Sistema SLAM acelerado por GPU para mapeo volumétrico globalmente consistente en tiempo real

Emiliano Höss
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Buenos Aires, Argntina
ehoss@dc.uba.ar

Pablo De Cristóforis
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Buenos Aires, Argntina
pdecris@dc.uba.ar

***Resumen*—El uso de GPUs integradas en robots móviles está en expansión debido a la creciente demanda de procesamiento embebido. La adaptación de algoritmos originalmente diseñados para CPUs a GPUs no es sencilla debido a heterogeneidad en las distintas arquitecturas de hardware y los recursos disponibles. En este trabajo, presentamos una versión acelerada del sistema de SLAM Voxgraph para mapeo volumétrico utilizando procesamiento en paralelo en GPUs. El sistema presentado se evaluó en una PC de escritorio con GPU y se comparó con la versión original de Voxgraph. Los resultados muestran que el sistema propuesto supera en 4x al front-end y en 8x al back-end a la versión original en términos de tiempos de ejecución, manteniendo la misma precisión. El nuevo sistema, acuñado coVoxgraph, se encuentra disponible como código abierto en GitHub[1].**

***Index Terms*—Mapeo, SLAM, SDF, GPU.**

I. Introducción

Para alcanzar el objetivo de navegar e interactuar de manera autónoma en entornos desconocidos, los sistemas de SLAM (Simultáneous Localization and Mapping) necesitan construir un modelo interno del mundo observado, es decir, un mapa. Las estimaciones que producen los robots de su propio movimiento se realiza de manera incremental a medida que se mueven por el entorno, esto conduce a errores de aproximación en la pose y a un mapeo inconsistente. Para mitigar los efectos inevitables que produce la incertidumbre sobre estos cálculos y mantener un mapa globalmente consistente se utilizan técnicas de cierre de ciclo (*loop closure*).

El mapeo basado en caracteristicas convierte los datos crudos del sensor en un conjunto de características del ambiente que utiliza para la construcción del mapa. Muchos sistemas SLAM bien conocidos siguen este enfoque y han demostrado crear mapas basados en características globalmente consistentes en tiempo real [1]–[3]. Aun así, son de uso restringido para otras tareas más allá de la localización, debido a las dificultades para extraer la forma y la conectividad de las superficies de los objetos en el entorno a partir de una representación dispersa del mapa.

Por otro lado, los mapas densos son adecuados no solo para la estimación de la pose, sino también para la reconstrucción de escenas, la planificación de trayectorias, la detección de objetos, la evasión de obstáculos y el control del movimiento. El primer sistema SLAM denso que utiliza la consistencia fotométrica de cada píxel para estimar la trayectoria de una cámara en mano fue [4]. Posteriormente, se desarrollaron varios sistemas SLAM densos basados en la técnica de ajuste de paquetes fotométricos (*Bundle Adjustment*) [5], [6] y actualmente utilizando aprendizaje profundo [7]–[9]. Sin embargo, estos sistemas aún requieren muchos recursos computacionales y no logran ejecutar en tiempo real en escenarios de gran escala. Tener la capacidad de crear mapas densos y globalmente consistentes en tiempo real sigue siendo un desafío para resolver el problema de la navegación autónoma a largo plazo en entornos complejos.

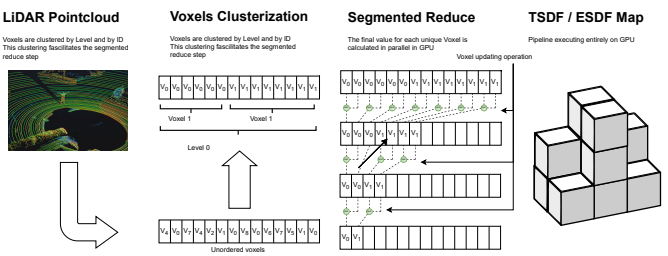

Figura 1: Flujo de datos de la implementacion en GPU de la integracion de nube de puntos del sensor en el sistema front-end de coVoxgraph.

