PRÁCTICA CUDA
JUEGO 16384

Laura Pérez Medeiro
Ángel Martín Segura

14 de marzo de 2019

En este documento se especificarán los apartados completados de la práctica, explicando con mayor detalle algunos de los procedimientos más complejos que podemos encontrar en la prácticas y se comentará la problemática que se ha tenido a la hora de elaborar algunos apartados.

1. Memoria Global

Para este apartado, las funciones que encontramos son:

- De ejecución en la CPU:
  - void generarTablero(int *tablero, int filas, int columnas, int dificultad, int vidas): Se encarga de inicializar el tablero vacío con el tamaño indicado por consola de filas y columnas, dentro de este método se llama a generar semillas para dar los valores iniciales.
  - void imprimirTablero(int *tablero, int filas, int columnas, int vidas): Se encarga de imprimir en pantalla el tablero, así como la información de las vidas que aún nos quedan.
  - void imprimirColumnas(int columnas); Para que sea más fácil imprimir el tablero se imprime el formato de las filas aparte.
  - int comprobarLleno(int *tablero, int filas, int columnas, int dificultad, bool &salida, int vidas): Se encarga de comprobar que aún podemos meter semillas nuevas en el tablero. Si es así genera semillas en función de la dificultad, de lo contrario, nos quita una vida y nos pregunta si deseamos seguir jugando.
  - void generarSemillas(int *tablero, int filas, int columnas, int dificultad): En función de la dificultad indicada por pantalla, genera 15 semillas con valores 2,4 u 8 en posiciones libres aleatorias del tablero u 8 semillas con valores 2 o 4.
  - void guardarPartida(int *tablero, int filas, int columnas, int dificultad): En caso de querer abandonar el juego antes de finalizar, existe la opción de guardar el tablero que se posee en momento en un fichero .txt que podemos cargar posteriormente.
  - void cargarPartida(); Si contamos con una partida guardada, podemos reanudarla.
  - void modoManual(int *tablero, int filas, int columnas, int dificultad, int vidas): Función que controla todos los movimientos que el usuario decida realizar, si se ha indicado como modo de juego manual.
  - void modoAutomatico(int *tablero, int filas, int columnas, int dificultad, int vidas): Se encarga de elegir los movimientos que se realizarán en el tablero de manera aleatoria.
  - void iniciar_partida(int vidas): Tras haber perdido una vida, se presenta la opción al usuario de seguir jugando (siempre y cuando aun conserve vidas para ello), volviéndose a pedir los datos acerca del tablero en el que se desea jugar.

- De ejecución en la CPU:
    - __global__ void juego(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): kernel encargado de lanzar los hilos para la ejecución del juego, tras ello llama a compruebaSemillas para realizar los movimientos.
    - __device__ void compruebaSemillas(int *tablero, int filas, int columnas, char movimiento): Se encarga de llamar a la función que desempeñará el movimiento a realizar según cuál le hayamos indicado.
    - __device__ void compruebaArriba(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): Se encarga de desplazar todas las filas hacia la fila más superior que le sea posible, comprueba si las semillas que tiene en esa fila es igual, y en tal caso procede a sumarlas.
    - __device__ void compruebaAbajo(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): Se encarga de desplazar todas las filas hacia la fila más inferior que le sea posible, comprueba si las semillas que tiene en esa fila es igual, y en tal caso procede a sumarlas.
    - __device__ void compruebaDerecha(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): Se encarga de desplazar toda las columnas hacia la columnas más a la derecha que le sea posible, comprueba si las semillas que tiene en esa fila es igual, y en tal caso procede a sumarlas.
    - __device__ void compruebaIzquierda(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): Se encarga de desplazar toda las columnas hacia la columnas más a la izquierda que le sea posible, comprueba si las semillas que tiene en esa fila es igual, y en tal caso procede a sumarlas.
    - __device__ void moverCeros(int *tablero, int fila, int columna, int filas, int columnas, char movimiento): Esta función es la verdadera encargada de mover las semillas dentro del tablero.

Una vez comentadas brevemente las funciones que se emplearán y los parámetros que utilizan, describiremos el funcionamiento a nivel general. El juego comienza preguntándonos si deseamos jugar una partida nueva o continuar jugando con una partida ya guardada, en caso de jugar una partida nueva nos pedirá una serie de valores: numero de filas y columnas del tablero (siendo el tamaño mínimo 4 y el máximo estará dado por las limitaciones de la tarjeta gráfica que posea nuestro ordenador), la dificultad (la cual determinará el numero de semillas que se lanzarán después de cada movimiento que se realiza) y el modo de juego (podremos elegir entre manual y ser nosotros quienes controlemos los movimientos o que estos se produzcan de manera aleatoria).
En este apartado tendremos un kernel que correrá un único bloque de hilos. Esto nos limita el tamaño del tablero que podemos imprimir, ya que no podrá exceder ese tamaño.

Problemas encontrados en el desarrollo de este apartado El correcto funcionamiento de los movimientos de la semilla dentro de la matriz y el sumarlos nos ha supuesto un auténtico reto, en el cual, hemos invertido mucho más tiempo del estimado y provocándonos un considerable retraso para la continuación de la práctica. Otro problema que nos conllevo bastante tiempo fue la detección de los movimientos utilizando las flechas, en sustitución a estas se utilizará para moverse 'w', 's', 'a', 'd' para los movimientos horizontales y verticales, además de 'z' como tecla de escape del juego.

Mejoras: Como se ha comentado, hemos sufrido un considerable retraso en la evolución de la práctica debido a los movimientos de las semillas y la elevada carga de trabajo de otras asignaturas. Aún así hay pequeñas mejoras que podemos encontrar como es la visualización de un tablero con semillas de colores en función del valor que tomen y unos corazones como indicadores de las vidas que aún poseemos.

2. Memoria por bloques

Para la realización de este apartado se contarán con las mismas funciones que en el apartado anterior pero con pequeñas modificaciones. Aquí, encontraremos la variable TILE_WIDTH la cual se la tomará el valor de 8 y que resultará de utilidad a la hora de definir el nuevo "DimGrid" y "DimBlock" que conformarán nuestro kernel.

En este apartado, el verdadero problema se ha encontrado con las matrices irregulares (es decir, aquellas que no tienen el mismo número de filas y de columnas). Ya que este tipo de matrices no llegan a completar el tamaño de la tesela.

Como apoyo a la resolución de este apartado se utiliza la multiplicación de matrices en memoria por bloques que se encuentra en las diapositivas.