Informe Tarea 02 Matriz Dispersa x Vector
INFO188 - Programación en Paradigmas Funcional y Paralelo

Acádemico: Cristóbal Navarro {cnavarro@inf.uach.cl}
Instituto de Informática, Universidad Austral de Chile
Alumnos: - Cristóbal Rebolledo {cristobal.rebolledo@alumnos.uach.cl}
- Carolina Obreque {carolina.obreque@alumnos.uach.cl}
- Felipe Córdova {felipe.cordova@alumnos.uach.cl}
- Sebastián Montecinos {sebastian.montecinos02@alumnos.uach.cl}

Diciembre 19, 2023

1 Introducción

En el presente informe, abordamos la implementación eficiente y paralela del producto de una matriz dispersa por un vector, un problema fundamental en el ámbito de la computación científica y la optimización de recursos de memoria. Con el objetivo de aprovechar las ventajas de las arquitecturas tanto de CPU como de GPU, proponemos un diseño estructural innovador para representar matrices dispersas, permitiendo un procesamiento simultáneo en ambas plataformas. La solución se materializa a través de implementaciones en paralelo utilizando OpenMP para la CPU y CUDA para la GPU, proporcionando una comparativa exhaustiva en términos de velocidad y uso de memoria.

2 Diseño de la solución

2.1 Librerías

Para el correcto funcionamiento de nuestro código, importamos las siguentes librerías

- `iostream`: Para la entrada y salida estándar.
- `vector`: Para el manejo de vectores dinámicos.
- `omp.h`: Para el soporte de OpenMP y la gestión de hilos en la CPU.
- `cstdlib`: Para funciones relacionadas con la generación de números aleatorios.
- `cuda.h`: Para el desarrollo de código en CUDA.

2.2 Estructuras de datos

El diseño de la solución se basa en dos estructuras fundamentales:

- `ElementoNoNulo`: Estructura que representa un elemento no nulo de la matriz dispersa, con información sobre su fila, columna y valor.
- `MatrizDispersa`: Estructura que representa una matriz dispersa, con información sobre el número de filas, columnas y un vector de elementos no nulos.

2.3 Funciones de impresión

Se han implementado funciones específicas para imprimir vectores y matrices dispersas de manera clara y legible, facilitando la verificación de resultados y la depuración del código.

2.4 Funciones de multiplicación

Se han desarrollado funciones tanto para la multiplicación en CPU como en GPU. La función `multiplicarMatrizPorVectorCPU` realiza la multiplicación de manera secuencial en CPU utilizando OPENMP, mientras que `multiplicarMatrizPorVectorGPU` implementa la multiplicación de manera paralela en GPU utilizando CUDA.

2.5 Generación de datos aleatorios

La función `llenarMatrizDispersa` permite llenar una matriz dispersa con elementos no nulos de manera aleatoria, utilizando una densidad proporcionada como parámetro.

2.6 Paralelismo y Modos de Ejecución

El programa ofrece tres modos de ejecución:

- Modo CPU (`modo = 0`): Realiza la multiplicación en CPU utilizando OpenMP para paralelismo.
- Modo GPU (`modo = 1`): Realiza la multiplicación en GPU utilizando CUDA.
- Modo CPU y GPU (`modo = 2`): Realiza la multiplicación en ambos dispositivos simultáneamente para comparar resultados y tiempos de ejecución.

El usuario puede seleccionar el número de hilos para el modo CPU mediante el parámetro `nt`.

2.7 Compilación y ejecución

Para compilar el programa, se proporciona un Makefile. Solo deberá ingresar el comando `make` en la consola. A continuación, deberá ejecutar el programa de la siguiente forma:

`./prog <n> <d> <s> <m> <nt>`

Donde:

