Cebrian Serrano, Guillermo
Gil Valverde, Alberto

antenas CUDA:

partes paralelizadas:

inicializar el mapa:
lo inicializamos en el vector del device mediante el kernel gpu_init(int *mapad, int max, int size);
siendo mapad el vector del device, max el numero al que queremos inicializar, y size el tamaño del vector.
para inicializarlo usamos un hilo para cada posicion accediendo a ella mediante el entero position:

int position = blockDim.x * blockDim.y * ((blockIdx.y * gridDim.x)+blockIdx.x)+((threadIdx.y*blockDim.x)+threadIdx.x);
y dando los respectivos valores:

if (position<size) mapad[position] = max;

actualizar:

lo actualizamos mediante otro kernel en el cual accedemos a cada posicion del vector usando un hilo por posicion como en el kernel de inicializar el mapa.

gpu_actualizar(int *mapad, int rows, int cols, Antena antena, int size)
mapad: vector del device.
rows y cols filas y columnas de la matriz.
antena es la nueva antena que se pone en el mapa.
size: tamaño del vector.

para calcular la distancia aplicamos el siguente algoritmo:

```
if(position<size)
	{
		int x,y;
		y=(int)position/cols;
		x=position-y*cols;//rows
		int dist = abs(antena.x -x) + abs(antena.y - y);
		int nuevadist = dist*dist;
		if(nuevadist<mapad[position])
```

```
		{
			mapad[position] = nuevadist;
		}
	}
```

reduce max:

para hacer el reduce tenemos 2 casos:

que el vector sea de tamaño par:

comparamos la primera mitad del vector con la segunda mitad y cada hilo compara un elemento de la primera mitad con el elemento correspondiente de la segunda mitad y se reduce a la mitad el tamaño para llamarlo de nuevo a no ser que se hayan comparado los 2 ultimos, es decir que se reduzca el tamaño a 1.

que el vector sea impar:

comparamos cada posicion del vector con el ultimo y restamos uno al tamaño.
Por ultimo lo copiamos el maximo a un vector de una unica posicon la cual sera el maximo de cada iteración.

```
__global__ void gpu_reduce(int *c, int size)
{
	int position = blockDim.x * blockDim.y * ((blockIdx.y *
gridDim.x)+blockIdx.x)+((threadIdx.y*blockDim.x)+threadIdx.x);

	if(position<size){
		if(size%2 != 0)
		{
			if(c[position]<c[size-1])
			{
				c[position]=c[size-1];
			}
		}else{

			if(c[position]<c[position+size/2])
			{
				c[position]=c[position+size/2];
			}
		}
	}
```

```
}

int reduce(int *maximo,int *c, int *v, int size,dim3 bd,dim3 gd)
{
	int t=size;

		while(t!=1){

			gpu_reduce<<<gd,bd>>>(c,t);
			cudaDeviceSynchronize();
			if(t%2==0){
				t=t/2;
			}else{
				t -= 1;
			}
		}

		cudaMemcpy(maximo,c,sizeof(int) * 1,cudaMemcpyDeviceToHost);
	return maximo[0];
}
```

el bucle while del main quedaria de la siguente manera:
while(1){
calculamos el maximo
copiamos la matriz del host a la matriz del device ya que al calcular el maximo esta se desordena.
si el maximo es menor que distMax se sale del bucle
incrementamos el numero de antenas
creamos una antena nueva
actualizamos
copiamos la matriz de device al host
}