Una perspectiva general sobre la gestión de memoria en arquitecturas de GPU

Rodrigo Stevenson Regla
Instituto de informática UACh
Valdivia, Chile
rodrigo.stevenson@alumnos.uach.cl

RESUMEN

En este trabajo de investigación se presenta la gestión de memoria en arquitecturas de GPU como propuesta a investigar con el fin de analizar los modos de gestión, en particular en este trabajo para un solo dispositivo, por lo cual se hará detalle de cual es el estado actual de la gestión, posteriormente se hará un planteamiento del problema con el fin de detallar como abordarlo, explicando además la implementación del problema del diagrama de Voronoi, incluyendo configuración de memoria; para someterlo a pruebas que responda a lo establecido previamente, por último se detallaran los resultados en los cuales la gestión manual rinde mejor que la gestión automática, pero sin tener diferencias significativas entre los modelos de tarjetas usados, cerrando todo esto se hace un análisis de lo observado en el experimento junto a una reflexión de que aspectos mejorar y como continuar a futuro.

1. INTRODUCCIÓN

El computo en unidades de procesamiento gráfico o GPU ha cobrado cada vez mas relevancia, esto en base a la alta capacidad de realizar operaciones aritméticas en comparación a los procesadores centrales o CPU, lo cual es muy atractivo para muchos científicos e investigadores ya que implica que simulaciones que toman horas en procesamiento normal, incluso aplicando paralelización, se pueden realizar en mucho menor tiempo técnicas de paralelismo en GPU.

El lider actual en el mercado de estos dispositivos es la multinacional NVIDIA y una de sus grandes novedades se da cuando presentan una nueva arquitectura para las tarjetas ya que esto suele suponer un aumento en las prestaciones de las mismas lo que significa mejores entornos para simulaciones y programas pesados que lo requieren.

En particular en el área de computación de alto rendimiento o HPC, el computo en GPU ha sido un tema muy abordado y bastante discutido, por ende es el área que mas se dedica a analizar las propiedades de estos dispositivos para procesar computo y como utilizarlas de manera eficiente, y uno de estos puntos es la gestión de memoria, la cual abarca todo lo relacionado con traspaso de datos, localización de los mismo, ordenamiento de los sectores de memoria y todo lo que se relacione con manipulación de datos, punteros o sectores de memoria, cabe mencionar que de aquí en adelante nos referiremos a la GPU como dispositivo y a la CPU como host. Ante esto, CUDA, la API de NVIDIA para procesamiento gráfico dispuso de 3 modos de gestión, de los cuales se tiene un modo automático llamado cudaMallocManaged, este modo lo que hace es de cierta manera compartir el segmento de memoria entre host y dispositivo de modo que cualquier modificación hecha en alguno de los dos lados se debe reflejar en el otro. El otro modo es el manual y en este se tienen dos formas, la primera es para un solo dispositivo, que se define en un principio por cudaMalloc, y partir de esto, todo lo que implique localización de memoria, traspaso de datos y coordinación entre dispositivo y host tiene que definirlo el programador; el segundo es para varios dispositivos, definido en primera manera como cudaHost y tiene las misma implicaciones del primero.

En particular en este trabajo se buscara investigar la gestión en un solo dispositivo, para esto se analizara el rendimiento entre dos modelos de tarjetas de distintas arquitecturas con el fin de apreciar diferencias entre estas, y también observar que tanto difieren los modos de gestión de memoria, todo esto con el fin de responder las siguientes interrogantes: Como se comparan en rendimiento las distintas arquitecturas? Cual modo de gestión refleja mejores resultados en ambas tarjetas? En lo que sigue de informe se detallara la metodología, desarrollo y resultados de pruebas que permitirán responder estas interrogantes, el interés de esto surge a partir de las discusiones observadas en las conferencias[1],[2].

2. METODOLOGÍA

Hay cuatro puntos claves que tratar en la metodología, el primero es como se va plantear el estudio, esto con el fin de establecer como llevar a cabo y ejecutar el mismo. El segundo punto es analizar el hardware de pruebas a usar ya que es bastante importante detallar con que elementos realizaremos las pruebas, hacer esto puede servir a futuro para sacar conclusiones una vez obtenidos los resultados. En tercer lugar se presentara el problema que se usara en las pruebas, se trata de los diagramas de Voronoi, y se detallara un poco acerca de que trata y la implementación que se usara para este trabajo. Por ultimo se detallara como se realizaran las pruebas una vez todo el entorno haya sido configurado.

