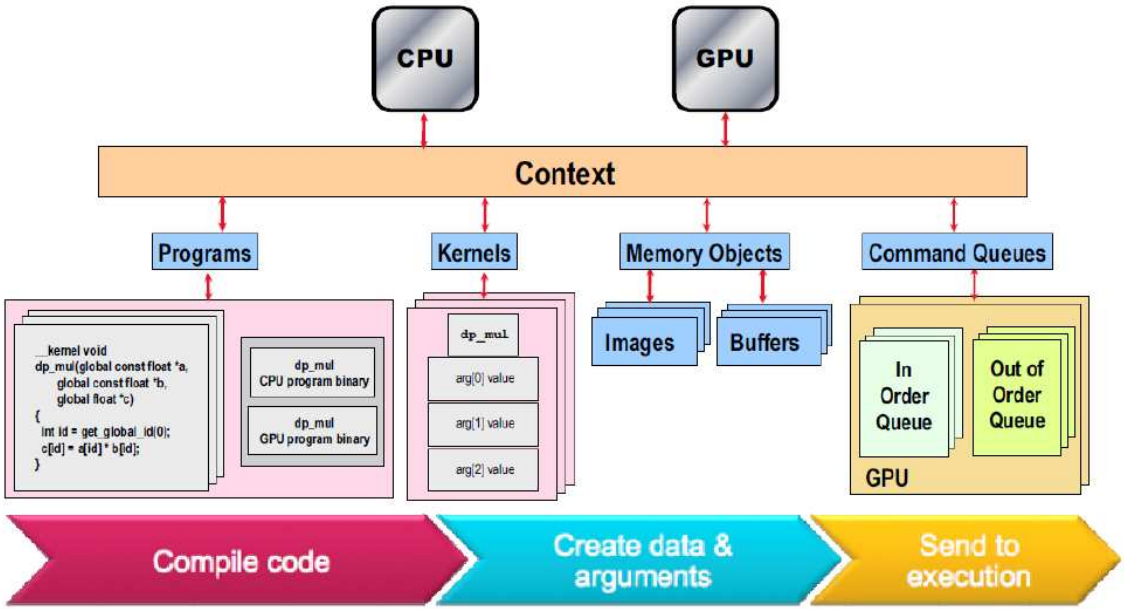

Figura 20. Esquema general d'OpenGL.

2.2 CUDA

Dins del context de les GPGPU apareix CUDA, amb l'objectiu d'aprofitar la potència dels dispositius de nVidia i poder processar en temps real operacions en una CPU no serien viables.

Compute Unified Device Architecture són les sigles d'aquesta solució que fa referència tant a un compilador com un seguit d'eines de desenvolupament creades per el fabricant de targetes gràfiques nVidia que permeten utilitzar una variant del llenguatge C per codificar algoritmes en les GPUs de nVidia.

CUDA intenta explotar els avantatges de les GPUs davant les CPUs de propòsit general utilitzant el paral·lelisme que ofereixen els seus múltiples nuclis, que permeten el llançament d'un altíssim nombre de fils simultanis. Per això mateix, si una aplicació està dissenyada per utilitzar nombrosos fils on cada un d'ells realitza una tasca independent, una GPU podrà oferir un millor rendiment.

CUDA també intenta aprofitar el gran paral·lelisme i el gran ample de banda de la memòria de las GPUs, en aplicacions que tenen un gran cost aritmètic en comptes de realitzar nombrosos accessos a memòria (possible cas de coll d'ampolla).

Abans de profunditzar amb el model de programació de CUDA, veurem les diferents capes que la componen (figura 21). Aquests programari està basat en:

- Una capa de controlador de maquinari.

- Una API i el seu runtime. Aquesta API és una extensió del llenguatge C, cosa que fa que la corba d'aprenentatge sigui mínima.

- Dos llibreries matemàtiques d'alt nivell com CUFFT i CUBLAS.*

*CUFFT és una llibreria per fer la transformada ràpida de Fourier (FFT) i CUBLAS és una llibreria que conté bona part de les funcionalitats de la

llibreria BLAS (Basic Linear Algebra Subprograms) utilitzada per realitzar operacions lineals bàsiques com multiplicació de vectors o matrius.

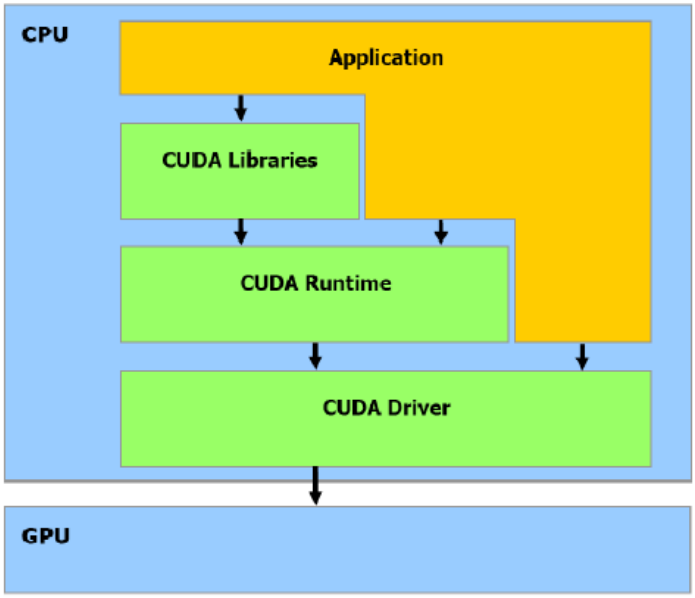

Figura 21. Jerarquia programari CUDA.

Aquest programari també permet un adreçament de caràcter general per la memòria DRAM, cosa que afavoreix a la flexibilitat de la programació, en el sentit que permet tant l'operació de repartir dades com la d'obtenir-ne (figura 22), dit d'una altre manera, permet llegir i escriure dades en qualsevol lloc de la DRAM (figura 23), igual que si de una CPU es tractés.

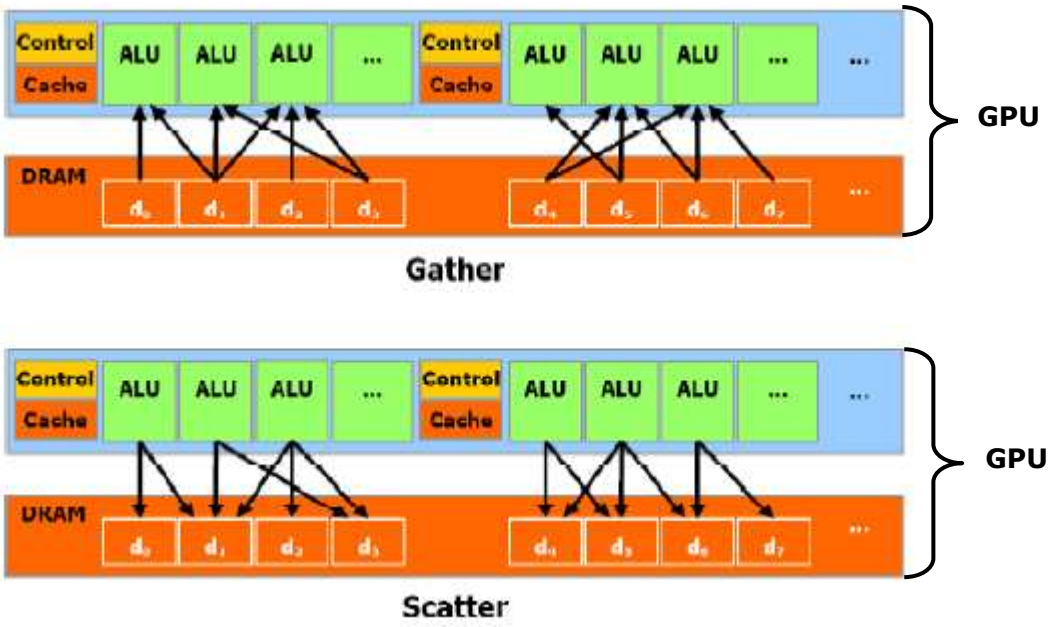

Figura 22. Operacions de memòria obtenir (Gather, literalment reunir) i repartir (scatter, literalment dispersió).

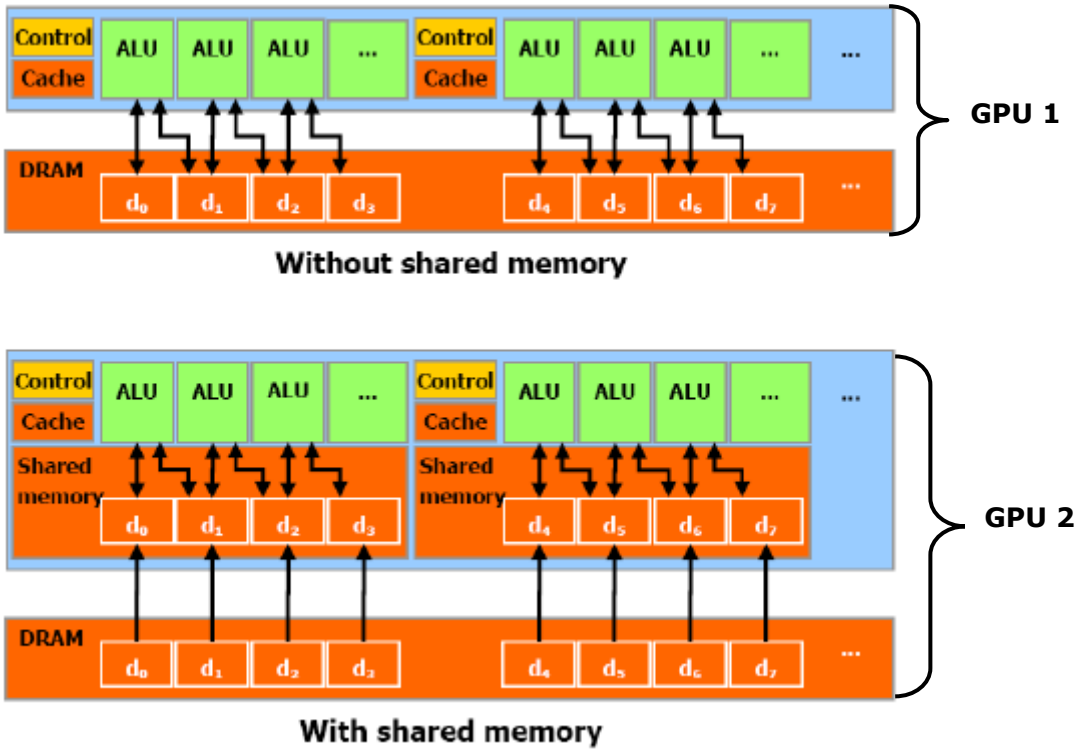

Figura 23. La memòria compartida proporciona les dades més a prop de les ALUs.

Les nVidia Quadro 5010M, 4000M, 3000M, 2000M i 1000M són alguns dels models que es beneficien de la memòria compartida.

Podríem doncs entendre CUDA com a un enllaç que ens permet fer un ús òptim de les característiques bàsiques de les GPUs comentades anteriorment a partir d'un llenguatge de programació d'alt nivell com és C.

2.2.1 Model de programació en CUDA

El primer que hem de diferenciar quan treballem amb CUDA és el *host* (nom que pertany al conjunt de la memòria principal i la CPU) i el *device* (GPU i memòria gràfica). La GPU només té accés a les dades que hi ha al dispositiu, per això CUDA proporciona les funcions ***cudaMalloc*** (utilitzada per reservar espai en el dispositiu) i ***CudaMemcpy*** (utilitzada per copiar informació entre el host i el device de forma bidireccional).

Per a crear codi amb el programari CUDA s'ha de tindre en compte que existeix una part del codi que s'executarà en la CPU i una altre part que s'executarà en la GPU. Normalment la part executada en la GPU seran funcions que s'anomenen ***kernel***, que inicien N threads que s'executen de forma paral·lela en N fils diferents. Un kernel es defineix fent servir la declaració **_global_** previ a les declaracions convencionals de C (com void, int...). En la crida a la funció es defineix l'estructura dels threads mitjançant la sintaxis **<<<Grid,Bloc>>>** on es pot configurar el tamany dels grids i els blocs.

L'arquitectura jeràrquica que segueix CUDA és bàsicament:

- En el primer nivell trobem el **grid** (reixeta). Un grid és un conjunt de blocs de fils. Disposem d'una variable anomenada gridDim que ens indica la dimensió del nostre grid en nombre de blocs. Exemple: gridDim.x

- En un segon nivell hi ha els **blocs**. Els blocs són un conjunt de fils i també disposem de les variables blockDim, que ens donen la dimensió del bloc en nombre de threads. Existeix una altre variable

que ens mostra el direccionament del block dins del grid anomenada blockIdx. Exemple: blockDim.y, blockIdx.y.

- En l'últim nivell hi trobem els **threads** que són la unitat mínima d'execució i que també es poden direccionar, amb la variable threadIdx, dins del bloc on es troben. Exemple: threadIdx.z

Per entendre més bé aquesta sintaxi cal saber que al executar un kernel es crea un grid (reixeta) de blocs, i que cada bloc conté diversos threads (fils) com podem veure en la figura 24.

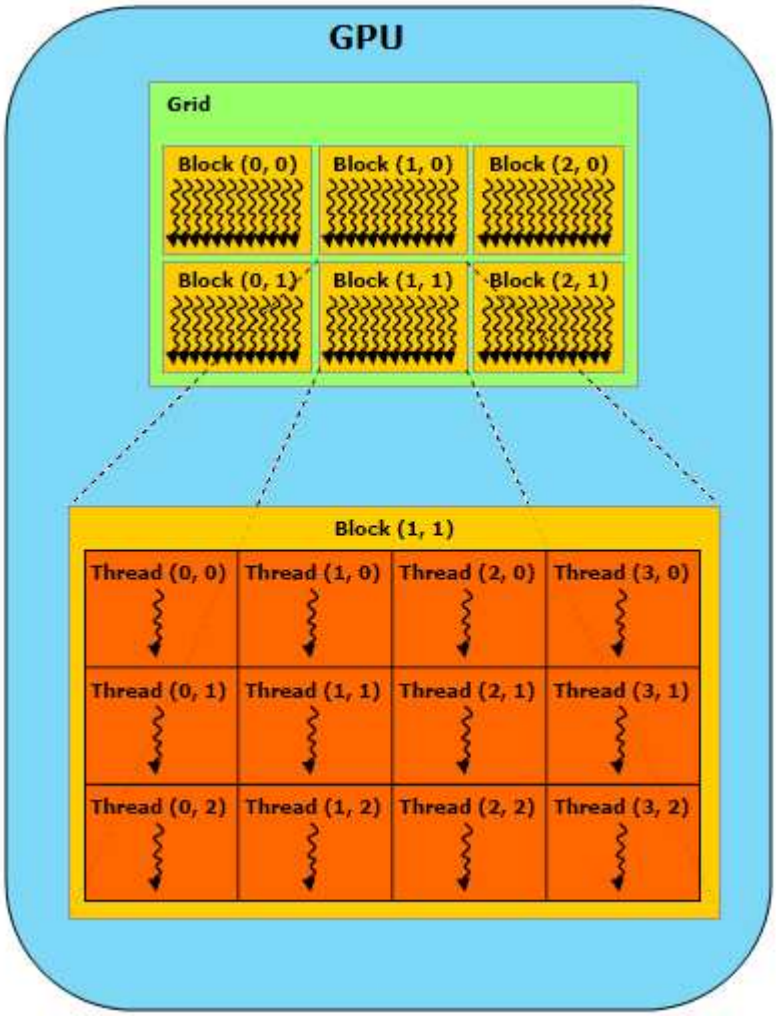

Figura 24. Estructura de reixeta (grid), blocs i fils (threads).Dins la GPU hi poden haver-hi diversos grids.

