Arquitectura de Computadoras 2013
Laboratorio Sistemas Heterogéneos: OpenCL
Juan A. Fraire

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.
- Saber desempeñarse con especificaciones estandarizadas.
- Ejercitar los contenidos teóricos brindados en clases a encontrar en la bibliografía de la materia[1]

Tareas:

- Se pide implementar los ejercicios 1 a 4 detallados a continuación. Los ejercicios deben poder compilarse y ejecutarse en el cluster MINI del Famaf cuyas cuentas son distribuidas en clases.
- Usar las especificaciones y plantilla de referencia subidas al moodle como consulta.

Condiciones:

- Las entregas serán a través del mail: **arq.famaf@gmail.com**. Deberán crear un archivo comprimido (tar, zip, rar, etc.) explicitando el número de laboratorio y sus integrantes (ApellidoNombre): ej: “lab1_FraireJuan_SanchezEduardo.tar.gz”. Dentro del archivo los ejercicios deberán llevar el nombre de ejercicio1.cpp, ejercicio2.cpp y así sucesivamente. Se valorará los códigos entregados con comentarios descriptivos.
- Junto con el código, se deberá entregar un pequeño **informe** en el cual se explique la estructuración del código, decisiones de diseño tomadas (si las hubiere), dificultades con las que se encontraron y cómo las resolvieron, etc.
- El formato de dicho informe queda a elección de ustedes, siempre y cuando sea un formato libre.
- El trabajo es grupal (Max. **3 personas**). Todos los integrantes del grupo deberán ser capaces de explicar el código presentado en la fecha de entrega.
- No está permitido compartir código entre grupos.
- Entrega y defensa de Laboratorio: **25 de Octubre**

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

Ejercicio 1:
Arme un programa usando OpenCL que:

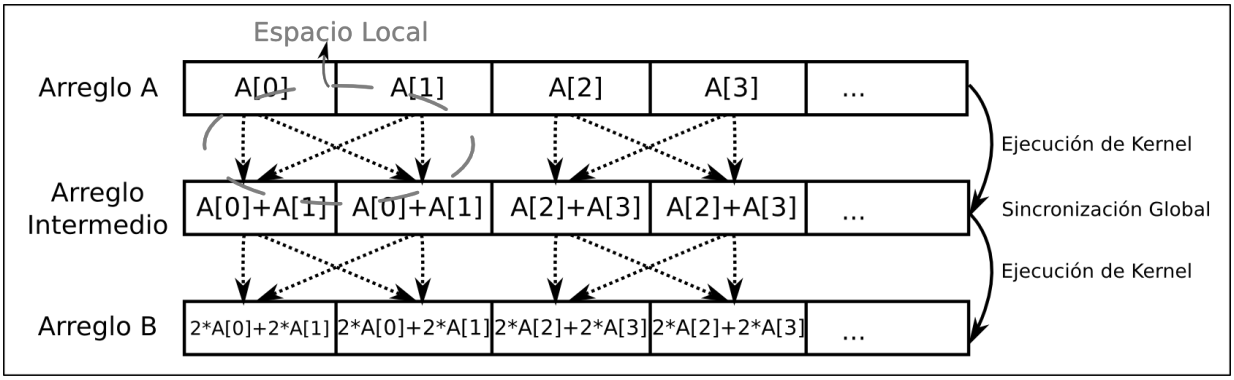
- Descubra la **cantidad de plataformas** disponibles y lo imprima en pantalla.
- Elija la **primera** que el sistema le ofrece para trabajar.
- Imprima en pantalla todas las consultas a plataformas posibles que permite la especificación.
- Consulte la **cantidad de dispositivos** compatibles con OpenCL y lo imprima en pantalla.
- Para cada uno de ellos, imprima en pantalla **al menos 10** de las consultas a dispositivos posibles que permite la especificación.
- Elija el dispositivo con **mejor perfomance** para trabajar.[2]
- Imprima en pantalla las salidas del programa de una manera legible.

Ejercicio 2:
Investigue algoritmos existentes para multiplicar matrices utilizando recursos de procesamiento paralelo, explique en el informe el adoptado. Luego, partiendo del código desarrollado en el práctico 1, desarrolle un programa OpenCL que:

- Genere dos **matrices cuadradas** con valores **flotantes** aleatorios de tamaño **NxN** donde N es una macro (#define N) definida en el programa.
- Utilice el dispositivo con mejor performance detectado en el **ejercicio 1** para ejecutar la multiplicación en paralelo.
- **Verifique** el resultado obtenido en el dispositivo OpenCL con el código serial clásico.
- Imprima en pantalla el resultado de una manera legible.

Ejercicio 3:
Para demostrar la capacidad de **sincronismo global y local** en OpenCL, se pide que arme un programa que realice dos veces la suma de los componentes vecinos de un arreglo A como se muestra en la figura. El sincronismo global se logra al ejecutar el mismo kernel en dos etapas.

- Se requiere utilizar un espacio **global** unidimensional de 10 elementos, y uno **local** de 2 elementos.
- Dentro del espacio local, cada kernel debe leer un elemento del arreglo de entrada almacenado en la memoria global para guardarlo en la memoria local. Implemente una barrera (`barrier(CLK_LOCAL_MEM_FENCE)`) para garantizar que cada kernel efectivamente escribió su espacio de memoria local correspondiente. Finalmente escriba el resultado de la operación usando los valores almacenado en la memoria local.
- Imprima en pantalla el resultado de una manera legible.



[2] Adoptando que la estimación de MFLOPS (mega-cálculo de punto flotante por segundo) como producto de la frecuencia de reloj por las unidades de cómputo disponibles.

Ejercicio 4:
Sabemos que la constante PI se puede determinar matemáticamente por medio de la siguiente ecuación:

$$\int_0^1 \frac{4}{(1+x^2)} dx = \pi$$

Y que por ende, se puede aproximar con:

$$\sum_{i=0}^{N} F(x_i) * \Delta x \approx \pi$$

Donde la aproximación se hace más precisa para grandes valores de N. El código serial para un programa que realiza este cálculo en C tradicional es el mostrado a continuación

```
static long int num_steps = 100000;
double step;
void main(){
	double x, pi, sum=0.0;
	step = 1.0/(double) num_steps;
	for(int i=0; i<num_steps;i++){
		x = (i+0.5)*step;
		sum = sum + 4.0/(1.0+x*x);
	}
	pi = step * sum;
}
```

- Se pide una implementación en OpenCL del algoritmo.
- Detalle en el informe las decisiones tomadas y detecte posibles mejoras u optimizaciones a realizar.
- Imprima en pantalla el resultado de una manera legible.