Memoria de la practica de OpenMP

Daniel de Vicente Garrote
Christian de la Puerta Verdejo

1:Explicación de la práctica

El programa consiste en un simulador de bombardeo de particulas sobre una superficie expuesta(tal como dice el enunciado).

Al programa se le administra un tamaño de array y uno o varios ficheros con las oleadas,el programa devuelve el tiempo de ejecución total,los máximos globales que se hayan producido,y donde se ha producido en el array el máximo global mas alto(si es cero es el primer elemento).

Ejemplo con tamaño 20 y un solo fichero(test1.txt)

0.2274 |oooooooooooooooooooooooooooooooooo

0.2404 |ooooooooooooooooooooooooooooooooooo

0.2566 |oooooooooooooooooooooooooooooooooooo

0.2821 |ooooooooooooooooooooooooooooooooooooo

0.2969 |oooooooooooooooooooooooooooooooooooooo

0.3054 |ooooooooooooooooooooooooooooooooooooooo

0.3054 |ooooooooooooooooooooooooooooooooooooooo

0.3214 |oooooooooooooooooooooooooooooooooooooooo

0.3387 |ooooooooooooooooooooooooooooooooooooooooo

0.3657 |oooooooooooooooooooooooooooooooooooooooooooo

0.3700 |ooooooooooooooooooooooooooooooooooooooooooooooooooox M0

0.3596 |oooooooooooooooooooooooooooooooooooooooooooooooooo

Time: 0.003276

Result: 10 0.370016

2:Objetivo de la practica

El programa ejecutado en secuencial tiene unos tiempos de ejecución muy altos(unos pocos minutos),y por ello es necesario reducirlo mediante técnicas de paralelización.

En esta práctica se empleará OpenMP,que permite mediante la declaración de directivas permite paralelizar partes del código,con el objetivo de reducir el tiempo de ejecución a valores aceptables sin alterar los resultados de la ejecución a valores incorrectos.

Lo ideal para esta práctica es cuanto menos tiempo tarde en ejecutarse el programa sin errores en los resultados,mejor. En las pruebas del leaderboard el programa sin paralelizar tarda unos 3 minutos y medio. El objetivo es reducirlo a mucho menos,por ejemplo unos 15 o 20 segundos sería muy óptimo. En nuestro caso solo llegamos a paralelizarlo a unos 33 segundos,una bajada de tiempo bastanta aceptable,pero que se puede mejorar.

3:Modificaciones realizadas en el código

Las modificaciones realizadas en nuestro caso son principalmente el uso de declarativas omp en secciones del código,junto con algunas retoques de elementos,principalmente variables.

- Existen dos bucles al principio de la sección paralelizable que se mueven de la misma manera(desde 0 hasta layer_size),y ambos ponen los elementos de dos arrays a cero. Asi que solo junte ambas operaciones en un bucle y utilicé una directiva omp parallel for,distribuyendo los hilos entre las iteraciones.

```
170     /*#pragma omp parallel for
171     for( k=0; k<layer_size; k++ ) layer[k] =0.0f;
172
173     #pragma omp parallel for
174     for( k=0; k<layer_size; k++ ) layer_copy[k] =0.0f;
175     */
176
177     #pragma omp parallel for
178     for( k=0; k<layer_size; k++ ){
179         layer[k] =0.0f;
180         layer_copy[k] =0.0f;
181     }
182
```

- En este caso de dos bucles no perfectamente anidados,probamos muchas cosas(uso de collapse,firstprivate,juntar manualmente los dos bucles en uno solo reajustando los índices…). Lo que mejor nos ha funcionado ha sido declarar las variables de energía y posicion fuera de los bucles,y dejar las inicializaciones dentro de estas. Ademas pusimos una declarativa omp parallel for en el bucle de índice k,ya que no tiene bucles en su interior y eso hacía que fuera fácil paralelizarlo,y eso no dio una gran mejora de tiempo.

```
189         for( j=0; j<storms[i].size; j++ ) {
190             energia = (float)storms[i].posval[j*2+1] / 1000;
191             posicion = storms[i].posval[j*2];
192             #pragma omp parallel for
193             for( k=0; k<layer_size; k++ ) {
194                 actualiza( layer, k, posicion, energia,distancia ,atenuacion,energia_k);
195             }
196         }
197
```