Cada bloc i cada fil tenen un únic ID i estan identificats dins de cada grid per les variables predefinides blockIdx (que pot ser 1D o 2D) i threadIdx (que pot ser 1D, 2D i 3D).

Al kernel cal passar-li una sèrie de paràmetres de tipus **dim3**, on aquest és una estructura (struct) de tres camps, on cada camp és assignat a una coordenada (x,y,z) i fan referència al nombre de blocs per cada grid i al nombre de threads per cada bloc.

El següent codi ajudarà a acabar d'il·lustrar aquesta informació. És un exemple de la suma de dos vectors A i B de mida N i acaba guardant el resultat en el vector C.

```
__global__ void vecAdd(float* A, float* B, float* C)
{
	int i = threadIdx.x;
	C[i] = A[i] + B[i];
}
int main()
{
	vecAdd<<<1, N>>>(A, B, C); // invocació del Kernel

}
```

També podem veure en el següent codi el mateix exercici però sumant matrius per poder veure com s'utilitzen les variables dimBlock.

```
__global__ void matAdd( float A[N][N], float B[N][N],float C[N][N])
{
	int i = blockIdx.x * blockDim.x + threadIdx.x;
	int j = blockIdx.y * blockDim.y + threadIdx.y;
	if (i < N && j < N)
	C[i][j] = A[i][j] + B[i][j];
}
int main()
{
	dim3 dimBlock(16, 16);
	dim3 dimGrid((N + dimBlock.x – 1) / dimBlock.x,(N + dimBlock.y – 1) / dimBlock.y);
	matAdd<<<dimGrid, dimBlock>>>(A, B, C); // invocació del Kernel
}
```

Cada fil pot accedir a les dades des de diferents espais de memòria durant la seva execució i cada un té la seva memòria privada local. Cada bloc de threads (fils) té una memòria compartida visible per a tots els fils d'aquest,

i tots els threads tenen accés a la mateixa memòria global (la memòria gràfica) com es pot observar en la figura 25.

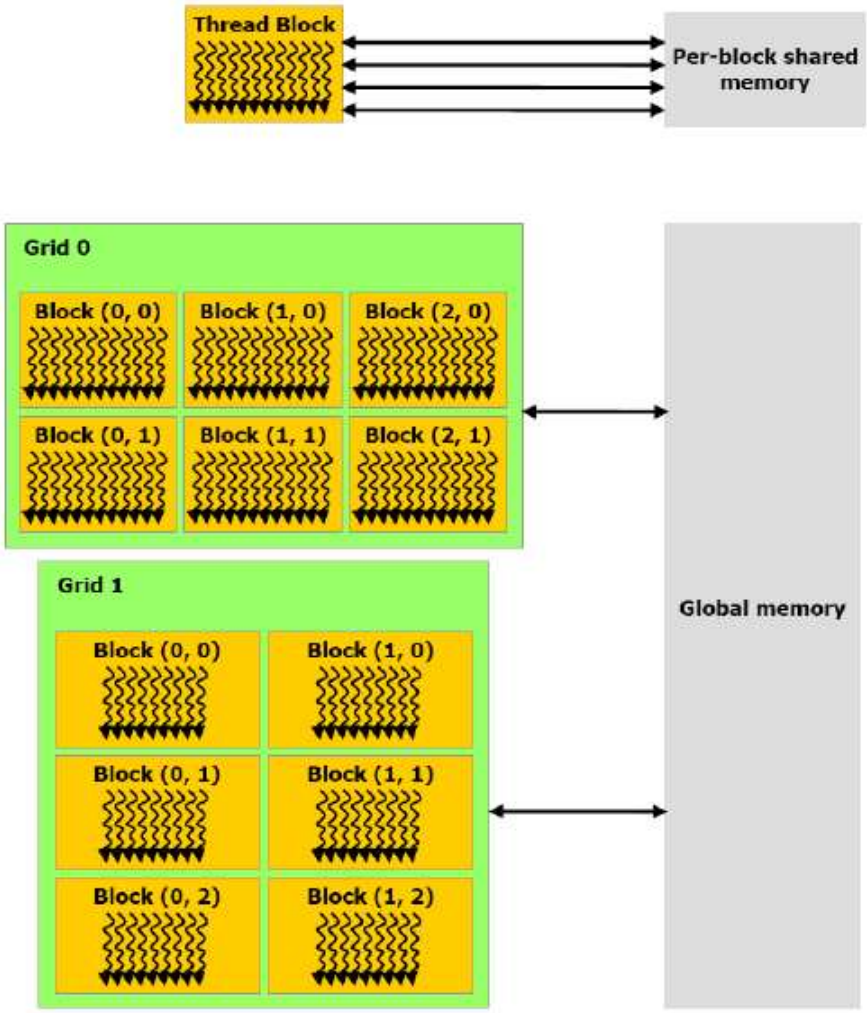

Figura 25. Jerarquia de la memòria.

Tal i com es pot veure en la figura 26, CUDA executa els fils sobre un dispositiu (device) físicament separat i que opera com a coprocessador d'ajuda al host, que estarà executant el programa en C, entenent així, que els kernels s'executaran en la GPU i la resta del programa en la CPU.

CUDA assumeix que tant el host com el device mantenen la seva pròpia DRAM, anomenada *host memory* i *device memory* respectivament.

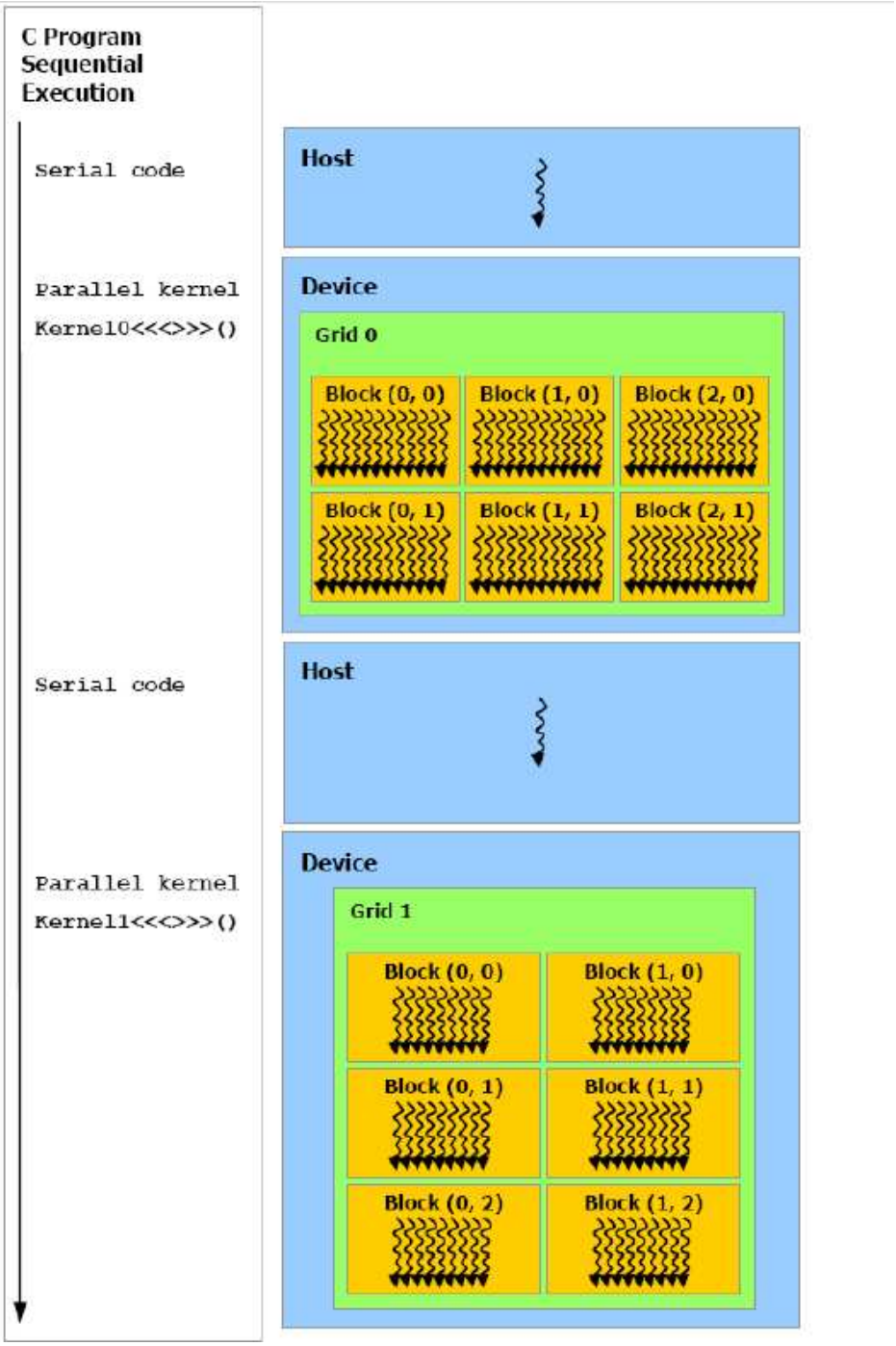

Figura 26. Exemple d'execució d'un programa. El codi s'executarà en sèrie en el device mentre que en el host s'executarà en paral·lel.

Cal tenir en compte que l'arxiu on hi ha el kernel ha de ser d'extensió .cu, ja que el codi és compilat per un compilador propi de CUDA (nvcc) que genera un codi concret per la GPU (codi amb extensió .cubin). Així doncs s'ha de vincular l'arxiu .cu en el codi principal en C per tal de que la crida al kernel tingui efecte (figura 27). Recordem que CUDA és una extensió de C i per tant permet definir les funcions (Kernels) en C.

```
kernel <<< dim3 numblocks, dim3 numthreads >>> (params);
```

Figura 27. Exemple de crida un kernel.

Un kernel però, mai podrà retornar cap tipus que no sigui void (no retorna res), no podrà ser una crida recursiva ni podrà tenir un nombre variable d'arguments. En canvi existeixen diferents tipus de kernels que es poden identificar fàcilment per la capçalera de la funció i que ens especifiquen en quin lloc s'executa la funció i des d'on l'han invocat.

- Global kernels (_global_): són funcions cridades des del host, però executades en el device.

- Device kernels (_device_): són funcions cridades des d'un mètode device, i executades en el mateix device. Aquest tipus mai pot ser cridat des del codi host.

- Host kernels (_host_): són funcions típiques de C, es criden i s'executen en el host.

També existeixen uns altres qualificadors que serveixen per especificar la localització en memòria en el dispositiu d'una variable.

- (_device_): la variable està en el dispositiu. Ha d'anar acompanyat d'algun altre qualificador que indiqui la mida que ocupa en la memòria. En cas contrari, la veriable està localitzada en la memòria global, té el temps de vida d'una aplicació i és accessible des de tots els fils del grid.

- (_constant_): la variable està localitzada en la memòria cau del dispositiu, té el temps de vida d'una aplicació i és accessible des de tots els fils del grid.

- (_shared_): la variable resideix en l'espai de memòria compartida d'un bloc de fils, té el temps de vida del bloc i només és accessible pel bloc.

En resum, el model d'execució de CUDA és el següent:

- Comença el programa i es fa la crida als kernels.

- Els kernels inicien els grids.

- Un bloc de threads s'executa en un multiprocessador.

- Diversos fils poden residir concurrentment en un multiprocessador.

- Els registres es reparteixen entre tots els fils residents.

- La memòria compartida es reparteix per a tots els blocs de fils residents.

2.3 ATI stream software

La plataforma de desenvolupament ATI stream proporciona a l'usuari final i als desenvolupadors de programari una suite d'eines per aprofitar la potència dels processadors stream de ATI. Aquest programari està format per:

- Un compilador per els arxius amb extensió dels dispositius ATI.

- Els drivers per els processadors stream del dispositiu (ATI Compute Abstraction Layer o CAL).

- Eines per el rendiment de profiling (serveix per analitzar kernels).

- Llibreries optimitzades per el rendiment dels processadors.

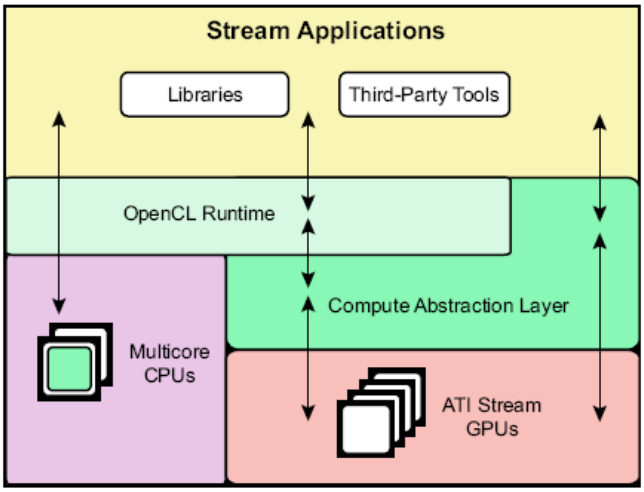

Figura 28. Sistema ATI stream software.

La última generació de processadors stream de ATI són programats amb un model de programació unificat de shader (ombrejat). Un llenguatge ombrejat (a partir d'ara shader) pot ser compilat independentment, i són

utilitzats bàsicament en operacions com la creació de efectes especials i per a la realització de transformacions.

Com que les GPU realitzen moltes operacions d'aquest tipus, els programadors utilitzen els llenguatges shader per a la creació d'instruccions que les realitzin, però per dur-ho a terme, aquests llenguatges necessiten enllaçar-se mitjançant una API.

El llenguatge que s'utilitza per a ATI és el GLSL (OpenGL Shading Language). És un llenguatge d'alt nivell basat en C i va ser creat per donar als desenvolupadors de programari un control més directe sobre el pipeline de OpenGL.

Amb el tipus de programació shader conegut amb el nom de stream computing, el arrays de dades d'entrada emmagatzemats en la memòria són assignats a un número de processadors determinats amb nuclis SIMD, cadascun dels quals executarà un kernel que proporcionarà un array de sortida amb les dades ja operades que es guardarà altre cop en la memòria. Cada instància del kernel quan s'executa dins d'un nucli del processador és el que s'anomena un thread (figura 29).

