ENGINYERIA EN GEOMÀTICA I TOPOGRAFIA
TREBALL DE FI DE GRAU

PROGRAMACIÓ EN PARAL·LEL. APLICACIÓ A PROCESSOS FOTOGRAMÈTRICS

Projectista: Hèctor Muro Mauri
Director: Albert Prades Valls
Convocatòria: Juny 2015

1. Resum

Aquest projecte pretén realitzar un nou enfocament a processos fotogramètrics típics, des de punt de vista de la programació en paral·lel, amb l'objectiu de demostrar que l'aplicació d'aquest mètode de programació a l'àmbit de la fotogrametria pot aportar millores substancials en el temps de càlcul.

Per tal de mostrar-ho s'han triat com a processos de prova, per una banda, la cerca de correspondències per correlació d'un parell d'imatges estereoscòpiques i, per una altra banda, la obtenció d'una ortofotografia. No s'ha pretès canviar l'arrel d'aquests processos, sinó fer una aplicació directa del mateix procés en un entorn en paral·lel. Per tant, es tracta d'un anàlisi comparatiu entre el mètode tradicional (en sèrie) i el nou (en paral·lel).

El treball té sentit perquè els darrers anys hi ha hagut un increment en la quantitat de dades amb les que ja es treballa en l'àmbit fotogramètric. El fet d'emprar una gran quantitat de fotogrames a l'hora de cercar correspondències, implica que els mètodes tradicionals esdevinguin cada cop més lents.

Els resultats s'analitzen en funció de la relació de temps obtinguda entre la programació sèrie i paral·lela. Ambdós casos es comparen en un mateix entorn, és a dir, amb el mateix ordinador i el mateix problema.

Les conclusions obtingudes respecte són que definitivament es tracta d'un mètode que s'ha d'implementar en aquests i altres àmbits ja que les reduccions de temps que implica són considerables.

Índex

1. Glossari

Kernel: Una funció Kernel especifica el codi que serà executat per tots els fils (threads) durant la fase en paral·lel.

Thread o fil o vector: un thread o fil és la seqüència més petita de les instruccions programades que pot ser duta a terme independentment de la resta.

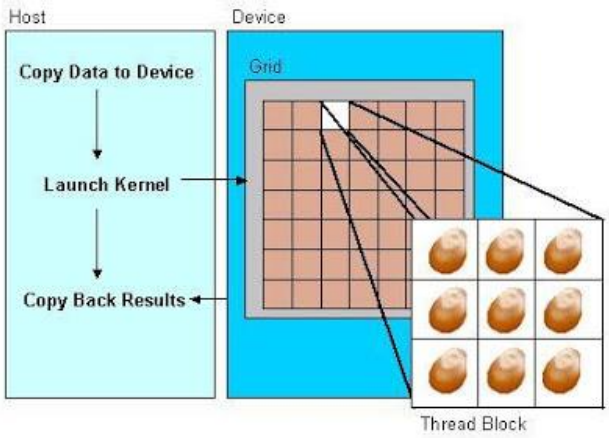
Block o bloc: Agrupació de threads, és el nivell jeràrquic superior. Cada bloc, per definició física, no pot tenir una mida mes gran de 512 o 1024 threads, en funció del processador.

Grid o malla: Agrupació de blocs.

CPU: Unitat de Processament Central (Central Processing Unit en les seves sigles en anglès) .

GPU: Unitat de Processament Gràfica (Graphic Processing Unit en les seves sigles en anglès).

GPU vs. CPU: Les GPU estan dissenyades (pel que fa a hardware) com a màquines de computació numèrica, i no faran bé certes tasques per a les que la CPU està especialment pensada, de tal manera, que un ha de pensar que la majoria d'aplicacions utilitzen tant la CPU com la GPU, executant les parts seqüencials del codi sobre la CPU i les parts "paral·lelizables" del codi sobre la GPU.



Il·lustració 1. Funcionament d'un programa CUDA (Kirk, 2010)

2. Introducció

Aquest projecte pretén explorar millores en el temps i en el rendiment que pot oferir la programació en paral·lel sobre certs processos relacionats directament amb la fotogrametria.

Es va decidir abordar els processos de la correlació de dos parells d'imatges, dins la cerca de correspondències, per una banda i de la creació d'una ortofotografia per una altra. La investigació només d'aquests dos processos no implica la seva exclusivitat dins el món de la programació en paral·lel, sinó un simple intent d'acotar el treball i fer-lo abastable per al temps estimat d'entrega.

D'entorns de programació en paral·lel n'hi ha molts. Des de OpenCL, que es tracta de programari lliure, fins a entorns creats per empreses privades com NVIDIA o AMD. En aquest projecte es va decidir utilitzar l'arquitectura CUDA, dins el compilador Visual Studio 2010, que empra les targetes gràfiques NVIDIA.

Per tal de comparar les millores, per a cada procés s'ha programat el mateix programa en sèrie i en paral·lel. A més, en el cas de la programació en paral·lel, ja que ofereix més llibertat a l'hora de programar, s'ha enfocat el mateix problema des de diferents perspectives, per tal de poder extreure més conclusions respecte al rendiment de la programació en paral·lel.

A l'hora d'iniciar la implementació del codi sempre s'ha partit del mateix origen. La obtenció primer dels processos en programació tradicional i després s'ha modificat aquest codi fins a obtenir el mateix procés, però completament paral·lelitzat.

3. Context global

En aquest apartat es contextualitzarà la programació en paral·lel, partint de les principals raons que van donar lloc al seu origen, per acabar al nivell actual de desenvolupament d'aquesta.

Durant més de dues dècades, els processadors basats en una única unitat de processament central (CPU), han patit grans increments i reducció de costos, en quant a aplicacions d'ordinador. D'aquesta manera, s'ha aconseguit portar a nivell d'usuari operacions de GFLOPS[1] per segon.

Aquesta millora en el rendiment i en la velocitat de càlcul dels processadors, ha provocat que s'hagi pogut produir un software de major funcionalitat, amb millors interfícies per a l'usuari, i amb resultats més potents durant molts anys. Els usuaris, tanmateix, un cop acostumats a aquestes millores acaben demanant-ne més, provocant un cercle d'oferta i demanda que és al voltant del que la indústria de la informàtica gira.

Per tant, durant molt de temps s'ha viscut sota la idea que a cada millora de hardware hi hauria un gran increment pel que fa a la velocitat i a la funcionalitat de les aplicacions que del hardware depenen. Si ha estat així és perquè fins fa ben bé pocs anys, a cada versió nova de processadors, el mateix software funcionava molt més ràpid que en l'anterior.

Aquesta corba de creixement, però, s'ha frenat una mica des de l'any 2003 degut al consum d'energia i a la dissipació de calor que han limitat l'increment de la

[1] FLOPS: floating-point operations: Les mesures de coma flotant per segon són una mesura de rendiment per a un ordinador, especialment en càlculs científics que requereixin un gran nombre d'operacions de coma flotant. 1 GFLOPS = 10^9 FLOPS.

freqüència del rellotge (clock-rates[2]) i el nivell d'activitat productiva que poden ser dutes a terme durant el mateix període de rellotge en una sola CPU. Es pot afirmar doncs que la temperatura és una de les principals culpables de la frenada en la millora de la velocitat de les CPU's. A més, si actualment s'incrementen les velocitats, és només degut a la creació de transistors més petits i per tant n'hi caben més, no perquè s'hagi aconseguit disminuir el període de rellotge.

Cal tenir present que, la gran majoria d'aplicacions de software estan escrites com a programes seqüencials, és a dir, que fan una operació o ordre després de l'altra. En aquest treball diferenciarem entre la programació seqüencial o tradicional i la programació en paral·lel.

Tot i que ja existeixin els ordinadors amb més d'un processador, un programa només s'executarà en un sol processador i el fet de tenir-ne quatre o vuit no farà que el programa funcioni més ràpid. El rendiment de l'ordinador en termes generals millorarà, ja que podrà fer altres operacions o comandes al mateix temps que executa un programa, però el programa en si no funcionarà més ràpid.

D'aquesta manera, sense millores en el rendiment, les noves aplicacions no podran ser actualitzades amb noves característiques ni capacitats dintre del seu software. Per a superar aquest límit de software és on rauen els inicis de la programació en paral·lel.

