COMPUTACIÓN PARALELA

Optimización de una simulación de bombardeo de partículas con MPI

Grupo 04:
Velasco Gil, Álvaro
Sanz Pérez, Alejandro

Introducción:

En esta práctica se solicitaba bajar el tiempo del simulador de bombardeo de partículas, consiguiendo bajarlo de una referencia de 44.785s, siendo algo que no se ha conseguido, a pesar de haber aprendido bastantes usos y manejo de MPI, con reducciones de tiempo considerables, que no eran presente en el leaderboard.
Por ello, se ha llegado a desarrollar hasta 3 versiones de paralelización y distribución de carga entre procesos, donde en ellas se ha conseguido reducir los tiempos con MPI entre 3 o 4, de los tiempos obtenidos ejecutándose en secuencial.
En esta memoria, se van a describir cómo se han realizado esas tres versiones, y cuál creemos que son las causas por las cuales en el leaderboard no se presenta ni aprecia esa reducción, que ejecutándose en local sí que es significativa.

Puesta a punto:

Lo primero, es dividir la carga de trabajo entre los diferentes procesos que van a ejecutarse, y para ello se ha seguido un reparto del tamaño de la capa principal **layer.**

```
*/
int proceso;
int *sendcount = (int *)malloc( sizeof(int) * size );
for (k=0; k<size; k++) sendcount[k]=0;
int *desplazamiento = (int *)malloc( sizeof(int) * size );
for (k=0; k<size; k++) desplazamiento[k]=0;

for (proceso = 0 ; proceso < size ; proceso++){
    sendcount[proceso] = layer_size/size;
    if (proceso < layer_size%size) sendcount[proceso] += 1;
    if (proceso > 0) desplazamiento[proceso] = desplazamiento[proceso-1] + sendcount[proceso-1];
}

float *layer_local = (float *)malloc( sizeof(float) * sendcount[rank] );
for (k=0; k<sendcount[rank]; k++) layer_local[k]=0.0f;

int inicio = desplazamiento[rank];
int dominio = sendcount[rank];
```

De esta forma, se procede a crear en cada proceso un **layer_local** de un tamaño más reducido que el tamaño de **layer** que es la capa principal, y haciendo un mejor suministro de la memoria. También se ha procedido a crear un array **desplazamiento** que indica en qué punto de **layer** empieza a operar cada proceso, y **sendcount**, que es un array que indica cuántas posiciones de layer usa cada proceso a partir de su desplazamiento.
Se puede ver la inicialización de **inicio** y **dominio**, para hacerlo más accesible desde cada proceso.
De este modo, solo haría falta la inicialización de **layer** en el proceso ROOT, ahorrando memoria.

Versión secuencial:

Se va a tener en cuenta a la hora de comparar resultados, constando de 7 tormentas de diferentes tamaños en una capa de tamaño 100.000, repartido entre 13 procesos, de modo que la división 100.000/13 no es exacta.

```
mpiexec -n 13 ./original 100000 ./test/testMedium_100000 ./test/testSmall_20000 ./test/testMedium_100000 ./test/testMPI_20_4p_3.txt
./test/testMPI_20_4p_2.txt ./test/testMedium_100000 ./test/testMPI_20_4p_1.txt

Time: 57.868921
Result: 26237 26.720068 10200 52.117203 13410 77.181709 13411 77.379143 13411 77.399254 15947 103.446716 15947 103.545403
```

V1: MPI_Scatterv y MPI_Gatherv

En esta versión se ha procedido a una división de la capa **layer** en las capas **layer_local** de cada proceso, aprovechando los arrays **desplazamiento** y **sendcount.**
La elección de MPI_Scatterv ha sido debido a que no se va a enviar siempre la misma cantidad de elementos a cada proceso, ya que **layer_local** de los primeros procesos, si a repartir no es exacta, tiene un tamaño mayor. Lo mismo ocurre para la elección de MPI_Gatherv en vez de MPI_Gather.
Tras el uso de MPI_Scatterv, se procede a actualizar los valores de **layer_local** tras la tormenta, y una vez acabada la tormenta, se juntan todos en **layer** del proceso ROOT con MPI_Gatherv, donde el proceso ROOT realiza la fase de relajación, y comprueba los máximos.

```
/* 4. Fase de bombardeos */
for( i=0; i<num_storms; i++) {
    /* Dividimos layer del proceso 0 en layer_local para cada proceso */
    MPI_Scatterv( layer, sendcount, desplazamiento, MPI_FLOAT, layer_local, sendcount[rank], MPI_FLOAT, 0, MPI_COMM_WORLD );
    /* a partir de aquí, cada proceso tiene un layer_local de un tamaño determinado con el que operara */
    /* Fase de actualización de layer_local en cada proceso */
    for( j=0; j<storms[i].size; j++ ) {…
    }
    /* Recopilamos los datos con un Gatterv */
    MPI_Gatherv( layer_local, sendcount[rank], MPI_FLOAT, layer, sendcount, desplazamiento, MPI_FLOAT, 0, MPI_COMM_WORLD );
    /* El proceso 0 tiene recopilado todo en layer */
    if (rank==ROOT_RANK){
        /* 4.1. Copia a layer_copy */
        for( k=0; k<layer_size; k++ )…
        /* 4.2. Relajacion de particulas */
        for( k=1; k<layer_size-1; k++ )…
        /* 4.3. Localizar maximo */
        for( k=1; k<layer_size-1; k++ ) {…
        }
    } //end for each particle in storm
    MPI_Barrier(MPI_COMM_WORLD);
} //end foreach storm
```