- Los dos primero bucles no se pueden juntar debido a que empiezan y terminan en índices diferentes,por ello simplemente los dejo separados y realizo una directivo omp parallel for para cada uno. En el caso del tercer bucle(el de los máximos) intentamos poner una declarativa omp parallel for reduction max,pero teníamos que almacenar cada valor de maximos[i] en una variable para cada iteración del bucle de índice i para que pudiese ser utilizada. El resultado es que la mejora de tiempo era muy pequeña,y además al probarlo en el tablón nos tiraba error,asi que al final no utilizamos directiva en este último bucle,ya que no encontramos una que funcionara correctamente. La mejora que si hicimos fue juntar los dos if anidados en uno solo,de forma que hubiera una mínima mejora en el tiempo.

```
198         #pragma omp parallel for
199         for( k=0; k<layer_size; k++ )
200             layer_copy[k] = layer[k];
201
202         #pragma omp parallel for
203         for( k=1; k<layer_size-1; k++ )
204             layer[k] = ( layer_copy[k-1] + layer_copy[k] + layer_copy[k+1] ) / 3;
205
206         for( k=1; k<layer_size-1; k++ ) {
207             if ( (layer[k] > layer[k-1] && layer[k] > layer[k+1]) &&  layer[k] > maximos[i] ) {
208             //if ( layer[k] > maximos[i] ) {
209                 maximos[i] = layer[k];
210                 posiciones[i] = k;
211             //}
212             }
213         }
```

- En esta función cambaimos el if por un valor absoluto(ya que es lo que hacia el if) e incluimos la suma en distancia en la misma linea,así se reduce el numero de instrucciones utilizadas,ademas en el if de abajo cambié la operación de suma a una mas compacta y moví las declaraciones de atenuación y energia_k fuera de los bucles,dejando las inicializaciones dentro de estas,de forma que no estuviera redeclarando las variables de forma constante y ganado una ligera mejora de tiempo. Tales variables se pasa como argumento en la función para que puedan ser utilizadas.

```
25  /* ESTA FUNCION PUEDE SER MODIFICADA */
26  /* Funcion para actualizar una posicion de la capa */
27  void actualiza( float *layer, int k, int pos, float energia,int distancia,float atenuacion,float energia_k ) {
28      /* 1. Calcular valor absoluto de la distancia entre el
29          punto de impacto y el punto k de la capa */
30      distancia = fabs(pos - k)+1;
31      //if ( distancia < 0 ) distancia = - distancia;
32
33      /* 2. El punto de impacto tiene distancia 1 */
34      //distancia = distancia + 1;
35
36      /* 3. Raiz cuadrada de la distancia */
37      atenuacion = sqrtf( (float)distancia );
38
39      /* 4. Calcular energia atenuada */
40      energia_k = energia / atenuacion;
41
42      /* 5. No sumar si el valor absoluto es menor que umbral */
43      if ( energia_k >= UMBRAL || energia_k <= -UMBRAL )
44          layer[k]+=energia_k;
45          //layer[k] = layer[k] + energia_k;
46  }
```

- Aparte,se añadió un #include<omp.h> para poder habilitar el uso de OpenMP

Anexo:Bibliografía y referencias

-Las diapositivas de clase sobre OpenMP

-La página online sobre GCC:http://www.network-theory.co.uk/docs/gccintro/

-StackOverflow:https://stackoverflow.com/questions/13290245/reduction-with-openmp4

https://stackoverflow.com/questions/15304760/how-are-firstprivate-and-lastprivate-different-than-private-clauses-in-openmp

-Otras páginas que hablan de OpenMP:
https://michaellindon.github.io/lindonslog/programming/openmp/openmp-tutorial-firstprivate-and-lastprivate/

http://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-reduction.html