2.1. Planteamiento

Como se definió en la introducción, los puntos claves de esta investigación se basan en la comparación de arquitecturas y de los modos de gestión de memoria, por ende, toda la definición y configuración de pruebas se hará en torno a esto. Por razones de complejidad y tiempo se hará el estudio en un solo dispositivo, por lo cual solo se implementaran los modos de gestión automático y manual para un solo dispositivo, esto también acotara el alcance del estudio ya que se excluyen modelos distribuidos de computo. Para este experimento se dispondrá del computador Çarbon"del Instituto de Informática de la Universidad Austral de Chile, los parámetros relacionados a las pruebas se definirán acorde a las prestaciones que ofrezca esta maquina.

2.2. Hardware de pruebas

Como se menciono, se usara un computador de una institucion para este experimento, en particular lo mas importante y lo que mas importara en este caso son las tarjetas que este sistema dispone, se tratan de dos tarjetas, la primera es una NVIDIA TITAN RTX arquitectura Turing del 2018, la segunda es una NVIDIA TITAN V arquitectura Volta del 2017, en la tabla.1 se pueden apreciar de mejor manera algunos de los detalles relevantes de ambas tarjeta.

Cuadro 1: Especificaciones de tarjetas grafica

Atributo	TITAN RTX	TITAN V
Arquitectura	Turing TU102	Volta GV100
CUDA Cores	4608	5120
Tensor Cores	576	640
RT Cores	72	-
FP32 Compute	16.3 TFLOPS	13.8 TFLOPS
Memory Bandwith	672 GB/s	653 GB/s
Memory Clock	14 Gbps	1.7 Gbps
Memory Bus	384-bit	3072-bit
Memory Capacity	24 GB	12 GB

Como se puede apreciar en la tabla.1, la RTX se presenta con mejores prestaciones en lo que a traspaso de datos y computo en FP32 respecta, mientras que la V presenta una mayor cantidad de CUDA cores y tensor cores, aunque para este trabajo sirven mas de referencia la cantidad de CUDA cores.

Como punto de referencia considerando el ano de salida de las respectivas arquitecturas y las prestaciones de cada tarjeta, a primera vista la TITAN RTX debería mostrarse mas potente en las pruebas.

2.3. Diagrama de Voronoi

El diagrama de Voronoi sera el conejillo de indias de esta investigación, pues este sera el problema que se usara en las pruebas para hacer las respectivas comparaciones.

El problema en si consiste en formar un plano bidimensional o tridimensional de polígonos a partir de un lote de semillas; las aristas de los polígonos formados serán equidistantes a las semillas de los poligonos respectivos como se aprecia en la figura.1.

Este problema forma parte del grupo de problemas basicos de la geometria computacional junto con el Convex Hull y la triangulacion de Delaunay, puesto que problemas de mayor grado se pueden representar con estas simulaciones basica, de hecho para el caso de la triangulacion de Delaunay, esta muy relacionada con el diagrama de Voronoi a tal punto que a partir de uno se puede obtener el otro.

Las aplicaciones del diagrama de Voronoi(en adelante los mencionaremos como VD) se extienden a bastantes áreas tales como la biología, la cristalografía y la geografía, en el caso de las dos primeras se le da bastante uso debido a la similitud de las formas de la naturaleza con los VD como conjuntos celulares, estructuras de paneles, formas cristalinas entre otros; en el caso de la geografía como se podría esperar se aprecia bastante en la confección de mapas, por ejemplo se puede usar VD para establecer áreas cercanas a aeropuertos usando estos mismos como semillas, el resultado