Al terminar esta versión comprobamos una notable reducción del tiempo, con resultados correctos, donde se baja de 57’868921 segundos a tan solo **17’41805s.**

```
mpiexec -n 13 ./energy 100000 ./test/testMedium_100000 ./test/testSmall_20000 ./test/testMedium_100000 ./test/testMPI_20_4p_3.txt
 ./test/testMPI_20_4p_2.txt ./test/testMedium_100000 ./test/testMPI_20_4p_1.txt

Time: 17.418052
Result: 26237 26.720068 10200 52.117203 13410 77.181709 13411 77.379143 13411 77.399254 15947 103.446716 15947 103.545403
```

Tras intentar subir esta versión al leaderboard, nos da un <<*error: time limit*>>, lo que nos hace buscar nuevas opciones.

V2: MPI_Gatherv y MPI_Bcast

Tras el timelimit recibido con la v1, se pensó que podía ser debido a que MPI_Scatterv realizaba una comunicación muy lenta a pesar de los buenos resultados obtenidos ejecutándose en local, por lo que se cambió la perspectiva:

```
/* 4. Fase de bombardeos */
for( i=0; i<num_storms; i++) {
    /* Rellenamos layer_local con la parte que le toca a cada proceso */
    for (j=0 ; j<dominio ; j++)
        layer_local[j] = layer[inicio+j];
    /* Actualización de cada layer_local */
    for( j=0; j<storms[i].size; j++ ) {
        int posicion = storms[i].posval[j*2];
        float energia = (float)storms[i].posval[j*2+1] / 1000;
        /* Cada proceso opera con su layer_local */
        for( k=0; k<sendcount[rank]; k++ ) {…
        }
    }
    /* Recopilamos los datos con un Gatterv */
    MPI_Gatherv( layer_local, sendcount[rank], MPI_FLOAT, layer_copy, sendcount, desplazamiento, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
    /* El proceso 0 tiene recopilado todo en layer_copy */
    if (rank==ROOT_RANK){
        /* 4.2. Fase de relajacion, se lo pasa a layer */
        for( k=1; k<layer_size-1; k++ )…
        /* 4.3. Localizar maximo */
        for( k=1; k<layer_size-1; k++ ) {…
        }
    } //end for each particle in storm
    /* Se comparte la capa layer con el resto de procesos */
    MPI_Bcast( layer, layer_size, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
    MPI_Barrier(MPI_COMM_WORLD);
} //end foreach storm
```

En esta versión, cada proceso tendría una copia de **layer**, pero a la hora de operar, actuaría solamente sobre **layer_local**, que sería rellenada en función del **inicio** y **dominio** de cada proceso a partir de **layer.**
Una vez terminado el golpeo de partículas, se reuniría cada **layer_local** de cada proceso en **layer_copy**, que pertenece sólo al proceso ROOT.
Posteriormente, el proceso ROOT realizaría la relajación y obtención de máximos, para, después, compartir con el resto de procesos la capa **layer**, y que la tengan actualizada en las siguientes tormentas.
Esta versión funcionaba también en local, dando unos tiempos un poco mejores que la versión anterior, pero casi imperceptible: **17’299859s.**

```
mpiexec -n 13 ./energy 100000 ./test/testMedium_100000 ./test/testSmall_20000 ./test/testMedium_100000 ./test/testMPI_20_4p_3.txt
 ./test/testMPI_20_4p_2.txt ./test/testMedium_100000 ./test/testMPI_20_4p_1.txt

Time: 17.299859
Result: 26237 26.720068 10200 52.117203 13410 77.181709 13411 77.379143 13411 77.399254 15947 103.446716 15947 103.545403
```

Como contra está el almacenamiento de memoria, ya que la capa **layer** debe reservar espacio en cada proceso, y si estamos hablando de tamaños grandes de **layer**, con muchos procesos, podemos quedarnos fácilmente sin memoria.