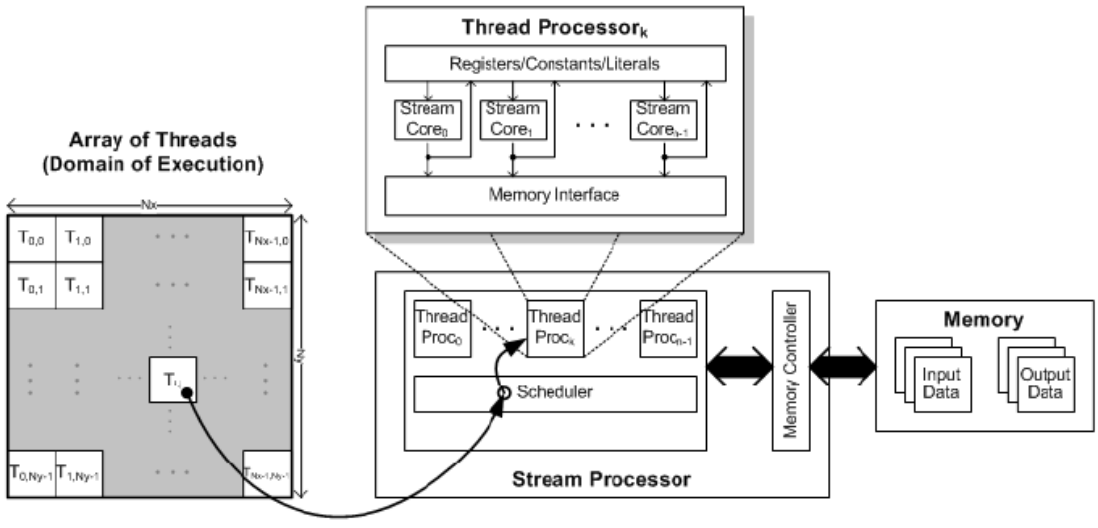

Figura 29. Es mostra com un array (T) entra en un processador stream i és assignat a un processador (thread Processor k) basat en SIMD (conté diversos stream cores).

En resum la API de CAL és ideal per als desenvolupadors sensibles al rendiment, ja que redueix al mínim la sobrecàrrega de programari i proporciona un control total sobre les característiques específiques del maquinari que podria no estar disponible amb eines de alt nivell.

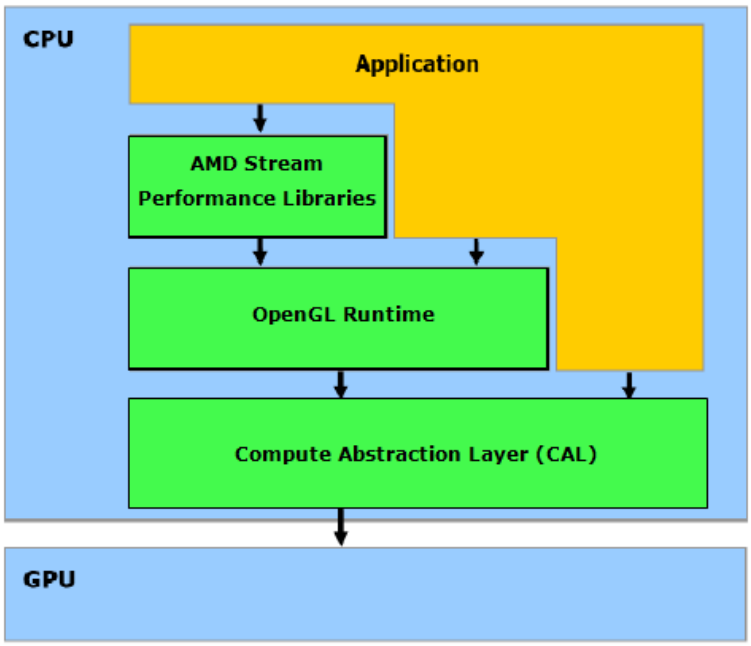

Figura 30. Jerarquia software ATI stream.

2.3.1 Model del sistema CAL (ATI Compute Abstraction Layer)

La CAL és una llibreria per un dispositiu que tingui una interfície de processadors stream ATI, que permet als desenvolupadors de programari interactuar amb els nuclis d'aquest tipus de processadors al nivell més baix per poder optimitzar el rendiment, mantenint al màxim la compatibilitat.

CAL incorpora:

- Generació de codi específic per a cada dispositiu.

- Gestió de diferents dispositius i recursos.

- Càrrega i execució de kernels.

- Suport multi dispositius.

- Interoperabilitat entre APIs gràfiques 3D.

Una aplicació CAL està composta de dues part ben diferenciades:

- Un programa que s'executa en el host del CPU (escrit en C o C++) que és l'aplicació principal.

- Un programa executant-se en el processador stream anomenat kernel.

La CPU executa la CAL en el host i controla els processadors stream mitjançant comandes enviades a través de l'API de CAL. El processador stream executa el kernel especificat per l'aplicació (figura 31).

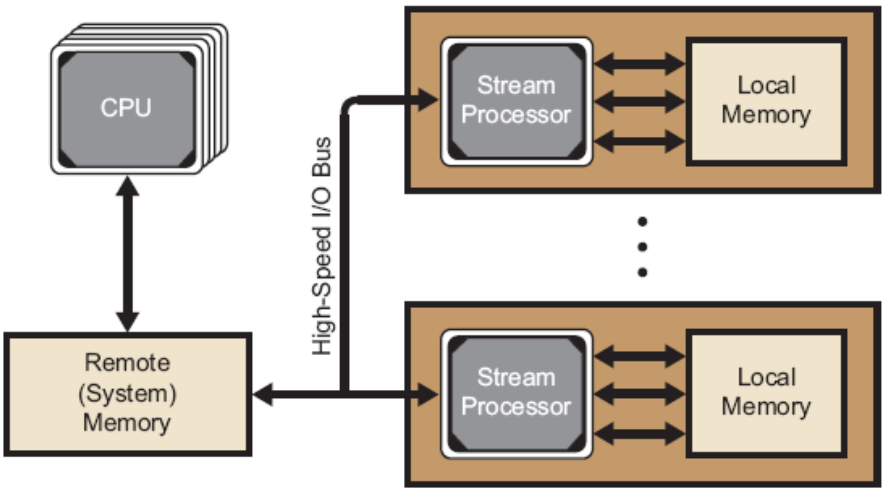

Figura 31. Arquitectura sistema CAL.

La API de CAL exposa els processadors stream com a una varietat de processadors de còmput del tipus SIMD. Aquests processadors executen el kernel prèviament carregat. El kernel llegeix les dades d'entrada d'un o més recursos d'entrada, realitza els càlculs i escriu el resultat en un o més recursos de sortida (figura 32). La computació paral·lela s'invoca mitjançant la creació d'una o més sortides i especificant el domini de l'execució per aquesta sortida. A més el dispositiu conté un planificador que distribueix la càrrega de treball dels processadors SIMD.

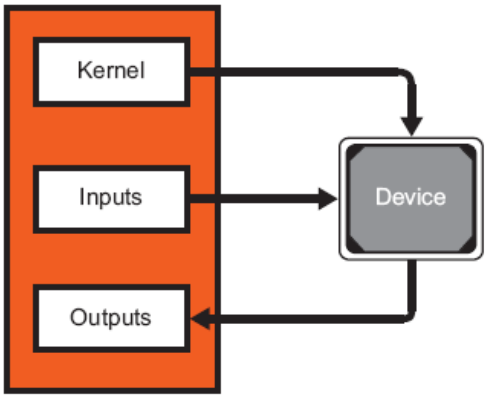

Figura 32. Esquema de l' execució del kernel.

Per entendre millor les execucions dels kernels veurem alguns exemples. El primer és una suma de dos matrius.

El codi de la CPU és:

```
void sum(float A[], float B[], float C[])
{
	for(int i=0; i<n; i++)
	{
		for(int j=0; j<m; j++)
		{
		float a0 = A[i][j];f

		loat b0 = B[i][j];

		C[i][j] = a0 + b0;
		}
	}
}
```

Podríem reescriure el codi per donar èmfasis a les operacions paral·leles de dades i executar una part del programa en la GPU (la part del kernel).

```
float sum_kernel(int y, int x, float M0[], float M1[])
{
	float a0 = M0[y][x];
	float b0 = M1[y][x];
	return a0 + b0;
}
void sum(float A[], float B[], float C[])
{
	for(int i=0; i<n; i++)
	{
		for(int j=0; j<m; j++)
		{
			C[i][j] = sum_kernel(i, j, A, B);
		}
	}
}
```

Encara que es paral·lelitzin càlculs, la CPU executa el codi igualment en sèrie calculant per exemple el C [0] [0] abans que el C [0] [1], però hi ha elements de C que poden ser calculats independentment els uns dels altres en qualsevol ordre. Podem veure-ho més gràficament en la figura 33.

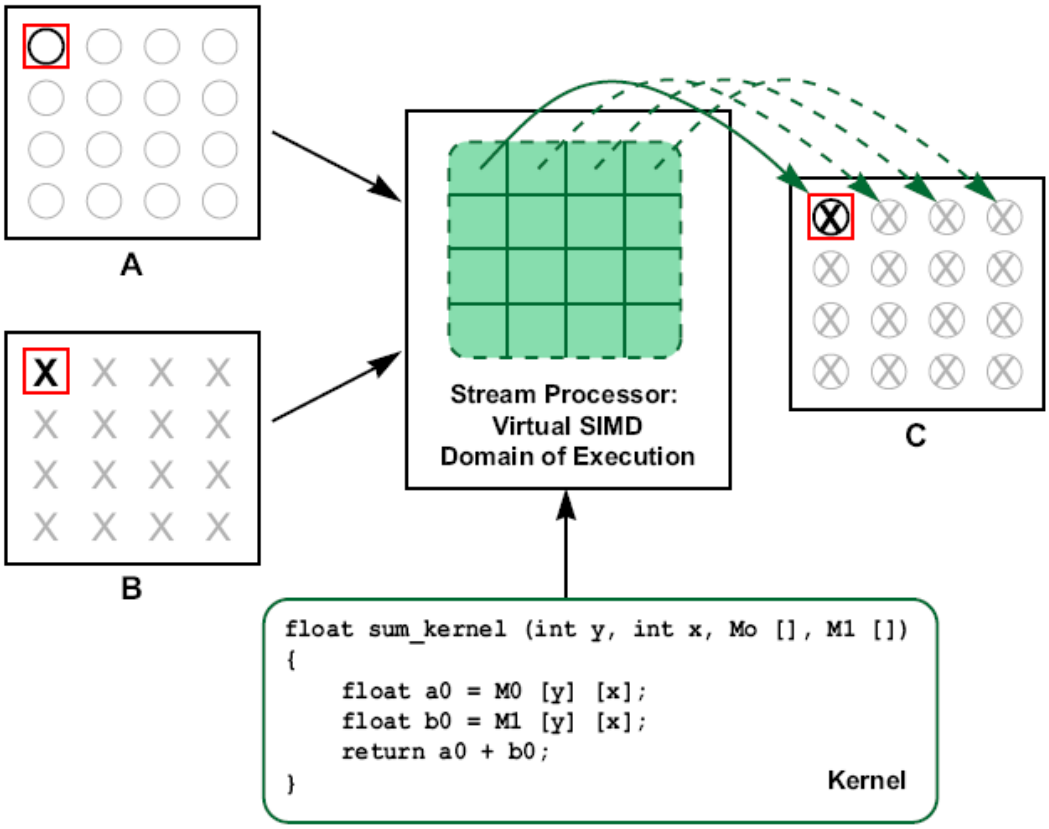

Figura 33. Execució del processador stream de l'exemple anterior.

Aquest altre exemple és el de la multiplicació de dues matrius (veure figura 34).

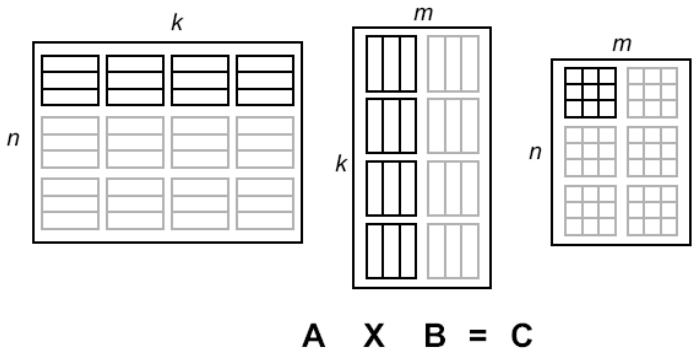

Figura 34. Recordatori multiplicació de dos matrius.

El codi de la CPU és el següent:

```
void matmult(float A[], float B[], float C[])
{
	for(int i=0; i<n; i++)
	{
		for(int j=0; j<m; j++)
		{
			float total = 0;
			for(int c=0; c<k; c++)
			total += A[i][c] * B[c][j];
			C[i][j] = total;
		}
	}
}
```

El codi que pot ser executat en el kernel és el indicat en negreta. Així doncs el codi reescrit quedaria de la següent manera:

```
float matmult_kernel(int y, int x, int k, float M0[], float M1[])
{
	float total = 0;
	for(int c=0; c<k; c++)
	{
		total += M0[y][c] * M1[c][x];
	}
	return total;
}

void matmult(float A[], float B[], float C[])
{
	for(int i=0; i<n; i++)
	{
		for(int j=0; j<m; j++)
		{
			launch_thread{C[i][j] = matmult_kernel(i, j, k, A, B);}
		}
	}
	sync_threads{}
}
```

Els kernels per aquest programari porten la terminació ".br" com a extensió d'arxiu. Aquests kernels són compilats amb un compilador específic anomenat Brook+ i tradueix els arxius .br a codi per els dispositius amb processadors stream (figura 35).

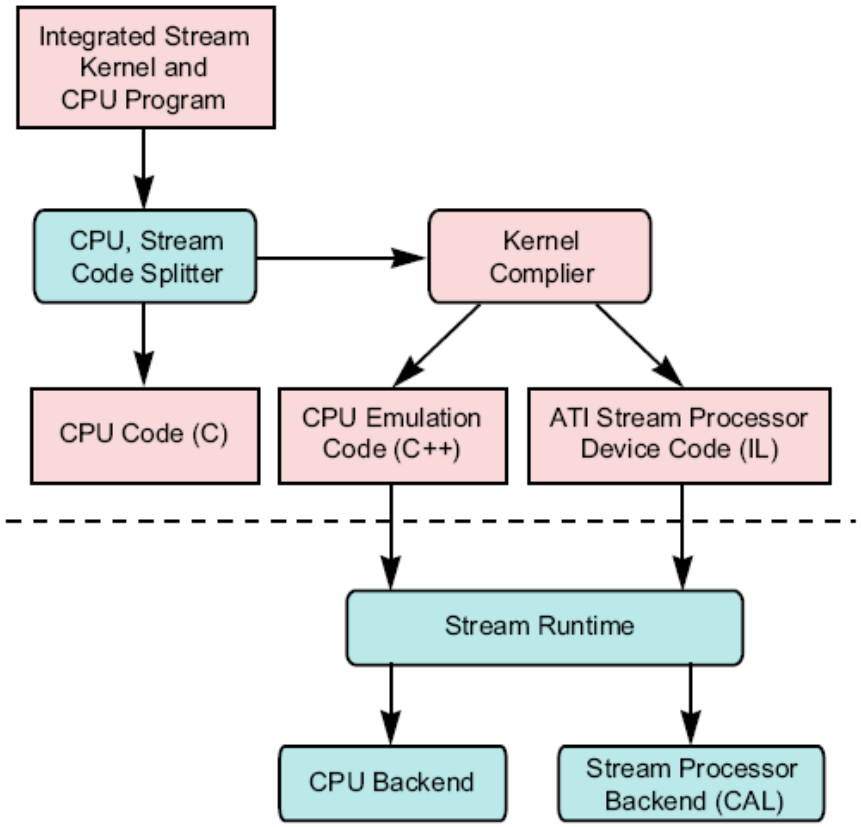

Figura 35. Diagrama de seqüència que seguiria un programa per ser executat amb processadors stream.

