INF-351: Computación de Alto Desempeño
Laboratorio 3
Multiplicación Matriz-Vector: ¿Qué método elegir?

Prof. Álvaro Salinas

25 de Noviembre de 2018

1. Descripción y Marco Teórico

En el siguiente laboratorio serán evaluados sus conocimientos sobre el uso de memoria compartida, memoria constante y operaciones atómicas. Claramente, todo su trabajo debe verse respaldado por los métodos y buenas prácticas aplicadas en laboratorios anteriores.

Multiplicación Matriz-Vector

La multiplicación matriz-vector es una operación simple que se encuentra en muchos métodos numéricos. En realidad, es un caso especial de la multiplicación de matrices, en donde la segunda matriz posee una sola columna. De esta forma, el producto $A\mathbf{x} = \mathbf{b}$, de una matriz $A_{N\times M}$ por un vector $\mathbf{x}$ de dimensión M da como resultado un vector $\mathbf{b}$ de dimensión N. Veamos un ejemplo:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,M} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

donde $a_{i,j}$ es el elemento en la i-ésima fila y j-ésima columna de A, x_j es el j-ésimo elemento de x y b_i es el i-ésimo elemento del vector b calculado según:

$$b_i = \sum_{j=0}^{M} a_{i,j} x_j$$

Debido a su presencia en una gran cantidad de métodos numéricos, es importante conocer una buena forma de implementar el producto matriz-vector. ¡Eso es lo que haremos a continuación!

2. Desarrollo

Según el problema descrito en la sección anterior, a continuación realizará distintas implementaciones y analizará su desempeño. Con esto en mente, para cada una de las implementaciones a desarrollar considere:

- Se trabajará con una matriz cuadrada de 10^8 elementos, i.e. $N = M = 10^4$.
- El tamaño de un bloque de hebras siempre será 256. A partir de este momento, se utilizará BS para hacer referencia a este valor.
- Los valores especificados en los dos puntos anteriores pueden ser declarados como constantes en tiempo de compilación.
- La matriz A y el vector x pueden ser inicializados con los valores que guste mientras no contengan valores nulos (0). Como consejo, se le recomienda que todos los valores sean 1 para entonces poder comprobar si el resultado es correcto cuando todos los valores en b sean 10^4.

Las implementaciones de producto matriz-vector que usted debe desarrollar son:

a) `__global__ void kernelA(int *A, int *x, int *b, int N)`:

Este kernel utiliza $N \times N = 10^8$ hebras. Cada hebra está asociada a un elemento $a_{i,j}$ de la matriz A, multiplicándolo por el valor x_j correspondiente y sumando este resultado al elemento en la i-ésima posición del vector b.

b) `__global__ void kernelx(int *A, int *x, int *b, int N)`:

Este kernel utiliza $N = 10^4$ hebras. Cada hebra está asociada a un elemento x_j del vector x, sumando a cada uno de los N valores b_i, la multiplicación de dicho x_j por el correspondiente elemento $a_{i,j}$.

c) `__global__ void kernelb(int *A, int *x, int *b, int N)`:

Este kernel utiliza $N = 10^4$ hebras. Cada hebra está asociada a un elemento b_i del vector b, sumando los N productos $a_{i,j}x_j$ requeridos en una variable local y asignando este resultado al elemento b_i mencionado.

d) `__global__ void kernelRed(int *A, int *x, int *b, int N)`:

Este kernel utiliza $N = 10^4$ hebras. Al igual que en `kernelx`, cada hebra está asociada a un elemento x_j. La diferencia radica en que, en vez de que cada hebra sume un producto a cada elemento b_i, solo cada bloque realizará esta acción. Para esto, para cada uno de los N valores de i se realizarán los siguientes pasos:

- Paso 1: Cada hebra de un bloque calcula el producto $a_{i,j}x_j$ y almacena este resultado en un array en memoria compartida de tamaño BS.
- Paso 2: Cada bloque de hebras realiza una reducción de su array, quedando finalmente la suma de los BS elementos almacenada en la primera posición de dicho array.
- Paso 3: Solo la primera hebra de cada bloque escribe el resultado descrito en el Paso 2 en el elemento b_i correspondiente.

