INF-351: Computación de Alto Desempeño
Laboratorio 1
Paralelizando el Método de Euler

Prof. Álvaro Salinas

27 de Septiembre de 2018

1. Descripción y Marco Teórico

En el siguiente laboratorio, usted realizará un análisis del paralelismo ofrecido por CUDA al resolver un determinado problema. Para lograrlo, deberá implementar CUDA kernels que permitan ejecutar un procesamiento paralelo sobre un conjunto de datos, para posteriormente comparar su desempeño con una versión secuencial que se ejecute en la CPU. Los objetivos principales de esta experiencia consisten en evaluar sus conocimientos sobre los fundamentos básicos de la programación en CUDA, tales como la elaboración de kernels, manejos de memoria, distribución de trabajo en CUDA threads y mediciones de tiempo.

Método de Euler

El método de Euler es el método numérico más simple para resolver un problema de valor initial (IVP), el cual corresponde a un tipo de ecuación diferencial ordinaria (ODE). Formalmente, un problema de valor inicial es una ecuación diferencial del tipo:

$$y'(t) = f(t, y(t))$$
$$y(t_0) = y_0$$

donde $y'(t) = dy/dt$ es la derivada de y con respecto a t, y $y(t_0) = y_0$ es lo que se denomina condición inicial. Tanto la función f como los valores t_0 e y_0 son conocidos.

Considerando $n + 1$ puntos t_i, con $0 \leq i \leq n$, el tiempo es discretizado en intervalos de ancho Δt calculado según:

$$\Delta t = t_i - t_{i-1} = \frac{t_n - t_0}{n}$$

por lo que es posible obtener cualquier valor t_i de acuerdo a $t_i = t_0 + i\ \Delta t$. Con esto mente, resolver numéricamente un problema de valor inicial se reduce a encontrar los valores $y_i = y(t_i)$, obteniendo así una aproximación discreta de la solución a la ecuación diferencial.

El método de Euler consiste en resolver un problema de valor inicial, considerando la discretización temporal descrita, mediante:

$$y_{i+1} = y_i + \Delta t\ f(t_i, y_i)$$

por lo cual, las aproximaciones a la solución son obtenidas mediante:

$$y_1 = y_0 + \Delta t\ f(t_0, y_0)$$
$$y_2 = y_1 + \Delta t\ f(t_1, y_1)$$
$$\vdots$$
$$y_n = y_{n-1} + \Delta t\ f(t_{n-1}, y_{n-1})$$

2. Desarrollo

A continuación, a través de CUDA kernels y funciónes secuenciales en CPU, usted resolverá dos problemas de valor inicial distintos mediante el método de Euler.

1. Comenzando por un caso sencillo, considere el siguiente problema de valor inicial:

$$y'(t) = e^{-t}$$
$$y(0) = -1$$

cuya solución exacta es $y(t) = -e^{-t}$ (puede utilizar esta solución para comprobar que sus implementaciones estén haciendo lo correcto). Considere también que el tiempo máximo hasta el cual se requiere aproximar es $t_n = 10$, por lo que para $n+1$ puntos t_i, se tiene que $\Delta t = 10/n$.

Este problema es muy sencillo de resolver, pues $f(t, y(t))$ es independiente de $y(t)$, i.e. $f(t, y(t)) = f(t)$, por lo cual, para cualquier i, con $0 \leq i \leq n$, se tiene:

$$\begin{aligned}
y_i &= y_{i-1} + \Delta t\ f(t_{i-1}) \\
y_i &= (y_{i-2} + \Delta t\ f(t_{i-2})) + \Delta t\ f(t_{i-1}) \\
y_i &= ((y_{i-3} + \Delta t\ f(t_{i-3})) + \Delta t\ f(t_{i-2})) + \Delta t\ f(t_{i-1}) \\
&\vdots \\
y_i &= y_0 + \Delta t \sum_{j=0}^{i-1} f(t_j) = -1 + \Delta t \sum_{j=0}^{i-1} e^{-(j\ \Delta t)}
\end{aligned}$$

Esto es de bastante utilidad, pues significa que es posible calcular los n valores y_i, con $1 \leq i \leq n$, de forma independiente (aparentemente una gran oportunidad para paralelizar).

a) Implemente una función secuencial de C/C++ que reciba la condición inicial (t_0 y y_0) y el valor de Δt, y almacene en un array los n valores y_i correspondientes, con $1 \leq i \leq n$. Mida el tiempo que demora la función en realizar su trabajo para $\Delta t = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$.

b) Implemente un CUDA kernel que reciba los mismos parámetros que la función anterior y obtenga los valores y_i de forma paralela, almacenándolos en un array. Mida el tiempo que demora el kernel en realizar su trabajo para $\Delta t = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ (es posible que para valores más pequeños de Δt, como 10^{-5} o 10^{-6}, la GPU no funcione bien debido al exceso de trabajo que supone el cálculo de la sumatoria realizado por cada hebra).