3. Informació programari GPGPU

Un cop situats dins de les diferents opcions que hi ha per utilitzar les GPUs com un processador de propòsit general, veurem com aquesta tecnologia pot influir en la programació de les aplicacions que un usuari pot arribar a fer servir en la seva vida quotidiana.

Moltes empreses de desenvolupament de programari han apostat per introduir la potència de càlcul que aporten les GPUs en les seves aplicacions per poder beneficiar-se de les millores de rendiment. Però realment funcionen millor aquests programes? Que aporta realitzar els càlculs en les GPUs?

Bé doncs, per poder donar solucions a aquestes preguntes analitzarem diversos programes i quin és exactament el benefici que dóna aquesta tecnologia.

3.1 Vreveal

Vreveal és un programari que permet fer algunes modificacions i correccions a arxius de vídeo digitals. Aquesta aplicació està dissenyada per estabilitzar, netejar i millorar les imatges dels vídeos, això sí, els arxius que ja tinguin una bona qualitat no podran ser millorats. Vreveal serveix doncs per a vídeo aficionats, ja que aquests usuaris no disposen d'equips professionals per filmar i per tant la qualitats dels vídeos podrà ser gairebé sempre millorat en algun sentit.

Vreveal pot millorar la imatge comparant cuidadosament cada fotograma amb els dos fotogrames adjacents (figura 36). Si el marc anterior o posterior és millor que el marc que s'està examinant, l'aplicació fa les modificacions necessàries al marc original permetent així il·luminar zones molt fosques, combatre amb èxit la difamació i les fotos mogudes, així com ajustar la nitidesa i el contrast.

Aquesta aplicació pot ser executada en qualsevol ordinador, no obstant, els usuaris que disposin d'una targeta gràfica que suporti la tecnologia CUDA, obtindran una gran millora de rendiment. Durant les proves, els desenvolupadors de l'aplicació van remarcar que l' utilització de la GPU GeForce GTX280 millorava el rendiment del producte 5 vegades en comparació de un processador Intel Core Duo E6600.

Figura 36. Exemple Vreveal.

3.2 Badaboom

Badaboom és una aplicació de conversió o transcodificació de formats multimèdia que agilitza la codificació dels vídeos per poder ser reproduïts en un dispositiu portàtil. El procés de conversió pot ser accelerat enormement si el programa és utilitzat en un ordinador on hi hagi una targeta gràfica nVidia compatible amb CUDA. La majoria de programes similars fan la conversió en la CPU, quedant així massa atrafegada per a qualsevol altre procés que requereixi còmput, però Badaboom al executar la conversió en la GPU deixa la CPU lliure per poder seguir utilitzant el PC mentre es realitza la transcodificació.

Com que la principal característica de l'aplicació és deixar la CPU lliure mentre es fa la conversió, el programa només pot funcionar si el PC disposa d'una targeta nVidia compatible amb CUDA (figura 37).

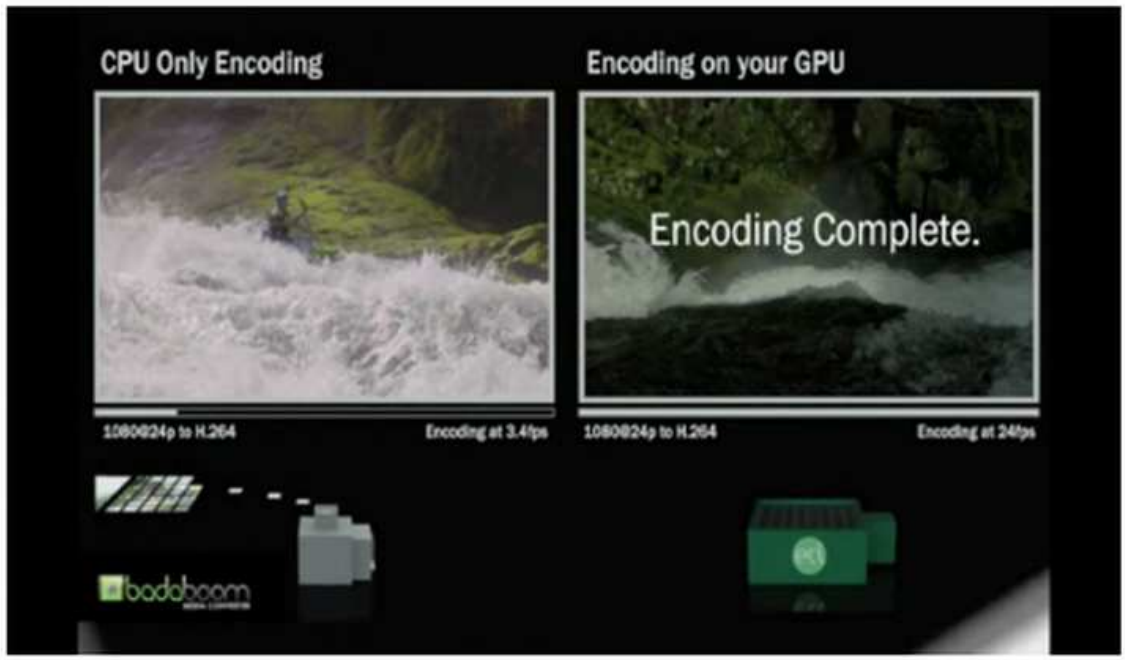

Figura 37. Exemple de codificació amb Badaboom. (Mentre que amb Badaboom ja s'hauria completat la codificació en la GPU, la CPU continuaria codificant el vídeo).

També es beneficien d'aquesta tecnologia programes amb molt més renom com són:

- Adobe Creative Suite CS4, on l'aplicació estrella es Photoshop. CUDA permet realitzar operacions de zoom, panoràmic d'imatge, composicions d'imatges 2D i 3D, moviment de imatges 3D i conversió de colors a temps real (figura 38).

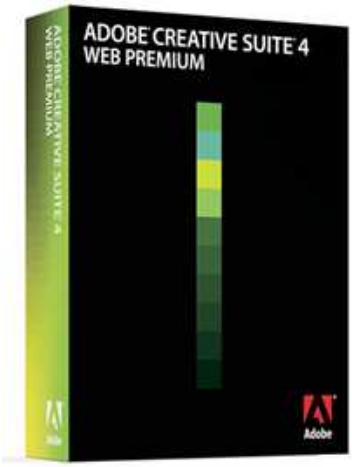

Figura 38. Aplicació Adobe Creative Suite.

- Suite Cyberlink PowerDirector 7, on CUDA i ATI stream proporcionen una millor velocitat de previsualització i de renderitzat en l'aplicació powerDVD, acceleren el reconeixement facial en el programa MediaShow y PowerProducer es beneficia d'uns processos de producció més ràpids (figura 39).

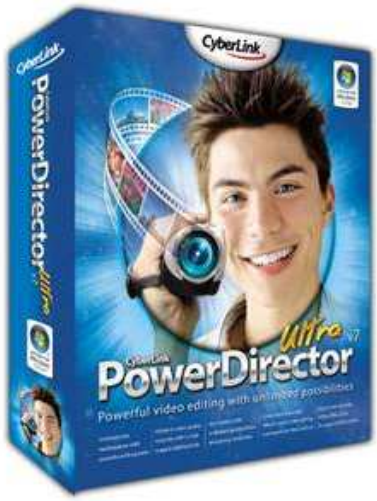

Figura 39. Aplicació Cyberlink PowerDirector.

- Nero Move It, redueix el temps de conversió de contingut multimèdia de vídeo fins a 5 vegades gràcies a CUDA (figura 40).

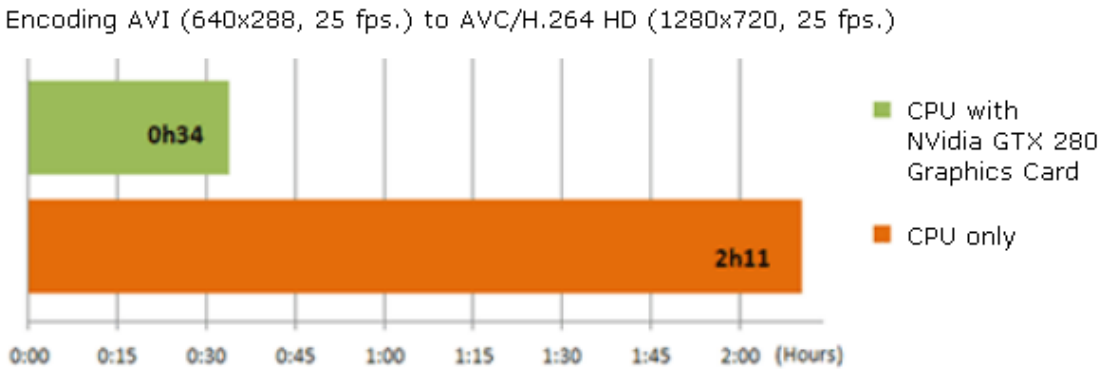

Figura 40. Comparació temps de conversió de un arxiu de vídeo.

Figura 41. Aplicació Nero Move It.

- ArcSoft Total Media Theatre té com a principal característica la tecnologia d'ampliació de l'escala de resolució, reproduint així arxius multimèdia de qualitat estàndard a una resolució pròxima a la alta definició utilitzant ATI stream (figura 42).

Figura 42. Aplicació Total Media Theatre.

- TMPGEnc 4.0 XPress que és un programari de codificació de vídeo multi format que ha demostrat que fent servir CUDA aconsegueix un augment de rendiment de les seves tasques de fins a un 446%, o expressat d'una altre manera, utilitzant CUDA tarda menys de una quarta part (figura 43).

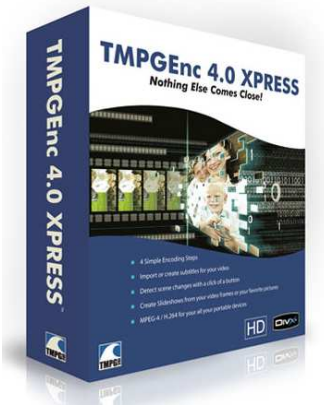

Figura 43. Aplicació TMPGEnc 4.0 Express.

- Movavi és un altre eina molt completa de conversió d'arxius de vídeo que compta amb una llarga llista de diferents tipus d'arxius compatibles. A part d'aquesta característica principal, l'aplicació permet unir arxius de vídeo, extreure l'àudio d'un clip, copiar el contingut d'un DVD al disc dur o capturar instantànies d'un fitxer. Utilitza la tecnologia CUDA per a realitzar les conversions arribant a augmentar la velocitat fins a 8 vegades segons el model de targeta gràfica nVidia utilitzat.(figura 44)

Figura 44. Aplicació Movavi

I també trobem programes més peculiars com SETI@home, una aplicació lliure que consisteix en el processament de senyals de radio per buscar una proba de que existeix intel·ligència extraterrestre. Qualsevol usuari es pot descarregar aquesta aplicació i entrarà a formar part de la xarxa de BOINC (Berkeley Open Infrastructure for Network Computing) en la qual SETI@home utilitzarà part de la potència del PC, espai del disc dur i ample de banda format així el que es podria anomenar un supercomputador, que amb els usuaris que hi ha actualment, té un rendiment mig de més de 2 petaFLOPS.

Figura 45. Logo Seti@home.

Després d'analitzar tots els avantatges que aporten les GPU al programari, és el moment de veure quins requisits són necessaris per continuar amb el projecte.

4. Requisits

4.1 Funcionals

Un requisit funcional és una característica requerida de la qual se sap que serà satisfeta mitjançant una addició d'un subsistema o un bloc de codi en el programari, en altres paraules, un requisit que denota la funcionalitat del sistema.

- Avaluar el programari. El principal requisit funcional serà veure les diferències entre el mateix programari executat en la CPU i executat en la GPU.

4.2 No funcionals

Anomenarem requisit no funcional a totes les exigències de qualitats que s'imposen al projecte, exigències com fer servir un cert llenguatge de

programació o plataforma tecnològica, o dit en altres paraules és un requisit que no podem satisfer afegint codi, per tant és una restricció.

Al tractar-se d'un projecte bastant específic, hi ha bastants restriccions.

- PC. El primer que necessitem és un PC per a poder testejar diferent programari i poder avaluar el rendiment d'aquest.

- GPU. El PC ha de disposar d'una targeta gràfica per poder fer les proves en la GPU, sinó no tindria sentit continuar el projecte.

- GPU compatible. La targeta gràfica ha de ser compatible amb la tecnologia d'utilitzar la GPU per a GPGPU, ja sigui utilitzant CUDA de nVidia o ATI stream d'ATI.

- Programari GPU de nVidia o ATI. Serà necessari disposar del programari proporcionat per el fabricant de la GPU per poder executar les proves en la targeta gràfica.

- Programari per programar. També es necessitarà un programa per poder fer modificacions en el codi font dels programes.

- Programari per avaluar. Tots els requisits anteriors no servirien de res si no hi hagués alguna aplicació a avaluar.

CAPÍTOL IV: DISSENY

Aquesta part del projecte està dedicada a la realització de les proves d'avaluació. Primerament es descriurà el procediment que es durà a terme fent un anàlisis dels requeriments funcionals i no funcionals. Més endavant a l'apartat de proves, es realitzaran tests amb diferents programes per veure les diferencies que existeixen entre utilitzar CUDA o no, els beneficis que aporta i una petita conclusió de cada apartat.

1. Requisits funcionals

El requisit funcional principal és de quina manera es podrà realitzar la connexió entre el codi en java del msms i la GPU, sabent que la interfície que hem d'utilitzar és CUDA , i aquesta no té suport per a java.

Per solucionar aquesta tasca, necessitem fer un binding de la biblioteca CUDA. Un binding és una lligadura o una referència a un altre símbol més complicat i que ha de ser usat freqüentment. En aquest cas el binding que hem de dur a terme serà una adaptació d'una biblioteca (CUDA, escrit en C) per a que es pugui fer servir en un llenguatge de programació diferent al que ha estat escrit (msms, escrit en java).

Existeixen eines que solucionen aquests problemes com JCUDA, JaCuda o Jacuzzi, però nosaltres ens decantarem per JCUDA, ja que és la més emprada per els desenvolupadors.

JCUDA és una llibreria de la biblioteca CUDA per poder ser usada en java. Per exemple té la possibilitat de comunicar-se amb les llibreries CUBLAS i CUFFT entre d'altres.

Per poder utilitzar JCUDA només ens hem de descarregar els fitxers des de la seva pàgina web (http://www.jcuda.org/) i incloure'ls en el projecte. En la web podem trobar ja fetes les implementacions d'algunes de les llibreries més utilitzades de CUDA i bastant exemples per poder entendre el funcionament d'aquestes.

