Práctica MPI

Daniel de Vicente Garrote
Christian de la Puerta Verdejo

1. Resumen del problema

Como en la práctica anterior,se nos ofrece un programa de simulador de bombardeo de partículas sobre una superficie expuesta. Dicho programa toma como parámetros de entrada un numero como tamaño y uno o varios ficheros de oleadas,y devolverá por pantalla los máximos que se han formado al final,su localización,y el tiempo total de ejecución del programa.

El problema del algoritmo es que se ejecuta de forma secuencial,y por ello con valores de array muy grandes o muchos ficheros de oleadas el programa tarda mucho(con un tamaño de 100000 y dos ficheros de oleadas grandes tarda 83 segundos).

Adicionalmente,si se compila con el modo debug(make debug),el programa mostrará por pantalla el bombardeo de partículas al finalizar,mostrando los máximos,pero solo funciona con ejecuciones de hasta 35 puntos(tamaño de array). En nuestro programa no funciona debido a que liberamos el espacio de los arrays al terminar el programa.

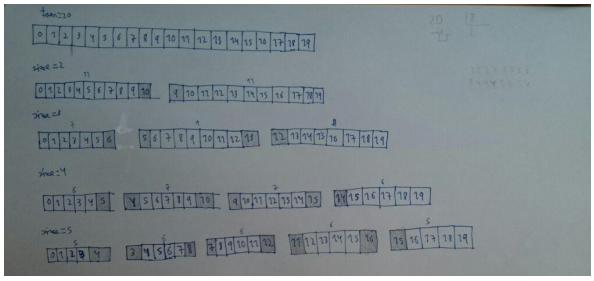
2. Descripción de la solución

Para reducir los tiempos de ejecución emplearemos MPI v2.2,concretamente la implementación de MPICH,de forma que podemos ejecutar varios procesos de forma concurrente para distribuir la carga de trabajo entre ellos y reducir el tiempo de ejecución.

La idea utilizada para la paralelización del código consiste en dividir el array de los bombardeos(y una copia que se utiliza de forma auxiliar) entre los procesos de la forma mas equitativa posible,pero incluyendo a cada trozo del array los valores contiguos que pertenezcan a procesos contiguos.
Por ejemplo un array de tamaño 20 tendría entre los cuatro procesos:

-

- Proceso 0:Posiciones 0 a 4 + posición 5 del proceso 1
- Proceso 1:Posición 4 del proceso 0 + posiciones 5 a 9 + posición 10 del proceso 2
- Proceso 2:Posición 9 del proceso 1 + posiciones 10 a 15 + posición 15 del proceso 3
- Proceso 3:Posición 14 del proceso 2 + posiciones 15 a 19



Ejemplo de reparto de un array de 20 con 2, 3, 4 y 5 procesos(lo sombreados son los «halos» de los trozos de array).

Tras cada fichero,los procesos envían al proceso 0 (proceso ROOT) los máximos que hayan encontrado en sus posiciones asignadas,y el proceso ROOT compara los diferentes máximos de todos los procesos para determinar cual sería el máximo de todo el array.
Cada proceso utiliza los campos contiguos de otros procesos para poder realizar comparaciones en las posiciones que tienen asignadas

3. Cambios realizados en el código

```
    int tam,resto;
    int gIni[size],gTam[size];
    for(i=0;i<size;i++){
        gIni[i]=0;
        gTam[i]=0;
    }

    struct max_local
    {
        int pos[1];
        float max[1];

    }max_local;


    tam=layer_size/size;
    resto=layer_size%size;
    for(i=0;i<size;i++)
        gTam[i]=tam;
    if(resto!=0)
        for(i=size-1;i>=size-resto;i--)
            gTam[i]++;

    gIni[0]=0;

    for(i=1;i<size;i++)
        gIni[i]=gIni[i-1]+gTam[i-1];


    gTam[0]=gTam[0]+1;
    gTam[size-1]=gTam[size-1]+1;
    if(rank>0 && rank<size-1)
        gTam[rank]=gTam[rank]+2;

    if(size==1){
        gTam[rank]-=2;
    }

    if(rank>0)
        gIni[rank]--;

    MPI_Barrier(MPI_COMM_WORLD);
```

Estos valores calculan el tamaño de cada trozo del array(incluido halos) y la posición inicial con respecto al array original.Primero se calcula el tamaño y el desplazamiento para cada trozo sin extremos contiguos, luego se añaden tamaños extra para los elementos contiguos(1 para los procesos de los extremos,2 para el resto) y se desplaza el inicio de cada trozo a la izquierda para todos los proceso excepto el cero.

```
206
207     float *layer_div = (float *)malloc( sizeof(float) * gTam[rank] );
208     float *layer_copy = (float *)malloc( sizeof(float) * gTam[rank] );
209     float Imaximos[size];
210     int Ipos[size];
211
212     if ( layer_div == NULL || layer_copy == NULL ) {
213         fprintf(stderr,"Error: Allocating the layer memory\n");
214         exit( EXIT_FAILURE );
215     }
216     for( k=0; k<gTam[rank]; k++ ) layer_div[k] = 0.0f;
217     for( k=0; k<gTam[rank]; k++ ) layer_copy[k] = 0.0f;
218     for( k=0; k<size; k++ ) Imaximos[k] = 0.0f;
219     for( k=0; k<size; k++ ) Ipos[k] = 0.0f;
220     max_local.max[0]=0.0f;
221     max_local.pos[0]=0;
222
223     /* 4. Fase de bombardeos */
224     for( i=0; i<num_storms; i++) {
225
226         /* 4.1. Suma energia de impactos */
227         /* Para cada particula */
228         for( j=0; j<storms[i].size; j++ ) {
229             /* Energia de impacto (en milesimas) */
230             float energia = (float)storms[i].posval[j*2+1] / 1000;
231             /* Posicion de impacto */
232             int posicion = storms[i].posval[j*2];
233             /* Para cada posicion de la capa */
234                 for( k=0; k<gTam[rank]; k++ ) {
235                     /* Actualizar posicion */
236                     int suma=k+gIni[rank];
237                     actualiza( layer_div, k, posicion, energia,suma);
238                 }
239
```