   [Pregunta] ¿Cuál de las dos implementaciones realizadas muestra un mejor desempeño? ¿Fue acaso conveniente utilizar la GPU para este trabajo? Comente.

c) Cree una solución híbrida al problema, en donde se utilice una función de CPU para calcular y almacenar en un array las n sumatorias. Luego, implemente un CUDA kernel que calcule los n valores y_i con los valores ya precomputados de las sumatorias. Mida el tiempo que demora esta solución en ejecutarse para $\Delta t = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. El tiempo medido debe considerar el cálculo en CPU, el traspaso de memoria de los valores precomputados a GPU, y la ejecución del kernel.

   [Pregunta] ¿Cómo se comporta esta solución en relación a las anteriores? Realice un gráfico o tabla en donde se muestren los tiempos de los 3 métodos implementados respecto al valor de Δt que se utilizó para cada medición. Si opta por realizar un gráfico, obtendrá 3 curvas distintas (una por cada implementación), conteniendo el eje x los valores de Δt, mientras que el eje y corresponderá a los tiempos medidos. *Hint: considere utilizar una escala logarítmica si las curvas no se aprecian correctamente.*

   [Pregunta] Imagine el caso en que para la misma ecuación diferencial y tiempos t_i, se requiere resolver muchos problemas de valor inicial que se diferencian por su condición inicial y_0. Argumente a favor de la solución que usted considera se demoraría menos en obtener los resultados.

2. Considere ahora el siguiente caso:

$$y_j'(t) = 4t - y_j(t) + 3 + j$$
$$y_j(0) = j$$

con $0 \leq j \leq m$ y cuya solución exacta es $y_j(t) = e^{-t} + 4t - 1 + j$. En este caso, cada y_j es una función, por lo que, en otras palabras, usted se enuentra frente a m problemas de valor inicial distintos:

$$\begin{aligned}
y_0'(t) &= 4t - y_0(t) + 3 & y_0(0) &= 0 \\
y_1'(t) &= 4t - y_1(t) + 3 + 1 & y_1(0) &= 1 \\
y_2'(t) &= 4t - y_2(t) + 3 + 2 & y_2(0) &= 2 \\
&\vdots & & \\
y_m'(t) &= 4t - y_m(t) + 3 + m & y_m(0) &= m
\end{aligned}$$

Para resolver esto con el método de Euler, considerando $y_{j,i} = y_j(t_i)$, se tiene que:

$$\begin{aligned}
y_{j,i} &= y_{j,i-1} + \Delta t \; f(t_{i-1}, y_{j,i-1}) \\
&= y_{j,i-1} + \Delta t \; (4t_{i-1} - y_{j,i-1} + 3 + j)
\end{aligned}$$

con $0 \leq i \leq n$ y $0 \leq j \leq m$. Es posible observar que, lamentablemente, esta vez f sí depende de las funciones y_j, por lo que, para un valor de j fijo, no se pueden calcular los n valores $y_{j,i}$ de forma independiente, pues cada uno necesita conocer el anterior $y_{j,i-1}$. La buena noticia es que, para un i fijo, sí se pueden calcular los m valores $y_{j,i}$ a la vez.

Para enfrentar este problema, considere $t_n = 1$ y $\Delta t = 10^{-3}$, es decir, $n = 1/10^{-3} = 10^3$.

a) Implemente una función secuencial de C/C++ que reciba el valor de m, el array con los m valores $y_{j,i}$ así como también el valor de Δt y t_i actual. Esta función debe realizar un paso de tiempo actualizando los m valores $y_{j,i}$ contenidos en el array a su correspondiente $y_{j,i+1}$. Su función debe ser llamada desde un ciclo que realice n iteraciones en las cuales el array con las aproximaciones debe irse sobreescribiendo. Mida el tiempo que demora el ciclo completo en realizar su trabajo para $m = \{10^4, 10^5, 10^6, 10^7, 10^8\}$. Antes de comenzar el ciclo, debe crear un array de tamaño m e inicializarlo con los valores $y_{j,0} = j$, es decir, `array[j] = j`. Se recomienda crear inmediatamente (antes de sobreescribirlo) una copia de estas condiciones iniciales en GPU para la siguiente pregunta. Este paso de generación de los datos no debe ser considerado en las mediciones de tiempo.

b) Implemente un CUDA kernel que reciba los mismos parámetros que la función anterior y actualice una vez los m valores $y_{j,i}$ de forma paralela, sobreescribiendo el array. Al igual que en el caso anterior, realice n llamadas al kernel dentro de un ciclo. Mida el tiempo que demora el ciclo completo en realizar su trabajo para $m = \{10^4, 10^5, 10^6, 10^7, 10^8\}$.

