Arquitectura del Computador 2016
Laboratorio N° 2: Programación en OpenCL

Objetivos:

- Diseñar programas basados en los modelos de *plataforma*, *memoria*, *programación* y *ejecución* del paradigma OpenCL.
- Desarrollar códigos y algoritmos capaces de explotar los beneficios del planteo concurrente dentro de un mismo dispositivo, y dentro de un sistema de heterogéneo.
- Ejercitar los contenidos teóricos brindados en clases a encontrar en la bibliografía de la materia[1]

Tareas:

- Se pide diseñar e implementar dos programas bajo el paradigma de programación OpenCL.
- Debe realizarse un **informe breve** (no más de dos páginas),
- Se debe entregar el **código del programa**, el cual debe poder compilar y ejecutar en el clúster MINI del Famaf cuyas cuentas fueron distribuidas en clases.

Condiciones de entrega

- El trabajo es grupal, en grupos de hasta 3 personas.
- Se debe entregar un **informe breve** (no más de dos páginas) donde 1) se discutan las decisiones de arquitectura adoptadas (sincronismos, tamaño de dimensiones de kernels, etc.) y 2) se respondan claramente las cuestiones de performance nombradas en la sección 3 de la parte 2.
- Debe entregarse el **código del programa**, el cual debe poder compilar y ejecutar en el cluster MINI del Famaf cuyas cuentas fueron distribuidas en clases. Los grupos no pueden compartir código entre sí. Cada grupo debe indicar sus integrantes al docente antes de la entrega.
- El trabajo será defendido en un examen oral. Todos los miembros del grupo serán evaluados y deben asistir y ser capaces de responder preguntas del proyecto.
- La entrega del proyecto debe enviarse al correo ***gonzalovodanovic@gmail.com*** en un archivo comprimido con el nombre “lab_Integrante1_Integrante2_Integrante3.tar.gz”. Además, las personas del grupo deben encargarse de traer el proyecto en un pendrive el día de la evaluación o entrega.
- Ambos programas deberán iniciar mostrando la información de la plataforma y dispositivos disponibles, además, se deberá elegir el mejor dispositivo de la plataforma a utilizar. Esto se deberá hacer de la misma forma que en el ejercicio 2 del practico de OpenCL.
- La aprobación del laboratorio es individual. Se evaluará correctitud, claridad de la solución y aspectos de performance.
- El proyecto se califica con la siguiente distribución de puntos:
    - Parte 1: 3 pts.
    - Parte 2: 5 pts.
    - Si ambos programas muestran la información de plataforma y dispositivos disponibles, y ademas, se elige el mejor dispositivo para trabajar. 2 pts. (Estos puntos no son divisibles)
- Se aplicarán posibles penalidades en caso de no cumplir con todas las especificaciones dadas:
    A. No respetar el método de entrada salida estipulado en la presente consigna (2 ptos).
    B. Entregar en la segunda fecha prevista (1 pto). Entregar en la fecha de recuperatorio (3 ptos).
    C. Otras penalidades, si el programa no cumple con lo expresado en esta consigna.
- La nota del laboratorio se promediará con la nota del segundo parcial.

---

[1] "Heterogeneous Computing with OpenCL", By Benedict R. Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry & Dana Schaa

Parte 1: Procesamiento de Imágenes

1. Descripción general

Se partirá de una imagen dada, se le aplicarán diversos procesos y se obtendrá una nueva imagen de salida. Dichos procesos, permitirán al usuario espejar verticalmente, horizontalmente y rotar un ángulo especifico dicha imagen. Estas tareas son altamente paralelizables, ya que, las operaciones se realizan pixel por pixel y son completamente independiente entre ellas.

2. Modelado del problema

El problema deberá dividirse en tres partes (kernels) donde cada una realiza distintos procesos, rotacion, espejado vertical y espejado horizontal. Dependiendo de los procesos que se deseen realizar, se deberán ejecutar uno más kernels. Los tres procesos, a partir de la posición de un pixel deberán calcular la nueva posición.
Para realizar el espejado, se debe trazar una linea imaginaria (vertical u horizontal dependiendo el tipo de espejado a realizar) e intercambiar cada pixel de un lado con su correspondiente del otro.
En el caso de la rotación de un ángulo Θ alrededor del punto (x_0, y_0), se puede lograr mapeando cada pixel (x_1, y_1) a su nueva posición (x_2, y_2) de la siguiente manera:

$$x_2 = cos(\theta) * (x_1 - x_0) - sin(\theta) * (y_1 - y_0) + x_0$$

$$y_2 = sin(\theta) * (x_1 - x_0) + cos(\theta) * (y_1 - y_0) + y_0$$

Con el objeto de simplificar la codificación en C, se provee de funciones de lectura y escritura de archivos de mapa de bits (extensión *.bmp*). Estas se encuentran en el archivo comprimido denominado "*Utilizades_para_el_laboratorio*".