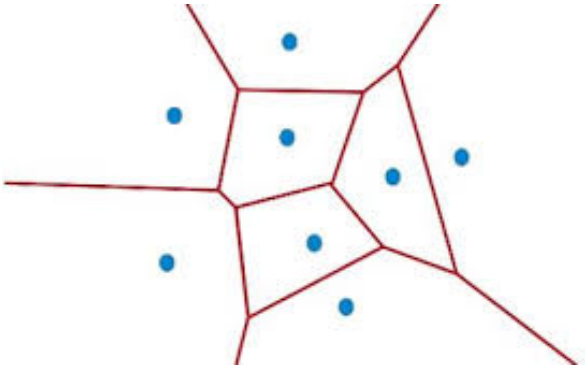

Figura 1: Representación en un plano de dos dimensiones del diagrama de Voronoi

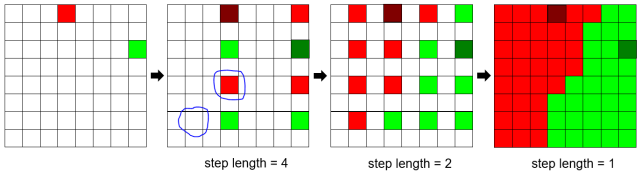

Figura 2: Construcción de un diagrama de Voronoi usando Jump Flooding Algorithm

serviría de guía para los aviones que en caso de emergencia tengan que buscar un aeropuerto cercano donde aterrizar; como se puede observar de acuerdo lo mostrado, los VD tienen mucho impacto en aspectos de la vida real.

Los algoritmos para construir VD son diversos, por ejemplo el mas eficiente es el de Fortune que logra construir un VD con una complejidad de O(nlog(n)), sin embargo este algoritmo no es muy óptimo para el computo de GPU puesto a que se limita a un barrido lo cual no termina siendo muy eficiente, en cambio los algoritmos de inundación o flooding son una propuesta mas atractiva para el paralelismo dado que se puede abarcar desde varios puntos a la vez, la idea es que a partir de cada semilla, se vaya llenando la información de los demás puntos en base a la semilla mas cercanan. Dentro de este tipo de algoritmos destaca el Jump Flooding Algorithm o JFA el cual logra crear un VD en similar a O(nlog(n)) como se demuestra en [3],[4] gracias a su aplicación en hardware de procesamiento gráfico.

La representación de como funciona un JFA se puede apreciar en la figura.2, la cantidad de pasos a ejecutar viene definida por un valor k que se calcula de la siguiente forma:

$$k = 2^{\lceil log2(n) \rceil - 1} \tag{1}$$

Donde n es el largo de la dimensión del cuadro que almacena todos los puntos, respecto a esto se usara un mapa de nxn, y se usara r para representar la cantidad de semillas; una vez configurado el mapa el algoritmo en si partirá usando las r semillas para empezara a marcar a sus 8 vecinos que tengan una coordenada en el eje x o y a k distancia, luego en la siguiente iteración el k se divide en dos y

los nuevos elementos marcados junto a las semillas marcaran mas elementos vecinos usando la misma lógica hasta que k sea igual a 1.

Para que este algoritmo haga sentido en el experimento, se establecerán dos modos de configuración, el primero sera el modo 0 el cual se referirá a la gestión automática y inicuamente se establecerá el cudaMallocManaged dejando la gestión en manos de la API; el segundo sera el modo 1 correspondiente a la gestión manual, adicionalmente se incluirán directivas para definir cuando enviar y recibir datos tanto en host como en el dispositivo.

2.4. Parámetros de prueba

Analizando las capacidades del hardware, en particular la capacidad de memoria de la GPU, se fue calculando que tanto espacio iban a ocupar en dispositivo el mapa entero de los puntos y el arreglo de semillas, y de acuerdo a lo estudiado se determino los siguientes rangos tanto para n como para r:

$$n \in \{8192, 16384, 32768\} \tag{2}$$

$$r \in \{256, 512, 1024, 2048\} \tag{3}$$

De acuerdo a lo estimado, probar con potencias de dos mayores a 32768 va a exceder la capacidad que permiten ambas tarjetas por lo que se establece ese valor como cota máxima, en tanto la elección de 8192 como cota mínima es porque sino el problema a resolver va ser muy pequeño y sus resultados no tendrían impacto para el estudio. Con el objetivo de tener resultados solidos, cada combinación de n y r sera puesta a prueba 10 veces, luego los resultados se promediaran y de esto se obtendrá el rendimiento para la combinación de n y r.

3. DESARROLLO

Esta sección constara de dos partes, en la primera se detallara como se estructuro la implementación para construir el diagrama de Voronoi y la segunda se explicara la configuración en memoria para cada modo.

3.1. Implementación de VD

Una vez estudiado y definido como se abarcara este problema se procede a implementar en codigo, el cual se divide en 3 secciones:

1. Crear semillas(Host)
2. Crear mapa(Dispositivo)
3. Crear VD usando JFA(Dispositivo)

Como se aprecia, se define también en que lado del hardware va estar cada sección, para la creación de semillas se estimo que la carga era baja incluso con el valor máximo de r definido, por lo que se opto por dejarlo en host, la creación del mapa es relevante ya que de acuerdo al modo elegido pueden ocurrir mas o menos coordinaciones entre host y dispositivo, para su implementación en si se opto por el dispositivo ya que al tratarse de valores de n grandes puede tomar buen tiempo en host, por ultimo el JFA se deja en dispositivo ya que al aplicarse en este sera mas rápido y además los cambios en cada iteración son lo que impactaran en la diferencia de rendimiento entre los dos modos de gestión. La duración del JFA como ya se menciono estará definido por el valor k obtenido en (1), y este también tiene su impacto en la posterior prueba ya que definirá la cantidad de veces que habrá que marcar vecinos y

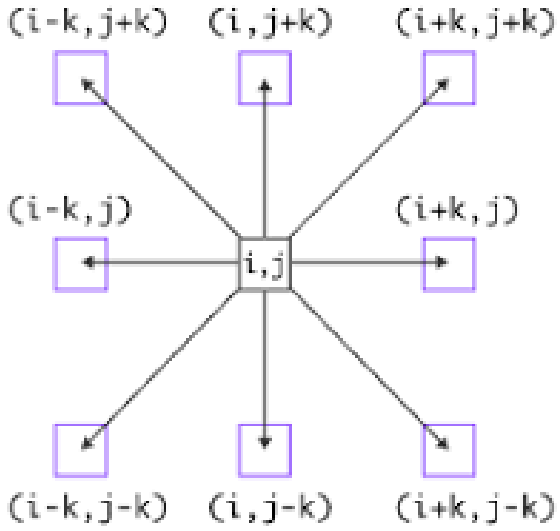

Figura 3: Los ocho k vecinos de un punto para JFA

a cuales marcar, respecto a estos, los vecinos a marcar se definen como en la figura.3.

Respecto a detalles mas técnicos usando CUDA, se programara de forma que cada thread mapee un píxel del mapa, el cual para poder manipularlo adecuadamente dada la naturaleza del problema se mapeara usando como base un bloque y una grilla bidimensional, para bloque se usara el máximo de hilos, osea 1024 threads en cada bloque por lo que los bloques se definen de 32x32.

Cabe mencionar que la elección de potencias de 2 para casi todos los parámetros relacionados al JFA y para los bloques viene dado por que los problemas con valores de potencias de 2 tienen buena afinidad con CUDA y las pruebas serian mas consistentes.

3.2. Configuración de memoria

Como se menciono ya, deben haber por lo menos dos espacios de almacenamiento, uno para el mapa y otro para las semillas y esto se refleja directamente en el codigo ya que para los dos modos de gestion se establecen dos arreglos:

1. map: Almacena todos los puntos y su información.
2. indx: Almacena información de las semillas.

Para el modo 0 solo se necesitan estos dos arreglos, puesto que la política de la gestión automática establece que esos arreglos y los cambios efectuados en estos serán visibles tanto para el host como para el dispositivo. En el caso del modo 1 se requerirán arreglos extras debido a que la gestión de memoria queda a manos del programador y no hay un arreglo que sea visible para el host y para el dispositivo a la vez, por lo cual se definen otro dos arreglos extras:

1. GPUmap: Versión en GPU de map.
2. GPUindx: Version en GPU de indx.

Estos son las únicos arreglos de memoria relevantes para este problema y sobre estas se definirá cual gestión y cual arquitectura rinde mejor, por un lado al incrementar la cantidad de semillas automáticamente habrá que hacer una mayor cantidad de modificaciones al mapa desde el primer valor de k lo que podría repercutir en la gestión automática, y por otro lado al aumentar el tamaño de n, el problema crece y es mas complejo para los dos modos.

4. RESULTADOS

Antes de mostrar los resultados definitivos, se hicieron unas pequeñas pruebas de análisis previo a las pruebas de interés y se detecto que el valor de 8192 para n con cualquier valor de r no reflejaba impacto en la gestión por lo que para las pruebas este valor se descarto dejando los valores para n en 16384 y 32768, y así como también se calculo, para n mayores a 32768 no se pueden realizar pruebas dado que el hardware no puede almacenar tanto.

