Grupo de laboratorio L2 – G4

Alejandro Sanz
Álvaro Velasco

PRÁCTICA 1:
Optimización con OpenMP de una simulación simplificada de bombardeo de partículas de alta energía

1) Puesta a punto:

En primer lugar, observamos el código para determinar que partes del código teníamos que optimizar.

Después, procedimos a mirar bien las diapositivas de la teoría de la asignatura para así poder saber qué, y cómo podíamos mejorar del código para que se ejecutase de una forma más rápida.

2) Qué se ha utilizado:

Se ha realizado varios intentos con diversas estructuras como:

- `parallel for`
- `barrier`
- `reduction`
- `single`

Además de el uso de variables compartidas con `shared`, o `firstprivate`.

Tras realizar distintos intentos con varias opciones de estas estructuras, probamos a cambiar la llamada a la función « `void actualiza()` » por el propio código de ella, ya que en la teoría del tema anterior lo citaba como una opción para mejorar el rendimiento.

Esta opción fue acertada, ya que además de eso, haciendo un parallel for en el bucle for el tiempo de ejecución bajaba sustancialmente.

```
162
165     /* PARALELIZAMOS LA FUNCION ACTUALIZA */
168    #pragma omp parallel for firstprivate(energia, posicion)
169    for( k=0; k<layer_size; k++ ) {
170            int distancia = posicion - k;
```

A continuación, intentamos paralelizar los bucles `for` restantes –*el que realiza la copia de layer, , el que rellena layer, y el que actualiza los máximos*– que quedaban por debajo

de la reestructuración de actualiza, metiéndolos en un `parallel for`, para que su trabajo se repartiese entre diferentes hilos, pero no era del todo acertado debido a que el último, el que realiza los máximos, no puede ser paralelizado, *–o no hemos dado con la clave para ello–,* debido a que si varios hilos van actualizando los máximos, se cumplen condiciones de carrera ya que uno se puede adelantar a otro, interfiriendo en el resultado.

Al descubrir esto, vimos que el tercero no se podía, pero que los dos anteriores si, por lo que se estableció paralelismo en ambos bucles:

```
194  #pragma omp parallel for shared(layer, layer_copy), firstprivate(layer_size)
195  for( k=0; k<layer_size; k++ )
196          layer_copy[k] = layer[k];
197
198  #pragma omp parallel for shared(layer, layer_copy), firstprivate(layer_size)
199  for( k=1; k<layer_size-1; k++ )
200          layer[k] = ( layer_copy[k-1] + layer_copy[k] + layer_copy[k+1] ) / 3;
201
```

.

Juntamos los dos bucles `for` de la *fase 3* en uno, ya que era una inicialización de vectores, y se podía hacer solo con un bucle, pensábamos que en teoría sería más rápido y más eficiente. También se podría paralelizar este, pero el tiempo que usaba era ínfimo.

En los siguientes intentos procedimos a intentar mejor el bucle `for` que se encuentra en la fase 4.3, el que nos daba problemas al paralelizar debido a la actualización de los máximos, pero en este caso se hizo una reestructuración del código:

```
204  for( k=1; k<layer_size-1; k++ ) {
205          /**
206           * Una reestructuacion de las comprobaciones:
207           * Primero que vea si es maximo, y luego que compruebe sus vecinos.
208           * Antes, estaba de forma:      (B&C), A.
209           * Ahora esta de forma:         A, B, C.
210           **/
211          if ( layer[k] > maximos[i] ) {                  // A
212                  if ( layer[k] > layer[k-1] ) {          // B
213                          if ( layer[k] > layer[k+1] ) {  // C
214                                  maximos[i] = layer[k];
215                                  posiciones[i] = k;
```

Este cambio en la estructura producía que si no era máximo ya dejaba de comprobar, en vez de comprobar primero sus vecinos y después si era máximo. Al implementarlo,

se pudo observar que se reducía apenas un segundo, por lo que se consideró como bueno.

También se intento en este bucle la paralelización usando la cláusula `reduction(max:layer),` con el fin de obtener el máximo de la capa, pero sin éxito.

Otra forma de intentar tratar este bucle fue usando condiciones `if` dentro del pragma, (referencia), pareciendo una opción válida en nuestros ordenadores a pesar de que luego, al subirlo al servidor, nos producía un error.

Como no encontrábamos forma de tratar ese bucle, que es el que nos ha dado problemas, no pudimos dar con la forma de paralelizar el bucle principal del número de tormentas, ya que afectaba a este último.

Para evitar este error se procedió a cambiar el nombre de todas las variables con el fin de evitar las condiciones de carrera, y privatizando en algunos casos la capa (`layer`), pero en todo caso, no encontrábamos la situación de éxito para reducir aún más el tiempo.