3. Requerimientos y sugerencias de implementación

Se pide realizar un programa OpenCL que lea una imagen, obtenga las acciones a realizarse (rotacion, espejado vertical y/o espejado horizontal) y en caso en que deba realizarse una rotación, debe pedirse el ángulo y el punto a partir del cual se hace la rotacion. El programa deberá realizar **todas** las tareas que se solicitaron en cadena sobre la misma imagen. Y finalmente deberá guardar el resultado como otra imagen del mismo tipo (*.bmp*).
Tanto para los procesos de espejado como el de rotación, se deberá crear un *work-item* por cada pixel, sin embargo, la cantidad de *work-item* en cada proceso difiere. Si llamamos W al ancho de la imagen (en pixels) y H al alto (en pixels), debido a que en los kernels del espejado se procesa la posición de dos pixeles en cada instancia del kernel, la cantidad de *work-item* será de (W x H) / 2. En el caso del de rotación se realiza de a uno, por lo tanto, será de W x H *work-item*.
Las naturaleza bidimensional de las imágenes sugiere de manera directa que las dimensiones del espacio indexado deberá ser de 2, por lo que cada *work-item* tendrá un identificador único del tipo *(x_i, y_i)* obtenible con las funciones get_global_id(0) y get_global_id(1).
Se deberá tener en consideración que en el caso de la rotación, algunos pixeles quedarán por fuera del marco de la imagen.

4. Formatos de entrada/salida

El programa recibirá datos por entrada estándar en un formato prefijado. El formato consiste, línea por línea:

1. Una terna de caracteres que contendrá los procesos a realizar, utilizándose el carácter Y para los proceso que desean realizarse y el N para los que no. De modo se obtendrá algo similar: (R, EV,EH) = (Y,N,N) si únicamente se desea rotar la imagen.
2. En caso de seleccionarse rotación entre los proceso a realizar, solicitar la terna A (ángulo de rotación),X0 ,Y0 (Coordenadas x e y del punto a partir del cual se realizará la rotación).

La salida del programa será una nueva imagen con los procesos aplicados.

Parte 2: Simulación física

1. Descripción general

Se dispone de una placa cuadrada plana de un material uniforme (ej: una plancha de metal). En algunos puntos de esta placa se aplica calor con una fuente de calor (ej: una llama) de modo de mantener la temperatura constante en ese punto. Si bien la aplicación de calor es puntual, el calor tiende a distribuirse alrededor de la fuente y luego de cierto tiempo, los lugares de la placa cercanos a la fuente estarán más calientes que los lejanos. En esta simulación modelamos como varía la temperatura en distintos puntos a lo largo del tiempo.

2. Modelado del problema

En primer lugar, si bien la temperatura varía continuamente a lo largo de la placa, nuestro modelo va a ser discreto, es decir, separar la placa en una cantidad finita de áreas, donde mediremos la temperatura promedio en cada una de esas áreas. La división de la placa se hará en una grilla uniforme de $N \times N$ bloques, donde para cada área se almacenará la temperatura en un *double*.

Llamaremos *j* a la cantidad de fuentes de calor. La posición de la *i*-ésima fuente de calor (con $0 \leq i < j$) estará dada por valores enteros (x_i, y_i), donde $0 \leq x_i < N$ y $0 \leq y_i < N$. El valor *x* representa la columna de la grilla donde está situada la fuente, y el valor *y* representa la fila. La temperatura de la *i*-ésima fuente de calor será denominada t_i, y será un valor de punto flotante (de tipo double).
Si *T* es la matriz de temperaturas, su estado inicial será:

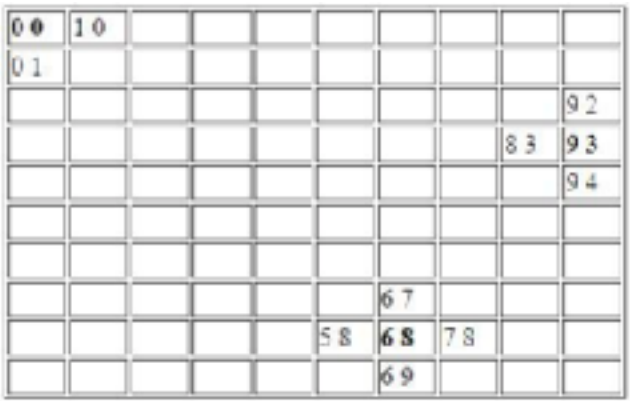
- $T_{pq} = t_i$ cuando $(q, p) = (x_i, y_i)$ para algún *i*
- $T_{pq} = 0$ caso contrario

La simulación a realizar será iterativa, en *k* pasos ($k \geq 0$). En cada paso se construye un *T'* a partir de *T*, que al final de la iteración sustituye al *T original*:

- $T'_{pq} = t_i$ cuando $(q, p) = (x_i, y_i)$ para algún *i*
- $T'_{pq} = promedio([T_{pq}] + + vecinos_T(q,p))$ caso contrario

En la definición anterior, *promedio* tiene su definición usual (*promedio(xs) = sum(xs) / len(xs)*). Los vecinos concretamente son los elementos inmediatamente adyacentes a una posición *(x, y)* (arriba, abajo, izquierda, derecha), pero teniendo cuidado con los bordes. Por ejemplo, si $N = 10$:

- $vecinos_T(0, 0)$ tiene 2 valores: $[T_{1\,0}, T_{0\,1}]$
- $vecinos_T(9, 3)$ tiene 3 valores: $[T_{8\,3}, T_{9\,2}, T_{9\,4}]$
- $vecinos_T(6, 8)$ tiene 4 valores: $[T_{6\,7}, T_{7\,8}, T_{6\,9}, T_{5\,8}]$



Un poco más formalmente:

- *vecinos$_T$(x, y)* contiene a $T_{y\ x-1}$ cuando $x > 0$
- *vecinos$_T$(x, y)* contiene a $T_{y\ x+1}$ cuando $x < N-1$
- *vecinos$_T$(x, y)* contiene a $T_{y-1\ x}$ cuando $y > 0$
- *vecinos$_T$(x, y)* contiene a $T_{y+1\ x}$ cuando $y < N-1$

En resumen, y desde muy alto nivel, el problema a calcular sin paralelismo y en pseudocódigo es:

```
    Leer datos de entrada N, k, j, (x[0], y[0], t[0]) ... (x[j-1], y[j-1],
t[j-1])
        Construir T inicial
        repetir k veces:
            Dado T, construir T'
            T := T'
        Mostrar T
```

3. Requerimientos y Sugerencias de Implementación

En general, el problema resulta altamente paralelizable dado que el cálculo de la temperatura de cada celda de la matriz *T,* se puede calcular independientemente leyendo los valores de los vecinos en el estado anterior *k-1*. Sin embargo plantea un desafío de sincronismo en el tiempo (estado a estado) dado que se debe garantizar que las temperaturas de *Tk-1* deben estar calculadas antes de calcular las de *Tk*.
En el modo de operación deseado, el usuario espera sólo el resultado final de la matriz temperatura *T*. Se deben recorrer todas los *k* pasos directamente hasta llegar al último, el cual será finalmente impreso en pantalla.
Es evidente que el planteo permite que el *Kernel* OpenCL itere internamente a lo largo de los pasos *k.* En otras palabras, se hace una sola copia de memoria (matriz resultado *Tk*) desde el dispositivo al host. Debe quedar claro a nivel conceptual que la propuesta requiere de sincronismos internos local en el *Kernel* (barreras).
Para explotar sistemas capaces de paralelización masiva (GPU), se pide diseñar un programa OpenCL en el que cada celda de la grilla *N×N* sea tratada por un *work-item*. Las naturaleza bidimensional del problema sugiere de manera directa que las dimensiones del espacio de kernel deberá ser de 2, por lo que cada *work-item* tendrá un identificador único del tipo (x_i, y_i) obtenible con las funciones get_global_id(0) y get_global_id(1). En particular, dado que el *Kernel* itera por todos los *k* pasos, necesitará de barreras que garanticen que todos los demás *work-items* han actualizado su temperatura *Tk-1* antes de calcular la *Tk* local. No olvidar que para que una barrera sea válida los *work-items* deberán estar agrupados en el mismo *work-group*.
En el informe se deberán discutir brevemente la siguiente cuestion:

1. El programa de la forma que está implementada, depende directamente del máximo tamaño de *work-group* que el hardware subyacente permita. Se deberá discutir alternativas de diseño de programa *host* y *kernel* que permita adaptarse a hardware con diferente tamaño de *work-group* para cualquier tamaño *N×N* de grilla ¿Es factible?.

4. Formatos de entrada/salida

El programa recibirá datos por entrada estándar en un formato prefijado, del cual se dan algunos ejemplos. El formato consiste, línea por línea

1. Un entero *N* (tamaño en filas/columnas de la grilla)
2. Un entero *k* (cantidad de iteraciones)
3. Un entero *j* (cantidad de fuentes de calor)
4. Una tripla $x_0\ y_0\ t_0$ separada por espacios; los dos primeros valores son enteros, el tercero es real.

Esto describe la fuente de calor 0

5. Una tripla x_1 y_1 t_1 separada por espacios; los dos primeros valores son enteros, el tercero es real. Esto describe la fuente de calor 1
6. ...
7. Una tripla x_{j-1} y_{j-1} t_{j-1} separada por espacios; los dos primeros valores son enteros, el tercero es real. Esto describe la fuente de calor j-1

La salida del programa mostrará una fila de la matriz por línea, en orden. Cada fila será una lista de valores separados por espacios, con cada valor impreso usando formato "%.2f" de C.