**Arquitectura del Computador 2015**

**Laboratorio N° 2: Programación en OpenCL**

**Objetivos:**

- Plantear un problema de simulación física, basado en un problema real de forma concurrente y paralela. El problema y modelo aquí brindado será compartido con el siguiente laboratorio de la materia (paralelización en MPI).
- 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 estándares.
- Ejercitar los contenidos teóricos brindados en clases a encontrar en la bibliografía de la materia[1]

**Tareas:**

- Se pide diseñar e implementar el modelo mostrado a continuación bajo el paradigma de programación OpenCL.
- Se debe entregar un **informe breve** (no más de una o 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.
- 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.

## 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).

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

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: $[T_{6\,7}, T_{7\,8}, T_{6\,9}, T_{5\,8}]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0 0** | 1 0 | | | | | | | | |
| 0 1 | | | | | | | | | |
| | | | | | | | | | 9 2 |
| | | | | | | | | 8 3 | **9 3** |
| | | | | | | | | | 9 4 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | 6 7 | | | |
| | | | | | 5 8 | **6 8** | 7 8 | | |
| | | | | | | 6 9 | | | |

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*.

Se pide implementar el modelo en OpenCL de dos formas diferentes:

- **a) Modo Express:** 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.
- **b) Modo Real-Time:** El usuario espera estudiar la evolución en el tiempo de la matriz temperatura *T*. Para esto, cada matriz *Tk* producto de la ejecución del paso *k*, deberá ser impresa en pantalla.

Es evidente que el planteo *Express* permite que el *Kernel* OpenCL itere internamente a lo largo de los pasos *k*, mientras que el *Real-Time* implica una ejecución diferente por cada iteración *k*. En otras palabras, en a) se hace una sola copia de memoria (matriz resultado *Tk*) desde el dispositivo al host, mientras que en b) se deben hacer tantas copias como iteraciones *k* existan. Debe quedar claro a nivel conceptual que la propuesta Express requiere de sincronismos internos local en el *Kernel* (barreras), mientras que en el modo Real-Time se deben utilizar herramientas de sincronización del lado del *host* (colas de comandos en-orden o eventos). Dada la diferente naturaleza de los enfoques, se sugiere implementar dos *Kernels* distintos, uno para cada modo.

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* del modo *Express* 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*.

Finalmente, se pide que el programa tome los valores de tiempos asociados a las transferencias de memoria host-device y viceversa, así como los intervalos de tiempo de ejecución de kernels (no se deben considerar los tiempos asociados a la impresión en pantalla). Para esto se deberán usar los modos de profiling explicados en clase.

En el informe se deberán discutir brevemente dos cuestiones:

1. Se deberán usar los resultados de timing obtenidos para explicar, analizar, y justificar detalladamente la diferencia (*overhead*) en performance del uso de los modos Express y Real-Time para diferentes valores de *N* y *k*.
2. El programa en modo Express, 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?.

### 3.1. 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 carácter ‘e’ o ‘E’, o ‘r’ o ‘R’ para especificar el modo de operación Express o Real-Time

respectivamente.

2. Un entero *N* (tamaño en filas/columnas de la grilla)
3. Un entero *k* (cantidad de iteraciones)
4. Un entero *j* (cantidad de fuentes de calor)
5. 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
6. 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
7. ...
8. 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. En el caso del modo Express sólo se mostrará la matriz *T* final, mientras que en el modo Real-Time se deberán ir mostrando las matrices intermedias *Tk* a medida que avanza la simulación.

## 4. Condiciones de entrega

- El trabajo es grupal, en grupos de hasta 3 personas.
- Se debe entregar un **informe breve** (no más de una o 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.
- Se debe entregar 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 podrá ser considerado para una evaluación 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 adcfamaf@gmail.com en un archivo comprimido con el nombre “lab_FraireJuan_SanchezEduardo_JosePerez.tar.gz” *antes* de la fecha de presentación. Además, las personas del grupo deben encargarse de traer el proyecto en un pendrive el día de la evaluación o entrega.
- La aprobación del laboratorio es individual. Se evaluará correctitud, claridad de la solución y aspectos de performance. El proyecto se califica como "no aprobado" o "aprobado" y se podrá recuperar en una fecha a determinar.