Pràctica de laboratori 2

Simulation of fire extinguishing

Marta Matute - 1496672
David Pacheco - 1565824

1 Plantejament del problema

En aquesta pràctica se'ns presenta un codi seqüencial escrit en llenguatge `C`. Aquest codi simula la propagació i extinció d'un incendi donada una matriu que representa la superfície del bosc, i d'altres paràmetres que ens donen la informació referent als focus d'ignició i als equips de bombers (posició, tipus, etc). Aquestes dades s'obtenen d'un arxiu de text que passem com a argument al programa.

La simulació avança en el temps mitjançant iteracions, i per cada iteració i posició dins del bosc s'actualitzen: la calor de cada posició dins la matriu (i la seva propagació a les posicions adjacents), i el moviment i l'acció dels bombers.

Trobem algunes condicions extres, com ara que el moviment i acció dels bombers no comença fins que no hagin passat 10 "tics de rellotge" (i.e. 10 iteracions) des que el primer focus s'ha activat.

El programa acabarà quan tots els focus de l'incendi es trobin estat "inactiu" o bé quan haguem arribat a un barem màxim d'iteracions, que queda especificat en el fitxer.

2 Anàlisi del problema

Es fàcil veure que moltes de les operacions que realitzem amb el programa de forma seqüencial es poden executar de fet en paral·lel, ja que no hi ha dependència entre elles.

Per exemple, una de les primeres operacions que fem en començar la simulació en sí, és activar els focus que correspongui. Per a fer això, per a cada iteració, recorrem tots els focus i si pertoca, els activem. Donat que cada focus té un temps d'inicialització diferent, que ve donat per argument, aleshores no ens cal tenir en compte la resta. Això vol dir que podríem fer la mateixa operació sobre tots els focus alhora i obtindríem el mateix resultat que fent aquesta operació de manera seqüencial.

Doncs, amb aquest objectiu, hem mirat cadascun dels apartats del codi seqüencial, i els hem analitzat per a veure si era adient paralelitzar-los, o si pel contrari era més òptim executar-los en sèrie.

Amb anterioritat, havíem realitzat aquests canvis per a paral·lelitzar a la CPU del ordinador (havíem fet servir comandes de `OpenOMP`). Ara, farem servir comandes de `OpenACC` per a fer aquestes paral·lelitzacions a la GPU.

3 Disseny de la solució

A diferència del que passava amb la CPU, paral·lelitzar a la GPU no sempre serà més ràpid. Això és perquè, tot i que l'execució de codi sí que és més ràpida a la GPU, és molt lent passar les dades d'un dispositiu a l'altre. Per aquest motiu només valdrà la pena paral·lelitzar a la GPU únicament aquelles parts del codi que facin moltes iteracions independents.

Al nostre codi hem trobat dues regions amb suficients iteracions com per a que ens sortís a compte fer una crida a la GPU i passar-hi les dades: quan actualitzem els valors de cada punt de la matriu segons la calor del punts que l'envolten. Aquest codi el trobem a l'apartat *4.2. Propagate heat (10 steps per each team movement)*, i quan fem el moviment dels equips a l'apartat *4.3. Move teams*.

Comencem explicant els canvis per a l'apartat *4.2. Propagate heat*. Originalment teníem un bucle de 10 iteracions, dins del quan teniem tres bucles més:

- Bucle per activar focus: depenent del la simulació que ens passin, aquest nombre pot ser molt petit, o considerablement gran. Per als tests que hem fet servir, aquest nombre anava des de 2 fins a 1000. itera sobre cada focus i actualitza el valor de la matriu en aquell punt si escau.
- Bucle per a copiar la matriu auxiliar: es crea una còpia de la matriu superfície. Aquest bucle té tantes iteracions com elements tingui la matriu, per tant, pot arribar a ser molt gran.
- Bucle per a actualitzar els valors: Un cop tenim la matriu copiada, fem servir els valors de la còpia per a calcular els nous valors de la matriu superfície. De nou, aquest bucle pot ser molt costós quan fem servir una matriu gran.

Una mica més avall, teníem també un quart bucle:

- Bucle per a calcular la calor residual global: itera també sobre totes les files i columnes. També ens interessa paral·lelitzar-lo per a matrius grans.

Finalment, doncs, la solució implementada executa els quatre bucles interiors a la GPU. Per a aconseguir-ho, primerament hem passat les dades de la CPU a la GPU, en particular hem enviat al dispositiu la matriu superfície, i la informació sobre els focus, i després hem tornat a copiar a la CPU només la matriu superfície. Notem que ens ha calgut crear una variable per a la matriu auxiliar. Això també ens ha ajudat a millorar el temps, ja que és més ràpid crear una variable a la GPU directament, que no pas passar-la d'un lloc a l'altre, i com que en aquest cas la còpia de la matriu encara no tenia guardat cap valor d'interès, hem pogut crear-la directament a la GPU.

Aquestes operacions les hem fet mitjançant la comanda:

```
#pragma acc data copy(surface[:mat_size]) create(surfaceCopy[mat_size])
                 copyin(focal[:num_focal])
```

A continuació, hem creat quatre regions paral·leles (una per a cada bucle) on fem les cridesde les directives d'OpenACC corresponents per a paral·lelitzar els bucles. A més a més, hem afegit el nombre de vectors del bucle niat. Així, aconseguim un guany en l'optimització i millor paral·lelització en conseqüència.

