#ifndef \_KERNEL\_CUH\_

#define \_KERNEL\_CUH\_

#include "common.h"

#include "libutils.h"

#include <cuda.h>

#include <curand.h>

#include <curand\_kernel.h>

#include <math.h>

/\*\*

\* Etiqueta para marcar elementos coincidentes en la matriz

\*/

#define COINCIDE 1

/\*\*

\* Etiqueta para marcar elementos que no coinciden en la matriz

\*/

#define NO\_COINCIDE 0

/\*\*

\* Realiza la llamada a la función CUDA y comprueba el valor devuelto. Si hay algún

\* error, devuelve (return) ERR\_CUDA.

\*/

#define CUDA(err, x) do { if ((err = (x)) != cudaSuccess) { \

imprimir (DETALLE\_LOG, "Error en la línea %d de '%s': %s\n", \

\_\_LINE\_\_, \_\_FILE\_\_, cudaGetErrorString (err)); \

return ERR\_CUDA;}} while(0)

/\*\*

\* Realiza la llamada al núcleo CUDA y comprueba el código de error. Si hay algún

\* error, devuelve (return) ERR\_CUDA.

\*/

#define KERNEL(err, nombre, bloques, hilos, ...) \

imprimir (DETALLE\_DEBUG, "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n" \

"Lanzando el núcleo '%s' con las " \

"siguientes dimensiones: \n" \

"\tBloques: x=%d, y=%d\n" \

"\tHilos (por bloque): x=%d, y=%d, z=%d\n" \

"+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n", \

#nombre, \

bloques.x, bloques.y, \

hilos.x, hilos.y, hilos.z); \

nombre <<< (bloques), (hilos) >>> (\_\_VA\_ARGS\_\_); \

\

if ((err = cudaGetLastError ()) != cudaSuccess) \

{ \

imprimir (DETALLE\_LOG, "Error en la línea %d de '%s': " \

"%s\n", \_\_LINE\_\_, \_\_FILE\_\_, \

cudaGetErrorString (err)); \

return ERR\_CUDA; \

}

/\*\*

\* Realiza la llamada al núcleo CUDA que usa memoria compartida

\* y comprueba el código de error. Si hay algún error, devuelve (return) ERR\_CUDA.

\*/

#define KERNEL\_COMP(err, nombre, bloques, hilos, tam, ...) \

imprimir (DETALLE\_DEBUG, "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n" \

"Lanzando el núcleo '%s' con las " \

"siguientes dimensiones: \n" \

"\tBloques: x=%d, y=%d\n" \

"\tHilos (por bloque): x=%d, y=%d, z=%d\n" \

"\tMemoria compartida: %d bytes\n" \

"+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n", \

#nombre, \

bloques.x, bloques.y, \

hilos.x, hilos.y, hilos.z, \

tam); \

nombre <<< (bloques), (hilos) , (tam) >>> (\_\_VA\_ARGS\_\_); \

\

if ((err = cudaGetLastError ()) != cudaSuccess) \

{ \

imprimir (DETALLE\_LOG, "Error en la línea %d de '%s': " \

"%s\n", \_\_LINE\_\_, \_\_FILE\_\_, \

cudaGetErrorString (err)); \

return ERR\_CUDA; \

}

/\* --------------------------------------- \*/

/\* DECLARACIÓN DE FUNCIONES DE DISPOSITIVO \*/

/\* --------------------------------------- \*/

/\*\*

\* Comprueba si es posible realizar un giro de 3x3 en la posición dada.

\*

\* @param posY

\* Coordenada Y del eje a comprobar.

\*

\* @param posX

\* Coordenada X del eje a comprobar.

\*

\* @param dimens

\* Dimensiones de la matriz a comprobar.

\*

\*

\* @return

\* true si es posible.

\* false si no lo es.

\*/

\_\_device\_\_ bool comprobar\_giro (int posY, int posX, Dim dimens);

/\*\*

\* Busca el primer elemento no vacío por encima de la posición especificada.