Las superficies implícitas modeladas como funciones de distancia con signo (SDF por su nombre en inglés *Signed Distance Functions*), introducidas por primera vez en [10], han demostrado ser una representación efectiva para el mapeo denso [11]. Sin embargo, producir un mapa globalmente consistente usando SDFs es costoso, ya que la optimización global del mapa se vuelve rápidamente intratable a medida que aumenta la cantidad de datos. Una alternativa para abordar este problema es representar el entorno reconstruido como una colección de submapas. La ventaja de este enfoque es que la pose del sensor, en el momento en que se realiza una nueva registración de un nuevo submapa, solo necesita ser registrada con respecto al submapa actual. Si la trayectoria completa no se considera durante la optimización del mapa

[1]https://github.com/lrse-uba/covoxgraph

denso, los submapas pueden ser restringidos entre sí a través de la alineación geométrica [12]. En [13] esta idea se extiende proponiendo usar una alineación libre de correspondencia basada en la Función de Distancia con Signo Euclidiana (ESDF) que representa una cuadrícula de voxels donde cada punto contiene su distancia euclidiana al obstáculo más cercano. El sistema presentado en este trabajo, llamado Voxgraph, formula el problema como una optimización de grafos de pose, incluyendo restricciones de odometría y cierre de ciclo para mantener la consistencia global ante la posibilidad de error introducidos por cierres de ciclo que involucren una larga cadena de submapas en su trayectoria. Comenzando con la implementación original de CPU de Voxgraph, desarrollamos una nueva versión basada en GPU acuñada coVoxgraph que supera en $8\times$ al back-end y $4\times$ al front-end del sistema original considerando los tiempos de ejecución manteniendo la misma precisión. Las contribuciones de este trabajo se resumen de la siguiente manera:

- Un módulo de Estructura-de-Arrays (SoA) para facilitar la gestión de memoria heterogénea y la coalescencia; y para reducir el costo de transferencia de datos entre CPU y GPU.
- Reimplementación parcial de la biblioteca Eigen para álgebra lineal y la biblioteca Kindr para operaciones de cuaterniones utilizando programación genérica para soportar el diseño de datos SoA.
- Una tabla hash de múltiples valores GPU-CPU que almacena valores e índices organizados en SoA para permitir el intercambio flexible de diseños de datos subyacentes.
- Un nuevo modelo de flujo de datos para convertir los datos de la nube de puntos en un mapa de Función de Distancia Firmada Truncada (TSDF) y para actualizar el mapa ESDF a partir del mapa TSDF.
- La implementación en GPU de un metodo de aproximacion por cuadrados mínimos, similar a Ceres, pero solo enfocado en la resolucion del problema presente en este trabajo.
- Un repositorio de acceso público con todo sistema desarrollado: https://github.com/lrse-uba/covoxgraph/

II. Trabajos Previos

El sistema de SLAM Voxgraph prioriza la legibilidad y la extensibilidad sobre el rendimiento, con una arquitectura que fue diseñada para ser ejecutada en CPU. El sistema es principalmente de un solo hilo y la concurrencia se utiliza solo para acelerar el procesamiento por lotes de datos dentro de los módulos de integración de nube de puntos del sensor, tal como se describe en [14]. Este sistema Voxblox se usa para construir incrementalmente las ESDFs directamente a partir de Campos de Distancia Truncada con Signo (TSDFs) y aprovechar la información de distancia ya contenida dentro del radio de truncamiento. Los mapas TSDFs resultan más rápidos que los Octomaps para construir y suavizar el ruido del sensor en muchas observaciones. Voxgraph se divide en un front-end y un back-end (ver Fig. 2).

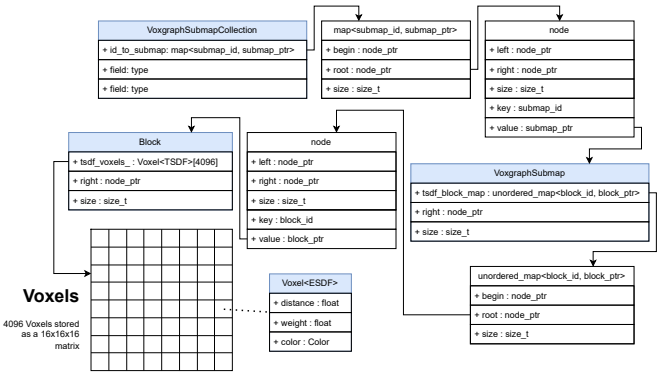

Figura 2: Ordenamiento de memoria en el sistema Voxgraph, las flechas representan las indirecciones necesarias para acceder a los datos, y los nodos sombreados en celeste representan los objectos definidos en el codigo de Voxgraph