Aqui en vez de crear un array entero de tamaño layer_size(el tamaño que le damos de entrada),cada proceso crea un array con su tamaño asignado(gTam tiene el tamaño de array de todos los procesos,pero se selecciona siempre por su rank de proceso,es decir gTam[0] es el tamaño del array para el proceso cero). Se realiza lo mismo para layer_copy.

```
28  void actualiza( float *layer_div, int k, int pos, float energia,int suma) {
29      /* 1. Calcular valor absoluto de la distancia entre el
30         punto de impacto y el punto k de la capa */
31
32
33      int distancia = pos - suma;
34      if ( distancia < 0 ) distancia = - distancia;
```

Luego en actualiza ademas se pasa como parámetro de enter suma,que es el indice k mas el desplazamiento del proceso que lo ejecuta,ya que esto se hacía dentro de la función actualiza(utilizando la posición del array en secuencial pero con el desplazamiento del proceso y el indice k).

```
243         for( k=0; k<gTam[rank]; k++ ){
244             layer_copy[k] = layer_div[k];
245         }
246
247         /* 4.2.2. Actualizar capa, menos los extremos, usando valores del array auxiliar */
248         for( k=1; k<gTam[rank]-1; k++ ){
249             layer_div[k] = ( layer_copy[k-1] + layer_copy[k] + layer_copy[k+1] ) / 3;
250         }
251
252         /* 4.3. Localizar maximo */
253         for( k=1; k<gTam[rank]-1; k++ ) {
254             /* Comprobar solo maximos locales */
255             if ( layer_div[k] > layer_div[k-1] && layer_div[k] > layer_div[k+1] ) {
256                 if ( layer_div[k] > max_local.max[0] ) {
257                     max_local.max[0] = layer_div[k];
258                     max_local.pos[0] = k+gIni[rank];
259                 }
260             }
261         } //end for each particle in storm
262
263         MPI_Gather(max_local.max,1,MPI_FLOAT,Imaximos,1,MPI_FLOAT,ROOT_RANK,MPI_COMM_WORLD);
264         MPI_Gather(max_local.pos,1,MPI_FLOAT,Ipos,1,MPI_FLOAT,ROOT_RANK,MPI_COMM_WORLD);
265         if(rank==ROOT_RANK){
266             for(j=0;j<size;j++)
267                 if(Imaximos[j]>maximos[i]){
268                     maximos[i]=Imaximos[j];
269                     posiciones[i]=Ipos[j];
270                 }
271         }
272         for( k=0; k<size; k++ ) Imaximos[k] = 0.0f;
273         for( k=0; k<size; k++ ) Ipos[k] = 0.0f;
274         max_local.max[0]=0.0f;
275         max_local.pos[0]=0;
276
277
278     } //end foreach storm
279
280     free(layer_div);
281     free(layer_copy);
```

Aquí en vez de guardar en maximos[i],cada proceso utiliza un struct(con un valor float llamad max y otro valor int llamado pos) para registrar los máximos y su posición con respecto al array secuencial,luego el proceso ROOT realiza un Gather de los valores del struct y los almacena en arrays temporales. Luego analiza los maximos de cada proceso y guarda en maximos[i] el maximo de entre los proceso,asi como su posicion con respecto al array en secuencial. Al terminar todo el programa, se liberan los arrays.

4. Fallos en el programa

El programa por norma general funciona correctamente cuando solo procesa un archivo,pero si hay dos archivos o mas,los resultados de estos son incorrectos,ya sea por que se desvían unas décimas del resultado correcto, o directamente devuelve cero como máximo. Existen un ligero problema al calcular los puntos de layer_div en los espacios contiguos.Una posible causa puede ser gIni,que toma los valores de desplazamiento desde la izquierda de cada array local(incluyendo los halos),pero no he podido asegurar que ese sea el fallo.
Ademas los resultados pueden variar si se cambia el numero de procesos en el programa,dando siempre resultados correctos con un solo proceso,pudiendo darlos incorrectos si hay dos o mas procesos. Probablemente causado por la particion del array con extremos contiguos(también llamados halos).

5. Bibliografía

https://www.mpich.org/ la documentacion de la pagina de mpich

https://stackoverflow.com/questions/31890523/how-to-use-mpi-gatherv-for-collecting-strings-of-diiferent-length-from-different?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa