COMPUTACIÓN PARALELA
SIMULACIÓN DE EXTINCIÓN DE INCENDIOS

Andrés Cabero Mata
Fernando Gigosos Ronda

Apartado 3:

Este apartado está formado por un solo bucle que debe rellenar con 0 la estructura de memoria
Se paraleliza con una única y sencilla directiva *for* que no cambia en el desarrollo de toda la práctica.

```
    /* 3. Initialize surface */
#pragma omp parallel for
    for( i=0; i<rows; i++ )
        for( j=0; j<columns; j++ )
            accessMat( surface, i, j ) = 0.0;
```

Apartado 4:

El apartado 4 y su respectivo bucle es el que abarca el recorrido de las iteraciones de la simulación y por tanto es necesariamente secuencial, ya que sin la información previa a cada paso no se puede dar el siguiente.
Es dentro de este bucle donde suceden las paralelizaciones más exhaustivas y cada uno de los bucles dentro de este son tratados de forma distinta.
Todo el bucle está en una región paralela.

Apartado 4.1:

El apartado 4.1 consta de un bucle que recorre los focos de calor verificando si en la iteración actual es cuando deben ser activados.
El primer cambio que se hizo fue añadir la siguiente directiva:

```
#pragma omp parallel for reduction(+:num_deactivated)
```

El bucle se paraleliza dividiendo grupos de iteraciones entre los hilos estas iteraciones individualmente suman valores leídos en el array, por ello es necesario la cláusula *reduction* para sumar todos los que se encuentran desactivados, teniendo en cuenta los diferentes hilos y la suma de cada uno.
Al principio se consideró como un *flag* inútil la variable *first_activation.* Por ello se borró del programa en un principio:

```
//  int first_activation = 0;
```

```
//              if ( ! first_activation ) first_activation = 1;
```

Sin embargo, debido al funcionamiento del algoritmo, los cálculos que se hacen en caso de que no haya focos activos son absolutamente inútiles, ya que se inicializa a 0 la temperatura, por ello si se utiliza ese flag (*first_activation) para indicar cuando hay que empezar a hacer la simulación, se pueden ahorrar los cálculos de una numerosa cantidad de iteraciones. Además de eso, la comprobación de activo del foco cambia de if a un else if*, pues el *if* anterior cambia a activado el foco, evitando tener que comprobar en esos casos si está desactivado.

```
        /* 4.1. Activate focal points */
        int num_deactivated = 0;
#pragma omp parallel for reduction(+:num_deactivated)
        for( i=0; i<num_focal; i++ ) {
            if ( focal[i].start == iter ){
                focal[i].active = 1;
                if ( ! first_activation ) first_activation = 1;
            }
            // Count focal points already deactivated by a team
            else if ( focal[i].active == 2 ) num_deactivated++;
        }
        if(! first_activation ) continue;
```

Apartado 4.2:

Es aquí donde se hace la mayor carga de trabajo, pues es un bucle que se ejecuta 10 veces más que el resto de la simulación. Pese a que se trata de un bucle, tiene dentro varios bucles que se ejecutan de distintas formas, por lo que una paralelización simple cambia el resultado esperado del programa. Es necesario paralelizar cada bucle interior de distinta forma.

```
    /* 4.2. Propagate heat (10 steps per each team movement) */
    float global_residual = 0.0f;
    int step;
    for( step=0; step<10; step++ )  {
```

Apartado 4.2.1:

Este primer bucle se paraleliza con la siguiente directiva:

```
#pragma omp for
```

El único cambio que sufre a lo largo de la práctica es eliminar la creación de las variables *x* e *y* utilizando directamente el valor en el acceso de la matriz a la hora de actualizar la temperatura de los focos activos. A la hora de optimizar el siguiente bucle (Apartado 4.2.2), este se ve afectado.

```
#pragma omp for
    for( i=0; i<num_focal; i++ ) {
        if ( focal[i].active != 1 ) continue;
        accessMat( surfaceCopy, focal[i].x, focal[i].y ) = focal[i].heat;
    }
```

Apartado 4.2.2:

Debido a la simpleza de este bucle, el único cambio que tiene en un principio es la siguiente directiva (junto con la cláusula *private(j)*):

```
            /* 4.2.2. Copy values of the surface in ancillary structure (Skip borders) */
#pragma omp for
            for( i=1; i<rows-1; i++ )
                for( j=1; j<columns-1; j++ )
                    accessMat( surfaceCopy, i, j ) = accessMat( surface, i, j );
```

Pero este bucle es eliminable, y hay dos formas de hacer el mismo algoritmo sin necesidad de copiar la matriz.