La programació en paral·lel consisteix bàsicament en realitzar una sèrie de processos literalment a l'hora, "paral·lelament". En contraposició al mètode seqüencial o tradicional. Això és possible gràcies a la utilització de les targetes gràfiques com a petits processadors. Aquesta realitza tot un seguit de processos paral·lelitzables amb la finalitat d'optimitzar el temps d'execució i aconseguir una major velocitat i rendiment en els processos.

La pràctica de programar en paral·lel a nivell d'usuari i a baix cost és molt recent. Fins fa relativament poc, la programació en paral·lel estava restringida als

[2] Clock-rates: És la freqüència en la que funciona una CPU, és a dir, la velocitat a la que realitza les seves ordres bàsiques. Es mesura en cicles per segon (Hz).

"superordinadors" situats en certes universitats del món, o bé a treballar amb diferents ordinadors connectats en paral·lel. És a dir, tenir moltes CPU's físicament. El que resulta realment "innovador" i assequible per a un públic molt més ampli i de base, és el fet de poder utilitzar una targeta gràfica, d'un ordinador portàtil, per exemple, com si es tractessin de molts processadors.

A l'actualitat, tal i com es pot veure en aquest projecte, qualsevol usuari amb un mínim coneixement de programació i un mínim hardware (gairebé qualsevol targeta gràfica amb menys de cinc anys d'antiguitat) pot fer i executar el seu programa en paral·lel. Això obre un gran ventall de possibilitats, degut al canvi de mentalitat que suposa el poder utilitzar una eina, abans sempre pensada i orientada als jocs i a la realitat virtual, en tants àmbits com els usuaris siguin capaços de trobar quelcom paral·lelitzable.

En la gran majoria d'àmbits relacionats amb la ciència, enginyeria o tecnologia, existeixen multitud de processos repetitius i que requereixen molt de temps de càlcul, hores i dies, per a arribar a resultats concloents. Són aquest tipus de processos els que han de ser paral·lelitzats i reduir els temps de càlcul a minuts o menys hores.

Això suposaria un estalvi de temps, i per tant econòmic, prou important com per a menystenir la millora que implica la programació en paral·lel. Més enllà de que sigui la solució actual al límit del hardware, es ben possible que en els pròxims anys o dècades, apareguin nous materials o conductors que faran tornar a millorar la funcionalitat del hardware. Ara per ara, saber programar en paral·lel i saber-ne fer un ús correcte de les eines serà sempre útil, independentment del hardware del que es disposi. És, en el fons, una manera d'emprar gairebé totes les possibilitats que el hardware ens dóna per a realitzar els processos que necessitem.

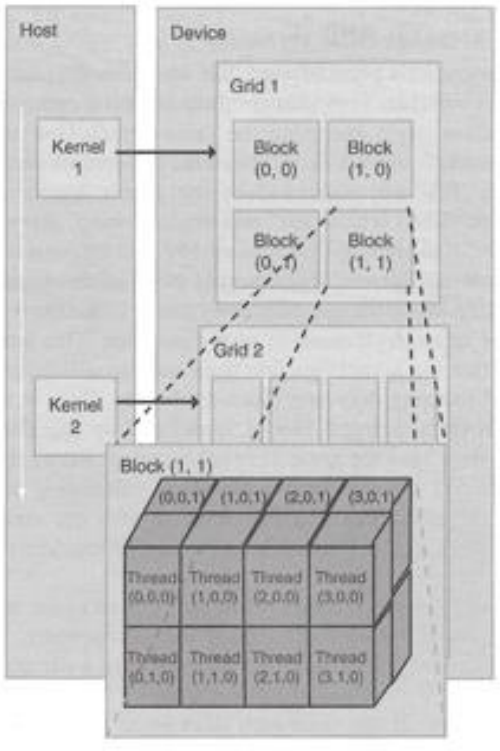
Tot i l'esmentat amb anterioritat, cal recordar i fer saber que les GPU's tenen els seus límits i que aquests són importants. Aquestes són processadors de vectors o fils que han de ser altament separables, o segmentats, optimitzats per a operacions

gràfiques i amb una programabilitat limitada. Per a que un programa sigui executable en paral·lel ha de ser fàcilment divisible en diferents unitats d'execució, petites o grans, però podent-hi encabir moltes operacions en un sol cicle de rellotge.

A més, el hardware té una sèrie de límits físics són absolutament necessaris de tenir en compte a l'hora de programar. És més, s'ha de tenir present com està organitzada la targeta gràfica per tal de programar eficientment i de manera paral·lela. Un d'aquests límits és que cada bloc només pot contenir 512 o 1024 fils, en funció de la targeta gràfica. Tot i això hi ha un nombre molt elevat de blocs, per tant és pot paral·lelitzar a molt més gran escala que 512 o 1024 operacions.

Cada targeta gràfica té un nombre determinat, però molt elevat, de fils. Quan s'invoca un nucli en CUDA, es crea automàticament una graella, que estarà subdividia en blocs i que aquests al mateix temps contindran els fils que executaran el programa.

La estructura física que tindrà la graella vindrà determinada pels blocs i podrà ser en 1 o 2 dimensions. Pel que fa als blocs, estaran determinats pels fils i aquests podran estar organitzats en 1, 2 o 3 dimensions i és el propi programador qui, a l'hora d'invocar el nucli, decideix quina estructura tindrà la graella i els blocs. La decisió d'una o altre estructura ve donada per el propi problema en que s'estigui treballant i no és més que un ajut per a la organització del problema o algorisme per al programador, ja que el programa s'executarà igual, estigui estructurat de la manera que sigui, mentre s'hi llencin els fils necessaris.



Il·lustració 2. Nucli, blocs i fils d'una GPU (Kirk, 2010)

Un altre inconvenient que un es pot trobar és el del temps de latència de la targeta gràfica. Tot i que se'n pugui fer ús per a la realització de processos de qualsevol

mena, aquesta està pensada i enfocada cap al món dels jocs d'ordinador. És per això que té una latència, que varia en funció de la targeta gràfica emprada. Això significa que si s'està executant un procés que triga més estona que la pròpia latència, la targeta gràfica deixarà de funcionar automàticament i, per definició, tot el procés. És un mecanisme de defensa per a que l'ordinador no es quedi bloquejat en el cas que un procés d'un joc falli en un moment determinat.

És per tot això que no s'ha d'entendre la programació en paral·lel com la solució única als problemes de rendiment que d'ara en endavant sorgeixin en els ordinadors. Aquesta possible solució o millora en el rendiment es tracta d'un ajut per als els processadors actuals i una descàrrega de feina a les CPU's a hores d'ara ja sobrecarregades. En definitiva, s'ha de passar del "processament central" a la CPU a realitzar un "coprocessament" repartit entre la CPU i la GPU, en el que ambdós processadors hagin de coexistir.

4. Programació en CUDA i estructura d'un programa.

Tot i haver fet una introducció i contextualització sobre el perquè de la programació, és també necessari parlar sobre els inicis de tot plegat. Els inicis es remunten a la creació d'una interfície de programació i llenguatge per poder programar en paral·lel. Aquest va ser creat per Apple i va ser l'anomenat OpenCL.

Per a la realització d'aquest projecte, després de valorar dues possibilitats (OpenCL i CUDA) es va decidir optar per CUDA per la gran quantitat de documentació, exemples disponibles i per la facilitat que ofereixen d'incorporar les llibreries en l'entorn del Visual Studio.

OpenCL forma part del sistema operatiu d'Apple, CUDA, (Compute Unified Device Architecture), en canvi, és l'arquitectura de càlcul o API en paral·lel d'NVIDIA (multinacional especialitzada en el desenvolupament de GPU's) i requereix d'un entorn NVIDIA, és a dir, una targeta gràfica d'aquesta companyia per a poder programar o executar programes en aquest llenguatge.

Aquesta eina va ser desenvolupada per AMD, IBM, Intel i NVIDIA, en l'anomenat grup Khronos, fins a convertir-se en un estàndard obert i lliure de drets, l'any 2008. Per tant, es tracta d'un estàndard totalment lliure i obert, sense cap restricció comercial. Això no vol dir que més endavant, diferents empreses decidissin crear les seves pròpies plataformes i eines per a comercialitzar-les, com és el cas de d'NVIDIA amb CUDA, per exemple.

Per tant, CUDA es tracta d'un compilador i d'una extensió de llenguatges com ara C++, per tal de fer possible el diàleg entre la CPU i la GPU. CUDA com a llenguatge ha estat creat a semblança directa de moltes ordres ja existents en C/C++, per tal de fer-ho més senzill per als programadors que s'iniciïn en CUDA, però ja hagin programat abans.

