Pol Gràcia Espelt - 1533358
Gerard Jover Pujol - 1527033

Programació Paral·lela

Pràctica OpenACC - 2020/21

1. Plantejament del problema

La pràctica consisteix en solucionar una Equació de Laplace 2D fent servir el mètode iteratiu de Jacobi de forma paral·lelitzada fent servir OpenACC. Se'ns proporciona un codi que resol aquest problema seqüencialment, i el nostre objectiu és paral·lelitzar aquest per tal d'aconseguir una millora en el temps utilitzant una GPU.

2. Anàlisi del problema

El problema de la paral·lelització es troba en que el algorisme va fent iterativament càlculs sobre la matriu i aquesta es va actualitzant, això es fa 'n' vegades o fins que la variació entre els diferents steps sigui menor que un cert valor. A diferència de la pràctica amb OpenMP, el nostre objectiu aquí serà utilitzar els recursos de la GPU per realitzar les operacions més simples dels bucles de forma paral·lela.

3. Disseny de la solució

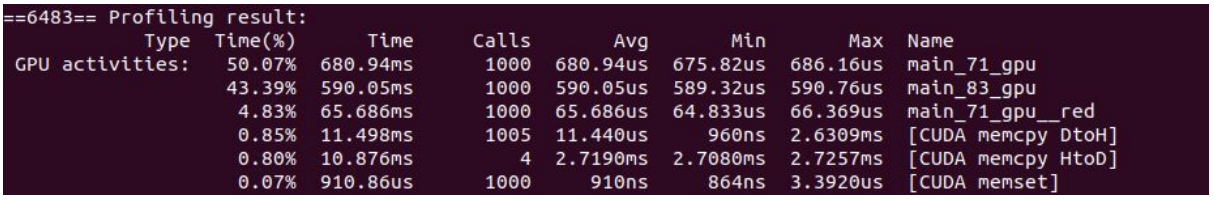
En un primer anàlisis, vam intentar paral·lelitzar la funció laplace step, tot i que aviat vam decidir eliminar les funcions i fer tot el codi dins del main, ja que el programa necessitava 10 segons per fer les crides a les funcions. Un cop vam tenir el codi seqüencial correctament estructurat dins del main i amb funcionament correcte, vam començar a definir les directives per a la paral·lelització. Vam veure que la part que es podria explotar per les acceleradores eren les operacions que feia la funció laplace_step, en els dos bucles fors nidats que feien n*n iteracions 1000 vegades o fins assolir un error més petit que el límit. Així, si podiem repartir totes aquestes operacions entre tots els threads de la gpu, obtindriem una gran millora de temps. Vam decidir posar els paradigmes 'acc parallel loop gang vector reduction(max: error) collapse(2)' sobre el for més extern. Vam decidir fer servir la directiva parallel sobre la kernels per així poder tenir més control sobre el nombre de threads que es creaven, el reduction prenia el valor màxim de tots els diferents valors locals de la variable 'error' que havia calculat cada thread i el collapse(2) indica que afagi els 2 bucles següents i els transformi en un sol bucle.
Una vegada teniem estructurada la paralelització dels càlculs, necessitavem indicar a la GPU com havia d'organitzar la memòria per tal de poder fer els càlculs pertinents i no tenir problemes de memòria ja que estem treballant sobre una matriu que te 16.000.000 d'elements. Per fer això vam decidir crear una regió parallela amb instruccions sobre com havia d'organtizar la GPU la memòria. Això ho vam fer fora del while, ja que només necessitem reservar l'espai un cop en la GPU. Els paradigmes que utilitzem són '#pragma acc data copy(A[:n*n]) create(temp[:n*n])'. El copy indica que volem copiar la matriu A (des del element 0 fins el element n*n, és a dir, la matriu en la seva totalitat) dins la memòria de la GPU i el create indica que volem reservar un espai igual al de la matriu temp[:n*n] per anar guardant els resultats. Finalment, al treballar sobre memòria de la GPU els intercanvis

d'apuntadors que feiem servir en la versió original per actualitzar les dades no eren funcionals, així que vam haver de fer els swaps de memòria manualment amb dos bucles fors i en aquest apartat també vam fer ús de les acceleradores.

4. Resultats (s'han assolit els objectius?, Quina acceleració obteniu respecte la versió seqüencial? I respecte la versió OpenMP? proves realitzades, errors coneguts, etc.)

Amb la versió paral·lelitzada amb OpenAcc obtenim un speedup de 44.3727/1.7523 = 25.3225x respecte l'original, i un 13.82666/1.7523 = 7.789x respecte OpenMP. Utilitzant el *pgprof* obtenim algunes dades sobre el rendiment del programa que ens donen informació sobre l'execució. Veiem que el compilador ha decidit crear grids de 65535x1x1, i blocks de 128x1x1. En total disposem de 65535 (grid size) * 128 (block size) = 8388480 threads. A més el compilador genera un altre grid per separat de 256 threads pel càlcul del reduction.

El *pgprof* mostra també resultats de temps consumit per la GPU, com es mostra en la següent imatge:



```
==6483== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   50.07%  680.94ms      1000  680.94us  675.82us  686.16us  main_71_gpu
                   43.39%  590.05ms      1000  590.05us  589.32us  590.76us  main_83_gpu
                    4.83%  65.686ms      1000  65.686us  64.833us  66.369us  main_71_gpu__red
                    0.85%  11.498ms      1005  11.440us     960ns  2.6309ms  [CUDA memcpy DtoH]
                    0.80%  10.876ms         4  2.7190ms  2.7080ms  2.7257ms  [CUDA memcpy HtoD]
                    0.07%  910.86us      1000     910ns     864ns  3.3920us  [CUDA memset]
```

El programa farà servir dues GPUs (la 71 i la 83) pel còmput de les operacions. La 71 requereix un 50% del temps, uns 680.94ms, i la 82 un 43.39%, uns 590.05ms. A més, observem que apareix una activitat amb el nom de "main_71_gpu_red", que indica el temps que es gasta amb la directiva "reduction": 65.686ms.

5. Principals problemes

El primer problema que ens vam trobar al intentar paral·lelitzar el codi va ser que OpenACC dona molts problemes si es treballa amb funcions (consumeix molt temps), pel que vam passar totes les funcions a codi dins del bucle while. Amb aquest canvi fet vam intentar afrontar el problema amb el kernels, pero ens va portar molts problemes a l'hora de compilar així que vam canviar a parallel. La paral·lelització dels bucles for ens va sortir ràpid, però el codi seguia tardant molt temps i no sabiem veure el perquè. També ens vam trobar amb que l'execució del swap amb punters relentitzava l'execució del programa, pel que vam decidir canviar-ho a un codi amb bucles for que si podíem paral·lelitzar. Mirant el codi de dalt a baix vam arribar a la conclusió de que era perquè la declaració del #pragma acc data es trobava dins del bucle while, pel que s'estava creant aquesta copia a cada repetició del while. Després de treure aquest a fora, i canviar el tipu de copia que feiem (inicialment intentavem fer un copyin i un copyout), vem aconseguir resoldre el problema.