El **front-end** de Voxgraph podría dividirse en dos módulos: uno responsable de convertir las mediciones de sensores en submapas y otro para crear las restricciones que se pasarán al back-end para corregir los errores durante la creacion de los submapas. Por su parte, el primer módulo también se divide en dos etapas. Primero, los datos del sensor entrante se integran en un volumen TSDF para incluir o actualizar los voxels que construyen el mapa TSDF y el segundo propaga los voxels actualizados desde el TSDF a la representación de volúmenes ESDF. Los voxels creados durante ambos pasos se agrupan en bloques de tamaño fijo que a su vez se indexan utilizando una tabla hash de bloques que representa internamente el mapa esparso global del entorno explorado. Este mapa global podría verse como una tabla hash de bloques dispersa en memoria (ver Fig. 2). El tamaño de los bloques puede ser cambiado por un parámetro de configuración, pero permanece igual durante la ejecución. Cuando una nueva nube de puntos llega desde el sensor, se lanzan múltiples hilos de ejecución que procesan un punto dentro de la nube y de manera serializada incorpora todos los voxels que intersectan con el rayo que se proyecta desde ese punto. Una vez que termina de incorporar todos los voxels, continúa con otro punto dentro de la nube de puntos. Este procesamiento se repite hasta alcanzar todos los puntos de la nube de puntos.

El **back-end** de Voxgraph es responsable de mantener el conjunto de restricciones generadas por el front-end (odometría, cierre de ciclo y registro de submapas) y estima la alineación de la colección de submapas más probable minimizando el error total de todas las restricciones del grafo de poses. Para hacer eso, tiene que resolver la aproximacion por mínimos cuadrados no lineales calculando las matrices de residuos y jacobianos para cada voxel y vértice de los submapas superpuestos. El cálculo no ponderado de los valores de los elementos de estas matrices es completamente independiente entre sí, el único punto en común es el valor de la sumatoria del peso de los voxeles o vértices relevantes, necesarios para calcular la media aritmética ponderada de los valores finales de las matrices de residuos y jacobianas. Sin

embargo, como este valor se precalcula para cada submapa cuando este es finalizado, no añade ninguna dependencia al cálculo, y por lo tanto, su implementación en GPU es perfectamente paralelizable [15].

Durante la última década, se han hecho esfuerzos para implementar algoritmos SLAM en múltiples unidades de procesamiento en sistemas heterogéneos CPU-GPU dado que esto representa un enfoque más realista para su uso en la vida real, destacando los beneficios y desventajas de las soluciones de computación embebidas para robots móviles. Aún así, la portabilidad de algoritmos originalmente diseñados para CPUs a GPUs no es sencilla debido a las diferencias en las arquitecturas de hardware y recursos disponibles. Tal es el caso de [16]–[18], donde se evalúan versiones aceleradas por GPU de sistemas SLAM basados en características. La GPU también ha sido utilizada para algunos componentes de SLAM RGB-D denso [11], [19]. En el contexto de SLAM LiDAR, el uso de GPU se limitó principalmente a acelerar la coincidencia de escaneo en el front-end [20], [21]. Sin embargo, en la mayoría de los trabajos anteriores, el back-end, y específicamente, la optimización del grafo de poses sigue siendo realizada en una CPU. En este trabajo se introduce coVoxgraph: un sistema de SLAM completo, enteramente acelerado por GPU (en su front-end y back-end) para el mapeo volumétrico globalmente consistente en tiempo real.

II-A. Profiling

Se realizó un analisis exhaustivo de los tiempos de ejecución de cada módulo de Voxgraph. Esto proporciona información valiosa sobre el consumo de tiempo y también sobre la distribución del uso de la CPU, detectando cuellos de botella, y guiando los esfuerzos para mejorar el sistema en términos de rendimiento para la versión acelerada en GPU.

Voxgraph requiere poder ser ejecutado en tiempo real para garantizar la estabilidad de la cantidad de datos de cada submapa. Las nubes de punto de entrada, destinadas a ser parte del submapa actual, se omitirán si el sistema no puede procesarlas en tiempo real. Esta variabilidad afecta la cantidad final de datos que se utilizan para cada submapa y, por lo tanto, el tiempo que demanda. Por esta razón, para realizar el análisi de tiempos de ejecución, se modificó el código oringal para evitar la omisión de nube de puntos, de manera similar a lo que se conoce como *process-every-frame* [22]. El conjunto de datos utilizado para el análisis de tiempos fue el mismo que Voxgraph (para más detalles, consulte la Sección IV).

