Universidad de La Laguna

# Programación en GPU con la plataforma CUDA

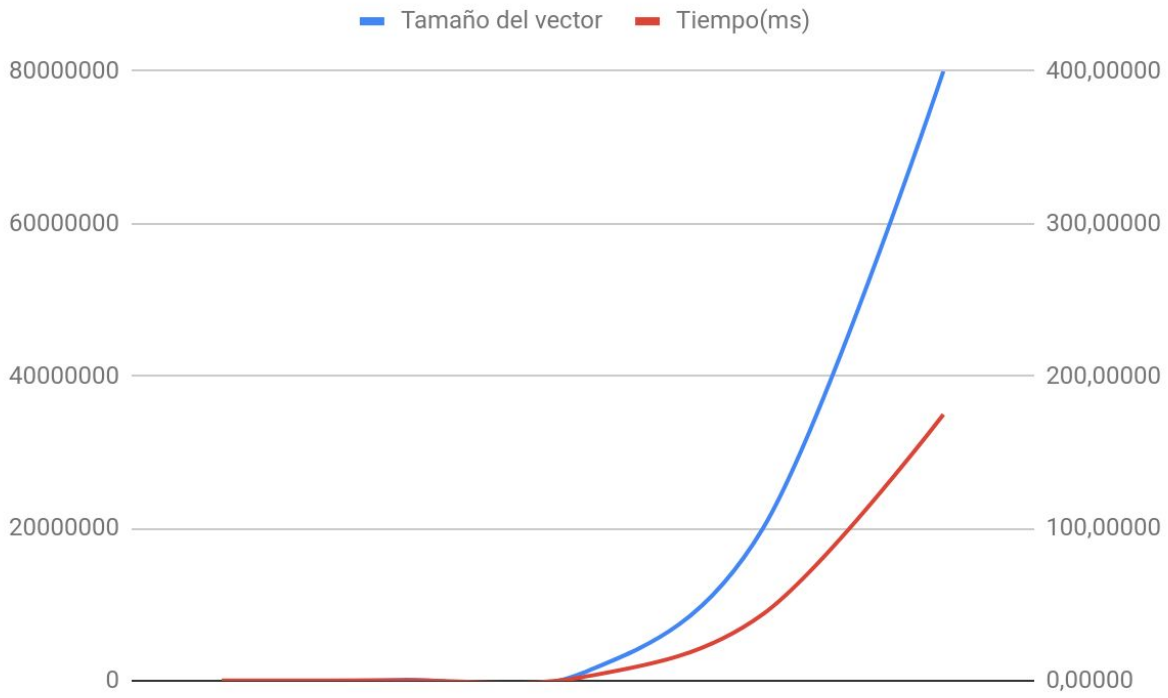
## Análisis de rendimiento en el cálculo de un histograma

Martín Belda Sosa

En esta práctica se verán distintas implementaciones para el cálculo de un histograma y se analizarán los tiempos que requiere cada una.

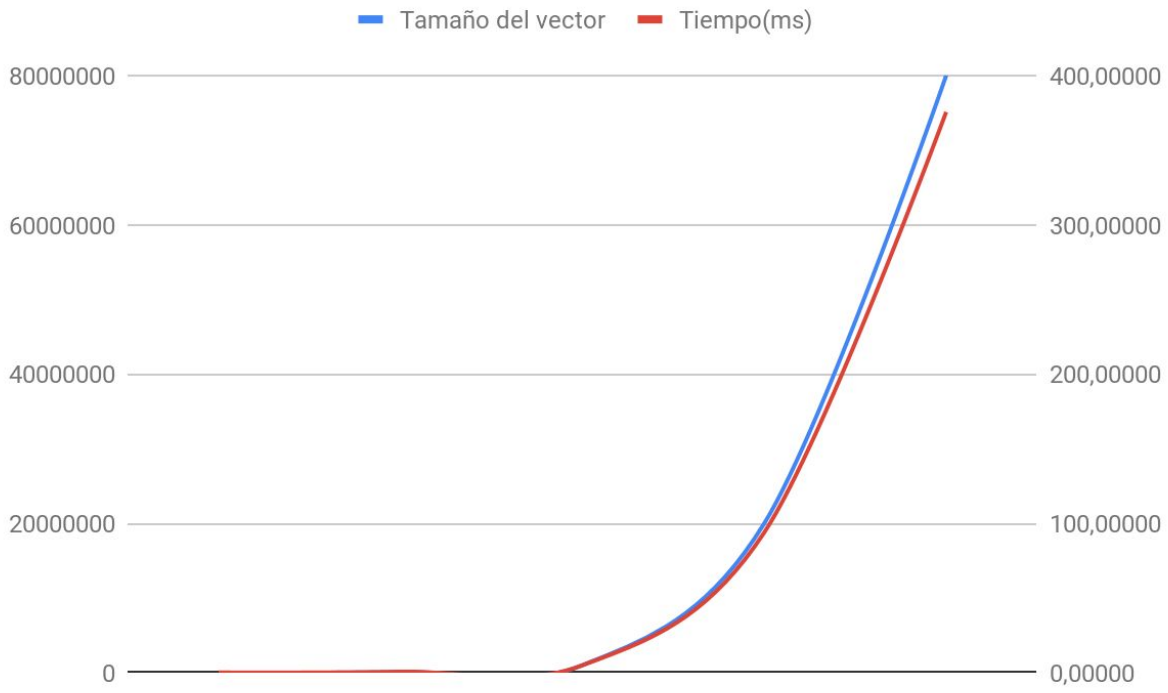
## Primera implementación: usando un único histograma global

En esta implementación se usará un único histograma global sobre el que se realizará la cuenta de los elementos del vector con una operación atómica. En teoría las operaciones se serializarán debido a que muchos hilos estarán intentando escribir en la misma posición del histograma.