A causa de la gran expansió comercial d'NVIDIA i en ser capdavanters en targetes gràfiques, es dóna la situació que actualment la gran majoria d'ordinadors estan dotats d'una targeta gràfica NVIDIA, condició indispensable per a poder executar un programa CUDA.

El codi font d'un programa que executa CUDA és molt similar al d'un programa escrit en C/C++, però amb unes característiques i estructura que sempre s'aniran repetint, tracti del que tracti el programa.

Per començar, la seva aparença és la d'un programa estàndard, on s'hi declararan les variables que necessitarem. Després, es generaran o s'obtindran totes les dades de partida per a poder realitzar els càlculs.

Un cop es declarin totes les dades necessàries per al càlcul, aquestes es trobaran a la CPU. El primer que s'ha de fer en CUDA és reservar espai a la memòria de la GPU. Això es fa mitjançant l'ordre "cudaMalloc", que és com l'ordre "malloc", però amb extensió CUDA (veure il·lustració 3).

```
cudaMalloc((unsigned char**) &variable_destí, mida_variable);
```

Il·lustració 2. Ordre "cudaMalloc" per a reservar espai a la memòria de la GPU. Elaboració pròpia.

Després s'ha de copiar o traslladar la informació de la CPU a la GPU. Això es realitza amb l'ordre "cudaMemcpy", que té 4 paràmetres: el primer és la variable de destí, el segon la d'origen, el tercer la mida en bytes que ocuparà i, per acabar, el quart indica el sentit de la còpia de dades, és a dir, si va de la GPU a la CPU o a l'inrevés (veure il·lustració 4).

```
cudaMemcpy(variable_destí, variable_origen, tamany_variable, cudaMemcpyHostToDevice);
```

Il·lustració 3. Ordre "cudaMemcpy" per a traslladar informació entre processadors. Elaboració pròpia.

Un cop hem fet això, inicialitzem el nucli. El nucli és el tros de codi que executaran tots els fils que es necessitin de forma paral·lela, ja que és el propi usuari el que n'indica el nombre en executar el nucli. Aquest nucli, per a l'usuari funciona com una funció que simplement es crida des de la funció principal, indicant les variables de la funció, per a que aquesta executi el que està programat (veure il·lustració 5). El nombre de blocs i fils que des d'un inici s'inicialitzin, no pot variar al llarg de l'execució del programa.

```
NomNucli<<< blocs, fils >>>(variables_funció);
```

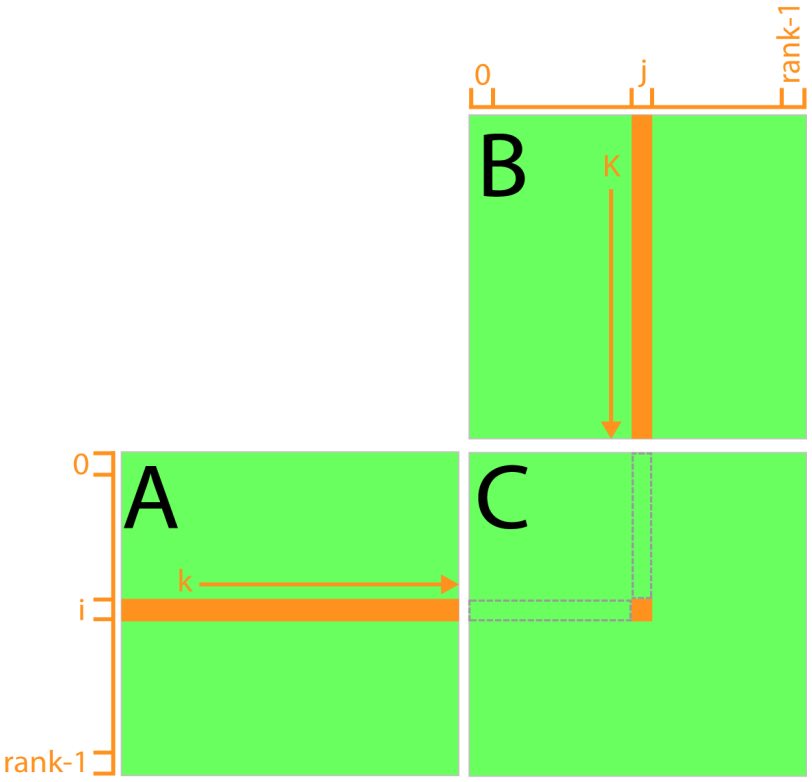
Il·lustració 4. Inicialització del nucli. Elaboració pròpia.

Aquest tros de codi s'ha d'escriure i plantejar com si fos un únic fil el que l'anés a executar, o com si la seva execució hagués de ser de manera seqüencial, però gràcies als índex "threadId" ("thread Identificator"), el nostre nucli s'executarà tantes vegades com li haguem demanat i de manera simultània i sempre sabent quina posició ocupa dins el bloc i la malla entre els centenars o milers de fils que estiguin executant qualsevol operació.

5. Primera aproximació a CUDA. La multiplicació de matrius

Per tal de començar a testejar el funcionament de CUDA i poder comprovar si es disposava del maquinari adient per a treballar-hi es va optar per la multiplicació de matrius.

Es tracta d'un cas senzill pel que fa a programació i a més és perfectament paral·lelitzable, ja que cada valor de la matriu resultant (C) ve donat per una operació que involucra a una fila i a una columna de les dues matrius que multipliquen (A i B), per tant, els resultats de la matriu final provenen d'operacions independents entre si (veure il·lustració 6).



Il·lustració 5. Multiplicació de matrius. (Kirk, 2010)

Com s'ha esmentat, totes les sumes i multiplicacions per a obtenir el resultat d'una multiplicació de matrius, són independents les unes de les altres. L'objectiu d'aquesta prova és paral·lelitzar aquests càlculs, és a dir, que en compte de fer un després de l'altre, es puguin calcular tots, o un gran nombre d'ells, a l'hora i obtenir una matriu amb el resultat correcte i amb els valors ben ordenats.

Això és possible perquè cada fil té un identificador únic (threadIdx), que es tracta d'un índex assignat a cada fil pel propi sistema de CUDA. No s'ha d'entendre com una cosa banal, sinó com la peça clau per a que el paral·lelisme obtingui un resultat coherent al final. No serviria de res realitzar processos a altes velocitats si després tot el resultat resulta ser un desordre de nombres i dades sense sentit.