Per a simplificar, es mostra a continuació un esquema de la estructura que hem fet servir:

```
#pragma acc data copy(surface[:mat_size]) copyin(surfaceCopy[:mat_size], focal[:
    num_focal])
{
    /* 4.2. Propagate heat (10 steps per each team movement) */
    for per als 10 steps // 4.2. Propagate heat
        #pragma acc parallel loop
        /* 4.2.1. Update heat on active focal points */
        //for per actualitzar valors de la matriu segons els focus

        #pragma acc parallel vector_length((columns)) present(surface)
        {
            #pragma acc loop gang
            /* 4.2.2. Copy values of the surface in ancillary structure */
            //for per a les files
                #pragma acc loop gang vector
                //for per a les columnes
        }

        #pragma acc parallel vector_length((columns)) if (mat_size > 1000)
        {
            #pragma acc loop gang
            /* 4.2.3. Update surface values (skip borders) */
            //for per a les files
                #pragma acc loop gang vector
                //for per a les columnes
        }
    }

    #pragma acc parallel
    {
        #pragma acc loop collapse(2) reduction(max: global_residual)
            //doble for de files i columnes i calcul del maxim residu global
    }
}
```

Per a la segona regió de codi que hem paral·lelitzat (la corresponent a l'apartat *4.3. Move teams*), originalment teníem un bucle per a cadascun dels equips, dins del qual trobàvem un altre bucle que calcula el focus més proper a l'equip.

De nou, com cada equip és independent l'un de l'altre, aleshores hi ha la possibilitat de fer aquestes operacions en paral·lel sense problema.

El primer de tot ha sigut passar les dades de la CPU a la GPU, en aquest cas hem hagut de passar la informació sobre els equips (el vector `teams`) i la informació sobre els focus. En acabar la computació a la GPU ens caldrà enviar a la CPU el vector `teams` amb els valors acualitzats.

Per a paral·lelitzar, en aquest cas simplement hem fer servir la comanda adient que crea una regió paral·lela alhora que entèn que és un bucle (també hem especificat el nombre de fils per a millorar el rendiment). Per al bucle interior, però, hem especificat que es tracta d'una regió que cal

executar de manera seqüencial. De nou, mostrem en un esquema l'estructura general del la regió paral·lelitzada:

```
#pragma acc data copy(teams[:num_teams]) copyin(focal[:num_focal])
{
    #pragma acc parallel loop gang vector vector_length(num_teams)
    /* 4.3. Move teams */
    // for per a cada equip
        #pragma acc loop seq
        /* 4.3.1. Choose nearest focal point */
        // for per a trobar el focus mes proper
}
```

4 Resultats

El que ens interessa a l'hora d'avaluar si hem obtingut bons resultats o no és el temps d'execució del programa, ja que busquem millorar el rendiment en aquest aspecte. Per avaluar aquests resultats, farem una comparació entre els temps d'execució que obteníem en executar seqüencialment, els temps que vam obtenir a la pràctica anterior fent servir `OpenOMP` (és a dir, paral·lelització a la CPU), i els que hem obtingut en aquesta pràctica fent servir `OpenOMP` (paral·lelització a la GPU).

	Temps d'execució en segons		
	Codi seqüencial	Codi paral·lel en CPU	Codi paral·lel en GPU
`test1` (56 iteracions)	0.001583	0.000596	0.122653
`test2` (30 iteracions)	105.962543	11.874965	5.274414
`test3` (4395 iteracions)	17.374399	3.067379	5.039005
`test4` (2707 iteracions)	19.956566	8.202045	10.841706

Observem que fent servir la GPU hem pogut millorar els temps per a tots els tests excepte el primer, que era unes 77 vegades més ràpid. Per a la resta de tests, hem millorat el resultat. En particular és notable la millora per al test 2, que passa a ser 20 vegades més ràpid. En resum tenim que:

`test1` Empitjora considerablement el resultat.

`test2` Millora molt en rendiment fent servir la GPU (més de la meitat del que aconseguíem amb la CPU).

`test3` Millora el rendiment de manera notable, però obteníem millors resultats fent servir la CPU.

`test4` Millora relativament en relació al temps seqüencial, però no hi ha molta diferencia amb el temps de la CPU.

El problema amb aplicar `OpenACC` per a matrius petites és que es triga moltíssim més en copiar les dades a la GPU i tornar-les a la CPU del que es triga en fer la computació en sí. Per tant en cap cas es millorable fent servir paral·lelització (recordem que la pràctica anterior vam aconseguir un millor temps especificant una dimensió mínima per a aplicar paral·lelisme i afegint optimitzacions al codi seqüencial). En canvi obtenim resultats molt bons per als casos en que trigava molt de manera seqüencial, fins i tot quan el nombre d'iteracions no és molt elevat.

5 Principals problemes

Hem tingut bastantes dificultat a l'hora d'entendre bé l'estructura de les regions paral·leles. En particular ens ha costat molt aconseguir que funcionés ni que sigui una regió petita. No teníem clares del tot quines eren les funcions de cada directiva. Per exemple, no havíem entès que en fer una crida a la directiva `data`, no es creava una regió paral·lela de forma implícita, aleshores tot allò que estigui dins la regió `data`, s'executarà a la CPU si no s'ha especificat la regió paral·lela.

En general, un cop hem tingut aquestes idees clares, no ens ha sigut difícil implementar regions paral·leles a diferents parts del codi.