Aránzazu Miguélez
Emma Juanico

INFORME PRÀCTICA Kmeans OpenACC

Introducció

En aquesta pràctica, hem portat a terme la paral·lelització de l'algoritme k-means utilitzant OpenACC per millorar el seu rendiment en GPU. El nostre objectiu ha estat, coneixent les seccions del codi que consumeixen més temps de computació, modificar-les per aprofitar les capacitats de processament en paral·lel de les GPU. A través d'aquesta paral·lelització, esperem aconseguir una reducció significativa dels temps d'execució i una millora del rendiment global de l'algoritme.

Paral·lelització del codi

Per tal de poder paral·lelitzar el codi a través d'OpenACC hem hagut de fer les següents modificacions.

Primer de tot, com ja es va veure a la pràctica anterior, la funció kmeans és la que consumeix un 99% del temps total. La funció kmeans té un bucle de *num_pixels* iteracions i a cada iteració fa una crida a la funció *find_closest_centroid*. Aquesta funció si l'analitzem podem observar que itera sobre tots els centroides per calcular la distància entre el píxel donat i cadascun dels centroides, és a dir, també està construïda amb un bucle for de *k* iteracions, així doncs, el bucle principal s'executarà *num_pixels* x *k* vegades, fet que fa augmentar la complexitat de l'algorisme. Per tant, aquesta part del codi és el bootleneck i s'han de dedicar els esforços per millorar el rendiment a aquesta zona del codi.

La primera modificació que hem realitzat, ha estat evitar la crida de *find_closest_centroid* dins la funció de kmeans perquè així la podem modificar i paral·lelitzar, és a dir, la funcionalitat de la funció (calcular la distància euclidiana al quadrat entre cada píxel i tots els centroides i després triar el centroide més proper en funció d'aquesta distància) l'hem escrit directament dins de la funció kmeans per també evitar crides externes.

Seguidament, hem utilitzat vectors auxiliars (red, green, blue i points) per acumular les sumes dels valors de píxels assignats a cada centroide, és a dir, s'utilitzen per acumular els valors dels components RGB i el nombre de punts assignats a cada centroide durant l'assignació de píxels a clusters. Per fer això hem reservat la memòria necessària per a cada array usant la funció malloc.

Un cop inicialitzats els centroides aleatòriament, primer de tot, el que hem fet ha estat utilitzar un pragma acc data copy:

- **#pragma acc data copy(...):** Aquest pragma s'utilitza per especificar les operacions de transferència de dades entre la memòria del host (CPU) i la memòria del dispositiu (GPU). Copy indica que cal copiar les dades des del host al dispositiu abans d'executar la regió paral·lela i viceversa després d'executar-la. És important gestionar adequadament la transferència de dades entre la CPU i la GPU per garantir que les dades estiguin disponibles al dispositiu quan es necessitin i es copiïn de tornada a la CPU quan es completin els càlculs en paral·lel.

Amb les dades necessàries ja transmeses, hem reiniciat totes les propietats dels centroides i els vectors auxiliars a 0. Això ho hem fet amb un bucle for paral·lelitzat amb el pragma:

- **#pragma acc parallel loop present(...)**: Aquest pragma s'utilitza per indicar que el bucle següent s'ha d'executar en paral·lel al dispositiu. *present* especifica quines variables han de ser presents al dispositiu abans d'executar el bucle. Això assegura que totes les variables necessàries estiguin disponibles al dispositiu abans que comenci l'execució en paral·lel.

Ara podem veure que en el codi seqüencial, es feia la crida de *find_closest_centroid* , però com hem dit, per trobar el cluster més proper ho hem fet directament dins la funció kmeans, i ho hem pogut paral·lelitzar amb la mateixa directiva utilitzada prèviament, pragma acc parallel loop present.

Finalment, un cop trobat el centroide més proper de cada píxel simplement hem fet les assignacions corresponents perquè els valors estiguin a les variables inicials.

L'última part del codi no l'hem modificada, tot i això al final del codi hem hagut d'alliberar l'espai de memòria que havíem reservat.

En usar aquests pragmes i modificacions, el compilador s'està instruint a generar codi paral·lel que aprofiti les capacitats de processament en paral·lel de la GPU per accelerar l'execució del codi. Això pot conduir a millores significatives en el rendiment, a continuació ho veurem.

Aquests canvis permeten que els bucles siguin executats en paral·lel al dispositiu, aprofitant així la capacitat de còmput de la GPU per accelerar el processament. Tot i això, és important tenir en compte que el rendiment i l'eficiència de la paral·lelització poden dependre de diversos factors, com la mida del conjunt de dades, l'arquitectura de la GPU i la implementació específica del compilador OpenACC.

Aránzazu Miguélez
Emma Juanico

GeForce RTX 3080 i GeForce RTX 1080Ti

En aquesta pràctica se'ns ha demanat l'execució del codi generat amb Open ACC amb dues GPU diferents. Per tant, abans d'analitzar els càlculs de temps, hem investigat sobre les característiques de cadascuna.

La GeForce RTX 3080, basada en l'arquitectura NVIDIA Ampere, representa un avenç significatiu, oferint una combinació de rendiment i eficiència energètica. Amb fins a 8704 nuclis CUDA, aquesta és capaç de gestionar una enorme quantitat de càlcul paral·lel, el que permet accelerar significativament el processament de tasques exigents. A més, amb una freqüència de GPU que pot arribar fins a 1,71 GHz i una memòria GDDR6X de fins a 10 GB amb una velocitat de fins a 19 Gbps, la GeForce RTX 3080 està perfectament equipada per manipular grans conjunts de dades amb rapidesa i eficàcia.

D'altra banda, la GeForce RTX 1080Ti, basada en l'arquitectura NVIDIA Pascal, continua sent una opció potent, tot i ser una generació anterior. Amb 3584 nuclis CUDA i una freqüència de GPU base de fins a 1,48 GHz, aquesta encara ofereix un rendiment notable en comparació amb les opcions més recents del mercat. Tot i que la seva memòria GDDR5X de 11 GB potser no és tan ràpida com la de la GeForce RTX 3080, encara proporciona un ample de banda suficient per a moltes aplicacions gràfiques i de càlcul.

Tant la GeForce RTX 3080 com la GeForce RTX 1080Ti podrien abordar aquest codi de manera semblant en termes d'acceleració de processament amb OpenACC, però podrien presentar algunes diferències degut a les seves especificacions de maquinari.

- La GeForce RTX 3080 pot tenir més nuclis CUDA que la GeForce RTX 1080Ti. Això significa que pot gestionar més threads simultàniament, el que pot accelerar el processament paral·lel.
- La GeForce RTX 3080 pot tenir una freqüència de GPU més alta que la GeForce RTX 1080Ti, el que pot millorar el rendiment en operacions de càlcul.
- La GeForce RTX 3080 pot tenir una memòria de GPU més ràpida i més gran que la GeForce RTX 1080Ti. Això pot ser beneficiós per a aplicacions que manipulen conjunts de dades més grans, ja que poden evitar transferències freqüents entre la memòria de la CPU i la GPU.
- La GeForce RTX 3080 pot presentar optimitzacions específiques del fabricant o de l'accelerador que millorin el rendiment en tasques específiques, com ara les optimitzacions per a càlculs de matrius o operacions aritmètiques.

Tot i que les dues podrien ser capaces d'accelerar el processament del codi Kmeans amb OpenACC, és possible que la GeForce RTX 3080 ofereixi un rendiment superior gràcies a la seva arquitectura més nova i a les seves especificacions més potents. No obstant això, el rendiment real dependrà de diversos factors, com ara la implementació específica del codi, la configuració del sistema i les optimitzacions del compilador. Fent el profiling i calculant els speedup ho podrem comprovar.

Càlcul de temps

TEMPS DE REFERÈNCIA (codi seqüencial modificat)						
Versió	k = 2	k = 4	k = 8	k = 16	k = 32	k = 64
Seq + Ofast	0,32	1,13	4,30	7,07	15,13	118,42
Iteracions	9	22	51	47	49	196
Checksum value	6405336	12689895	22329779	42559141	73490886	124269810

versió	Temps de referència					
OpenACC	AOLIN					
num threads	k = 2	k = 4	k = 8	k = 16	k = 32	k = 64
GeForce RTX 3080	0,83	1,56	3,18	2,91	3,69	11,42
GeForce RTX 1080Ti	1,19	2,15	4,34	4,61	4,25	15,76
Iteracions	9	22	51	47	49	196
Checksum Value	6405336	12689895	22329779	42559141	73490886	124269810

Speedup						
num threads	k = 2	k = 4	k = 8	k = 16	k = 32	k = 64
GeForce RTX 3080	0,38	0,72	1,35	2,43	4,1	10,37

GeForce RTX 1080Ti	0,27	0,53	0,99	1,53	3,56	7,51

Conclusions

Els resultats dels speedups mostren que la versió paral·lela del codi amb OpenACC ofereixen un rendiment millor en comparació a la versió seqüencial modificada. A mesura que augmenta el nombre de la k, és a dir, de clusters, el speedup també augmenta, això indica que la paral·lelització és més beneficiosa quan hi ha més càlculs a repartir entre els diferents threads, ja que com podem veure amb un nombre de clusters elevat podem arribar a un speedup de 10,37, mentres que amb un nombre de clusters petit (2, 4, 8) casi no hi ha speedup o directament no n'hi ha.

També podem veure que els resultats mostren diferències en els temps d'execució i speedups entre la GeForce RTX 3080 i GeForce RTX 1080Ti, és a dir, el rendiment pot variar segons l'arquitectura de la GPU utilitzada.

Les optimitzacions de transferència de dades, com s'ha comentat prèviament, s'han fet amb el pragma #pragma acc data copy, és a dir s'han especificat les dades que es necessiten en la regió paral·lela, evitant així transferències de dades innecessàries entre la CPU i la GPU. La paral·lelització dels bucles crítics s'ha fet mitjançant el pragma #pragma acc parallel loop, que ha paral·lelitzat els bucles indicats, és a dir, els ha executat de manera simultània.

En resum, la paral·lelització del codi amb OpenACC ha proporcionat millores significatives en el rendiment quan el valor dels clusters és elevat.