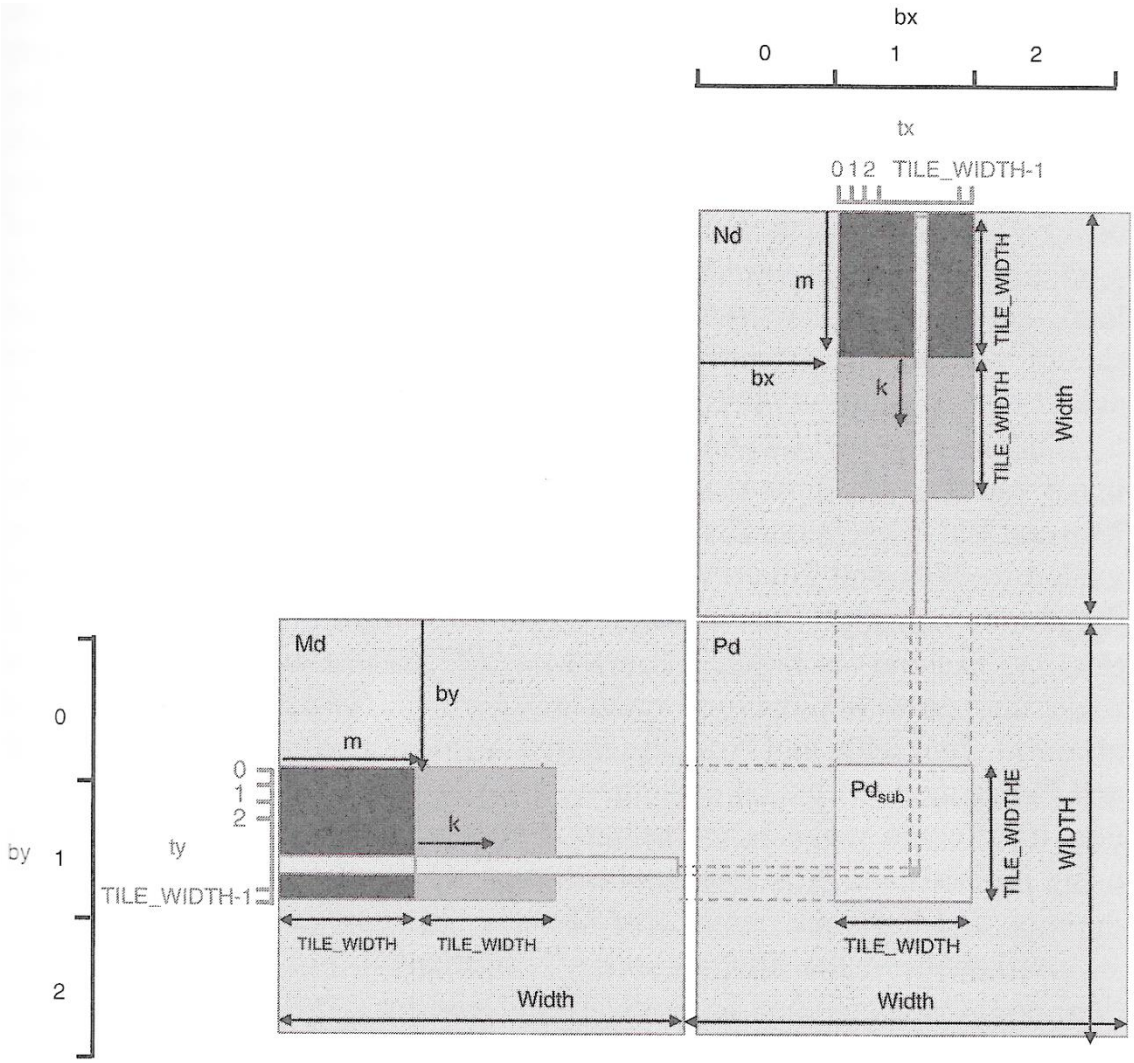
Per tant, l'identificador assigna a cada un dels fils llençats a l'iniciar el nucli, un identificador únic que serveix al fil per a saber quina posició ha d'ocupar en el resultat final, en funció de les operacions que s'hagin dut a terme.

A més, aquest identificador s'ha utilitzat per a realitzar els bucles necessaris per a la multiplicació de matrius. Això significa un canvi de mentalitat, ja que en una multiplicació de matrius en un programa seqüencial, és el propi programador qui assigna les variables que aniran identificant cada posició de cada valor (per exemple, "i" per a les files de la matriu "A", "j" per a les columnes de la matriu "B" i "k" per a cada posició de la matriu "C") (veure il·lustracions 8 i 9).

En aquesta nova situació, és el propi identificador del fil el que assigna a cada fil una fila i columna sobre la que haurà de realitzar la multiplicació i després extreure'n el resultat i escriure'l a la posició que correspon en la matriu resultat.

Durant el procés de programació es va poder comprovar que un dels límits de la programació en paral·lel és que cada bloc només té 512 fils. Tot i això, hi ha mètodes per a poder treballar amb quantitats superiors a 512 fils i de manera paral·lela.

Una d'elles es enviar més d'un bloc. En una malla hi ha un elevat nombre de blocs, així que s'hi s'envien 20 blocs amb 512 fils cada un, estem generant 10240 fils, aconseguint així evitar el límit físic dels fils i ampliant l'abast dels càlculs que es vulguin realitzar.



Il·lustració 6. Multiplicació per rajoles. (Kirk, 2010)

També s'ha optat per una altra solució, que va més enllà i soluciona el problema a grans escales. Aquesta consisteix en la subdivisió de la matriu resultant en diferents rajoles o "tiles" de tal manera que tots els elements d'una rajola són computats per un sol bloc, de tal manera que un fil pot trobar el seu índex "x" dins del bloc i al mateix temps dins la rajola. El que es fa, bàsicament, és subdividir el problema, la matriu en aquest cas, en problemes o matrius més petites per tal de simplificar el global.

Les rajoles sempre hauran de ser més petites que els blocs. La seva dimensió dependrà del tipus de problema plantejat i de la quantitat de subdivisions que es considerin necessàries per a la correcta execució del programa.

5.1. Anàlisi dels resultats de la multiplicació de matrius

Per tal de poder comprovar la efectivitat i el rendiment de la programació en paral·lel, s'han realitzat multiplicacions de matrius quadrades, de diferent mesura i sempre de manera seqüencial i després en paral·lel. D'aquesta manera podem comprovar la millora, pel que a temps respecta, de realitzar operacions en paral·lel o seqüencialment.

Com es pot observar en els resultats (veure Taula 1), quan es tracta de matrius petites (fins a 400 o 500 files/columnes) la diferència és negativa, trigant més en la multiplicació en paral·lel que no pas en sèrie.

Files	Blocs	Fils	Temps sèrie (s)	Temps paral·lel (s)	Ràtio (sèrie/paral·lel)
64	4	16	0.001	0.442	**0.002**
128	4	32	0.008	0.504	**0.015**
192	4	48	0.03	0.52	**0.057**
256	4	64	0.071	0.551	**0.13**
384	8	48	0.241	0.637	**0.38**
512	8	64	0.637	0.827	**0.77**
640	12	64	1.246	0.983	**1.27**
768	12	64	2.108	1.082	**1.95**
896	24	64	3.486	1.434	**2.42**
1024	24	64	9.659	1.68	**5.75**
1536	32	48	36.987	3.219	**11.49**
2048	32	64	92.888	4.855	**19.15**
3072	32	96	308.289	10.843	**28.43**
4096	32	128	808.587	22.681	**301.60**

Taula 1. Comparació de temps entre la multiplicació de matrius en sèrie i en paral·lel. Elaboració pròpia.

Això és a causa del temps que suposa l'enviament d'informació de la CPU a la GPU i viceversa (cudaMemcpy). Quan es tracta de fer el producte de matrius molt grans, en l'enviament es segueix perdent el mateix temps, però és compensat ràpidament pel càlcul massiu.

També s'ha pogut comprovar que com més s'ajusten el nombre de fils llençats al nombre que realment es necessita, el rendiment és molt més elevat i el temps de càlcul menor.

5.2. Implementació pràctica de la multiplicació de matrius

A continuació es mostren dues parts petites del codi que s'ha utilitzat per a la multiplicació de matrius tan en sèrie (il·lustració 6), com en paral·lel (il·lustració 7). El que es mostra és el canvi en els bucles que realitzen la multiplicació, que és on rau l'arrel de la paral·lelització del problema.

```
for( int j=0 ; j<WA ; j++ )
    for( int i=0 ; i<WA ; i++ )
    {
        h_C[WA*j+i]= 0;

        for( int k=0 ; k<WA ; k++ )

            h_C[WA*j+i]+= h_A[WA*j+k]*h_B[WA*k+i];
    }
```

Il·lustració 7. Multiplicació de matrius en sèrie. Elaboració pròpia.

```
for (int i = 0; i < wA; ++i)
{
   float elementA = A[ty * wA + i];
   float elementB = B[i * wB + tx];
   value += elementA * elementB;
}
```

Il·lustració 8. Multiplicació de matrius dins el nucli de la funció paral·lela, on les variables "tx" i "ty" són identificadors únics de cada fil. Elaboració pròpia.

Tal i com es pot veure desapareixen els dos bucles més externs (els que van corrent per les files de la primera matriu i les columnes de la segona), ja que ara no "corre" fila per fila o columna per columna, sinó que es calculen totes a l'hora i només es necessita l'identificador únic de cada fil, aquí escrits com a "tx" i "ty" (veure il·lustració 7), per a poder representar correctament el resultat final.

6. Segona aproximació a CUDA. Correlació de dues fotografies.

En aquest apartat s'ha tractat el primer cas fotogramètric, la correlació de dues fotografies, en paral·lel. Es parteix d'un programa que realitza la correlació en sèrie (obtingut en assignatures anteriors) i a partir d'aquest es van fent modificacions i explorant metodologies per a programar la correlació en paral·lel.

La cerca de correspondències és aquell procés fotogramètric que consisteix en cercar i trobar punts, línies o superfícies equivalents en una o més imatges. Els elements a cercar poden ser patrons radiomètrics (zones equivalents en nivells digitals) o bé elements i objectes reals que es trobin en ambdues imatges.