4.1. Resultados TITAN V

Las pruebas en este dispositivo se realizaron sin problemas, incluso habiendo procesos que la usaban de forma pasiva, esto no supuso desmedro alguno en las pruebas, los resultados se tabularon para posteriormente ser graficados con el fin de tener una mejor apreciación visual de lo experimentado.

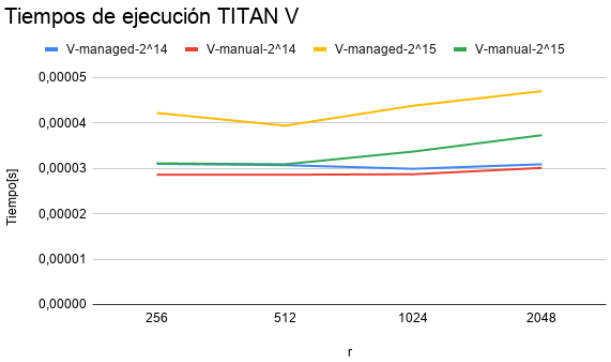

Figura 4: Gráfico de tiempos de ejecución para TITAN V

Como se puede apreciar en la figura.4, se tienen los resultados, el modo automático se representa con managed y el modo manual con la misma palabra, esto se repetirá en los otros gráficos. En cuanto a los resultados en si, para 16384(o 2exp14) se aprecia una mejora de tiempo en la gestión manual con 512 semillas o menos, mientras que para mas semillas no se aprecia una diferencia significativa, sin embargo con n=32768 si se aprecian rendimientos notoriamente distintos entre los modos de gestión siendo claramente mejor el modo manual, esto se podría explicar debido a que al aumentar la dimensión del mapa, la sincronización que usa por debajo el modo automático termina generando una sobrecarga de tiempo ya que necesita reflejar estos tiempos tanto en host como en el dispositivo.

4.2. Resultados TITAN RTX

De la misma forma que con la TITAN V, no hubo problemas a la hora de trabajar con la TITAN RTX, por lo que las pruebas no fueron alteradas por algún aspecto interno del sistema, los resultados de estas tambien se tabularon y graficaron.

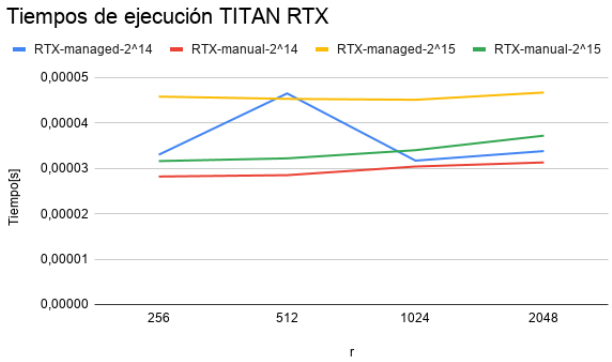

Figura 5: Gráfico de tiempos de ejecución para TITAN RTX

Observando la figura.5 lo primero que se puede destacar que tanto para n=16384 como para n=32768, con todos los valores posibles de semillas definidos, la gestión manual mostró un mejor rendimiento que la automática en todos los casos con diferencias apreciables. Un punto interesante de analizar es en el caso de la gestión automática con n=16384 y r=512 se puede observar un peak negativo de rendimiento de la gestión automática, este punto llamo bastante la atención y se repitió el experimento para esa combinación de n y r para observar si fue un caso particular o se repetía y mostró el mismo comportamiento, se cree que se debió que justo para esta combinación de n y r con ese modo de gestión en particular podía ocurrir que la tarjeta haga un trabajo extra por ende el resultado es bastante peor que lo observado en la TITAN V.

4.3. Comparativa

Al haber obtenido los resultados de las gestiones en las dos tarjetas se puede proceder a hacer la comparación para definir cual rindió mejor, cabe mencionar que se dijo en un principio de acuerdo a las especificaciones de fábrica que la RTX debería rendir mejor que la V. Para poder comparar visualmente de mejor manera se hicieron 3 gráficos, el primero de gestión automática, el segundo de gestión manual y el tercero de speed up.