\* Además, este elemento se convierte a DIAMANTE\_VACIO.

\*

\* @param matriz

\* Matriz en la que se ha de buscar el elemento.

\*

\* @param fila\_ini

\* Fila del primer elemento a comprobar.

\*

\* @param columna

\* Columna a comprobar.

\*

\* @param dimens

\* Dimensiones de la matriz.

\*

\*

\* @return

\* El primer elemento encontrado, si había alguno.

\* -1 si no se encontró ningún elemento no vacío.

\*/

\_\_device\_\_ int buscar\_lleno (int \*matriz, int fila\_ini, int columna, Dim dimens);

/\* ---------------------- \*/

/\* DECLARACIÓN DE NÚCLEOS \*/

/\* ---------------------- \*/

/\*\*

\* Genera un número aleatorio en base a la secuencia especificada y al índice del hilo.

\*

\* @param estado

\* Estado utilizado para generar la secuencia.

\*

\* @param resultado

\* Vector en el que se almacenará el número generado.

\*

\* @param min

\* Límite inferior para generar un número (inclusivo).

\*

\* @param max

\* Límite superior para generar un número (inclusivo).

\*/

\_\_global\_\_ void gen\_aleat\_cuda (curandState \*estado,

int \*resultado,

int min,

int max);

/\*\*

\* Mueve todos los elementos a la izquierda de fila\_bomba hacia su derecha. Cuando llega

\* al primer elemento, genera un nuevo elemento.

\*

\* @param semilla

\* Elemento inicial para generar la secuencia.

\*

\* @param resultado

\* Vector en el que se almacenarán los números generados.

\*

\* @param min

\* Límite inferior para generar un número (inclusivo).

\*

\* @param max

\* Límite superior para generar un número (inclusivo).

\*

\* @param dimens

\* Dimensiones de la matriz resultado.

\*

\* @param fila\_bomb

\* Fila a eliminar.

\*/

\_\_global\_\_ void eliminar\_fila\_cuda (unsigned long semilla,

int \*resultado,

const int min,

const int max,

const Dim dimens,

int fila\_bomba);

/\*\*

\* Mueve todos los elementos a la izquierda de fila\_bomba hacia su derecha. Cuando llega

\* al primer elemento, genera un nuevo elemento.

\*

\* @param semilla

\* Elemento inicial para generar la secuencia.

\*

\* @param resultado

\* Vector en el que se almacenarán los números generados.

\*

\* @param min

\* Límite inferior para generar un número (inclusivo).

\*

\* @param max

\* Límite superior para generar un número (inclusivo).

\*

\* @param dimens

\* Dimensiones de la matriz resultado.

\*

\*

\* @param fila\_bomb

\* Fila a eliminar.

\*/

\_\_global\_\_ void eliminar\_columna\_cuda (unsigned long semilla,

int \*resultado,

const int min,

const int max,

const Dim dimens,

int col\_bomba);

/\*\*

\* Gira todos los elementos posibles en grupos de 3x3 (bomba III).

\*

\* @param resultado

\* Vector que almacena la matriz que va a ser cambiada.

\*

\* @param dimens

\* Dimensiones de la matriz.

\*/

\_\_global\_\_ void girar\_matriz\_cuda (int \*resultado, Dim dimens);

/\*\*

\* Comprueba si la fila contiene elementos repetidos.

\*

\* @param matriz

\* Matriz con los valores actuales de los diamantes.

\*

\* @param dimens

\* Estructura con las dimensiones de la matriz.

\*

\* @param coincidencias

\* Matriz en la que se va a indicar si había alguna coincidencia.

\*/

\_\_global\_\_ void busar\_coinc\_cuda\_fila (int \*matriz,

Dim dimens,

int \*coincidencias);

/\*\*

\* Comprueba si la columna contiene elementos repetidos.

\*

\* @param matriz

\* Matriz con los valores actuales de los diamantes.