V3: MPI_Scatterv y MPI_Reduce

Visto que en las versiones anteriores se nos reducía el tiempo en local pero no en el leaderboard, nos planteamos un nuevo reorientación de la práctica, intentando optimizar la fase de obtención de máximos, en vez de la fase de golpeo de partículas.
Para ello, la fase de golpeo de partículas se realiza en el proceso ROOT, que una vez hecha la fase de relajación, comparte la capa **layer** con el resto de procesos, dando a cada uno -con MPI_Scatterv- la división que le pertenece para que la guarde en **layer_local.** A partir de ahí, cada proceso obtiene el máximo de su capa, y se obtiene el máximo global gracias a la MPI_Reduce operando con MPI_MAXLOC, y pasándoselo al ROOT.
Para un reparto de la posición y el valor del máximo, se ha necesitado la implementación de una estructura.

```
/* 4. Fase de bombardeos */
for( i=0; i<num_storms; i++) {
    /* El proceso ROOT realiza la simulación */
    if (rank==ROOT_RANK) {
        /* 4.1. Fase de golpeo de particulas */
        for( j=0; j<storms[i].size; j++ ) {…
        } /* 4.2. Copia de layer_copy */
        for( k=0; k<layer_size; k++ )
            layer_copy[k] = layer[k];
        /* 4.3. Fase de relajacion */
        for( k=1; k<layer_size-1; k++ )
            layer[k] = ( layer_copy[k-1] + layer_copy[k] + layer_copy[k+1] ) / 3;
    }
    /* Repartimos la capa entre los procesos */
    MPI_Scatterv( layer, sendcount, desplazamiento, MPI_FLOAT, layer_local, sendcount[rank], MPI_FLOAT, 0, MPI_COMM_WORLD );
    /* 4.3. Cada proceso localiza el maximo de su layer_local */
    /* El máximo no puede ser ni la primera ni la ultima posicion de LAYER (afecta al primer y ultimo proceso) */
    int inicio = rank == 0 ? 1 : 0;
    int final = rank == size - 1 ? sendcount[rank]-1 : sendcount[rank];
    /* Estructura para poder enviar el maximo y su posicion */
    struct {
        float maximo;
        int posicion;
    } local, global; // Dos variables que contienen el máximo y posicion local, y otra la global por tormenta
    local.maximo = 0;
    for( k=inicio; k<final; k++ ) {
        /* Comprobación de maximo */
        if ( layer_local[k] > local.maximo ) {…
        }
    }
    /* Reducimos los maximos locales sobre el máximo global */
    MPI_Reduce(&local, &global, 1, MPI_FLOAT_INT, MPI_MAXLOC, ROOT_RANK, MPI_COMM_WORLD);
    /* Actualizacion del maximo global */
    if (rank==ROOT_RANK) {
        maximos[i] = global.maximo;
        posiciones[i] = global.posicion;
    } //end for each particle in storm
} //end foreach storm
```

Tras realizarlo, se ve en los resultados que no hay una reducción de importancia, pero algo sí que reduce, por lo que podría aplicarse a otra versión diferente si se quisiese reducir aún más el tiempo: **56’273376s.** *(secuencial* = 57’868921s.)

```
mpiexec -n 13 ./energy 100000 ./test/testMedium_100000 ./test/testSmall_20000 ./test/testMedium_100000 ./test/testMPI_20_4p_3.txt
 ./test/testMPI_20_4p_2.txt ./test/testMedium_100000 ./test/testMPI_20_4p_1.txt

Time: 56.273376
Result: 26237 26.720068 10200 52.117203 13410 77.181709 13411 77.379143 13411 77.399254 15947 103.446716 15947 103.545403
```

A la hora de subirlo al leaderboard, obviamente, también nos da timelimit.
En cuanto a memoria, solo el ROOT reserva **layer** y **layer_copy**.