Como se podrá observar en el gráfico de la figura.6, para n=16384 la TITAN V presento mejor rendimiento en gestión automática que su contra parte Turing, lo mismo se aprecia para n=32768 pero solo hasta r=512, para valores de r superiores los dos se comportan prácticamente igual por lo que se estima que a medida que crece el n para mayores valores o se comportan de una misma manera o la RTX empieza a tener mejor rendimiento, sin embargo dadas las prestaciones de las tarjetas disponibles esto no se podrá saber, no al menor ahora. Lo destacable es que la TITAN V rindió mejor que la TITAN RTX, aunque por poco; lo cual se contrapone a lo estipulado en un principio.

A diferencia de lo observado en la gestión automática, en este caso fue mucho mas parejo el rendimiento, al punto de que en ciertos intervalos de r ambas tarjetas rinden de forma muy poco distinguible como se puede observar en el gráfico de la figura.7,

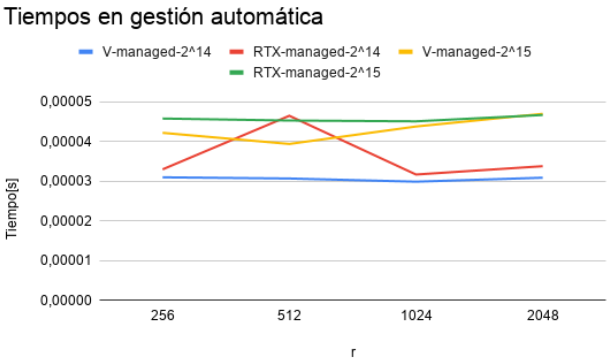

Figura 6: Gráfico de tiempos de ejecución para gestión automática en las dos tarjetas.

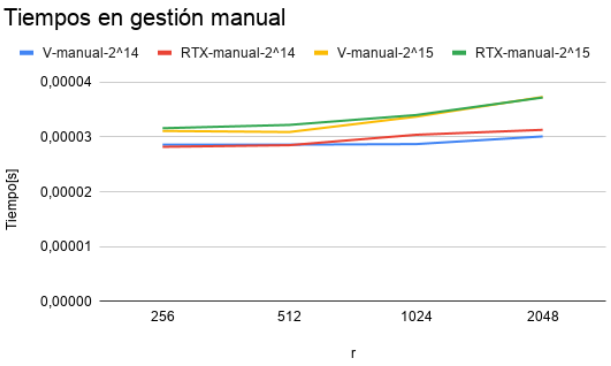

Figura 7: Gráfico de tiempos de ejecución para gestión manual en las dos tarjetas.

de lo que se puede mencionar se tiene que en el caso de semillas mayores a 512 para n=16384 rindió ligeramente mejor la TITAN V, lo mismo para una cantidad menor o igual a 512 de semillas, en resumen de nuevo la TITAN V venció a la RTX pero por un margen muy menor, se podria decir que incluso es indistinguible.

Lo primero que se deduce al analizar lo representado en el gráfico de la figura.8 es que en todos los casos efectivamente la gestión manual venció a la automática y también se puede observar que a medida que se incremento el tamaño de n, en este caso de 16384 a 32768, el speed up aumento por lo que se puede decir que a medida que crece el tamaño del problema n, mejor rinde el modo manual respecto al automático, lo otro que también se observa es que mientras mas crece la cantidad de semillas r, mas disminuye el speed up aunque se mantiene en margenes positivos por lo que sigue siendo mejor la gestión manual.

Para ir cerrando esta sección se tienen una seria de deducciones respecto a lo observado, lo primero y lo mas notorio fue que la gestión manual desplazo a la automática en rendimiento para todos los casos, y la diferencia aumenta si el tamaño n de un problema aumenta, en este caso no se diferencian tanto ya que se trata de un O(nlog(n)) pero con problemas de mayor complejidad se podría

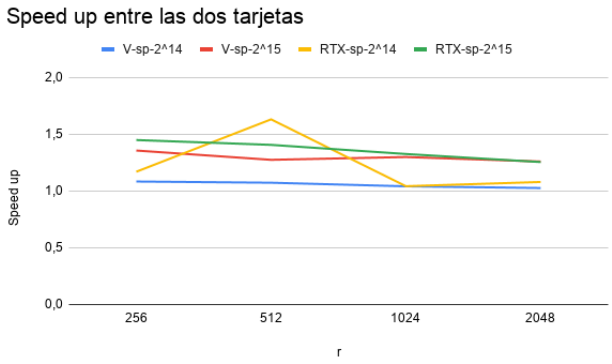

Figura 8: Gráfico comparativo de speed up entre las dos tarjetas.

hacer mas notorio. Respecto a las arquitecturas representadas por sus tarjetas se observo que si bien hubieron ligeras diferenciar que favorecían a la TITAN V, no se logro observar un resultado definitivo que pudiese determinar cual trabajaba de mejor manera.

5. CONCLUSIÓN

Respecto a lo planteado en el comienzo se puede confirmar que efectivamente las sobrecargas de coordinación que tiene que hacer la gestión automática le terminaron jugando en contra, lo que quedo demostrado en todos los experimentos realizados, por lo que se puede decir que si bien programar usando gestión automática puede ser cómodo, a la larga puede volverse en la contra ya que si o si para lograr mejores resultados hay que saber gestionar la memoria por cuenta propia, en síntesis, se puede usar gestión automática cuando se empieza a programar para guiarse pero después hay que aprender a hacerlo por cuenta propia con el fin de obtener mejores beneficios en lo que se esta programando.

Otro punto a remarcar también es la poca diferencia que hubo entre las dos tarjetas pese a ser de arquitecturas distintas, lo cual podría indicar que pese a las diferencias en cuanto a las especificaciones, estas no tienen mucha diferencia a la hora de procesar computo, sin embargo asumir esto podría ser erróneo, ya que para empezar esta investigación no considero el desgaste por tiempo activo o por uso en las tarjetas, esto es importante ya que a medida que se explota el uso de una tarjeta, lógicamente estas se desgastan y al pasar el tiempo terminan con un rendimiento menor al ofrecido por el fabricante, otro aspecto que podría influenciar es el tipo de problema, no es que el problema usado sea un mal determinante, sino que para poder realizar una perspectiva mas completa, es importante someter los sujetos de estudio, en este caso las tarjetas, a problemas de diversa índole, en lo cual se apunta a buscar variedad en la complejidad del problema.

Entre otras cosas, este trabajo también se podría mejorar, como se menciono antes, aumentar la diversidad de problemas permitiría tener una mejor vista general de como se diferencian los modos de gestión, también ocupar mas tarjetas para este tipo de estudio seria un buen aporte, por ejemplo, se disponía de acceso a otro computador que disponía de cuatro NVIDIA 2080 Ti de arquitectura

Turing también, por temas de tiempo no se hicieron pruebas en estas tarjetas. Otro punto que falto cubrir en esta investigación es la gestión entre dispositivos distintos, lo cual podría ayudar bastante cuando el problema es demasiado grande, como lo que paso cuando se querían crear mapas con un n que sea potencia de 2 y además mayor a 32768; haber estudiado este aspecto también habría sido un punto extra para este artículo.

Como propuesta personal a futuro, se propone seguir investigando aspectos importantes de tener en cuenta cuando se trabajar con hardware de este tipo que permite acelerar el computo, pero también se incita a seguir investigando los problemas relacionados a geometría computacional, en este caso el diagrama de Voronoi es una propuesta interesante de abordar, pero también se debe extender el alcance a otros problemas como el Convex Hull o la triangulación de Delaunay dada su importancia en la geometría computacional.

6. BIBLIOGRAFÍA

1 Memory management on modern GPU architectures; GTC keynote NVIDIA 2019; URL: developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9727-memory-management-on-modern-gpu-architectures.pdf

2 Memory management on modern GPU architectures; GTC keynote NVIDIA 2020; URL: developer.nvidia.com/gtc/2020/video/cwe21754, min: [40:00, 55:00].

3 Guodong Rong and Tiow-Seng Tan. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In Proceedings of the 2006 symposium on Interactive 3D graphics and games (I3D '06). Association for Computing Machinery, New York, NY, USA, 109–116. DOI:https://doi.org/10.1145/1111411.1111431.

4 G. Rong and T. Tan, "Variants of Jump Flooding Algorithm for Computing Discrete Voronoi Diagrams,"4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007), Glamorgan, 2007, pp. 176-181, doi: 10.1109/ISVD.2007.41.