ITESM CSF
Programación Multinúcleo Lu 4
Prof. Octavio Navarro
Noviembre 28 2018
Arthur Alves A01022593
Adrián Biller A01018940

Documentación proyecto final

Descripción de proyecto

Antecedentes
2048 es un juego reciente el cual ha ganado mucha fama por su simplicidad y facilidad de jugar en cualquier plataforma. Este juego consiste en una matriz de 4x4 el cual contiene tiles con números de 2^n . Estos tiles se pueden combinar cuando son del mismo número para crear un número más grande. El objetivo del juego es llegar a tener un tile con el número 2048.

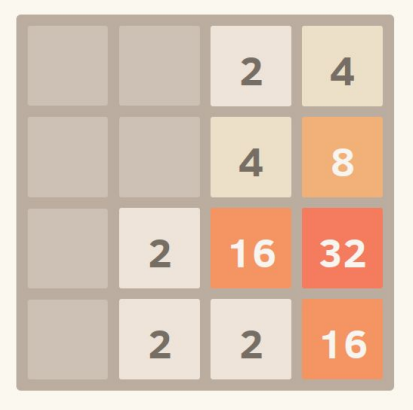

imagen de tablero en juego 2048

Implementación
La simplicidad de este juego nos permite crear una versión en terminal en la cual podemos usar algoritmos de inteligencia artificial para lograr que esta misma tome las decisiones de los movimientos que tomar y poder ganar con el menor número de movimientos posibles.

El propósito de este proyecto es crear dos versiones del mismo juego, una corriendo algoritmos con CPU y otro corriendo el mismo algoritmo con componentes optimizados al correrlo en hilos de GPU.

Manual de usuario

Para correr esta implementación del juego de 2048 se requieren dos archivos. El primero es el archivo de *game2048.py* y es un archivo llamado *autoPlay2048.py*. Los archivos se corren en Python versión 2.7 con el comando *python2.7 <nombre de archivo>*.

Hay diferentes formas de experienciar este proyecto. Primeramente se puede jugar el juego de 2048 en la terminal y buscar lograr tu propio puntaje más alto. Esto se hace con el comando *python2.7 game2048.py* en una terminal en el directorio donde se encuentran los archivos. Los controles para jugar manualmente son las teclas 8, 4, 5 y 6 que equivalen a los movimientos arriba, izquierda, abajo y derecha, respectivamente.

Por otro lado está la implementación del juego para que pueda jugar por su cuenta. Esta implementación se contiene en el archivo *autoPlay2048.py*. Este archivo puede tener dos argumentos que se usan con el comando *python2.7 autoPlay2048.py <numero de juegos hipoteticos por acción (numérico)> <tipo de procesamiento para los movimientos (cpu o gpu)>.* Por ejemplo si se desea jugar usando CPU con 50 juegos hipotéticos por estado de juego se escribe el comando como "*python2.7 autoPlay2048.py 50 cpu*".

Ejemplo 1:

Línea de comando:

```
Projects$ python autoPlay2048.py 50 cpu
```

Salida:

```
[[0 0 0 0]
 [0 0 0 2]
 [0 0 0 0]
 [0 0 0 2]]<- scr: 0 mv: 2

Average time to act: 0.0217008590698
[[0 0 0 4]
 [0 0 0 0]
 [0 0 0 0]
 [2 0 0 0]]<- scr: 4 mv: 0

Average time to act: 0.0223944187164
[[2 0 0 4]
 [0 0 0 0]
 [0 0 0 0]
 [0 2 0 0]]<- scr: 4 mv: 0
```

Ejemplo 2:

Línea de comando:

```
Projects$ python autoPlay2048.py 50 gpu
```

Salida:

```
[[0 0 0 0]
 [2 0 0 0]
 [0 0 0 0]
 [0 0 2 0]]<- scr: 0 mv: 3

Average time to act: 1.24186301231
[[0 0 0 2]
 [0 0 0 2]
 [0 0 0 0]
 [0 0 0 2]]<- scr: 0 mv: 1

Average time to act: 0.866533517838
[[0 0 0 4]
 [0 0 0 2]
 [0 0 0 0]
 [0 0 0 2]]<- scr: 4 mv: 0

Average time to act: 0.739688634872
```

Resultados

	CPU	GPU
Prueba 1	0.1164	3.0472
Prueba 2	0.1189	2.8929
Prueba 3	0.1176	2.5686
Prueba 4	0.1208	2.3769
Prueba 5	0.1196	2.5098
Promedio	**0.11866**	**2.67908**

Speedup

$$\frac{Tiempo\ promedio\ en\ CPU}{Tiempo\ promedio\ en\ GPU} = \frac{0.11866}{2.67908} = 0.044291$$

Conclusión

Como inicialmente se había definido el alcance del programa se buscaba paralelizar las instancias de juego para así obtener un árbol de decisión más grande con mayor utilidad. Sin embargo debido a la estructura que tiene el programa, las clases complicaron la implementación de este. Posteriormente se buscó paralelizar algunas de las operaciones que son realizadas con respecto a las instancias de juego. Se implementó paralelización en una de las operaciones básicas de todas las acciones que se pueden realizar en el juego.

Al jugar 2048, como mencionado anteriormente, todos los tiles de una columna o fila (dependiendo de la dirección jugada) son enviados a un extremo del tablero. Este movimiento básico consideramos es uno de los que más tiempo de ejecución puede tomar pues son varias condiciones y se debe hacer por cada movimiento a todos los cuadros del tablero. Posterior a esto, también se buscó optimizar el cálculo del *score* del juego.

Estas propuestas fueron implementadas correctamente utilizando pyCUDA. Sin embargo, al ejecutar el programa se pudo notar un speedup menor al 100%. Al analizar la implementación, pudimos notar que las actividades que se estaban realizando aunque sí fueron paralelizadas, no iban a ser capaces de disminuir el tiempo de ejecución del programa. Debido al acceso constante de memoria global entre GPU y CPU y el uso exclusivo de la memoria compartida para el *score* se pudo notar que los constantes accesos a memoria, no permitían un speedup.

En conclusión, las ideas si tienen bases e implementadas de otra manera pueden haber tenido un speedup mayor a la implementación en CPU. Sin embargo, hubo implicaciones que no fueron tomadas en cuenta que aumentaron el tiempo de ejecución.