Computación Paralela: MPI

JHON STEEVEN CABANILLA ALVARADO
MIGUEL CHAVEINTE GARCÍA
G01

Punto de partida, división por procesos

```
#define min(X, Y) (((X) < (Y)) ? (X) : (Y))  //MODIFICACION
```

```
//Comenzamos realizando la division por procesos
int resto = rows % nprocs;
int my_size = rows / nprocs;     //Tamaño de cada uno de los procesos
int my_begin = rank * my_size;   //Comienzo de cada uno de los procesos


//Caso en el que el size no sea divisible entre el numero de procesos
if(resto!=0){
        if(rank < resto){
                my_size += 1;
        }
        my_begin += min(rank, resto);
}
```

División array searchers

```
/*3.3 Searchers initialization */
int search;
int cont = 0;
for( search = 0; search < num_searchers; search++ ) {

        searchers[ search ].id = search ;
        searchers[ search ].pos_row = (int)( rows * erand48( random_seq ) );       //Posicion global
        searchers[ search ].pos_col = (int)( columns * erand48( random_seq ) );    //Posicion global
        searchers[ search ].steps = 0;
        searchers[ search ].follows = -1;
        total_steps[ search ] = 0;


        if(searchers[ search ].pos_row >= my_begin && searchers[ search ].pos_row < (my_begin + my_size)){
        //El searcher pertenece al proceso
                searchers[cont] = searchers[ search ];
                cont ++;
        }
}

num_searchers = cont;
```

Accesos

```
/* Get starting position */
int pos_row = searchers[ search ].pos_row;
int pos_col = searchers[ search ].pos_col;

int pos_row_local = pos_row-ini; //MODIFICACION: restamos para obtener la coordenada local
```

Otro punto importante, es que se deben modificar los accesos que se hacen a las distintas matrices ya que necesitamos conocer las coordenadas locales correspondientes.

Movimiento de searchers

```
if(climbing_direction==0){
        searchers[ search ].follows = searchers[ search ].id;
        search_flag = 1;
}
```

```
/*MODIFICACION*/
if(ini>pos_row ){
        search_flag = 2;
        //printf("por abajo");
}

if(pos_row >=fin){
        search_flag = 3;
        //printf("por arriba");
}
```

Comprobar los searchers que se quedan, los que se van por arriba o los que se van por abajo.

```
//Declaracion arrays -- Comunicaciones
Searcher *terminados = NULL;            // Searcher terminados
Searcher *EnviadosDelante = NULL;       // Searchers que no pertenecen
Searcher *EnviadosDetras = NULL;        // Searchers que no pertenecen


//Memory allocation
terminados = (Searcher *)malloc( sizeof(Searcher) * num_searchers_total );
EnviadosDelante = (Searcher *)malloc( sizeof(Searcher) * num_searchers_total );
EnviadosDetras = (Searcher *)malloc( sizeof(Searcher) * num_searchers_total );
```