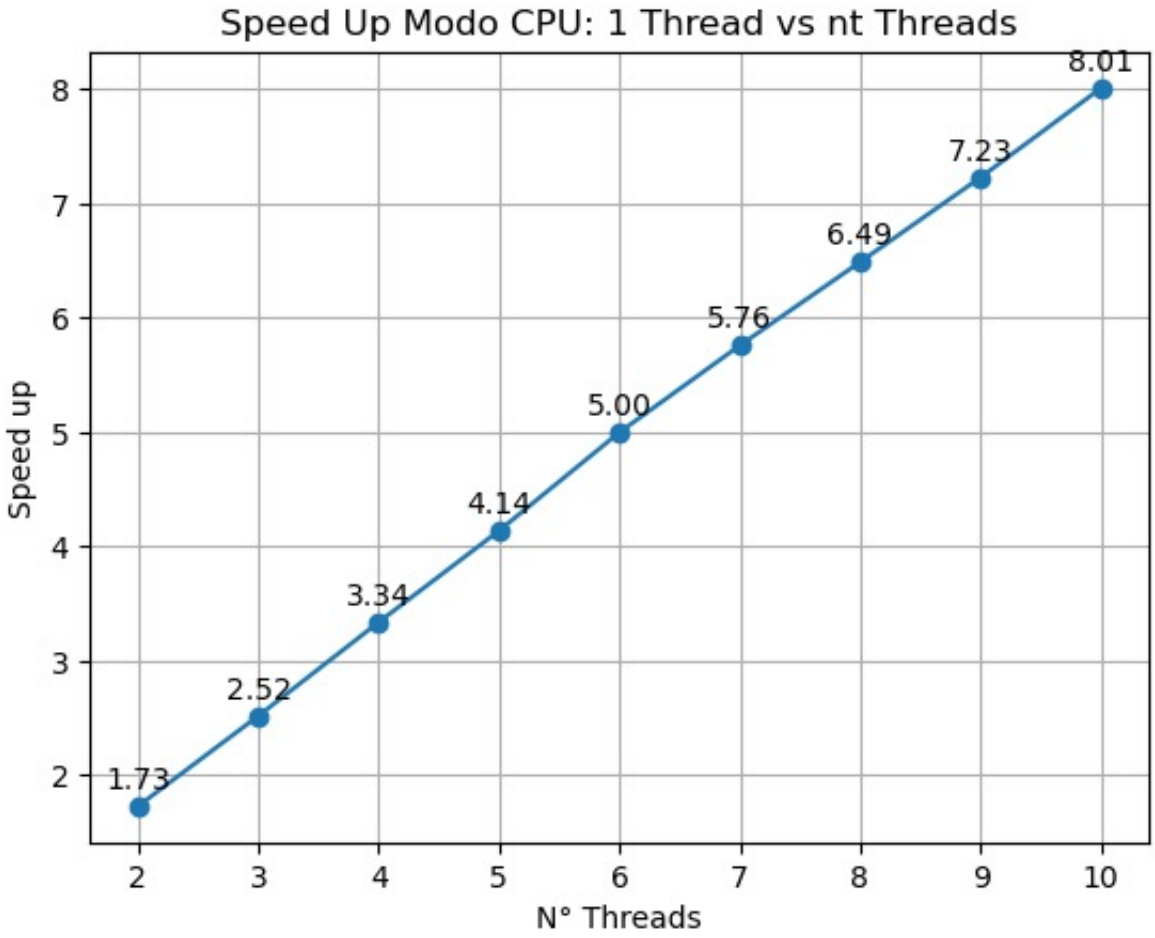
- `n` es el tamaño de la matriz (n x n).
- `d` es la densidad de números no nulos (0 ¡ d ¡ 1).
- `s` es la semilla de números aleatorios.
- `m` es el modo de ejecución (0 = modo CPU, 1 = modo GPU, 2 = modo CPU y GPU).
- `nt` es el número de threads de la CPU.