Com que una de les coses que busquem és transferir les operacions de molta càrrega de còmput a la GPU, no podem utilitzar aquestes llibreries ja

implementades, sinó que hem de crear un kernel propi que executi aquesta càrrega en la GPU. Així doncs hem de localitzar en quines operacions el programa requereix de més temps i crear un kernel que pugui ser enviat mitjançant CUDA a la GPU.

Per l'operació que busquem el sin(cos(tang(x))) on x és el valor de cada posició de un vector tindríem l'estructura següent:

```
1. private void Solució(float input[], float output[])
2. {
3.     for (int i=0; i<input.length; i++)
4.     {
5.         output[i] = (float)Math.sin(Math.cos(Math.tan(input[i])));
6.     }
7. }
```

Aquesta part del codi s'haurà de reescriure en un kernel com aquest, que estarà en un fitxer apart amb extensió .cu.

```
__global__ void CalculsSolucio(float* input, float *output, int n)
{
    int i = threadIdx.x;
    output[i] = sin(cos(tan(input[i])));
}
```

I on hi havia el codi inicial, haurem de realitzar la crida al kernel que hem escrit, i encara que no estigui escrit en java, gràcies a JCUDA la crida podrà ser realitzada.

```
1. private void CalculsSolucioCUDA(float input[], float output[])
2. {
3.     copyMemoryToDevice();
4.     computeSolutionKernelLauncher.call(deviceInput, deviceOutput, n);
5.     copyMemoryToHost();
6. }
```

2. Requisits no funcionals

El primer pas és avaluar els requisits no funcionals per començar a dissenyar un pla d'acció.

1- PC. Disposem d'un PC per realitzar tests al programari. El PC està format per una CPU Intel Core 2 Quad 6600 a 2.4GHz i 2GB de RAM.
2- GPU .El PC disposa d'una targeta gràfica.
3- GPU compatible. Podem saber si la nostre gràfica és compatible mirant si el model del que disposem està en la llista que hi ha en l' enllaç següent (http://www.nvidia.es/object/cuda_gpus_es.html) si és nVidia, o fixant-nos si el nostre model de ATI és superior al model HD 4000. Casualment es disposa d'una nVidia GeForce 8400GS que apareix a la llista i és totalment compatible amb CUDA, així doncs queda automàticament descartat utilitzar algun element de la casa ATI en aquest projecte per un tema purament de cost.
   La GeForce 8400GS disposa de 512 MB de memòria gràfica dedicada i la freqüència del nucli és de 450 MHz.
4- Programari GPU. Programari necessari pel funcionament de CUDA disponible gratuïtament a la web de nVidia.
5- Programari per programar recomanat per nVidia. Serà necessària l'aplicació Microsoft Visual Studio 2008 per a la compilació de les proves que es realitzin amb CUDA, i l'aplicació Eclipse, un entorn integrat de desenvolupament de codi obert per desenvolupar projectes.
6- Programari per avaluar. Es vol avaluar el rendiment i el benefici que aporta CUDA en el mateix programa i intentar aplicar aquesta tecnologia a algun programa que no ho incorpora. Realitzarem la tasca d'avaluació amb algun dels programes esmentats en l'anàlisi, i intentarem aportar aquesta tecnologia al msms.

Com que es compleixen tots els requisits no funcionals, podem continuar endavant amb el projecte.

3. Proves

Per poder realitzar les proves, és necessari tenir instal·lat una sèrie de programari per tal de que CUDA funcioni correctament.

La primera aplicació a instal·lar ha de ser Microsoft Visual Studio 2008. Aquesta aplicació serà necessària per que un cop creat el fitxer que conté el

codi que no serà executat en la GPU, es compili amb el compilador que porta incorporat el Visual i es creï un fitxer apte per ser executat en la CPU. Al ser estudiant de la UAB podem accedir al repositori de Microsoft i descarregar-nos l'aplicació amb la llicència d'estudiant.

El segon pas és instal·lar l'últim driver de nVidia. El driver és el que permet accedir a totes les característiques de la targeta nVidia, inclòs el suport de CUDA.

El tercer pas és instal·lar el CUDA toolkit, que és un conjunt d'eines de desenvolupament, llibreries i documentació que es necessiten per crear aplicacions per l'arquitectura CUDA i és el que més tard ens permetrà compilar els programes CUDA (recordem: NomDelFitxer.cu). Aquest toolkit inclou:

- Compilador CUDA C/C++.
- GPU Debugging i eines profiling.
- Llibreries matemàtiques accelerades per GPU.
- Primitives accelerades de rendiment per GPU.

El quart pas és totalment opcional, però molt recomanable. Es poden descarregar una sèrie d'exemples anomenats SDK code examples que són útils per veure des de codis senzills fins a més complexos, i així començar a introduir-se en CUDA.

El cinquè pas és també opcional i requereix haver descarregat el paquet SDK code examples, però també és força recomanable. Es tracta de comprovar que tot el procés d'instal·lació ha anat correctament compilant un exemple dels que hi ha al paquet SDK amb el Visual Studio. Si la compilació no retorna cap error, la instal·lació és correcte.

Si s'ha seguit aquest ordre no ha d'haver sorgit cap problema a l'hora de compilar un exemple del SDK amb el Visual Studio. Si estava instal·lat el toolkit de CUDA abans que el Visual, serà necessari afegir l'arxiu "cuda.rules" situat en la carpeta on s'ha instal·lat CUDA al path del projecte

que es vol compilar. Un cop realitzats els passos anteriors, ja ens podem disposar a avaluar diversos programes.

3.1 Movavi

Per avaluar l'aplicació Movavi, s'han realitzar diverses proves de conversió de format a partir de 5 arxius de vídeo de diferents mides.

Un cop realitzada la instal·lació, Movavi automàticament detecta si hi ha alguna GPU compatible amb CUDA i l'habilita per defecte (figura 46).

Figura 46. Missatge on Movavi mostra l'habilitació de CUDA (només apareix el primer cop que s'inicia l'aplicació).

Per poder realitzar les proves comparant els resultats de l'utilització de CUDA, hem hagut de deshabilitar la opció d'acceleració de la GPU, accedint-hi des de la pestanya "editar" del menú principal i entrant a "preferencias" (figura 47).

Figura 47. Imatge menú superior aplicació Movavi.

En la finestra de "preferencias" hem desmarcat la opció "habilitar acceleración de GPU nVidia" (figures 48 i 49).

Figura 49. Finestra de preferències on la GPU està habilitada

Figura 48. Finestra de preferències on la GPU està deshabilitat

Un cop establerta la forma per habilitar i deshabilitar CUDA, hem importat un per un els vídeos per avaluar-los individualment mitjançant el botó d'afegir arxius (figura 50).

Figura 50. Botó afegir arxiu.

Per seleccionar el format en que volem que transformi l'arxiu, ens hem de dirigir a la part inferior del programa, on apareix un selector per poder escollir quin format volem. El formats aptes per a la utilització de CUDA estan especificats (figures 51 i 52).

Figura 52. Selector de formats amb la opció de CUDA habilitada.

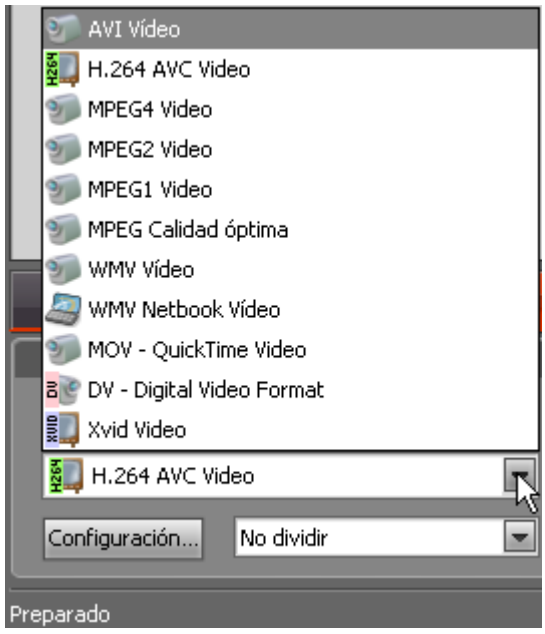

Figura 51. Selector de formats amb la opció de CUDA deshabilitada.

Un cop seleccionat el format, ja podem iniciar la conversió amb el botó "Convertir" (figura 53).

Figura 53. Botó convertir.

A partir de totes les conversions realitzades hem elaborat la següent taula per observar els resultats.

Mida de l'arxiu de vídeo	Conversió GPU CUDA	Conversió CPU	% de millora
4,04 GB	32 min	46 min	43,7
2,34 GB	20 min	33 min	65,0

1,47 GB	17 min	30min	76,5
533 Mb	5 min	7 min	40,0
140 Mb	2 min	2 min	0,0

Com mostren les dades obtingudes, executar el programa amb la tecnologia CUDA habilitada, redueix de forma considerable el temps de còmput. Aquest canvi de temps de càlcul es pot apreciar en els arxius més grans, degut a que per mides petites, es consumeix més temps en transferir el arxiu fins la targeta gràfica que fer el càlcul en si.

Una altre dada important és que al executar el programa en la CPU, l'ordinador queda bastant saturat, deixant-nos bastant limitades les accions que podem fer si no volem que augmenti el temps. En canvi si el programa és executat en la GPU, el ordinador està més alliberat i es pot utilitzar d'una manera més àgil.

3.2 vReveal

Per avaluar vReveal, seguirem els mateixos passos que hem utilitzat en el cas anterior. Per defecte després de la instal·lació també queda habilitada la utilització de CUDA, així doncs per habilitar i deshabilitar la GPU haurem de dirigir-nos a la pestanya "Herramientas" del menú, i des d'alla mateix podem seleccionar-la o desseleccionar-la (figura 54).

Figura 54. Figura 54. Menú superior vReveal.

Per afegir els vídeos cal anar al botó "importar" i en aquest cas si que ja els podem importar tots de cop, ja que la millora només es realitzarà als arxius seleccionats. Per seleccionar les millores, fem click amb el botó dret del ratolí i premem "mejorar" (figura 55).

Figura 55. Vista de les opcions que ofereix vReveal per als arxius.

Ens situem al menú de millores bàsiques per seleccionar quines millores volem efectuar. Per realitzar aquest test s'han seleccionat totes les millores possibles.

Figura 56. Millores que ofereix vReveal per als arxius importats.

Per últim, un cop seleccionades les millores, només cal especificar en quin lloc es guardarà el nou arxiu de vídeo millorat (figura 57).

Figura 57.Opcions per guardar el nou arxiu.

Ja només caldrà esperar a que es realitzi tot el procés (figures 58 i 59) .

Figura 58. Inici del procés de millora d'un arxiu amb el suport GPU activat.

Figura 59. Fi del procés de millora d'un arxiu amb el suport GPU desactivat.

Com en el cas anterior, hem elaborat una taula utilitzant els mateixos arxius de vídeo per avaluar els resultats obtinguts.

Mida de l'arxiu de vídeo	GPU CUDA	CPU	% de millora
4.04 GB	2h 47 min	6h 24min	129,9
2,34 GB	1h 32 min	3h 56 min	156,5
1,47 GB	1h 15 min	3h 5 min	146,7
533 mb	19 min	52 min	173,7
140 mb	5 min	14 min	180,0

Com en el cas anterior, en la taula observem que CUDA influeix positivament en el temps de còmput. També hem comprovat que mentre s'executa l'aplicació amb CUDA habilitada, es pot seguir utilitzant el PC a un nivell raonable, mentre que si no s'utilitza la GPU, cada moviment que es realitzi en el PC, fa enrederir el temps de finalització de la tasca.

A diferència de movavi, el canvi en l'utilització de la GPU o no es nota des del arxiu més petit fins al més gran, comprovant que en la taula de movavi el arxiu de 140 MB casi no aportava benefici, i en canvi en aquesta de vReveal existeix una diferència notable.

3.3 msms

Un cop vistes les millores que presenten els programes de vídeo que utilitzen CUDA, procedim a intentar aplicar aquesta tecnologia a un programa totalment diferent. Per continuar, és necessari saber què fa exactament el programa per veure què podem executar en la GPU i què no.

El criteri que seguirem per avaluar el msms serà, en línies molt generals, el següent:

- Executarem el programa normalment amb diferents entrades de dades i mesurarem el temps que tarda en cada una de elles.

- S'estudiarà el comportament del programa, diferenciant les parts i intentant entendre al màxim el codi font.

- Es faran les modificacions necessàries per poder executar la part de càlcul a la GPU.

- Es compilarà de nou el programa per que pugui ser executat part del codi en la GPU.

- Executarem altra vegada les mateixes entrades de dades que en el primer apartat mesurant el temps.

- Analitzarem els resultats.

El primer que es necessita per començar és el codi font del programa msms que el podem trobar de forma totalment gratuïta a la seva web (http://www.mabs.at/ewing/msms/index.shtml). Necessitem també un programa per poder veure, executar, modificar i fer diferents proves a aquest codi, i utilitzarem per tot això l'aplicació eclipse que també és totalment gratuït. Com que el codi del msms és java, descarregarem la versió "Eclipse IDE for java EE developers". La versió que s'utilitzarà durant aquest projecte és l'anomenada Helios (figura 60).

Figura 60. Logo eclipse Helios.

L'eclipse no necessita ser instal·lat, per la seva execució només cal descomprimir l'arxiu descarregat de la web, i buscar la icona "eclipse.exe"(figura 61)

Figura 61. Icona eclipse.exe

Per poder obrir el codi font del msms hem de seguir els següents passos:

- Creem un nou projecte java (figura 62).

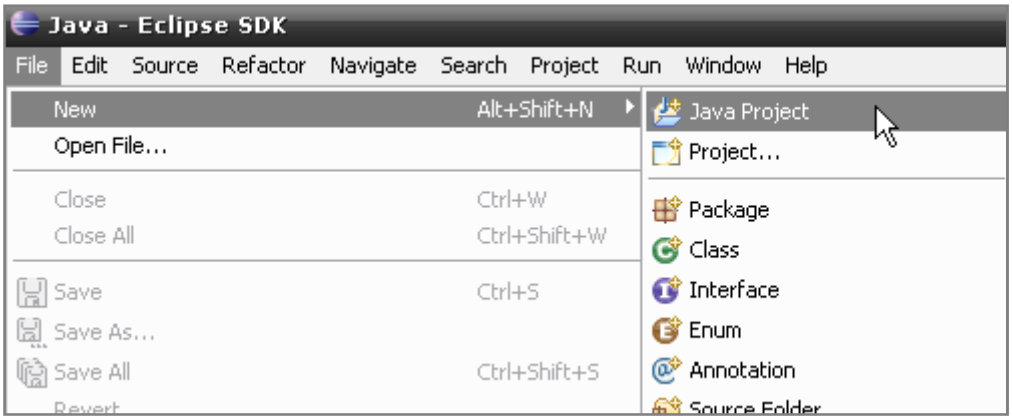

Figura 62. Imatge menú superior eclipse per crear un nou projecte.

- Adjudiquem un nom, en aquest cas msms, i especifiquem la ruta on hi ha el codi font del msms (figura 63).