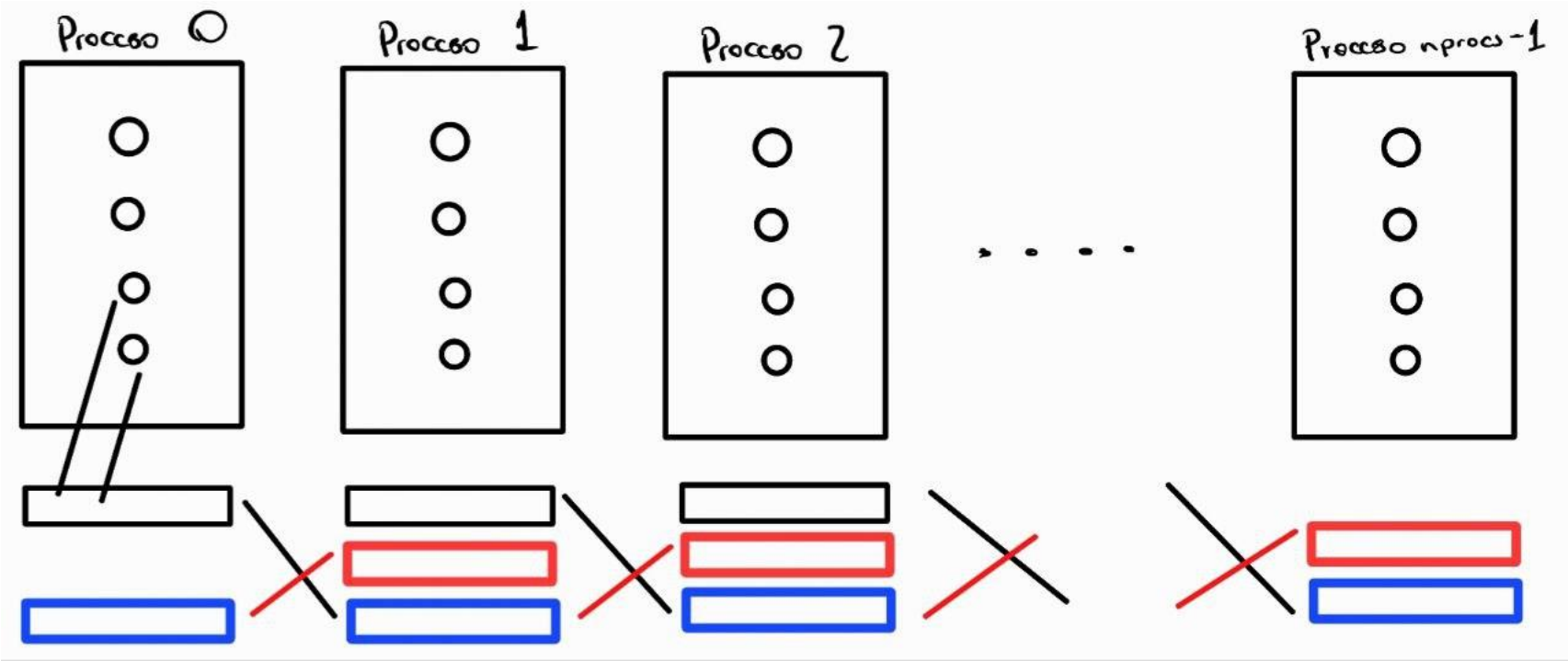
Declaración de arrays

```
//Comprobar posicion de columnas
if(search_flag==2){
        //Se pasa por debajo
        EnviadosDetras[index_detras] = searchers[ search ];
        //printf("detras");
        index_detras++;

} else if(search_flag==3 ){
        //Se pasa por encima
        EnviadosDelante[index_delante] = searchers[ search ];
        //printf("delante");
        index_delante++;
} else{
        //Pertenece
        terminados[index_terminado] = searchers[ search ];
        //printf("terminado");
        index_terminado++;
}
```

En función del valor que tome search_flag, copiaremos el searcher en el array correspondiente y utilizamos un index para cada uno, que nos dirá la nueva cantidad de searchers de ese array.

Comunicaciones: Planteamiento



Implementación

```
int searchers_llegados=0;
int total_recibidos=0;

if(rank != nprocs - 1){

        MPI_CHECK("Envios delante menos del ultimo proceso", MPI_Isend(EnviadosDelante, index_delante, MPI_Searcher, rank+1, 1, MPI_COMM_WORLD, &request1) ); /*CHECKS*/


        MPI_CHECK("Recibo delante menos del utlimo proceso", MPI_Recv(searchers, num_searchers_total, MPI_Searcher, rank+1, 0, MPI_COMM_WORLD, &status1); ); /*CHECKS*/

        MPI_CHECK("Obtenemos cantidad de buscadores, proceso 0", MPI_Get_count(&status1, MPI_Searcher, &searchers_llegados) ); /*CHECKS*/
        total_recibidos=total_recibidos+searchers_llegados;
}

MPI_Wait(&request1, MPI_STATUS_IGNORE);
index_delante=0;


if(rank != 0){
        MPI_CHECK("Envios detras menos del primer proceso", MPI_Isend(EnviadosDetras, index_detras, MPI_Searcher, rank-1, 0, MPI_COMM_WORLD, &request2); ); /*CHECKS*/



        MPI_CHECK("Recibo detras menos del primer proceso", MPI_Recv(&searchers[total_recibidos], num_searchers_total-total_recibidos, MPI_Searcher, rank-1, 1, MPI_COMM_WORLD, &status2) );

        MPI_CHECK("Obtenemos cantidad de buscadores, proceso intermedio", MPI_Get_count(&status2, MPI_Searcher, &searchers_llegados) ); /*CHECKS*/
        total_recibidos=total_recibidos+searchers_llegados;
}

MPI_Wait(&request2, MPI_STATUS_IGNORE);
index_detras=0;


num_searchers=total_recibidos;
```

Fin iteraciones

```
int reduction = 0;
MPI_CHECK("Reduccion terminados", MPI_Allreduce(&index_terminado, &reduction, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD) ); /*CHECKS*/
//printf("iteracion\n");

if(num_searchers_total == reduction){
        flag_terminados = 0;
}
```

```
int flag_terminados = 1;
while(flag_terminados != 0){

        for( search = 0; search < num_searchers; search++ ) {
                int search_flag = 0;
                int pos_row=searchers[ search ].pos_row;
                int pos_col = searchers[ search ].pos_col;
                /* Compute the height */
                int pos_row_local = pos_row-my_begin; //MODIFICACION: restamos para obtener la coordenada local
                float altura=accessMat( heights, pos_row_local, pos_col );
                if(altura==INT_MIN) accessMat( heights, pos_row_local, pos_col ) = get_height( pos_row, pos_col, rows, columns, x_min, x_max, y_min, y_max );
                while( ! search_flag ) {
                        search_flag = climbing_step( rows, columns, searchers, search, heights, trails, x_min, x_max, y_min, y_max, my_begin, (my_begin + my_size));
                }
```