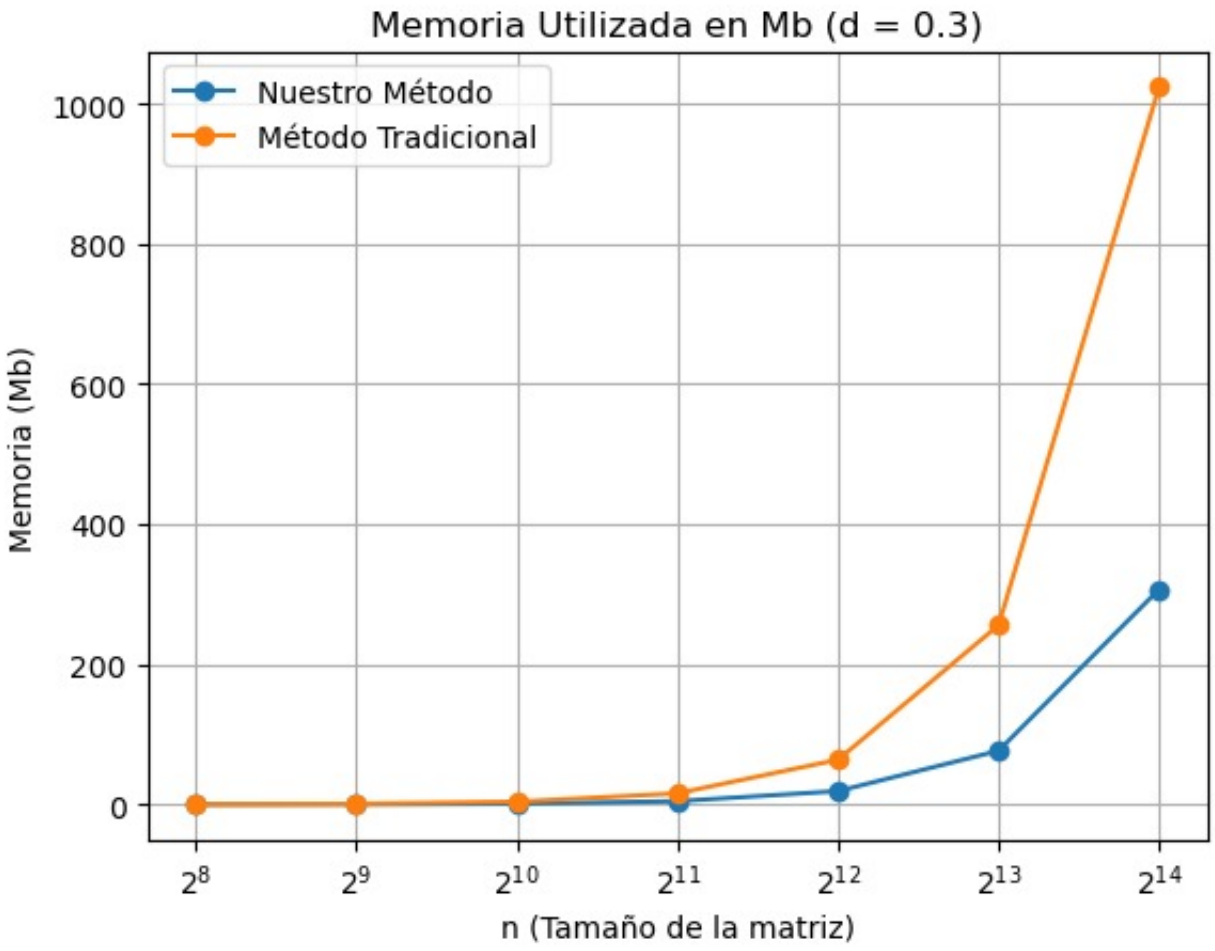
2.8 Resultados y Verificación

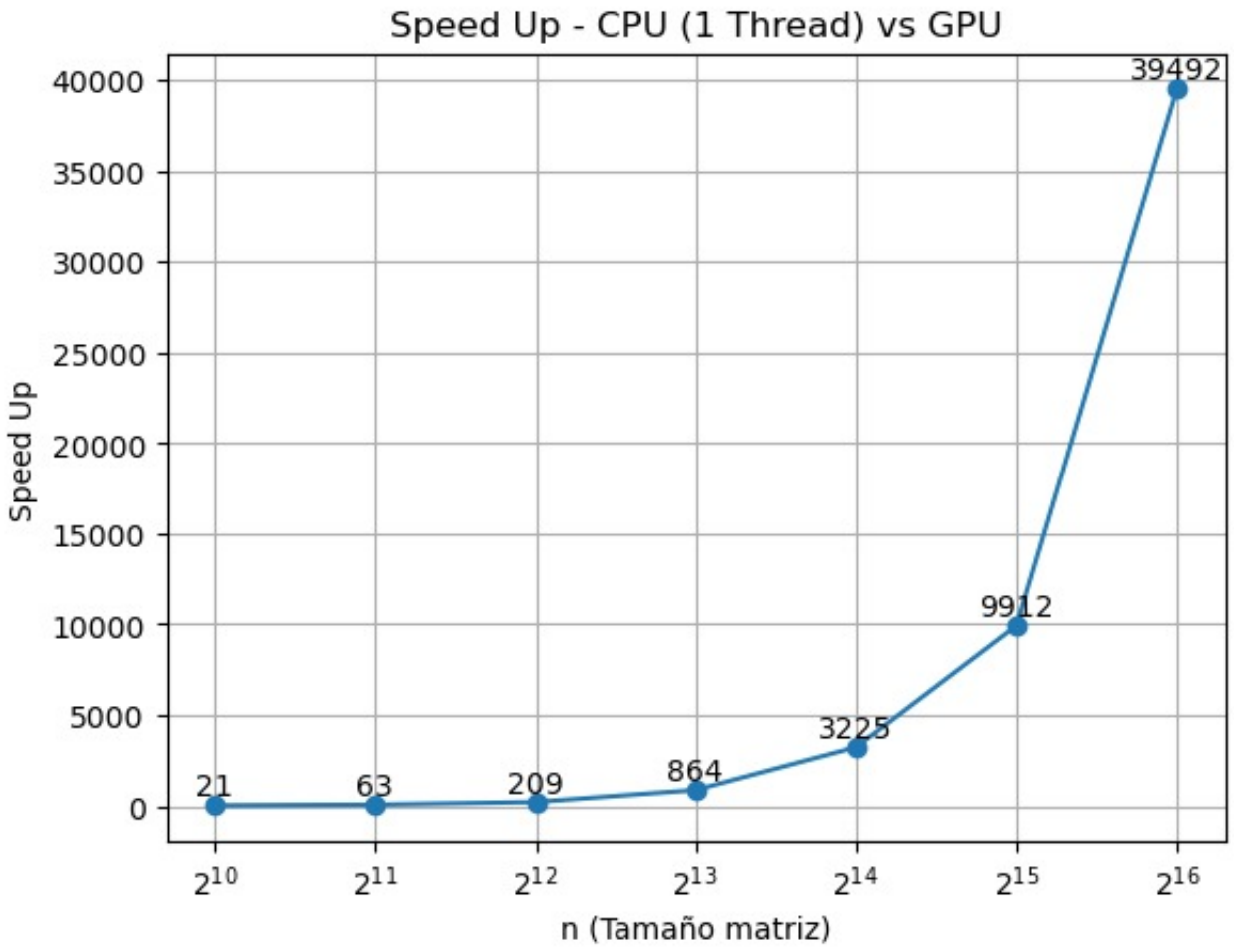
El programa imprime los resultados de la multiplicación y proporciona información sobre el tiempo total de ejecución en cada modo. Para verificar la corrección de los resultados, se compara la salida del modo CPU y GPU en el modo combinado. Además, se proporciona la proporción de elementos no nulos para verificar la densidad de la matriz generada aleatoriamente.

Este diseño busca aprovechar al máximo las capacidades de paralelismo de CPU y GPU, brindando una solución eficiente y comparativa para el problema del producto de una matriz dispersa por un vector.

A continuación se presentan los resultados obtenidos mediante la experimentación.


Speed Up Modo CPU: 1 Thread vs nt Threads
Speed up
N° Threads
1.73
2.52
3.34
4.14
5.00
5.76
6.49
7.23
8.01


Memoria Utilizada en Mb (d = 0.3)
Nuestro Método
Método Tradicional
Memoria (Mb)
1000
800
600
400
200
0
2^8
2^9
2^{10}
2^{11}
2^{12}
2^{13}
2^{14}
n (Tamaño de la matriz)



3 Conclusión

En este proyecto, abordamos de manera efectiva el desafío del producto de una matriz dispersa por un vector, aprovechando las capacidades de paralelismo de CPU y GPU. La implementación eficiente de este problema fundamental en el ámbito de la computación científica y la optimización de recursos de memoria se llevó a cabo mediante un diseño estructural innovador y el uso de tecnologías como OpenMP y CUDA.

La utilización de las estructuras de datos ElementoNoNulo y MatrizDispersa demostró ser esencial para representar de manera eficiente matrices dispersas y facilitar la implementación de las funciones de multiplicación en ambos dispositivos. La generación de datos aleatorios mediante la función llenarMatrizDispersa permitió evaluar el rendimiento del programa en situaciones diversas.

La posibilidad de ejecución en tres modos diferentes (CPU, GPU, CPU y GPU) brinda flexibilidad al usuario, permitiéndole ajustar la configuración según sus necesidades y recursos disponibles. La comparativa exhaustiva de resultados y tiempos de ejecución en estos modos proporciona una visión clara del rendimiento relativo de la CPU y la GPU.