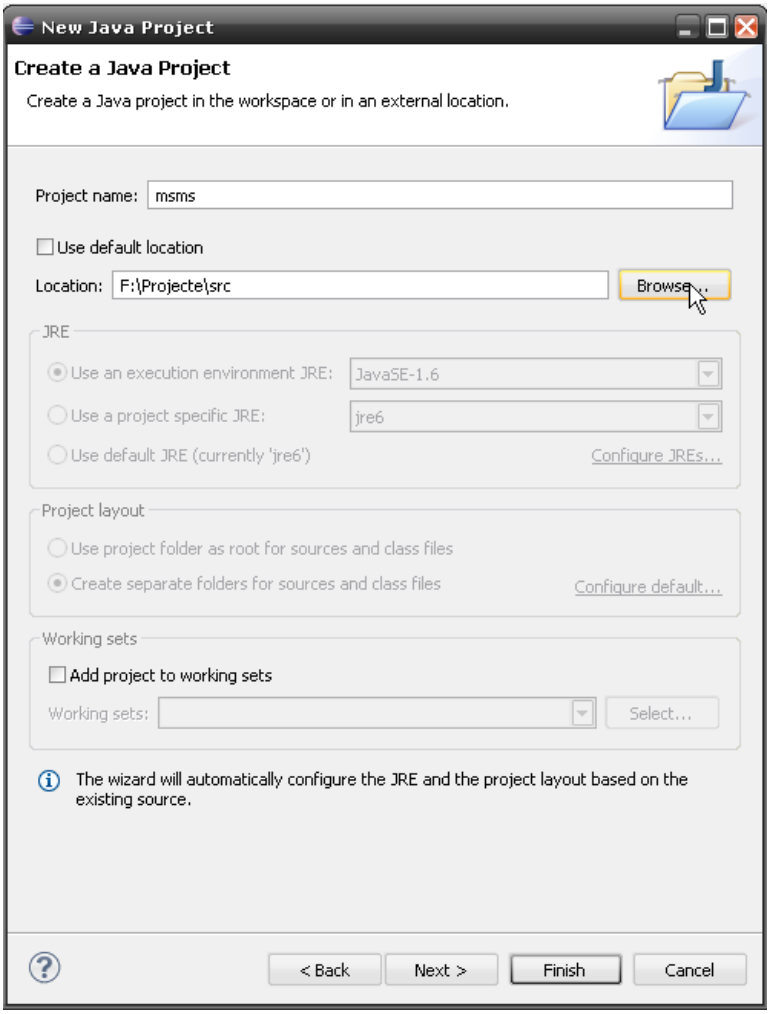

Figura 63. Finestra per la creació d'un nou projecte d'eclipse.

- Un cop premut el botó finalitzar ja haurem importat el codi font (figura 64).

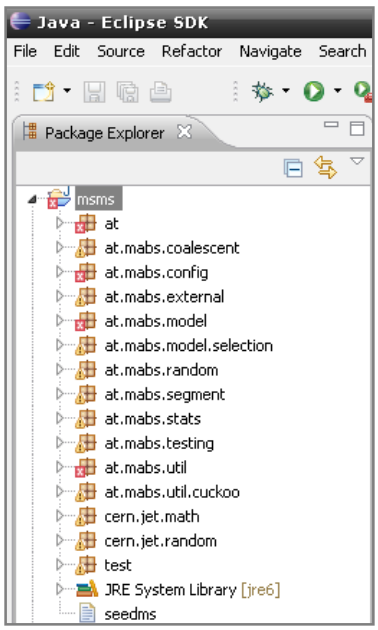

Figura 64. Imatge dels arxius importats amb errors.

- Veiem però que hi ha errors (creuetes vermelles, figura 64). Això és degut a que falta una llibreria pel funcionament del programa. Per adjuntar la llibreria hem d'anar a les propietats del projecte (figura 65).

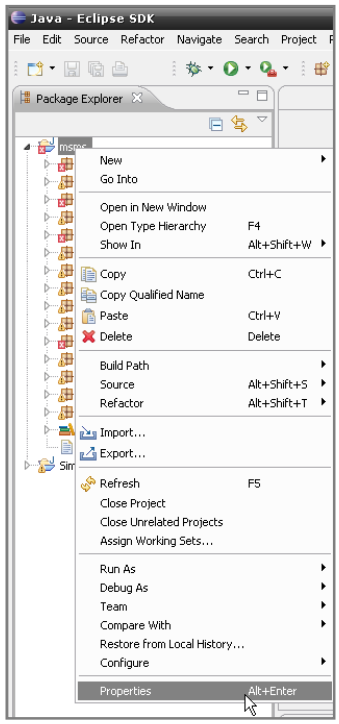

Figura 65. Imatge de la selecció de propietats del projecte.

- I en el apartat *java build path*, seleccionar la pestanya llibreries (figura 66).

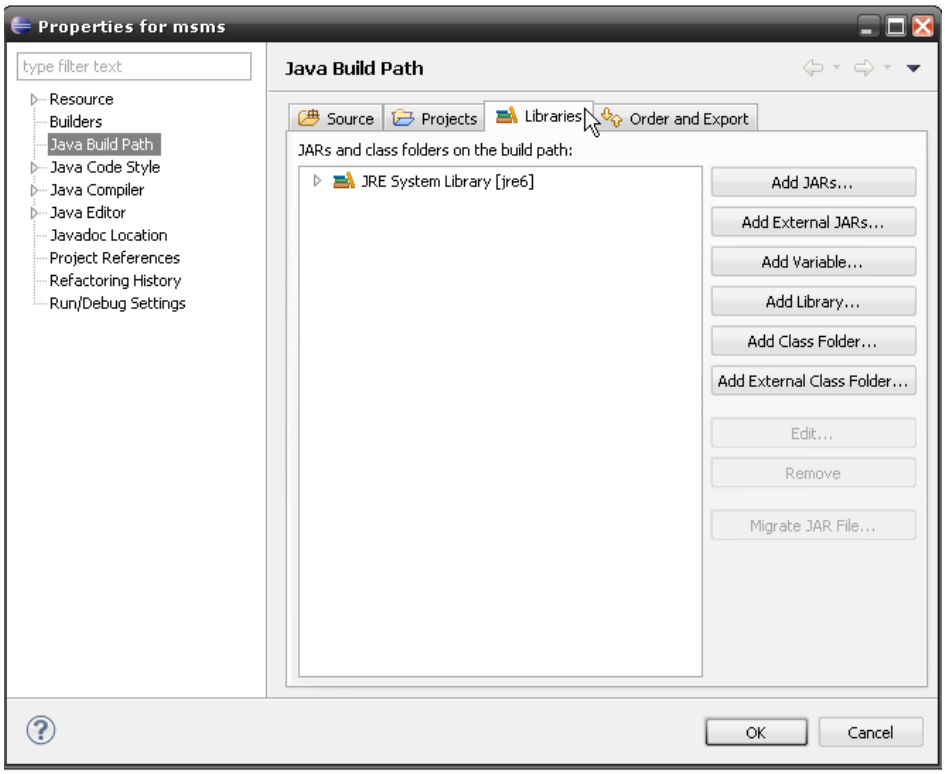

Figura 66. Finestra propietats del projecte (pestanya llibreries).

- I prémer el botó "add external JARs" (figura 67).

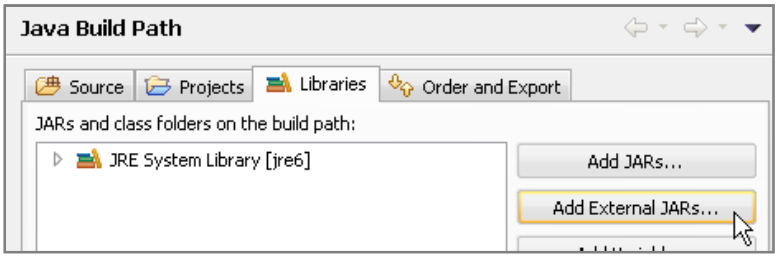

Figura 67. Botó afegir llibreries jar.

- Busquem l'arxiu msms.jar i l'afegim a les nostres llibreries (figura 68).

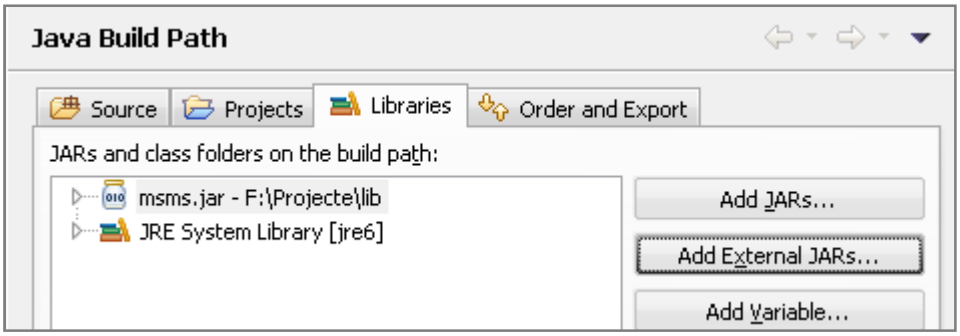

Figura 68. Llibreria msms.jar afegida al projecte.

Ara ja si que el codi font està preparat per poder ser compilat (figura 69).

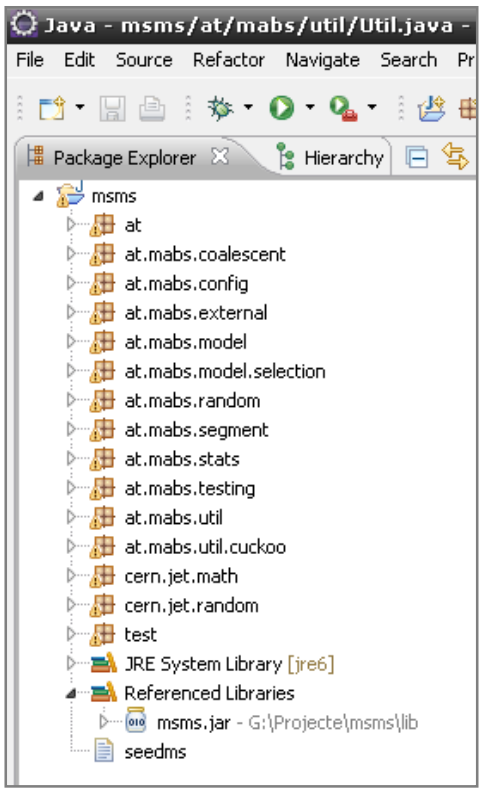

Figura 69. Codi font importat correctament.

Com que ja tenim el codi font per poder-nos familiaritzar amb el programa i anar fent proves, podem anar duent a terme execucions del msms amb diferents entrades de dades, mentre anem estudiant el codi.

3.4 Proves amb diferents entrades de dades

El msms al tractar-se de un programa en java, el podem executar des del símbol del sistema (figura 70)

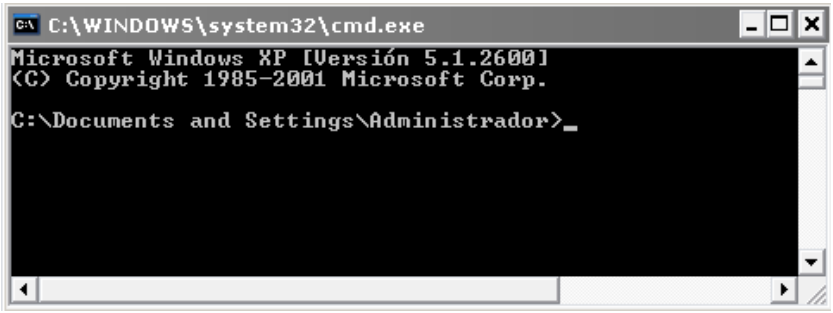

Figura 70. Símbol del sistema (Command prompt en anglès).

Per executar-lo només és necessari situar-se dins de la carpeta on està situat el fitxer msms.jar i executar la següent comanda:

Java –jar msms.jar –ms valor1 valor2 –t valor3

Recordem que l'execució:

Java –jar msms.jar –ms N r –t l

Denota que N és el número d'individus, r el número de repliques i l el número de llinatges.

Per poder mesurar el temps d'execució s'ha creat un fitxer .bat des del qual es crida l'execució del fitxer msms.jar amb els valors que es vulguin introduir, i crea un fitxer amb el dia i l'hora d'inici i fi de l'execució. Aquest fitxer l'anomenarem exec.bat i s'executarà des del símbol del sistema amb la comanda "exec.bat valor1 valor2 valor3". El exec.bat conté el següent codi:

```
echo %date% > "ms %1 %2 -t %3".txt
echo %time% >> "ms %1 %2 -t %3".txt
echo.
java -jar msms.jar -ms %1 %2 -t %3 > dades.txt
echo.
echo %date% >> "ms %1 %2 -t %3".txt
echo %time% >> "ms %1 %2 -t %3".txt
```

La primera línia escriu el dia en que comença l'execució en un fitxer .txt que s'anomenarà "ms primer_valor_introduït segon_valor_introduït –t tercer_valor_introduït.txt"

La segona línia afegeix l'hora d'inici de l'execució al fitxer txt.

La tercera afegeix una línia en blanc al fitxer txt.

La quarta línea executa el msms.jar amb els valors introduïts i guarda el resultat en un altre fitxer anomenat dades.txt que ens permetrà saber la mida de les dades de sortida.

I les restant són iguals que les anteriors, la cinquena afegeix una línea en blanc i la sisena i la setena afegeixen el dia i l'hora en que acaba l'execució.

Un exemple d'aquesta execució seria :

exec.bat 20 100 50

Aquesta execució crearia un fitxer anomenat "ms 20 100 –t50" amb l'hora i el dia de l'inici i la fi de l'execució, i un fitxer dades.txt amb els valors de sortida del msms.

Tot seguit es mostraran les taules que hem creat amb tots els resultats obtinguts després de fer diverses execucions amb diferents valors d'entrada.

SIMPLE AUGMENTEM N			
Comanda	CPU	GPU	Tamany dades.txt
-ms 20 100 –t 50	1 seg	-	0.5Mb
-ms 20 1000 –t 50	3 seg	-	5MB
-ms 20 10000 –t 50	6 seg	-	48 MB
-ms 20 100000 –t 50	35 seg	-	479 MB
-ms 20 1000000 –t 50	15 min	-	5,15 GB
-ms 20 10000000 –t 50	2h 27 min	-	47 GB
-ms 20 100000000–t 50	43h 42min	-	469 GB
-ms 20 1000000000–t 50	----	----	No cap en un HDD de 1TB

SIMPLE AUGMENTEM r			
Comanda	CPU	GPU	Tamany dades.txt
-ms 20 100 –t 50	43cs	-	472 Kb
-ms 200 100 –t 50	80 cs	-	5,98 Mb
-ms 2000 100 –t 50	6 seg	-	77,6 Mb
-ms 20000 100 –t 50	1 min 45 seg	-	0,99 GB
-ms 30000 100 –t 50	3 min 3 seg	-	1,5 GB
-ms 40000 100 –t 50	"out of memory"	-	---
-ms 200000 100 –t 50	"out of memory"	-	---

SIMPLE AUGMENTEM l			
Comanda	CPU	GPU	Tamany dades.txt
-ms 20 100 –t 50	43 cs	-	470 Kb
-ms 20 100 –t 500	1 seg 17 cs	-	4,52 MB
-ms 20 100 –t 5000	9 seg 30 cs	-	46,8 MB
-ms 20 100 –t 50000	1 min 33 seg	-	473 MB
-ms 20 100 –t 100000	3 min 13 seg	-	912 MB
-ms 20 100 –t 200000	5 min 45 seg	-	1,79 GB
-ms 20 100 –t 300000	"Out of memory"	-	---
-ms 20 100 –t 500000	"Out of memory"	-	---

COMBINAT			
Comanda	CPU	GPU	Tamany dades.txt
-ms 200 100 –t 500	5 seg 70 cs	-	55,9 MB
-ms 2000 1000 –t 5000	"out of memory"	-	---
-ms 2000 1000 –t 500	9 min 35 seg	-	7,64 GB
-ms 20000 1000 –t 500	"out of memory"	-	---
-ms 2000 10000 -t 500	1h 54 min	-	76,3 GB
-ms 2000 100000 -t 500	67h 13 min	-	765 GB
-ms 2000 1000000 –t500	---	-	No cap en 1TB

Com es pot observar en les anteriors taules, continuar fent execucions requeria de molt espai lliure en el disc dur, però el principal problema és que en tal quantitat de dades escrites en el disc dur, el temps real d'execució es va distorsionant cada cop més (els càlculs es fan molt més ràpid que l'escriptura de les dades en el disc dur), així que amb les dades obtingudes en tindrem prou per més endavant poder acabar d'omplir les mateixes taules amb els valor de les execucions en la GPU.