[Pregunta] ¿Cuál de las dos soluciones es más rápida? ¿A qué se debe esto? Comente los resultados y realice un gráfico o tabla en donde se muestren los tiempos de ambas implementaciones respecto al valor de m que se utilizó para cada medición. Si opta por realizar un gráfico, obtendrá 2 curvas distintas (una por cada implementación), conteniendo el eje x los valores de m, mientras que el eje y corresponderá a los tiempos medidos. *Hint: considere utilizar una escala logarítmica si las curvas no se aprecian correctamente.*

c) Utilizando el mismo kernel de la pregunta anterior. Con un valor de $m = 10^8$ fijo, mida los tiempos que demora el ciclo completo al variar el número de hebras por bloque entre los valores 64, 128, 256 y 512.

[Pregunta] ¿Con que configuración se obtuvo el mejor resultado? Concluya sobre lo observado. Realice un gráfico o tabla en donde se muestren el tiempo medido respecto a los tamaños de bloque utilizados. Si opta por realizar un gráfico, obtendrá 1 curva, conteniendo el eje x los tamaños de bloque, mientras que el eje y corresponderá a los tiempos medidos.

3. Reglas y Consideraciones

Entrega

- La entrega debe realizarse en un archivo de nombre Lab1-X.tar.gz (formatos rar y zip también son aceptados), donde X debe ser reemplazado por el número de su grupo. Diríjase a la inscripción de grupos para consultar su número.
- El archivo de entrega debe contener un informe en formato pdf junto con el código implementado para resolver el laboratorio. Se le ruega entregar un código ordenado.
- El informe debe contener:
  - Título y número del laboratorio.
  - Nombre y rol de todos los integrantes del grupo.
  - Modelo y compute capability de la tarjeta gráfica que fue utilizada para ejecutar el código. Si se probó con más de una tarjeta gráfica, incluya los datos de todas y específique qué tarjeta se utilizó en cada pregunta y resultado reportado.
  - Desarrollo. Preocúpese de responder cada pregunta señalada con el tag [**Pregunta**] en el enunciado.
  - Conclusiones. Incluya comentarios, observaciones o supuestos que surgieron durante el desarrollo del laboratorio.
- El descuento por día de retraso es de 30 puntos, con un máximo de 1 día de retraso. No se aceptarán entregas posteriores.
- En caso de copia, los grupos involucrados serán evaluados con nota 0. No hay problema en generar discusión y compartir ideas de implementación con sus compañeros, pero códigos copiados y pegados no serán aceptados.
- El no cumplimiento de estas reglas implica descuentos en su evaluación.
- La fecha de entrega es el día Lunes 8 de Octubre. Se habilitará la opción de entrega en aula.

Consideraciones

- Para el número e utilice una constante definida con la directiva #define en global scope (fuera de cualquier función, en la zona de los #include). Ejemplo: `#define E 2.71828182845904523536`
- Trabaje con flotantes de precisión simple (`float`).
- Para el calculo de potencias, utilice la función `float powf(float x, float y)` definida en la librería math.h. También funcionará dentro de un kernel.
- Para mediciones de tiempo, utilice `clock()` de la librería time.h en caso de mediciones en CPU y `cudaEvent()` para códigos de GPU. Trabaje con milisegundos $[ms]$.
- En caso de optar por la realización de gráficos, puede optar por la herramienta que más le acomode. La librería matplotlib de Python siempre es una buena opción.
- Los parámetros solicitados para cada función o kernel son los mínimos exigibles. Usted puede agregar los que estime conveniente siempre y cuando no impliquen un procesamiento de datos distinto al señalado. Un claro ejemplo de parámetros que usted debería agregar a los señalados son las dimensiones de los arrays.
- Utilice 256 hebras por bloque en sus implementaciones. La única excepción a esto es la última pregunta, en la cual se solicita variar este valor.