La forma más “elegante” es utilizar las direcciones de los dos arrays (*surface* y *surfaceCopy)* y cambiar la una con la otra utilizando un puntero auxiliar, ahorrandonos el trabajo de copiar la matriz entera:

```
 #pragma omp single
 {
            float *s = surface;
            surface = surfaceCopy;
            surfaceCopy = s;
 }
```

La otra tiene otro punto de vista distinto. Como se hace 10 veces el bucle completo del apartado 4, y es necesario hacer la copia de la matriz únicamente por cálculo que se hace en el 4.2.3, se puede desdoblar el dónde hacer el cálculo, cambiando de la matriz *surface* a la matriz *surafaceCopy* dependiendo de qué paso del bucle se está iterando. Si es un paso par, haces el cálculo en *surfaceCopy*, si es impar, en *surface*.

Este cambio afecta tanto al apartado anterior, como al siguiente.

Apartado 4.2.3:

Este bucle le pasa lo mismo que al anterior (4.2.2), que su paralelización es tan simple como añadir la siguiente directiva (también incluida la cláusula):

```
            /* 4.2.3. Update surface values (skip borders) */
#pragma omp for nowait
            for( i=1; i<rows-1; i++ )
                for( j=1; j<columns-1; j++ )
                    accessMat( surface, i, j ) = (
                        accessMat( surfaceCopy, i-1, j ) +
                        accessMat( surfaceCopy, i+1, j ) +
                        accessMat( surfaceCopy, i, j-1 ) +
                        accessMat( surfaceCopy, i, j+1 ) ) / 4;
```

Se añade la cláusula nowait debido a que teniendo en cuenta que la repartición de iteraciones del bucle es static, el resultado de este cálculo afecta a los mismos índices que a los que afecta el siguiente bucle (4.2.4).

Como hemos visto este bucle se veía afectado por el anterior. En conjunto, los apartados 4.2.1, 4.2.2 y 4.2.3 terminan de la siguiente manera:

```
#pragma omp parallel private(j)
{
            if(step % 2 != 0){
                /* 4.2.1. Update heat on active focal points */
            #pragma omp for
                for( i=0; i<num_focal; i++ ) {
                    if ( focal[i].active != 1 ) continue;
                    accessMat( surfaceCopy, focal[i].x, focal[i].y ) = focal[i].heat;
                }

                /* 4.2.3. Update surface values (skip borders) */
    #pragma omp for nowait
                for( i=1; i<rows-1; i++ )
                    for( j=1; j<columns-1; j++ )
                        accessMat( surface, i, j ) = (
                            accessMat( surfaceCopy, i-1, j ) +
                            accessMat( surfaceCopy, i+1, j ) +
                            accessMat( surfaceCopy, i, j-1 ) +
                            accessMat( surfaceCopy, i, j+1 ) ) / 4;

            }else{

                /* 4.2.1. Update heat on active focal points */
    #pragma omp for
                for( i=0; i<num_focal; i++ ) {
                    if ( focal[i].active != 1 ) continue;
                    accessMat( surface, focal[i].x, focal[i].y ) = focal[i].heat;
                }

                /* 4.2.3. Update surface values (skip borders) */
    #pragma omp for nowait
                for( i=1; i<rows-1; i++ )
                    for( j=1; j<columns-1; j++ )
                        accessMat( surfaceCopy, i, j ) = (
                            accessMat( surface, i-1, j ) +
                            accessMat( surface, i+1, j ) +
                            accessMat( surface, i, j-1 ) +
                            accessMat( surface, i, j+1 ) ) / 4;
            }
```

Apartado 4.2.4:

Como se indicó en el servidor de Discord, el cálculo del absoluto, tanto en la condición como en la igualdad es innecesario, pues sólo se necesita saber ese equilibrio de temperatura cuando todos los focos han sido desactivados.
Esa es la razón de que se incluya la condición del primer if. Se cambia también la igualdad, ya que no queremos encontrar el máximo posible menor que el límite (*THRESHOLD)* sino encontrar un valor mayor que el límite, y entonces dejar de buscar. Por ello se añade el break, y, sobre todo se añade un if previo al bucle interno para que una vez encontrado el valor, todas las iteraciones pasen en blanco y no entre en el bucle anidado.
Hay una opción de openmp, con la directiva *cancellation point*, que avisa al resto de hilos, pero su uso es más lento (teniendo en cuenta el tamaño del array y el contenido del bucle) que añadir el break y la condición de superar el límite con el *continue* para que no ejecuten los hilos.
Al final hemos mantenido el absoluto (arreglado ahora que solo afecta a la operación, debido al cambio de los tres apartados anteriores.

```
            /* 4.2.4. Compute the maximum residual difference (absolute value) */
            if(num_deactivated == num_focal)
            if(global_residual < THRESHOLD){
#pragma omp for nowait
                for( i=1; i<rows-1; i++ ){
                    if (global_residual>=THRESHOLD) continue;
                    for( j=1; j<columns-1; j++ ){
                        if ( fabs(accessMat( surface, i, j ) - accessMat( surfaceCopy, i, j )) >= THRESHOLD  ){
                            global_residual = THRESHOLD;
                            break;
                        }
                    }
                }
```

Apartado 4.3:

```
            /* 4.3. Move teams */
  if(num_deactivated<num_focal){
  #pragma omp parallel private(i, j)
  {
  #pragma omp for
                for( t=0; t<num_teams; t++ ) {
```

Este bucle abarca el movimiento de los equipos de bomberos, para ello, se realiza un bucle iterándolos y otro anidado que itera los focos para ver cuál está más cerca del equipo en cuestión, El bucle paralelizado finalmente ha sido el más exterior. Este bucle sólo es necesario cuando hay focos activos, así que se hace cuando no todos están desactivados.
Para ello se añade un if de entrada a 4.3 y 4.4, ambos en la misma región paralela, y si no se cumple se hace solo el 4.4, la acción de los equipos de bomberos.

Apartado 4.4:
En este apartado se efectúan las acciones de los equipos de bomberos, la paralelización es también en el bucle general donde cada hilo hace las acciones de cada equipo. Es necesario usar la siguiente directiva :

```
#pragma omp atomic
```

Ya que si dos equipos afectan al mismo punto, hay que asegurar que se cambian el valor de la posición según el factor de cambio de ambos.

```
#pragma omp for nowait
        for( t=0; t<num_teams; t++ ) {
            int target = teams[t].target;
            int radius;
            /* 4.4.1. Deactivate the target focal point when it is reached */
            if ( target != -1 && focal[target].x == teams[t].x && focal[target].y == teams[t].y
                && focal[target].active == 1 )
                focal[target].active = 2;

            /* 4.4.2. Reduce heat in a circle around the team */
            // Influence area of fixed radius depending on type
            if ( teams[t].type == 1 ) radius = RADIUS_TYPE_1;
            else radius = RADIUS_TYPE_2_3;
            for( i=teams[t].x-radius; i<=teams[t].x+radius; i++ ) {
                if( i<1 || i>=rows-1) continue; //Out of the heated surface
                for( j=teams[t].y-radius; j<=teams[t].y+radius; j++ ) {
                    if (j<1 || j>=columns-1 ) continue; // Out of the heated surface
                    float dx = teams[t].x - i;
                    float dy = teams[t].y - j;
//                  float distance = sqrtf( dx*dx + dy*dy );
                    if ( sqrtf( dx*dx + dy*dy ) <= radius ) {
#pragma omp atomic
                        accessMat( surface, i, j ) = accessMat( surface, i, j ) * (1-0.25); // Team efficiency factor
```

Se ha cambiado también la posición del if que comprueba si el punto al que afecta no entra en la tabla, comprobando antes de entrar en el for de la dimensión j, para evitar comprobaciones innecesarias.

Bibliografía:

- http://jakascorner.com/blog/2016/08/omp-cancel.html
- http://jakascorner.com/blog/2016/07/omp-thread-cancellation-model.html
- https://stackoverflow.com/questions/28258590/using-openmp-to-get-the-index-of-minimum-element-parallellylelly
- https://stackoverflow.com/questions/44197573/performance-openmp-collapse-vs-no-collapse-for-large-nested-loops
- OPENMP Parte1-3 Computación Paralela (Universidad de Valladolid)