En algunes execucions, apareixia l'error "out of memory" (sense memòria). L'hem pogut solucionat en alguns casos proporcionant més memòria al programa amb comandes com per exemple ***java -Xmx1600m*** que

proporciona al programa 1,6 GB de memòria RAM (si canviem el 1600 per 1700 proporcionarem 1,7 GB de RAM). Però en altres casos el msms requeria més de 2GB de memòria per crear el arbre i com que el PC només compta amb 2 GB de RAM, no s'han pogut fer més execucions.

Podem veure com a exemple del contingut del fitxer "ms 20 10000000 –t 50.txt" en la figura 71 on es pot observar que el temps d'execució ha estat aproximadament de dos hores i mitja.

Figura 71. Contingut del fitxer "ms 20 10000000 –t 50.txt"

3.5 Examinar el codi font

Com que el codi font del msms és molt complex amb múltiples fitxers font, utilitzarem una eina que permet veure l'execució del programa pas a pas.

Chronon es una extensió per l'eclipse que permet enregistrar tota l'execució del programa en el disc dur, i després reproduir-la pas a pas, marcant en cada moment la línea de codi que s'està executant, els mètodes, les excepcions i els valors de les variables entre algunes altres coses, permetent així tenir un control lineal del que està executant l'aplicació.

Chronon és un programa de pagament, però tenim la opció de descarregar-nos la versió de prova que és vàlida per 30 dies i la versió per estudiant que dóna llicència durant tot un any. Chronon es pot aconseguir des de la web http://www.chrononsystems.com/ on hi ha una petita demostració del que pot fer aquesta eina. (figura 72)

Figura 72. Logo chronon.

Un cop instal·lada l'aplicació i aconseguida la llicència, ens apareixen tres botons nous a la barra d'eines d'eclipse:

El botó d'enregistrar. Aquest és el botó que ens fa falta per poder enregistrar l'execució del programa en el disc dur. (figura73)

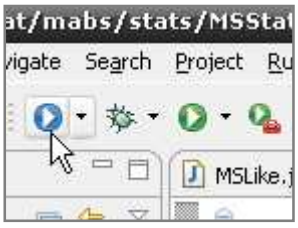

Figura 73. Botó per enregistrar l'execució.

El botó de càrrega. Aquest botó és el que ens permet seleccionar quina de les execucions guardades en el disc dur volem reproduir. Automàticament quan carregem una execució guardada, apareix la perspectiva de Chronon (figura 74).

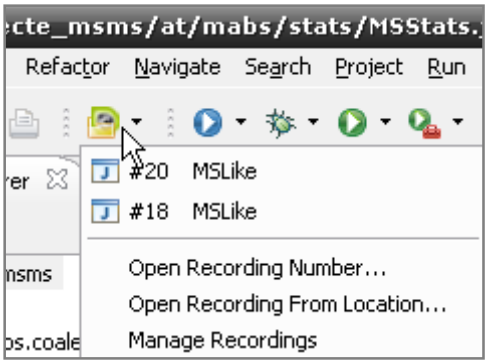

Figura 74. Botó de càrrega d'execucions guardades.

El botó de la perspectiva Chronon. La perspectiva de chronon és el que ens permetrà veure pas a pas l'execució (figura 75).

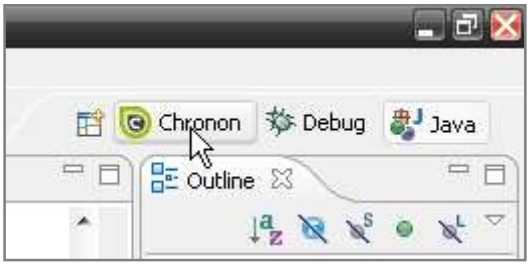

Figura 75. Botó perspectiva chronon.

3.6 Explicació bàsica de la perpectiva de chronon

Les fletxes ens permeten moure per cada línea de codi. Les dos del mig ens porten a la seguent o a l'anterior línia, mentre que la primera ens permet seguir les crides a altres funcions i la última tornar d'aquestes crides (figura 76).

Figura 76. Fletxes de la perspeciva chronon.

A la part dreta de la perspectiva chronon es mostra el temps transcorregut respecte el temps total de l'aplicació. (figura 77)

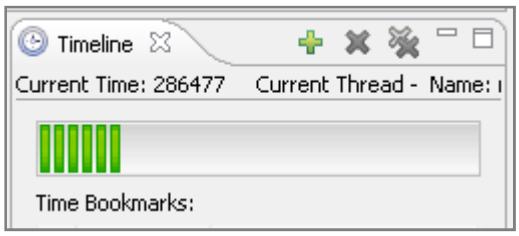

Figura 77. Barra de temps transcorregut.

Al mig de la pantalla es mostra la línia de codi que s'està executant remarcada en verd. La columna vertical també verda ens mostra les línies de codi executades en la crida (figura 78).

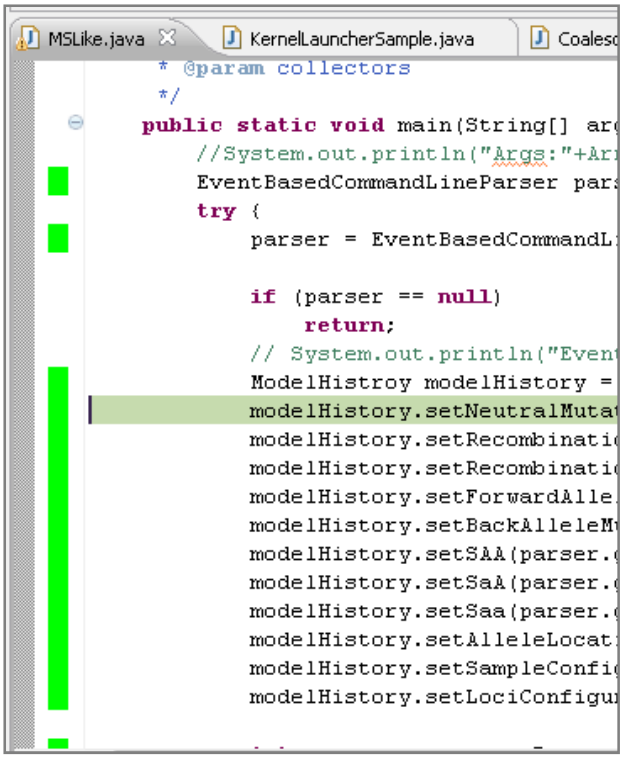

Figura 78. Vista de la línea de codi que s'està executant

Amb la perspectiva de Chronon ens és molt més senzill veure en quins punts el programa executa els càlculs i consumeix la major part del temps.

L'execució principal es porta a terme en el `MSlike.java`, on s'executa un bucle de tantes repeticions com el segon paràmetre que s'ha introduït com a valor d'entrada.

Aquest bucle, al mateix temps, crida a dues funcions situades en els arxius `coalescentEventCalculator.java` i `MSStats.java` respectivament, que dins seu executen un bucle i dos bucles més respectivament. Dins de la funció situada en el coalescentEventCalculator.java, es crida a una altre funció situada en l'arxiu LineageState.java, on de nou hi ha un altre bucle. Aquest arbre es pot veure més clarament en la figura 79.

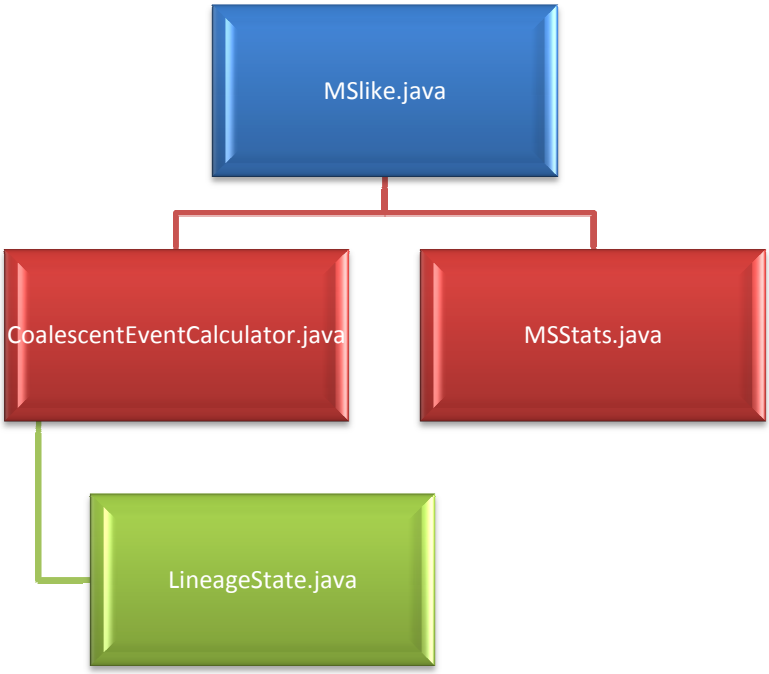

Figura 79. Arbre execució msms.

En aquests quatre arxius és on es consumeix la major part del temps de l'execució, per tant ens hem de centrar en aquest nuclis que són els que haurem de transferir a la GPU per tal de que els calculi.

Si els anem desglosant mica en mica anem arribant a les fulles d'aquest arbre, i podem observar com el que inicialment podria semblar una massa d'operacions es redueix a un munt de iteracions amb moviments de dades, i operacions bàsiques, com per exemple sumar 1 al total que portem acumulat, o fer una divisió entre 64. (figura 80 i figura 81 respectivament)

```
public static final int sum(int[] array){
    int cumulant=0;
    for(int i=0;i<array.length;i++)
        cumulant+=array[i];
    return cumulant;
}
```

Figura 80. Exemple codi msms on es realitza una suma.

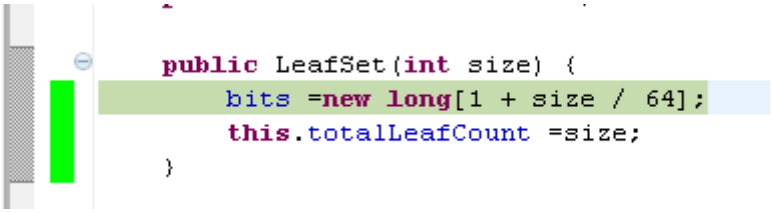

Figura 81. Exemple codi msms on es realitza una divisió.

Després d'analitzar profundament el msms no s'ha trobat el què s'esperava, una massa de còmput intenssiu que pogués ser enviada a la GPU i aquesta mitjançant els càlculs paral·lels retornés la resposta en un menor temps que el quin s'havia mesurat anteriorment en la CPU. A més el fet de que el codi inclogui molts condicionals (if) com és el cas, no permet la execució síncrona de tots els threads, i això implica una pèrdua de rendiment molt important. En aquest programa la implementació de CUDA no seria efectiva, ja que per realitzar una operació de sumar una unitat, hauriem de copiar els valors que tenim per sumar a la memòria principal, enviar-los a la memòria gràfica, la GPU ho calcularia, i de la memòria gràfica es tornaria a copiar a la memòria principal. Tot aquest procés surt a compte si el càlcul que s'ha de dur a terme és intenssiu, però en aquest cas, el msms no milloraria al ser executat en la GPU, tot el contrari, el temps d'execució augmentaria molt més.

Seria el cas que s'ha vist anteriorment en l'avaluació de movavi, que per a arxius més petits, CUDA no aportava un benefici de temps de còmput. En aquest cas inclús podria tardar més del que tarda en la CPU.

Per tant amb el msms hem pogut comprovar que no tots els programes es poden beneficiar de les noves tecnologies de les GPU. Crear o modificar una aplicació per ser executada en la GPU, implica realitzar un gran estudi del programa, tenir un gran coneixement de com treballa i com es té que fer servir CUDA, ATI stream o OpenCL segons sigui el cas, i sobretot entendre quins beneficis pot aportar aquesta tecnologia i quines són les seves debilitats. Obviament no qualsevol aplicació millorarà el seu rendiment en una GPU i fins i tot ens podem trobar el cas que sabent que en un determinat tipus de còmput, una GPU és millor, el resultat no es millori (com s'ha pogut comprovar en el cas de l'avaluació de l'aplicació movavi).

CAPÍTOL V: CONCLUSIONS

1. Conclusió general

Després d'haver realitzat les proves pertinents amb CUDA, podem concloure que la utilització de les GPUs en programes que anteriorment es computaven en la CPU no és efectiva per totes les aplicacions, ja que només ens aporta beneficis en casos concrets. Hem mostrat que CUDA accelera notablement els processos de codificació, modificació i correcció en arxius de vídeo, però al mateix temps no ha pogut ser útil per realitzar el còmput d'un programa de coalescència.

L'augment de rendiment que experimenten les aplicacions en les targetes gràfiques és degut a las diferències fonamentals que existeixen entre les arquitectures de les CPUs i les GPUs. El problema més gran que sorgeix és l'accés a memòria i mentre que les CPUs estan dissenyades per accedir aleatòriament a memòria afavorint la creació d'estructures de dades complexes amb els punters en posicions arbitràries en la memòria, les GPUs tenen un accés a memòria molt més restringit, podent igualment llegir varies posicions arbitràries però sempre escrivint en la mateixa posició predeterminada.

Així doncs, el dissenyador de programes GPGPU té la tasca d'adaptar els accessos a memòria i les estructures de dades a les característiques de la GPGPU. La majoria d'algoritmes executats en una CPU es poden arribar a implementar en una GPU, però això no sempre aportarà un augment d'eficiència. És necessari que les aplicacions que es vulguin executar en la GPU tinguin un elevat grau de paral·lelisme i una gran densitat aritmètica per tal de que executar un programa en la targeta gràfica aporti beneficis.

