PRIMERA PRACTICA
DE
COMPUTACIÓN PARALELA

Roberto Gozalo Andrés & Javier Gutiérrez Rodríguez

Índice

1. Introducción **2**

2. Modificaciones realizadas **2**

3. Otras Consideraciones **4**

4. Bibliografía **4**

1. Introducción

En esta práctica partimos de un código secuencial que simula la evolución de un incendio con varios focos en cuya extinción participan equipos de bomberos. La superficie es representada en un array bidimensional y mediante el uso de parámetros se establece la localización de los focos y equipos de bomberos. El objetivo de la práctica es conseguir una mejora sustancial en el tiempo de ejecución del programa usando técnicas de computación paralela, en específico de OpenMP. En nuestro caso hemos logrado alcanzar un tiempo de ejecución de 22.386 que nos ha asegurado la posición 15 en el leaderboard.

2. Modificaciones realizadas

En el punto 3 se inicializa el array bidimensional que representa la superficie a 0,0. Nuestra modificación consiste simplemente en paralelizar el bucle más externo, distribuyendo así las iteraciones entre los distintos hilos

```
1     /* 3. Initialize surface */
2       float *copia;
3       #pragma omp parallel for
4       for( i=0; i<rows; i++ )
5         for( j=0; j<columns; j++ )
6           accessMat( surface, i, j ) = 0.0;
7
8
```

En el punto 4.1 intentamos paralelizar el bucle, pero finalmente descartamos esta opción puesto que los resultados eran contraproducentes y ocasionaba un aumento en el tiempo de ejecución.
En el punto 4.2 se encuentra el bucle encargado de simular 10 pasos de propagación del fuego. Como en cada iteración es necesario tener listo el resultado de la iteración anterior para para poder ser realizada correctamente no se ha aplicado ninguna directiva OpenMP. Por mera curiosidad hicimos alguna prueba, pero causaba resultados erróneos.
En el punto 4.2.2 se utilizan dos bucles para copiar la superficie (surface en surfaceCopy). Hemos prescindido de este bucle usando una variable auxiliar float* copia con la cual intercambiamos los valores de surface y surfaceCopy

```
1     copia = surface;
2       surface = surfaceCopy;
3       surfaceCopy = copia;
```

Los puntos 4.2.3 y 4.2.4 son comprimidos en el mismo bucle puesto que ambos eran iguales y no hay dependencia de datos. Adicionalmente se inicia una ejecución paralela del bucle y se aplica una reducción sobre global_residual. La directiva utilizada es pragma omp for reduction (max:global$_r esidual$).

```
    /* 4.2.2. Copy values of the surface in ancillary structure (Skip borders) */
    /* 4.2.3. Update surface values (skip borders) */
  #pragma omp for reduction(max:global_residual)
    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;
          if ( fabs( accessMat( surface, i, j ) - accessMat( surfaceCopy, i, j ) > global_residual ))
          {
            global_residual = fabs( accessMat( surface, i, j ) - accessMat( surfaceCopy, i, j ) );
          }
      }
```

En el punto 4.3 donde comienza el movimiento de los equipos de bomberos se ha paralelizado el bucle usando la directiva pragma omp parallel for private(t), la indicación de que la variable t es privada es redundante puesto que por defecto el índice del bucle es privado.

```
/* 4.3. Move teams */
    #pragma omp parallel for private(t)
    for( t=0; t<num_teams; t++ ) {
```

En el punto 4.3.1 hemos optado por una ejecución paralela del bucle y aplicar una reducción sobre distance La directiva utilizada es pragma omp for reduction (min:distance).

```
/* 4.3.1. Choose nearest focal point */
      float distance = FLT_MAX;
      int target = -1;

      #pragma omp parallel for private(j) reduction(min:distance)
      for( j=0; j<num_focal; j++ ) {
        if ( focal[j].active != 1 ) continue; // Skip non-active focal points
        float dx = focal[j].x - teams[t].x;
        float dy = focal[j].y - teams[t].y;
        float local_distance = sqrtf( dx*dx + dy*dy );
        if ( local_distance < distance ) {
          distance = local_distance;
          target = j;
        }
      }
```

En el punto 4.4 hemos paralelizado el bucle usando la cláusula firstprivate para utilizar variables que ya tenían un valor definido antes de empezar el bucle y la cláusula shared para variables que son compartidas por todos los hilos. La directiva usada fue pragma omp parallel for default(none) firstprivate $(\text{num}_teams, rows, columns, teams, focal) shared(surface) private(i, j)$.

```
/* 4.4. Team actions */
    #pragma omp parallel for default(none) firstprivate (num_teams, rows, columns, teams,
     focal ) shared (surface) private (i, j)
    for( t=0; t<num_teams; t++ ) {
      /* 4.4.1. Deactivate the target focal point when it is reached */
      int target = teams[t].target;
      if ( target != -1 && focal[target].x == teams[t].x && focal[target].y == teams[t].y
       && focal[target].active == 1 )
        focal[target].active = 2;
```

Finalmente, en el punto 4.4.2 se hace uso de la directiva pragma omp atomic para conseguir evitar las condiciones de carrera que se hayan podido dar debido a la paralelización realizada. También hemos optado por no realizar la operación sqrt, elevando el radio al cuadrado, ya que así tenemos un menor coste computacional.

```
float distance = ( dx*dx + dy*dy );
          if ( distance <= radius*radius ) {
            #pragma omp atomic
            accessMat( surface, i, j ) = accessMat( surface, i, j ) * ( 1 - 0.25 ); //
    Team efficiency factor
          }
```

3. Otras Consideraciones

- Hemos intentado usar la cláusula collapse en diversos casos, pero no hemos conseguido encontrar alguno en el que redujese el tiempo de ejecución.
- También hemos observado que la ejecución del programa en el servidor en ciertas horas donde la carga era mínima mejoraba los resultados, también la ejecución de varias pruebas seguidas proporcionaba cierta mejoría, intuimos que debido a la cache.
- Las pruebas realizadas localmente con el caso de prueba nos quedaron inservibles en cierto momento ya que era difícil apreciar mejoras en el tiempo de ejecución, desarrollar un caso de prueba más extenso nos permitió visualizar mejor la influencia de los cambios implementados.
- Planteamos la posibilidad de desenrollar algún bucle, pero por falta de tiempo no llegamos a llevarlo a cabo.

4. Bibliografía

- Apuntes de la asignatura
- Página de IBM sobre la sentencia atomic `https://goo.gl/gnbr8h`
- Página de stackoverflow sobre collapse `https://goo.gl/Yd26hM`