La correspondència d'imatges es troba implícita en la gran majoria de processos fotogramètrics, ja sigui en l'orientació relativa (cerca i localització de punts homòlegs), en la triangulació aèria, en l'orientació externa o en la generació de models d'altures.

Tanmateix, en funció de com i que s'empra per a la cerca de correspondències, es diferencien diferents tipus de metodologies de cerca:

- **Area-based matching (ABM) o cerca basada en àrees:** Aquest mètode es basa en comparacions a nivell local sobre petites porcions prèviament seleccionades.
- **Feature-based matching (FBM) o cerca basada en elements:** Aquest mètode es basa en la cerca d'elements reals i tangibles del terreny. Entenem com a aquests elements carreteres, ponts, tapes d'enllumenat, llindars de camps,...
- **Relational-based matching (RBM) o cerca basada en relacions:** A diferencia de la resta, aquest es basa en la detecció topològica d'estructures i objectes definits en bases de dades amb descripcions d'objectes[3].

[3] Lerma Garcia, J. L. Fotogrametria moderna: analítica y digital. 1ª ed. Valencia: Universidad Politécnica de Valencia, Servicio de publicación, 2002. ISBN 97-884-9705-21-8.

El fet d'utilitzar un o altre mètode nomes es veurà influenciat pel tipus i qualitat de dades de partida de que disposem i l'enfocament final que es vulgui donar.

Existeixen molts tipus i criteris de cerca de similituds: la covariància, les diferències absolutes de nivells digitals, la correlació o el mètode d'ajust mínim quadràtic. No hi ha cap mètode bo o dolent, si no que cadascun d'ells s'adapta més o menys a les necessitats particulars de les diferents situacions que es poden donar a l'hora de cercar correspondències en fotogrametria.

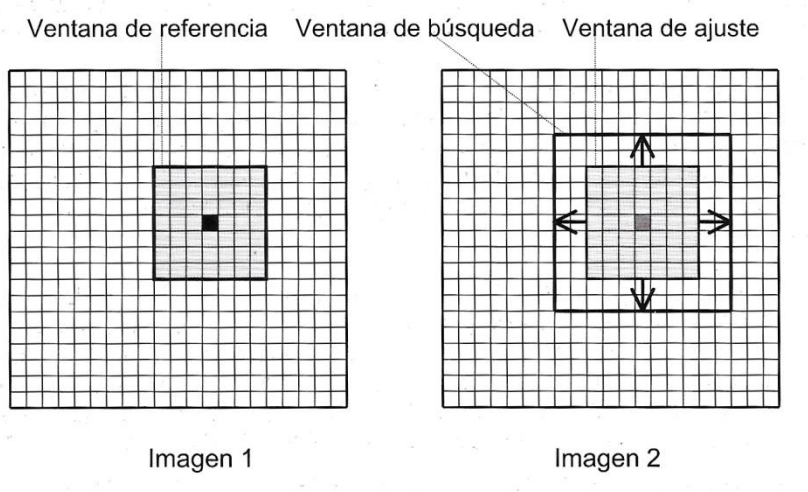
Si bé és cert que qualsevol lloc on hi hagi solapament hauria de ser suficient, s'han d'evitar les zones fosques amb poca senyal radiomètrica i zones repetitives o amb canvis de pendent molt acusats.

En aquest projecte treballarem amb la funció de correlació, comparant sempre un petit bocí d'una de les imatges (al que anomenarem matriu patró), amb una part més gran de l'altra imatge (part que anomenarem matriu de cerca) (Il·lustració 10). La mida, localització del patró i la matriu de cerca o referència tindran una gran influència en el bon resultat final i saber-ho escollir bé agilitzarà o alentirà en gran mesura el procés.

La correlació consta bàsicament de tres càlculs, que són: el valor mig dels píxels de la zona d'estudi, la desviació tipus i la covariància. Cal recordar que la correlació es tracta de la covariància normalitzada per les desviacions tipus del patró i de la cerca.

De tal manera, obtindrem com a resultat final una matriu amb els valors de la correlació entre cerca i patró normalitzats entre -1 i 1. És obvi que, com més a prop de la unitat es trobi el valor de la correlació, més semblança hi haurà entre les dues porcions comparades. En contraposició, quan el valor sigui pròxim a 0 significarà que no hi ha cap mena de similitud. El valor negatiu fa referència a una correlació negativa, que succeeix quan es treballa simultàniament amb un positiu i un negatiu entre els dos fotogrames.

En aquest tipus de correlació s'assumeix que en dues fotografies preses consecutivament, hi haurà una sèrie de blocs de píxels (patrons) que es repetiran en ambdues fotografies, és a dir, que tindran valors radiomètrics semblants. La cerca d'aquest patró es pot fer sobre tota la imatge homòloga o només sobre certes zones prèviament definides, tot depèn de la informació de partida que es disposi.



Il·lustració 9. Esquema de la cerca de correspondències. Lerma (2002).

6.1. Paral·lelització del procés

En aquesta primera aproximació, es va agafar una matriu patró (31 x 31 píxels), de la qual es busca la correlació per a una imatge relativament gran de cerca de mides 255 x 255 píxels. D'aquesta manera, el que es fa és calcular la correlació per bocins de 31 x 31 en la imatge més gran, obtenint un resultat d'una imatge de 240 x 240 on cada valor correspon a la correlació normalitzada entre -1 i 1 entre el patró i la matriu de cerca en aquell punt.

En la primera aproximació al procés de transformació del codi sèrie a paral·lel s'ha escollit expressament un exemple en que a la funció del nucli només li passem el patró i la cerca. D'aquesta manera abordem l'essència del problema que és la cerca d'una correspondència per a un sol punt. En estadis més avançats treballarem amb les imatges senceres.

Per començar, es va partir d'un programa que realitzés la correlació en sèrie, ja que a partir d'aquest s'iniciaria el procés de paral·lelitzar.

La hipòtesi inicial era que la millor funcionalitat seria la que executés tots els càlculs dins el nucli de la GPU, deixant la CPU només per a llegir i enviar informació en els dos sentits.

Un cop la informació es troba al nucli de la GPU, cada fil de la funció nucli s'encarrega d'un càlcul complet de correlació, és a dir, cada fil s'encarrega de comparar una porció de 31 x 31 píxels de la matriu de cerca amb la matriu 31 x 31 píxels del patró.

Els resultats de l'exemple anterior, pel que fa a temps i rendiment, han estat molt bons i superiors al que s'esperava, ja que partíem d'un temps pròxim als 4 s en els càlculs en sèrie i s'ha arribat a obtenir temps pròxims als 0.8 s, el que suposa una millora d'entre 4 i 5 vegades respecte a la velocitat de càlcul.

6.2. Primera metodologia. Càlcul píxel a píxel

Un cop aconseguida la correlació per a un cas molt reduït, es va procedir a extrapolar-lo a un cas real, amb dues imatges d'un vol fotogramètric, d'una mida de 7500 x 11500 píxels, per a poder comprovar d'aquesta manera el funcionament i rendiment en condicions reals de la correlació en CUDA.

En aquest cas el procediment ha estat el següent:

S'han obert, llegit i emmagatzemat les imatges amb les que es treballa. Un cop fet això, reservem memòria per a la totalitat de les dues imatges a la GPU, això significa 250 Mb per cada imatge.

Un cop tenim l'espai reservat, hi enviem les dades, les imatges. S'ha optat per enviar massivament les dues imatges de cop a la GPU perquè el que fa perdre temps i rendiment, com ja s'ha esmentat amb antelació, no són pas els càlculs, sinó el trànsit d'informació que pot existir entre la CPU i la GPU. D'aquesta manera, només es fa un enviament de les dades inicials, amb les quals ja es poden calcular tantes correlacions com es necessitin.

Quan les dades ja són a la GPU, fem els retalls de patró i de cerca de la mida que interessi i es calcula la correlació en la zona desitjada.

La primera metodologia per al càlcul de la correlació consisteix en definir un punt "A" de la fotografia 1 al voltant del qual es crea un patró. Aquest patró es tracta d'una matriu de 31 x 31 que després serà correlada sobre una matriu de cerca de 255 x 255 píxels a la fotografia 2. De tal manera que cada fil farà una de les 224 correlacions per a aquella zona en concret i escriurà un valor de correlació en la matriu resultant.