V4: Reparto de todas las fases

Después de probar muchas opciones, se procede a repartir en cada proceso todas las fases posibles, incluyendo la fase de relajación. De modo que cada proceso opera con su **layer_local,** después, se juntan todos en **layer** de ROOT con un MPI_Gatherv, que se comparte a todos los procesos con un MPI_Bcast.
Una vez compartido, cada proceso realiza la relajación de la zona que le toca en la repartición, guardando cada uno en su **layer_local** y calcula de ahí el máximo local.
Se realiza una reducción para obtener el máximo global, y se usa un MPI_Gatherv para reunir de nuevo los valores en **layer** que han sufrido la fase de relajación en cada proceso, compartiendo posteriormente otra vez la capa con MPI_Bcast**.**

```
/* 4. Fase de bombardeos */
for( i=0; i<num_storms; i++) {
    /* Rellenamos el layer local con la parte que le toca a cada proceso */
    for (j=0 ; j<dominio ; j++)
        layer_local[j] = layer[inicio+j];
    for( j=0; j<storms[i].size; j++ ) {
        int posicion = storms[i].posval[j*2];
        float energia = (float)storms[i].posval[j*2+1] / 1000;
        /* Cada proceso opera con su layer_local */
        for( k=0; k<sendcount[rank]; k++ ) {…
        }
    }
    /* Recopilamos los datos en el ROOT con un Gatterv */
    MPI_Gatherv( layer_local, sendcount[rank], MPI_FLOAT, layer, sendcount, desplazamiento, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
    /* Compartimos la capa LAYER para que cada proceso haga la atenuacion en la suya propia */
    MPI_Bcast( layer, layer_size, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
    /* Fase de relajacion en cada proceso */
    for( k=0; k<dominio ; k++ ){…
    }
    /* Comprobacion de cada maximo local */
    int inicio_max = rank == 0 ? 1 : 0;
    int final_max = rank == size - 1 ? sendcount[rank]-1 : sendcount[rank];

    struct {…
    } local, global;
    local.maximo = 0;
    /* Comprobacion de maximos en cada layer_local */
    for( k=inicio_max; k<final_max; k++ ) {
        if ( layer_local[k] > local.maximo ) {
            local.maximo = layer_local[k];
            local.posicion = k + desplazamiento[rank];
        }
    }
    /* Reducimos obteniendo el maximo global */
    MPI_Reduce(&local, &global, 1, MPI_FLOAT_INT, MPI_MAXLOC, ROOT_RANK, MPI_COMM_WORLD);
    /* El proceso raiz actualiza el maximo */
    if (rank==ROOT_RANK) {
        maximos[i] = global.maximo;
        posiciones[i] = global.posicion;
    }
    /* Recopilamos los datos finalizando la tormenta con un Gatterv */
    MPI_Gatherv( layer_local, sendcount[rank], MPI_FLOAT, layer, sendcount, desplazamiento, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
    /* Compartimos la capa para que cada proceso haga la siguiente tormenta */
    MPI_Bcast( layer, layer_size, MPI_FLOAT, ROOT_RANK, MPI_COMM_WORLD );
} //end foreach storm
```

Tras probarlo, no se aprecia casi reducción del tiempo respecto a la versión 2, dando un tiempo de **17’320884s** en local, pero no consiguiendo bajar de los dos minutos en leaderboard, siguiendo saliendo el error por timelimit.

```
mpiexec -n 13 ./energy 100000 ./test/testMedium_100000 ./test/testSmall_20000 ./test/testMedium_100000 ./test/testMPI_20_4p_3.txt
 ./test/testMPI_20_4p_2.txt ./test/testMedium_100000 ./test/testMPI_20_4p_1.txt

Time: 17.320884
Result: 26237 26.720068 10200 52.117203 13410 77.181709 13411 77.379143 13411 77.399254 15947 103.446716 15947 103.545403
```

Después de estas cuatro versiones, ya no se nos ha ocurrido forma alguna de poder tratar el programa y que consiga aunque sea bajar de los dos minutos en leaderboard, siendo algo que nos asombra ya que en local los tiempos si que disminuían notablemente.

V0: Versiones fallidas

En las primeras versiones se procedió a la resolución de la práctica repartiendo la carga del array de tormentas entre procesos, con la finalidad de reducir todas las capas **layer** obtenidas en una suma de ellas con un MPI_Reduce, que se guardaría en otro array que sería el histórico de las tormentas, y se reiniciaría a 0 **layer** en cada nueva tormenta para sumar posteriormente el histórico. Este método no se consiguió implementar de nuevo ya que acababa fallando en los valores de algunas milésimas del máximo, y tras no encontrar la solución, se acabó desechando este método.

Conclusiones:

Tras consultar en bibliografía, se consideró que las comunicaciones con MPI tienen un retardo, y que posiblemente no se nos aprecia la reducción en el leaderboard debido a que MPI_Scatterv o MPI_Gatherv son comunicaciones pesadas.
Se iba a proceder a quitar estas opciones y hacer todas las versiones con simples MPI_send pero deberíamos implementar demasiados, por lo que tendrían también un retardo.

Entrega:

En las condiciones de entrega solo se permite la subida de un archivo, asi que se procede a entregar la versión 4.
En cualquier caso que se desee, se proporcionará acceso a cualquiera de las otras versiones.

Bibliografía:

- MPI_Gatherv:
  http://mpi.deino.net/mpi_functions/MPI_Gatherv.html
- MPI_Scatterv:
  http://mpi.deino.net/mpi_functions/MPI_Scatterv.html
- MPI_Reduce and MPI_Reduceall:
  http://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/
- Broadcast and collective communication:
  http://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/