Follows

```
if(rank != 0){
        MPI_CHECK( "Enviar cont_acabados a proceso 0", MPI_Send(&index_terminado, 1, MPI_INT, 0, 10, MPI_COMM_WORLD) );
        MPI_CHECK( "Enviar acabados a proceso 0", MPI_Send(terminados, index_terminado, MPI_Searcher, 0, 11, MPI_COMM_WORLD) );
}

if(rank == 0){

        //Searcher *llegados = (Searcher *)malloc( sizeof(Searcher) * num_searchers_total );

        int vamos_teniendo=index_terminado;
        //printf("index_termin: %d\n",index_terminado);

        for(i = 0; i<vamos_teniendo; i++){
                searchers[i] = terminados[i];
        }

        for(i = 1; i<nprocs; i++){
                MPI_Status status1;
                MPI_Status status2;

                int llegan=0;

                MPI_CHECK( "Recibir index_terminado de cada uno", MPI_Recv(&llegan, 1, MPI_INT, i, 10, MPI_COMM_WORLD, &status1) );
                MPI_CHECK( "Recibir terminados de cada uno", MPI_Recv(&searchers[vamos_teniendo], llegan, MPI_Searcher, i, 11, MPI_COMM_WORLD, &status2) );
                /*for(j = 0; j<llegan; j++){
                        searchers[j + vamos_teniendo] = llegados[j];
                }*/
                //printf("llegan: %d\n",llegan);
                vamos_teniendo = vamos_teniendo + llegan;

        }
```

Reducciones finales

```
/* If this searcher found a maximum, check the maximum value */
for( search = 0; search < index_terminado; search++ ) {
        int id = terminados[ search ].id; // teniamos --> searcher[ search ].id;
        if ( terminados[ search ].follows == id ) {
                int pos_row = terminados[ search ].pos_row - my_begin;
                int pos_col = terminados[ search ].pos_col;
                int valor_altura=accessMat( heights, pos_row, pos_col );
                if ( maximo_local < valor_altura )
                        maximo_local = valor_altura;
        }
}
//printf("maximo  %d\n",maximo_local);

MPI_CHECK( "reduce de las alturas maximas", MPI_Reduce(&maximo_local, &max_height, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD) );


for( i=0; i<my_size*columns; i++ ) {
        if ( trails[i] !=-1 )
                recorridos_local++;
}
MPI_CHECK( "reduce de casillas manchadas", MPI_Reduce(&recorridos_local, &total_tainted, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD) );
```

Otras optimizaciones

- Implementación de Struct Searcher (*)
- Implementación de Halos (**)
- Reserva necesaria de cada matriz
- Utilizar variables para ahorrarnos llamadas
- Eliminar la estructura switch
- Utilizar #define para la función get_height
- Eliminar la matriz Tainted

*

```
///*Implementacion Struct Searcher*///
int fields = 5;

int array_of_blocklengths[] = {1,1,1,1,1};

// Block displacements
MPI_Aint array_of_displacements[] = {
        offsetof( Searcher, id ),
        offsetof( Searcher, pos_row ),
        offsetof( Searcher, pos_col),
        offsetof( Searcher, steps ),
        offsetof( Searcher, follows)
        };

// Block types
MPI_Datatype array_of_types[] = { MPI_INT, MPI_INT, MPI_INT, MPI_INT, MPI_INT };

MPI_Datatype MPI_Searcher;

MPI_CHECK( "creacion MPI_Searcher",MPI_Type_create_struct(fields, array_of_blocklengths, array_of_displacements, array_of_types, &MPI_Searcher ));

MPI_CHECK( "subida MPI_Searcher",MPI_Type_commit( &MPI_Searcher));
```

**

```
//Reservamos solo la parte necesaria de cada matriz
heights = (int *)malloc( sizeof(int) * (size_t)(my_size+2) * (size_t)columns );
trails = (int *)malloc( sizeof(int) * (size_t)my_size * (size_t)columns );
//tainted = (int *)calloc(my_size *columns,sizeof(int));


if ( heights == NULL || trails == NULL ) {
        fprintf(stderr,"-- Error allocating terrain structures for size: %d x %d \n", rows, columns );
        MPI_Abort( MPI_COMM_WORLD, EXIT_FAILURE );
}
```

Fuentes

- Apuntes de teoría.
- https://www.rookiehpc.com/mpi/docs/index.php