Una vegada obtinguda la correlació, aquesta s'envia novament a la CPU, on s'escriu en un arxiu en format binari per a la seva visualització posterior.

Com a resultat, s'ha obtingut un temps màxim de 0.65 s.

6.3. Segona metodologia. Càlcul per zones

En aquest cas, el procediment és semblant al de la primera metodologia. Les fotografies son obertes, llegides i emmagatzemades de la mateixa manera que en el cas anterior.

El que aquí canvia és que en comptes de treballar amb un sol punt (i el seu respectiu patró), agafem una zona de 100 x 100 punts al voltant d'un punt, no cal que sigui el mateix. L'objectiu d'aquest procediment és que cada fil calculi una correlació per a un punt en concret, amb tots els càlculs que això suposa, en lloc d'utilitzar tots els fils per al càlcul de la correlació en un sol punt.

El que s'està fent és que, de forma individual, cada fil treballi molt més i, per tant, que el procés trigui molt més. Tot i això, el que s'aconsegueix és paral·lelitzar a "gran escala". Tanmateix, això servirà per a veure el rendiment d'aplicar el paral·lelisme a càlculs massius de dades (recordem que estem parlant de 10.000 punts).

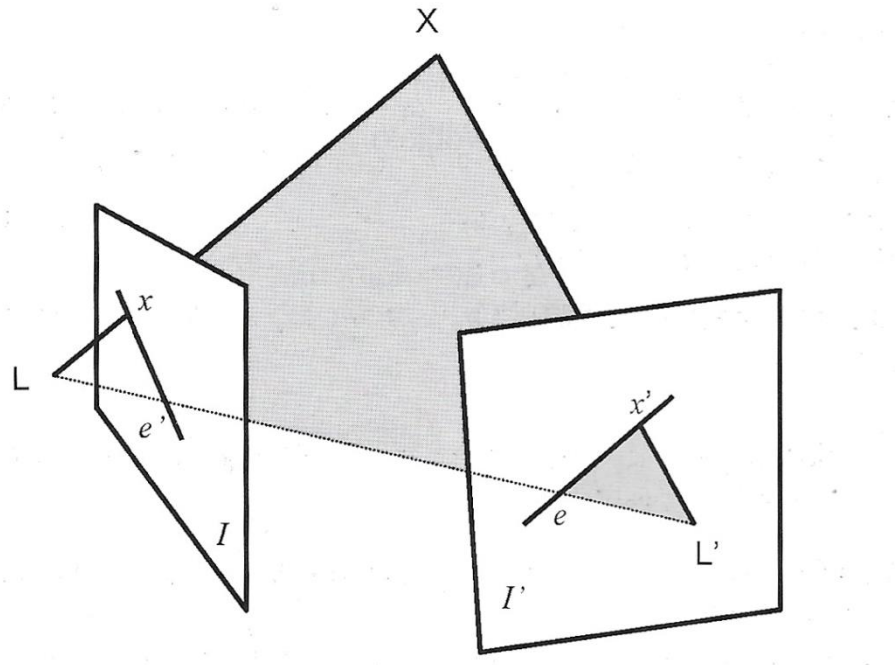
En aquesta segona metodologia, tenint en compte que treballem amb un nombre molt gran de punts, el temps màxim ha estat de 367.28 s. Si dividim per el nombre de punts total dóna com a resultat 0.03 s per punt.

6.4. Tercera metodologia. Imatges epipolars

Per tal de comprovar la diferència de rendiment quan ja es tenen més dades de partida, en el moment que la informació d'inici és suficient com per a acotar la zona de cerca a petites matrius, es va voler posar a prova el càlcul en paral·lel de la correlació. Aquest cop amb un parell d'imatges epipolaritzades.

Diem que dues imatges tenen una geometria epipolar quan un punt qualsevol de la imatge esquerra troba el seu punt homòleg traçant una línia horitzontal (línia epipolar) (veure il·lustració 9). Són, per tant, unes imatges que han patit una sèrie de translacions fins a quedar orientades en el mateix sentit vertical i l'únic desfasament que existeix és en el sentit horitzontal.

El que això suposa és que en comptes d'haver de cercar el punt homòleg en un pla bidimensional (tota la matriu de cerca) ara tot es redueix a un espai unidimensional. Si les fotografies han estat ben normalitzades, els punts homòlegs no presenten paral·laxi i, per tant, coincideixen fila a fila i la correlació o correspondència es produeix sobre un sol eix, l'horitzontal.



Il·lustració 10. Geometria epipolar. Lerma (2002)

Per a aquest cas, en conseqüència, la matriu patró passa de tenir una mida de 31 x 31 píxels a 1 x 21 píxels i la matriu de cerca passa de 256 x 256 a 3 x 301. Això implica passar de treballar amb 961 píxels en la matriu patró i 65.536 píxels en la matriu de cerca, a treballar-ne amb sols 21 i 903 píxels respectivament. Esperem d'aquesta manera que cada fil pugui processar un sol punt de correspondència i que ho faci molt ràpidament. Aquest model està pensat per calcular models d'altures on es requereix literalment milions de correspondències.

6.5. Comparació dels resultats

El que s'ha pogut comprovar i constatar un cop contrastats amb els tres mètodes és que no hi ha un que sigui millor que l'altre, sinó que s'ha de saber en quin moment convé emprar-los.

En el cas que ens trobéssim davant d'una situació en el que necessitéssim unes matrius de cerca més grans, degut a una desconeixença total de la orientació d'ambdós fotogrames i, per tant sense saber on cercar correspondències, el càlcul píxel a píxel respon molt millor pel que fa a rendiment.

D'altra banda, si la situació donés peu a una possible fragmentació en matrius de cerca més petita, però necessitéssim molts més càlculs de correlacions, aleshores és el càlcul per zones el que aconsegueix millors resultats.

Tal i com es pot observar a la taula 2, el rendiment és superior com més massiu és el càlcul de dades. En el cas de les zones, on s'estaven calculant correlacions per a 10.000 punts, en paral·lel ha trigat 6 minuts i escaig, mentre que en sèrie el cost de temps ha estat d'una hora. Això no fa més que reafirmar el fet que el paral·lelisme funciona igual o millor per a rendiments més llargs. A més resulta espectacular pel fet de realitzar coses quasi instantàniament, en el cas de càlculs massius de dades, la rendibilitat supera totes les expectatives.

Val a dir que aquest cas era un test bastant extrem, ja que normalment no buscarem 10.000 punts per a cercar, sinó uns pocs centenars. Tot i diferir sobre el que seria un cas real, val la pena com a prova del rendiment que es pot obtenir.

Pel que respecte a la última correlació provada, amb imatges epipolars com a dades de partida, els resultats indiquen que s'ha aconseguit realitzar el mateix procés, però deu vegades més ràpid (veure taula 2).

Simulació d'una orientació relativa: No sabem a priori on és la correspondència. Ens cal grans matrius de cerca. Fem dos enfocaments:

	Temps sèrie	Temps paral·lel	Relació
Càlcul píxel a píxel	4.8 s	0.65 s	**7.38**
Càlcul per zones	3320.48 s	367.28 s	**9.04**

Simulació d'una extracció de model d'altures: sabem aproximadament on és la correspondència. Ens és suficient petites matrius de cerca sobre la línia epipolar.

	Temps sèrie	Temps paral·lel	Relació
Càlcul epipolars	28.86 s	2.73 s	**10.57**

Taula 2. Comparació de resultats entre el càlcul de correlacions en sèrie i en paral·lel**.**

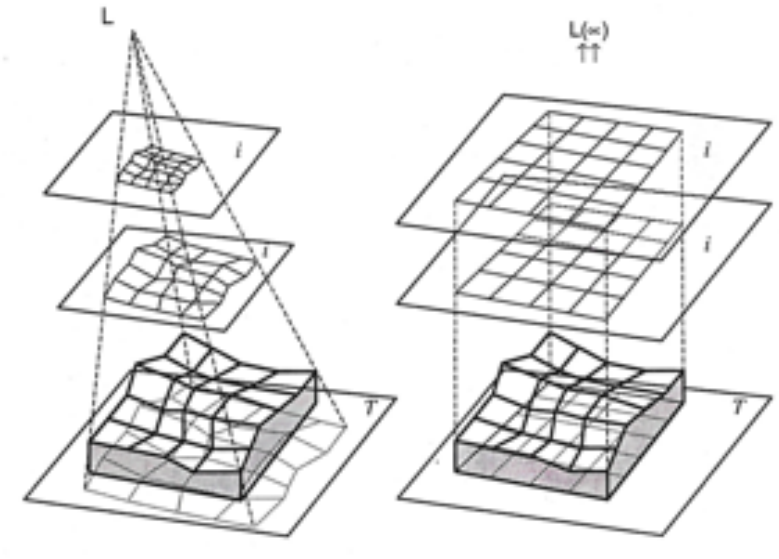
El fet que les dades de partida tinguin tanta influència en el temps i rendiment total que fa que no es pugui menystenir de cap de les maneres. Si bé ja partíem de la base que, com més preparades fossin aquestes dades de partida, més es redueix la feina a l'hora de cercar correspondències i, per tant més ràpid funciona un procés, el que ha sorprès és el fet que la ràtio augmenti d'aquesta manera.