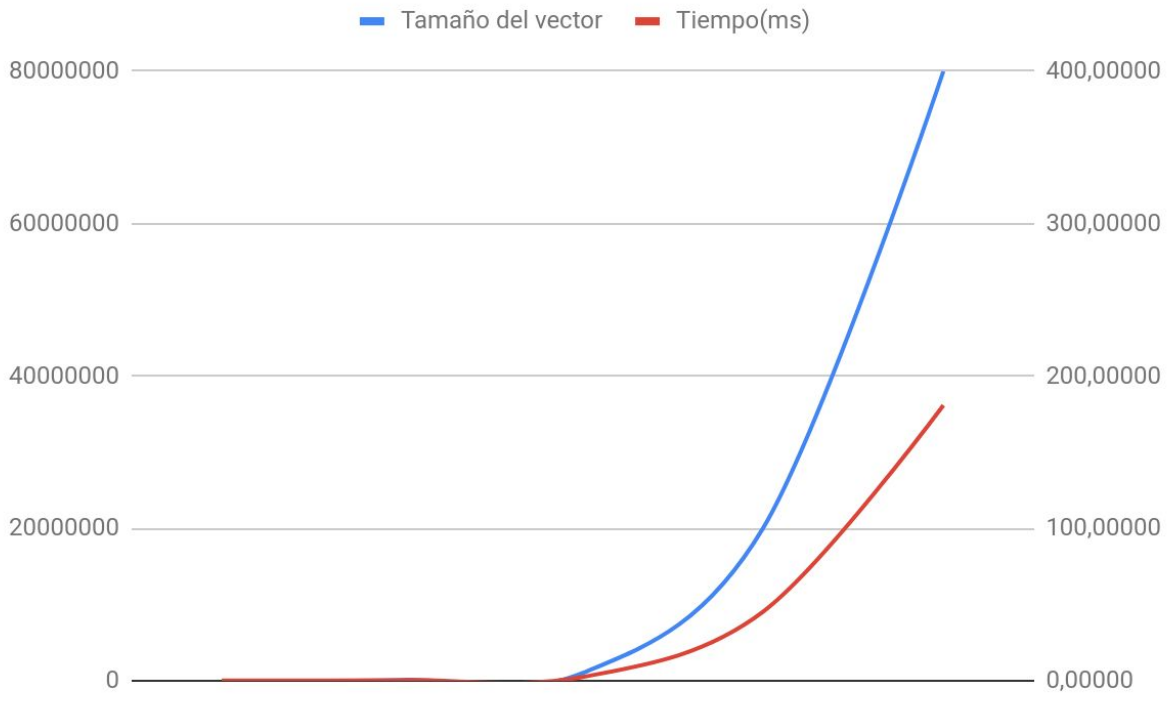
## Segunda implementación: histogramas locales y suma por reducción

Esta implementación tiene un determinado número de histogramas locales entre los que se reparten subconjuntos del vector para tratar de aumentar el paralelismo a la hora de hacer operaciones atómicas. Una vez calculados los histogramas locales se hace una suma por reducción para calcular el histograma global final. Las pruebas se han hecho con 10 histogramas locales.



## Tercera implementación: histogramas locales y suma por reducción con memoria compartida

Esta implementación trata de aprovechar la arquitectura de la GPU usando un histograma local por cada bloque, de esta manera se puede almacenar el histograma local en memoria compartida acelerando los accesos y sincronizar los hilos mediante barreras. En este caso, debido a que hay un elevado número de histogramas locales se ha optado por hacer una reducción basada en operaciones atómicas y no en recursividad porque se han obtenido los mejores resultados.



## Conclusiones

Los mejores resultados se obtienen para la primera implementación, donde hay un único histograma que es incrementado mediante operaciones atómicas por todos los hilos.

La peor implementación es la segunda, donde hay 10 histogramas locales sobre los que se realizan las operaciones atómicas para luego sumarlos en un histograma global.

La tercera implementación tiene un rendimiento similar a la primera a pesar de que está pensada para aprovechar la arquitectura y usa memoria compartida para acelerar los accesos.

Es posible que la arquitectura de la GPU optimice las operaciones atómicas de forma que su ejecución tenga un rendimiento similar a la ejecución paralela de operaciones. Esto explicaría el motivo por el cual la primera implementación tiene el mejor de los tres rendimientos.

### Recursos


https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

https://devblogs.nvidia.com/cooperative-groups/

https://devblogs.nvidia.com/gpu-pro-tip-fast-histograms-using-shared-atomics-maxwell/

https://devblogs.nvidia.com/cuda-dynamic-parallelism-api-principles/

https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

https://devblogs.nvidia.com/introduction-cuda-dynamic-parallelism/

https://devblogs.nvidia.com/faster-parallel-reductions-kepler/