Es importante recalcar que cada uno de los tres pasos recién explicados debe realizarse para cada valor de i.

Hint: *Revise el código de Reducción y Operaciones Atómicas.*

e) `__global__ void kernelSM(int *A, int *x, int *b, int N)`:

Este kernel utiliza $N = 10^4$ hebras. Su funcionamiento es muy similar al de `kernelb`, en el sentido de que cada hebra debe sumar N productos $a_{i,j}x_j$ y almacenarlos en el elemento b_i correspondiente. La

principal diferencia es el hecho de que en esta ocasión se tomará conciencia de que todas las hebras están leyendo los mismos valores x_i, por lo que, en vez de realizar estos accesos directamente a memoria global, estos valores serán primeros almacenados en memoria compartida. Notar que para lograr que cada hebra en un bloque tenga asociado un elemento del array de memoria compartida, es necesario dividir las N sumas en grupos de BS. De esta manera, podemos explicar el kernel de la siguiente forma:

- Paso 1: Cada hebra de un bloque lee uno de los primeros BS elementos del vector x y lo almacena en su correspondiente posición del array de memoria compartida.
- Paso 2: Cada hebra de un bloque suma a una variable local los BS productos $a_{i,j}x_j$ (esta vez leyendo los valores x_j desde memoria compartida) para su valor de i asociado.
- Paso 3: Se repite el Paso 1, pero para los elementos x_i con $BS \leq i < 2BS$.
- Paso 4: Se repite el Paso 2 de forma idéntica, pero en esta ocasión los elementos de x presentes en el array de memoria compartida han sido modificados.

Y así sucesivamente hasta que se haya hecho para los N valores en x.

Hint: *Revise el último ejemplo en los apuntes de memoria compartida.*

f) `__global__ void kernelCM(int *A, int *b, int N)`:

Este kernel utiliza $N = 10^4$ hebras. Al igual que `kernelSM`, su funcionamiento es muy similar al de `kernelb`, pero en esta ocasión simplificaremos un poco las cosas. Dado que el vector x no posee un tamaño muy grande, y cada elemento de él no es solo leído por las hebras de un mismo bloque, sino por todas las hebras en ejecución, esta vez almacenaremos el vector x en memoria constante (notar que `int *x` ha desaparecido de los parámetros del kernel). El resto del código debiese ser idéntico a `kernelb`.

Verifique que con cada implementación desarrollada se obtenga el resultado correcto y ejecute la herramienta de profiler `nvprof`. Visualice los resultados en el Visual Profiler.

[**Pregunta**] ¿Cuál de las implementaciones realizadas muestra un mejor desempeño? Muestre una tabla o gráfico con los tiempos obtenidos y comente sobre ellos.

[**Pregunta**] Analice las relaciones entre los distintos tiempos observados (ejemplo: `kernelXXX` se demora más o lo mismo que `kernelYYY`). ¿Qué puede decir sobre estas relaciones? ¿A qué cree usted que se deben? No es necesario que comente sobre las 15 relaciones presentes entre los 6 kernels desarrollados, sino solo las que considere más importantes o aquellas que le parezcan extrañas o diferente a lo que hubiese esperado.

3. Reglas y Consideraciones

Entrega

- La entrega debe realizarse en un archivo de nombre Lab3-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 3 de Diciembre. Se habilitará la opción de entrega en aula.

Consideraciones

- Trabaje con arrays de enteros de 4 bytes (`int`).
- Para mediciones de tiempo utilice la herramienta Visual Profiler.
- Si lo desea, puede agregar capturas de pantalla del reporte del Visual Profiler. Éstas incluso pueden reemplazar las tablas solicitadas siempre y cuando se entienda claramente qué implementación está siendo analizada.