Aquest fet és degut a que s'ha trobat la metodologia de càlcul i de programació que millor encaixa per a la programació en paral·lel i en aquest cas concret, obtenint així el major rendiment possible.

Com a conclusió general de les tres proves fetes, es pot afirmar que no existeix una única manera de paral·lelitzar un procés, sinó que hi ha diferents graus de paral·lelisme, diferents escales i en diferent mesura. No hi ha una manera bona o dolenta de fer-ho, sinó que depèn del nombre i mida de les dades que s'estiguin emprant, de les característiques particulars dels càlculs del procés i de la rendibilitat que es vulgui obtenir. És a dir, paral·lelitzar més o a gran escala no significa millor rendiment final.

7. Tercera aproximació a CUDA. Ortoprojecció fotogramètrica

S'entén com a ortoprojectar o a rectificar diferencialment, aquell procés que consisteix en transformar una fotografia en perspectiva, que fuga, en una projecció ortogonal, realitzant una deformació de la imatge, píxel a píxel, fins a aconseguir que cada objecte de la imatge estigui representat de tal manera que la fotografia queda reproduïda com si aquesta hagués estat presa sobre la vertical de tots els punts que en ella hi surten, amb la focal situada en l'infinit (Il·lustració 114) .



Il·lustració 11. Imatge en perspectiva respecte a una ortofotografia. Lerma (2002)

D'aquesta manera, s'elimina el desplaçament imatge produït per la inclinació de la càmera fotogràfica i per l'efecte orogràfic i, per tant, la imatge es converteix en un instrument apte per a la mesura.

Si en la ortofotografia s'hi col·loquen, a més a més, elements i simbologies, com poden ser corbes de nivell, toponímia o altres elements d'interès, aleshores parlem d'ortofotomapes.

Cal recordar que la ortofotografia s'ha convertit en els darrers anys en una eina molt útil i ha acabat essent molt necessària en àmbits com la cartografia i la fotogrametria. Aquesta situació s'ha donat degut a que el seu cost, comparat amb la qualitat final, és baix i a més, cal sumar-hi la facilitat pel que fa a la interpretació a nivell de l'usuari.

A més, es tracta d'una eina que permet la obtenció d'informació georreferenciada d'una manera molt més ràpida que no pas seguint els mètodes de la cartografia tradicional. Aquesta qüestió és fonamental per a aquelles situacions on el temps és la variable principal per a la solució del problema.

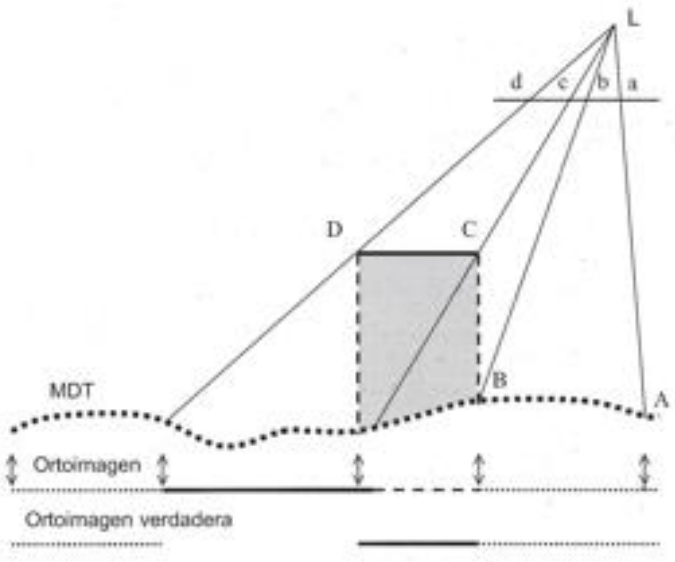
7.1. La producció d'ortofotografies

Per a obtenir ortofotografies o ortofotomapes necessitem una informació prèvia. Aquestes dades són les imatges en si, orientades interna i externament i un model d'altures.

Conèixer les dades d'orientació de la càmera en el moment en que es va realitzar la fotografia així com la seva coordenada terreny (la coordenada de la focal en aquell instant), és fonamental per a poder traslladar les coordenades fotograma a les coordenades terreny gràcies a les equacions de col·linealitat.

De la mateixa manera i en el mateix ordre d'importància, cal conèixer els angles inercials d'orientació, els quals són coneguts gràcies a la IMU (Unitat de Mesura Inercial, de l'anglès *Inertial Measurement Unit)*.

Tota aquesta informació és fonamental, ja que si no, no hi hauria ortofotografia, però el que aconsegueix que una ortofotografia sigui de més o menys qualitat i precisió és un bon model d'altures.



Il·lustració 12. Elements d'una ortofotografía. Lerma (2002)

L'elecció i combinació del model o models d'altures és el que, en definitiva, definirà la qualitat final de l'ortofotografia. És obvi que necessitem un Model Digital del Terreny (MDT), com a mínim. Recordem que un MDT és un model que ens donarà informació sobre la cota, ja que es tracta d'un conjunt de dades que extrapolen la coordenada Z. És a dir que per cada coordenada X, Y, existeix un únic valor de Z. Un bon model d'altures està format per una malla densa de punts, regular o irregular, que defineixen l'orografia del terreny.

Si bé no és l'únic model digital disponible i se'n poden emprar d'altres com els Models Digitals d'Edificis (MDEd) o Models Digitals de Superfície (MDS)[4].

Un MDEd representa en tres dimensions tots aquells objectes i detalls que no són part de la orografia natural, com per exemple edificacions, ponts, viaductes, etc.

Pel que fa a un MDS és aquell que envolta i modela per complet la superfície tridimensional d'una zona, hi hagi els elements que hi hagi. S'entén normalment com el model digital més complet i, sovint aquest s'obté de la conjunció d'un MDT amb un o més MDEd.

Amb aquestes dades de partida ja podem generar les ortofotografies. El mètode clàssic, que és el que aquí s'ha utilitzat és el que aplica les equacions de col·linealitat. Aquestes equacions usen com a coordenades terreny els centres que s'han obtingut un cop projectada una retícula sobre el model digital. D'aquesta manera, el valor radiomètric de la imatge final, s'extreu directament de la inicial. La producció de l'ortofotografia segueix un procés repetitiu i paral·lelitzable fins que s'han calculat els nous valors de tots els píxels.

[4] Lerma Garcia, J. L. Fotogrametria moderna: analítica y digital. 1ª ed. Valencia: Universidad Politécnica de Valencia, Servicio de publicación, 2002. ISBN 97-884-9705-21-8.

A més de les equacions de col·linealitat, s'ha utilitzat un interpolador, ja que la malla que s'empra per a l'MDT és regular i els punts amb cota no tenen perquè coincidir amb el centre dels píxels. D'aquesta manera s'han donat valors reals de cota als centres dels píxels.

Tanmateix, ens podem preguntar si, un cop el procés estigui paral·lelitzat i si la reducció de temps és important i suficient, per què no calcular i representar les ortofotografies a temps real? Imaginem que tenim un visualitzador, on es poden veure imatges aèries o de satèl·lits i que a mesura que nosaltres visualitzem o simplement ens col·loquem sobre la zona geogràfica que més ens interessi, el que veiem en pantalla aparegui automàtica i instantàniament ortoprojectat.

Aquest procés també s'ha provat amb dues metodologies diferents per tal de comprovar el seu rendiment en diferents condicions i situacions, pel que fa a les dades de partida (mesura de les imatges, nombre de correlacions i d'altres).