Les GPUs aporten una sèrie d'avantatges, com són la ràpida evolució que tenen (gairebé unes tres vegades més ràpid que el marcat per la llei de Moore), el baix cost, la gran millora en la programabilitat que s'ha fet en els últims anys i l'alliberació de tasques de la CPU. Però al mateix temps tenen una sèrie d'inconvenients com per exemple que no sempre es pot generar una solució factible o convenient, tenen unes certes limitacions del

maquinari (cada model de targeta gràfica és diferent i té diferents prestacions) i que el model de programació ha de seguir el SIMD, però té un alt nivell de dificultat, ja que requereix de molts conceptes de baix nivell.

Així doncs tot el programari que tingui una gran quantitat de còmput aritmètic que pugui ser executat de forma síncrona, en el qual no hi hagin dependències ni condicionals i que tingui un gran nivell de paral·lelisme es podrà aprofitar dels avantatges de ser executat en la targeta gràfica. Conseqüentment existiran moltes aplicacions que no compliran part d'aquests requeriments i mai s'executaran més ràpid en una GPU. Actualment els programes que obtenen més benefici estan enfocats en el tractament de imatges i de vídeo degut a que compleixen el tres principals requisits esmentats.

La importància de la tecnologia GPGPU anirà creixent a la velocitat que marquin els consumidors de programari que veuran incrementada la velocitat de les seves tasques diàries utilitzant el tàndem que formen la CPU i la GPU per als còmputs de propòsit general. Els principals fabricants de GPUs saben que el futur passa per el GPGPU i per això cada vegada donen més facilitats per a la utilització dels seus productes, així com treballen per introduir més tecnologia en els seus dispositius.

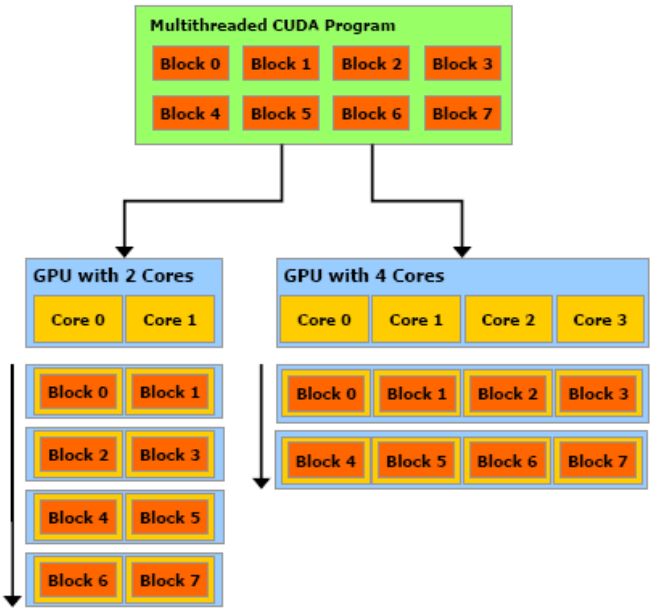

Figura 82. Comparació de l'execució de un programa amb CUDA , utilitzant una GPU de 2 nuclis (a l'esquerra) o una de 4 nuclis (a la dreta).

Les motivacions que han vist els fabricants de programari per desenvolupar aplicacions executades majoritàriament en la GPU, és que les targetes gràfiques tenen un baix cost, experimenten una gran i ràpida evolució a causa de la demanda del món dels videojocs, tenen una altes prestacions amb els processadors stream que només estan aprofitades en videojocs i aplicacions molt específiques, i tenen una bona capacitat de programació.

El cas particular del projecte ha demostrat que no tot programa pot ser executat en una GPU i augmentar el seu rendiment. No ha fet falta continuar desenvolupant aquest apartat per poder concloure que el temps d'execució del msms en la GPU hagués estat més lent. A conseqüència d'aquest fet, la planificació inicial que s'havia fet en l'estudi de viabilitat s'ha vist lleugerament afectada, augmentant considerablement el temps destinat a la tasca de proves amb CUDA intentant buscar una solució, i reduint sensiblement el temps destinat a l'anàlisi de resultats degut a que s'han produït menys comparacions.

Tasques	Durada	Inici	Fi	Predec.	Cost	Treball
Projecte final de carrera	**70,5 días**	**mar 01/02/11**	**mar 28/06/11**		**8.939,65 €**	**199 horas**
Inici del projecte: assignació i matriculació del projecte	0,4 días	mar 01/02/11	mar 01/02/11		128,00 €	2 horas
Planificació	**3,75 días**	**mar 01/02/11**	**mar 15/02/11**	**2**	**980,00 €**	**16 horas**
Estudi de viabilitat	3,75 días	mar 01/02/11	lun 14/02/11		900,00 €	15 horas
Aprovació Estudi Viabilitat (Punt de control)	1 día	lun 14/02/11	mar 15/02/11	4	80,00 €	1 hora
Anàlisi de l'aplicació	**17,6 días**	**mar 15/02/11**	**vie 18/03/11**	**5**	**2.352,00 €**	**51 horas**
Anàlisi de requisits fuincionals i no funcionals	1,67 días	mar 15/02/11	lun 21/02/11		225,00 €	5 horas
Anàlisi de dades	5 días	jue 24/02/11	mié 09/03/11	7	900,00 €	20 horas
Anàlisi de la seguretat i legalitat	2 días	lun 21/02/11	jue 24/02/11	7	360,00 €	8 horas
Documentació de l'anàlisi	3,75 días	mié 09/03/11	vie 18/03/11	9;7;8	675,00 €	15 horas
Aprovació de l'anàlisi (Punt de control)	0,6 días	vie 18/03/11	vie 18/03/11	10	192,00 €	3 horas
Disseny de l'aplicació	**46,45 días**	**vie 18/03/11**	**mié 22/06/11**	**11**	**4.599,65 €**	**116 horas**
Proves programari	25,2 días	vie 18/03/11	lun 30/05/11		1.537,50 €	35 horas
Proves programari amb CUDA	21,48 días	vie 18/03/11	vie 20/05/11		1.317,86 €	30 horas
Programació amb CUDA	18,94 días	vie 18/03/11	vie 15/04/11	11	750,00 €	25 horas
Conclusions	2 días	lun 30/05/11	jue 02/06/11	15;13;14	0,00 €	4 horas
Document del disseny	7,09 días	jue 02/06/11	vie 17/06/11	16	873,93 €	20 horas
Aprovació del disseny (Punt de control)	0,52 días	vie 17/06/11	mié 22/06/11	17	120,37 €	2 horas
Finalitzar projecte	**4,79 días**	**vie 17/06/11**	**mar 28/06/11**	**17**	**880,00 €**	**14 horas**
Generació de documents	2,5 días	vie 17/06/11	mié 22/06/11		600,00 €	10 horas
Tancament del projecte	2 días	mié 22/06/11	lun 27/06/11	20	160,00 €	2 horas
Entrega del projecte	0,5 días	mar 28/06/11	mar 28/06/11	21	120,00 €	2 horas

Figura 83. Descripció de les tasques realitzades durant el projecte.

Com que la tasca de programació amb CUDA es realitzava mentre es duien a terme les proves del programari, la data d'entrega del projecte no s'ha

vist modificada. Així doncs un cop finalitzat el projecte el diagrama de Gannt ha resultat ser el següent:

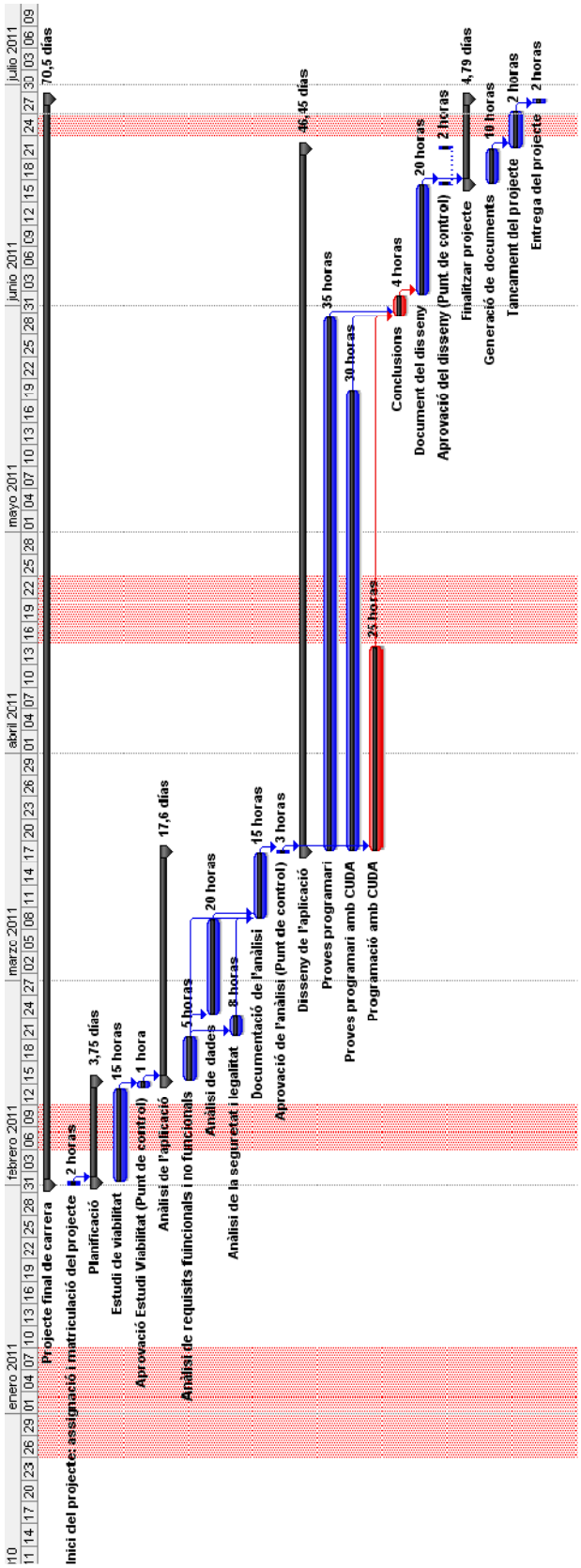

Figura 84. Diagrama de Gannt al finalitzar el projecte.

En resum, podem assegurar que la tecnologia de la GPGPU ens assegura un millor rendiment en els casos de còmput intensiu i aritmètic, on es puguin processar elements en paral·lel i on hi aparegui un baix nivell de dependències i condicionals.

2. Valoració personal

L'elecció d'aquest projecte va vindre motivada per la desconeixença total sobre el ús de les targetes gràfiques per executar programes de propòsit general. No coneixia gairebé res sobre aquest tema, però em va captivar l'idea de que una GPU pogués alleujar la càrrega de còmput que suporten les CPU.

Durant el desenvolupament del projecte he pogut comprovar que aquesta tecnologia tot i ser relativament nova, està evolucionant molt ràpidament, prova d'això és que CUDA per exemple ja ha tret quatre versions diferents en els últims anys, i que té totes les possibilitats de ser la tecnologia que s'imposarà en un futur no molt llunyà (AMD, un dels fabricants de CPUs més importants, va comprar ATI en l'any 2006).

Per aquestes raons, crec que ha estat una bona experiència realitzar el projecte sobre aquesta tecnologia que donarà un nou món de possibilitats als programadors de programari, si es que ja no els hi dóna.

3. Millores futures

Seguint la línea d'investigació que s'ha portat a terme durant el projecte, un treball futur es podria basar en portar aquest estudi un pas endavant, comparant directament la mateixa aplicació amb diverses targetes gràfiques d'nVidia i d'ATI utilitzant CUDA, ATI Stream i OpenCL.

ANNEX

1. Abreviacions

ALU: La Unitat Aritmètico Lògica (Aritmetic Logic Unit) és un circuit digital que calcula operacions aritmètiques i lògiques entre dos nombres.

API: Application Programming Interface. Conjunt de crides a certes biblioteques que ofereixen accés a certs serveis des dels processos i representa un mètode d'abstracció en la programació.

CAL: Fa referència tant al compilador com a un conjunt de eines de desenvolupament creades per ATI /AMD que permeten als programadors fer servir una variació del llenguatge de programació C per codificar algoritmes en GPUs de ATI/AMD.

CPU: Unitat Central de Processament. És el component del ordinador i d'altres dispositius programables, que interpreta les instruccions contingudes en els programes i processa les dades.

CUDA: Compute Unified Device Architecture. Fa referència tant al compilador com a un conjunt de eines de desenvolupament creades per nVidia que permeten als programadors fer servir una variació del llenguatge de programació C per codificar algoritmes en GPUs de nVidia.

DNA: (ADN) L'àcid desoxiribonucleic és un àcid nucleic que conté les instruccions genètiques utilitzades en el desenvolupament i funcionament de tots els éssers vius conegut.

DRAM: Dynamic Random Access Memory és una memòria electrònica d'accés aleatori que s'usa principalment en els mòduls de memòria RAM com a memòria principal del sistema.

GFLOPS: GigaFLOPS és una unitat de mesura de rendiment de còmput, especialment en càlculs científics que requereixen un gran ús d'operacions en coma flotant.

GLSL: OpenGL Shading Language, és una part de l'API estàndard d' OpenGL que permet especificar segments de programes gràfics que seran executats en una GPU.

GPGPU: General Purpose computing on Graphics Processing Units. Computació de propòsit general sobre unitats de processament gràfic. És la tècnica de fer servir GPUs per executar aplicacions tradicionalment tractades per la CPU.

GPU: Unitat de Processament Gràfic. És un processador dedicat al processament de gràfics o operacions de coma flotant.

HDD: Hard Disk Drive, Disc dur. És un dispositiu d'emmagatzemament no volàtil.

MIMD: Multiple Instruction Multiple Data, qualsevol processador pot executar diferents instruccions sobre diferents dades.

MISD: Multiple Instruction Single Data, diverses unitats funcionals realitzen diferents operacions sobre les mateixes dades.

Msms: és un programa de simulació coalescent (unitari), que inclou recombinació, estructura demogràfica i selecció en un sol llloc.

nVidia: Empresa multinacional especialitzada en desenvolupar unitats de processament gràfic.

SIMD: Single Instruction Multiple Data, una sola instrucció i múltiples dades. És una tècnica emprada per aconseguir paral·lelisme a nivell de dades.

SISD: Single Instruction Single Data, una sola instrucció una sola dada. Un únic processador executa una única dada.

SO: Sistema Operatiu.

OpenGL: És una especificació estàndard que defineix una API per escriure aplicacions que produeixen gràfics 3D.

2. Referències

Arxius en format pdf obtinguts de www.amd.com

-AMD Compute Abstraction Layer (CAL) (2010).

-Technical overview ATI Stream Computing.

-Programming guide ATI Stream Computing OpenCL.

Arxius en format pdf obtinguts de www.nvidia.com

-CUDA quickstart guide (2008).

-CUDA C best practices guide (2011).

-nVidia CUDA C programing guide (2011).

-nVidia CUDA reference manual (2010).

Pàgines web consultades en l'any 2011:

- www.khronos.org/opencl
- www.vreveal.com
- www.badaboomit.com
- www.adobe.com
- www.es.cyberlink.com
- www.nero.com
- www.arcsoft.com

- www.tmpgenc.net
- www.movavi.com
- www.setiathome.berkeley.edu
- www.jcuda.org
- http://www.mabs.at/ewing/msms/index.shtml
- www.eclipse.org
- www.chrononsystems.com
- www.wikipedia.org

Autor: Jordi Mitjana Trullàs