\*

\* @param dimens

\* Estructura con las dimensiones de la matriz.

\*

\* @param coincidencias

\* Matriz en la que se va a indicar si había alguna coincidencia.

\*/

\_\_global\_\_ void busar\_coinc\_cuda\_col (int \*matriz,

Dim dimens,

int \*coincidencias);

/\*\*

\* Elimina todos los elementos que se haya visto que han coincidido.

\*

\*

\* @param matriz

\* Matriz con los valores actuales de los diamantes.

\*

\* @param dimens

\* Estructura con las dimensiones de la matriz.

\*

\* @param coincidencias

\* Matriz con las coincidencias encontradas.

\*/

\_\_global\_\_ void eliminar\_coinc\_cuda (int \*matriz,

Dim dimens,

int \*coincidencias);

/\* ----------------------------------- \*/

/\* DECLARACIÓN DE FUNCIONES AUXILIARES \*/

/\* ----------------------------------- \*/

/\*\*

\* Rellena la matriz de juego con diamantes aleatorios.

\*

\* @param malla

\* Estructura de tipo Malla (definida en 'common.h') con las dimensiones de

\* la matriz y su contenido.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_CUDA si alguna función CUDA ha fallado.

\*/

int matriz\_aleat (Malla \*malla);

/\*\*

\* Obtiene las dimensiones de los hilos necesarias para ejecutar la matriz con las

\* dimensiones especificadas, teniendo en cuenta las limitaciones del dispositivo.

\*

\* @param bloques

\* Elemento de tipo dim3 para almacenar las dimensiones de los bloques

\* dentro de la rejilla (2D).

\*

\* @param hilos

\* Elemento de tipo dim3 para almacenar las dimensiones de los hilos dentro

\* de los bloques (3D).

\*

\* @param tam\_matriz

\* Estructura Dim (definida en 'commno.h') con las dimensiones de la matriz

\* que se desea usar en el dispositivo.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_CUDA si hubo algún error al obtener las características del

\* dispositivo.

\* ERR\_TAM si la matriz especificada sobrepasa las capacidades del

\* dispositivo.

\*/

int obtener\_dim (dim3 bloques, dim3 hilos, Dim tam\_matriz);

/\*\*

\* Función para ejecutar la bomba I (eliminar fila).

\*

\* @param fila\_bomba

\* Fila que se debe eliminar (poner a DIAMANTE\_VACIO).

\*

\* @param malla

\* Estructura con la información del juego.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* CUDA\_ERR si hubo algún error relacionado con CUDA.

\*/

int bomba\_fila (int fila\_bomba, Malla \*malla);

/\*\*

\* Función para ejecutar la bomba II (eliminar columna).

\*

\* @param col\_bomba

\* Columna que se debe eliminar (poner a DIAMANTE\_VACIO).

\*

\* @param malla

\* Estructura con la información del juego.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* CUDA\_ERR si hubo algún error relacionado con CUDA.

\*/

int bomba\_columna (int col\_bomba, Malla \*malla);

/\*\*

\* Función para ejecutar la bomba III (girar en grupos de 3x3).

\*

\* @param malla

\* Estructura con toda la información del juego (matriz, nivel

\* y dimensiones).

\*/

int bomba\_giro (Malla \*malla);

/\*\*

\* Busca coincidencias en la matriz y marca las casillas para ser eliminadas (las deja

\* como DIAMANTE\_VACIO.

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_CUDA si hubo algún error al obtener las características del

\* dispositivo.

\* ERR\_TAM si la matriz especificada sobrepasa las capacidades del

\* dispositivo.

\*/

int eliminar\_coincidencias (Malla \*malla);

/\*\*

\* Rellena los diamantes vacíos en la matriz.

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_CUDA si hubo algún error al obtener las características del

\* dispositivo.

\* ERR\_TAM si la matriz especificada sobrepasa las capacidades del

\* dispositivo.

\*/

int llenar\_vacios (Malla \*malla);

#endif