7.2. Primera metodologia. Càlcul per píxels

Aquest és el mètode que resulta més lògic a l'hora de pensar en paral·lelitzar el procés de la correlació. Senzillament s'ha estructurat el nucli del codi de tal manera que cada fil calcula un nivell digital d'un dels píxels de la ortofotografia resultant.

En aquest cas s'ha procedit a treballar amb una imatge molt més gran, de 4000 x 3000 píxels cada una. Els resultats que es van obtenir van ser passar de 9 s quan era calculat en sèrie a tan sols 1.51 s en paral·lel.

La millora de temps és molt mes que substancial i tenint en compte que s'estava treballant amb imatges reals, de mida real, es pot afirmar que el test va resultar tot un èxit en el primer intent.

A més, si aquests 7 s de diferència els extrapolem a una situació on no només estiguem ortoprojectant una imatge, sinó una sèrie o un vol fotogramètric sencer, la quantia de temps que és guanya és molt gran.

7.3 Segona metodologia. Càlcul per files

En aquest cas, hem seguit el procediment en el que cada fil calculés una fila sencera de la imatge resultant. S'ha partit de la hipòtesi de que si s'augmenta el nombre de càlculs que cada fil ha de realitzar individualment, el rendiment final augmenta.

Això es fonamenta sobre la base que es necessitaran menys fils i a que és possible que al paral·lelitzar a una escala tan petita (per cada píxel) s'estigui infrautilitzant els fils, ja que tot i estar molt més limitats a l'hora dels càlculs comparat amb la CPU, encara se'ls pot carregar de més feina.

Amb aquesta idea en ment, s'ha replantejat de nou el nucli que s'executa a la GPU. A nivell físic, la graella de blocs ja no es tracta d'una graella amb dues dimensions, sinó que passa a tenir-ne una, degut a que al treballar per files, estem eliminant la coordenada horitzontal i només ens interessa la vertical.

Els resultats obtinguts en ambdues metodologies es poden veure en el següent apartat.

7.3. Comparació de resultats

Quan es va començar a paral·lelitzar el procés de generació d'una ortofotografia, hi havia molta esperança a arribar a uns resultats clarament positius, ja que es tracta d'un procés que consta d'una sèrie de processos que, al mateix temps, contenen càlculs altament paral·lelitzables.

	Temps sèrie	**Temps paral·lel**	**Ràtio**
Càlcul per píxels	9.12 s	1.51 s	**6.04**
Càlcul per files	9.12 s	1.62 s	**5.63**

Taula 3. Comparació de resultats entre el càlcul de la orotofotografia en sèrie i en paral·lel.

Altrament, tal i com ha succeït en el cas de la correlació, hem pogut constatar que, si bé en sèrie moltes vegades hi ha una sola opció i a l'hora de programar hom està limitat i restringit a seguir una certa metodologia, en el cas de la programació en paral·lel, CUDA en aquest cas, hi ha molt més marge a la creativitat, ja que és el propi programador el que decideix quin nivell de paral·lelisme hi haurà en l'aplicació.

En el que ens ocupa, teníem la hipòtesi que, si bé els resultats de l'ortofotografia emprant un càlcul píxel a píxel, és a dir, on cada fil s'ocupava sols d'un píxel, eren bons, vam decidir que un fil podia calcular tota una fila i així el rendiment final milloraria.

Tal i com es pot veure a la taula 3 no va resultar així. Tot i parlar de poc més d'una dècima de segon, el càlcul de l'ortofotografia píxel a píxel és més ràpid que fila a fila. Això pot ser degut a que els càlculs que ha de realitzar cada fil són masses i probablement no s'estigui utilitzant en excés aquest recurs.

8. Conclusions

Després d'haver implementat algunes funcions en paral·lel, podria dir que les premisses inicials que em feia respecte al potencial de les targetes gràfiques s'han acomplert satisfactòriament. Tal i com s'ha vist en els casos mostrats al llarg del treball els resultats són d'una reducció de temps de càlcul molt notable.

Tot i això no s'ha de comprendre a CUDA i a la programació en paral·lel en general com a la solució única i definitiva, degut a la seva potència o als bons resultats aconseguits. S'ha d'entendre sempre com una eina que acompanya a la programació tradicional, ja que la necessita. La necessita perquè la GPU té una programabilitat molt limitada i tot el que sigui obtenció de dades i mostra de resultats després, s'ha de fer mitjançant la CPU. És per això que podem dir que es tracta d'una unió de forces entre dues peces claus en el funcionament d'un ordinador i ara, de molts programes.

La valoració en positiu dels resultats també ho és pel fet que s'ha emprat una targeta gràfica bastant simple, d'un ordinador portàtil. D'aquesta manera, si s'hagués disposat d'un hardware encara més potent (avui dia trobem targetes gràfiques amb 10 vegades més nuclis), els resultats probablement haurien estat encara millors.

Pel que fa al llenguatge propi de CUDA, tot i que per lèxic o sintaxi no difereixi en gran mesura de C/C++, la programació en paral·lel és molt diferent a la programació tradicional, tant en forma com en plantejament. S'ha de comprendre que el que estem executant, serà llençat milers de vegades a l'hora. Això implica canviar per complet la manera de pensar i plantejar els algorismes quan es programa.

A més, el propi concepte de paral·lelisme dona joc a una gran llibertat a l'hora de programar, per tant per a cada problema que s'abordi, s'hauran de provar moltes metodologies i entorns fins a trobar el més adequat per a cada situació en concret.

Els nous horitzons que aquest nou paradigma obre són molt grans, ja que si bé en aquest projecte el que s'ha fet ha sigut millorar una sèrie de processos ja existents, encara que s'hagi fet per a poder comparar i demostrar que és molt més ràpid, d'aquí a pocs anys, de ben segur, la majoria de programes estaran basats en aquesta tècnica de programació, perquè cal recordar que s'està arribant al límit de hardware i, per tant, només queda reinventar-se en el software.

9. Bibliografia

Buill F, Núñez M. A., Rodríguez J. J. *Fotogrametría analítica*: 1ª edición. Barcelona: Edicions UPC, 2003. ISBN: 84-8301-671-0.

Ceballos F. J. *C/C++ Curso de programación*. 3ª edición. Madrid: RA-MA Editorial, 2007. ISBN: 978-84-7897-762-8.

Kirk, D. B.; Wen-mei, W. H. *Programming massively parallel processors*. Burlington: Morgan Kauffmann, 2010. ISBN 978-0-12-381472-2.

Lerma Garcia, J. L. *Fotogrametria moderna: analítica y digital.* 1ª ed. Valencia: Universidad Politécnica de Valencia, Servicio de publicación, 2002. ISBN 97-884-9705-21-8.

Parsons, A. *Professional Visual Studio 2005*. Indianapolis: Willey Publications, 2007. ISBN 978-0-76-459846-3.

Sanders, J, Kandrot E. *CUDA by Example. An introduction to General-Purpose GPU Programming*. Upper Saddle River, NJ. Pearson Education, Inc. 2010. ISBN 978-0-13-138768-3.

Wilt, N. The CUDA Handbook. *A comprehensive guige to GPU Programming*. Upper Saddle River, NJ. Pearson Education, Inc. 2011. ISBN: 978-0-321-80946-9.

Zaius John. Confessions of a Speed Junkie. Seattle. 2009. (Consulta, 25 de març de 2015). Disponible a: http://gpgpu-computing4.blogspot.com.es/

10. Agraïments

A l'Albert Prades, tutor i company d'aquest viatge en la descoberta d'un món completament desconegut. La seva passió i implicació per a aquest tema van aconseguit captar tot el meu interès i dedicació. No em quedo curt al dir que sense ell, aquest projecte no hagués estat possible.

A la Paula Muro, que gràcies al suport tècnic i moral aquest projecte va poder seguir quan es trobava en les hores més baixes. Als meus pares, que sempre m'han animat a seguir fent allò que m'agrada i m'han donat aquell consell mes necessari quan més calia.

A en Cesc Masdeu, company i amic de molts viatges i aventures aquests últims dotze mesos. I és gràcies a aquests viatges que he pogut pensar en fred i veure les coses d'una altra manera, per així seguir endavant. Excels.

I per últim a tots aquells amics i amigues que sempre m'han animat i s'han interessat per mi i per aquest projecte, a en Gabri, a la Xènia, a la Mònica i a la Marta.

A tots vosaltres, i d'altres que em deixo, moltes gràcies.