Universitat Autònoma de Barcelona

Facultat de Ciències

Pràctica 2

Autors:

Gerard Lahuerta & Ona Sánchez

1601350 — 1601181

27 de Novembre del 2022

1 Introducció

L'objectiu d'aquesta pràctica és optimitzar el temps d'execució del programa energystorm, programat al fitxer energy_storm.c.
Per tal d'optimitzar el funcionament del programa, es recurrirà a la paralel·lització de totes les instruccions repetitives, sempre i quan sigui possible, que generin un cost computacional elevat, al fitxer energy_storm_acc.c.

2 Anàlisi del problema

A l'estudi realitzat anteriorment sobre els temps de diversos bucles del programa[1], es va veure que la fase que més tarda en executar-se en la simulació és el bucle iniciat a la linia **183**, ja que el percentatge de temps que s'hi inverteix és casi del 100% del total del temps d'execució, pel que els esforços en optimitzar el codi haurien d'anar enfocats a aquest bucle.

Per altra banda, també es va concloure que el bucle iniciat a la línia **198** podria ser paral·lelitzat, pel que s'ha d'estudiar quina és la millor manera d'optimitzar-lo.

S'observen també bucles diversos que podrien unir-se en un de sol, com és el cas dels fors de les línies **175** i **176** (ja que usen el mateix rang de valors per l'índex k i no depenen entre ells), es procedirà, també, a paral·lelitzar aquest nou bucle conjunt per agilitzar els càlculs quan s'ha de fer moltes iteracions del mateix (ja que depen el nobre d'iteracions en funció del nombre de cèlul·les que hi introduïm).

[1] L'estudi esmentat és el ja entregat anteriorment, si es vol consultar és: Estudi Pràctica 1 CAP

3 Disseny de la solució

S'ha simplificat la paral·lelització de la millor manera possible per tal d'optimitzar al màxim el codi. S'ha separat el codi en tres blocs, dos paral·lelitzats amb la clàusula *#pragma acc kernels* i un bloc que no es pot paral·lelitzar de manera senzilla amb *OpenACC* i que per tant s'hauria d'utilitzar la programació amb cuda.

En el primer bloc, s'ha requerit la creació dins la memòria col·lectiva dels threads del vector *layer_copy*, aquest pas s'ha fet per a optimitzar la paral·lelització i evitar copiar un vector que inicialment està a 0.

L'altre bloc paral·lelitzable amb *OpenACC* ha requerit afegir un bucle més del que disposavem inicialment per a poder trobar la posició a on es troba el màxim del vector *layer*. Tot i afegir aquest nou bucle, la paral·lelització ajuda a disminuir el seu cost.

El bloc que no es pot paral·lelitzar amb OpenACC és degut a que conté un bucle que requereix l'utilització d'un vector (*posval*) que està contingut dintre d'una estructura que a la vegada és guardada dins d'un altre vector (*storms*). Aquest procés fa que el compilador no pugui accedir correctament a la posició de memòria que necessitem per fer els càlculs, pel que únicament es pot paral·lelitzar mitjançant cuda, assegurant-te que hi accedeixes correctament a la posició.

En cas de voler consultar les zones modificades, en les seccions de codi hi ha indicades les files on es troben les comandes al fitxer energy_storms_acc.c.

Exposem ara les seccions del codi modificades:

```
180 #pragma acc kernels
181 {
182     #pragma acc data create(layer_copy[0:layer_size])
183     #pragma acc loop
184     for( k=0; k<layer_size; k++ ){
185         layer[k] = 0.0f;
186         layer_copy[k] = 0.0f;
187     }
188 }
```

```
212 #pragma acc kernels
213 {
214     #pragma acc loop
215     for( k=0; k<layer_size; k++ )
216         layer_copy[k] = layer[k];
217 
218     /* 4.2.2. Update layer using the ancillary values.
219                Skip updating the first and last positions */
220     for( k=1; k<layer_size-1; k++ )
221         layer[k] = ( layer_copy[k-1] + layer_copy[k] + layer_copy[k+1] ) / 3;
222 
223     /* 4.3. Locate the maximum value in the layer, and its position */
224     float m;
225     int p;
226      #pragma acc loop reduction(max:m)
227      for( k=1; k<layer_size-1; k++ ) {
228         /* Check it only if it is a local maximum */
229         if ( layer[k] > layer[k-1] && layer[k] > layer[k+1] ) {
230 
231             if ( layer[k] > maximum[i] ) {
232                 m = layer[k];
233             }
234         }
235     }
236     maximum[i] = m;
237 
238     #pragma acc loop
239     for (k=1; k<layer_size-1; k++){
240             if(layer[k] == m){
241                     p = k;
242             }
243     }
244     positions[i] = p;
245   }
246 }
```

4 Resultat

	TEMPS	
	SEQÜENCIAL	PARAL·LELITZAT
Test 2	43.369	3.865
Test 7	241.569	21.568
Test 8	5.260	70.749
	Acceleració	
Test 2	-	11.22
Test 7	-	11.20
Test 8	-	0.07

S'observa com s'ha optimitzat el programa obtenint una millora de fins a 11.22 vegades més ràpid. Tot i així s'observa com depenent del cost computacional del programa no obtenim una millora significativa (o fins i tot no obtenim millora, com en el cas del test 8) quan és suficientment baix.

Concluim doncs que hem assolit els objectius de paral·lelitzar el programa i recomanem no paral·lelitzar-lo mitjançant *OpenACC* quan el cost del programa no és alt (triga menys de 30 segons aproximadament).

5 Principals problemes

Enumerem ara els principals problemes que han aparegut en el nostre procés d'optimització del codi:

1. Optimització del bucle de la linia 180:

   El problema principal va ser optimitzar l'inicialització dels vector ja que, per exemple, el vector *layer_copy* no és necessari que es guardi a memòria. Per aquest motiu s'ha utilitzat la clàusula *create*, per a crear el vector en la memòria global dels threats y així evitar copiar-lo de manera inecessària.

2. Optimització del bucle de la linia 226:

   Inicialment es va probar a optimitzar el bucle utilitzant clàusules com *reduction* i *routine seq* però no aconseguiem obtenir la posició màxima ja que intentavem tractar el procés de compilació del *OpenACC* com el de *OpenMP*. Finalment és va recorrir a utilitzar la clàusula *reduction* de manera exclusiva per a trobar el màxim i després crear un bucle for després encarregat de trobar la posició on s'assoleix. Aquesta implementació de dos bucles fors en comptes d'un augmenta la complexitat del programa però gràcies a la paral·lelització amb *OpenACC* i de l'algorisme de reducció que s'aplica amb la clàusula *reduction*.