Cada determinado tiempo se crea un nuevo submapa en Voxgraph, en ese momento el sistema necesita realizar la integración de las nubes de puntos recolectadas en esa ventana de tiempo, la actualización del mapa ESDF y la optimización del grafo de poses. La Fig. 3 muestra la distribución de tiempo en el momento en el que se crea un nuevo submapa. Considerando que, entre una medición y otra, hay alrededor de 20 ejecuciones de la integración de la nube de puntos, los tres módulos que demandan más tiempo de ejecición son: la integración de la nube de puntos, la actalización del mapa ESDF y la optimización del grafo de poses, como ya fuera

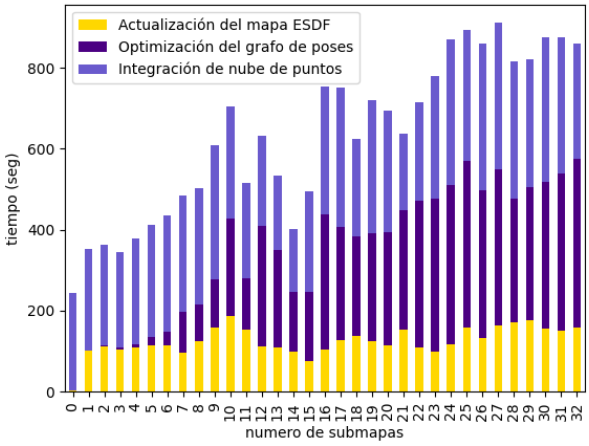

Figura 3: Analisis de mediciones de tiempo de cada modulo de Voxgraph.

mencionado tambien por los autores de Voxgraph [13]. Como puede observarse, el tiempo de ejecución del front-end (actualización del mapa ESDF y la integración de nube de puntos), se mantiene constante debido a que no hay cambios significativos en la cantidad de nube de puntos que se integran. Sin embargo, el tiempo de ejecución del back-end (optimización de grafo de poses) aumenta a medida que crece la cantidad de submapas totales en el mapa global.

III. Sistema Propuesto

En las siguientes subsecciones detallamos las principales contribuciones que permitieron portar el sistema Voxgrpah de su versión original para CPU a la versión acelerada en GPU.

III-A. Struct of Arrays

Los elementos de una colección de objetos pueden disponerse en memoria siguiendo dos enfoques distintos: por un lado SoA (por su nombre en ingles Struct of Arrays) donde los objetos se separan de acuerdo a sus campos en distintos arreglos; y por otro lado, AoS (por su nombre en ingles Array of Structs) donde los objetos son almacenados individualmente y de manera consecutiva. Para este trabajo se decidio utilizar el enfoque SoA debido a que ofrece varias ventajas. Primero, mejora la localidad de los datos, ya que cada componente de la estructura de datos de un objeto se almacena en un array separado, aumentando la utilización de la memoria caché y la coalescencia de memoria. En segundo lugar, la disposición SoA puede facilitar la vectorización, que es la ejecución paralela de operaciones en múltiples elementos de datos utilizando núcleos SIMD en procesadores tanto de GPU como CPU. En tercer lugar, mejora el rendimiento de los datos, minimizando el número de transacciones de memoria y disminuyendo el tamaño total de cada transacción, debido a la reducción del relleno que se utiliza dentro de la estructura de cada objeto para alinearlo en memoria (*padding*).

III-B. *Hashing Espacial*

Voxgraph utiliza una tabla hash de tamaño dinámico que hace uso del enfoque presentado por [23]. Hay tres niveles de particiones al indexar posiciones en la representación del mapa global. El primer nivel utiliza una colección de submapas superpuestos, junto con sus restricciones y poses relativas para estimar su alineación más probable. Esta colección se implementa utilizando una tabla hash de la Biblioteca Estándar C++ para indexar cada submapa. Dentro de cada submapa hay un segundo nivel de partición representado por bloques, que son esencialmente una matriz de un número fijo de voxels. Los bloques se almacenan en una tabla de hash de la Biblioteca Estándar C++ utilizando dos funciones de hash. La primera función transforma las coordenadas continuas representadas como una tupla de puntos flotantes de 64 bits, a coordenadas globales discretas representadas por una tupla de enteros sin signo de 64 bits. Esto crea un segundo nivel de partición llamado Discrete Global Grid (DGG), después de eso se aplica la segunda función hash denominada DECO para mejorar la eficiencia al recorrer los datos, siguiendo el metodo de [24]. El problema con este enfoque es que cuando se crea un bloque, se crea un nuevo objeto y la tabla de hash guarda el puntero a su posición en memoria. Creando así un patrón de memoria dispersa dentro de cada submapa. El último nivel de partición se realiza dentro de los bloques, utilizando una matriz cuadrada densa de voxels del mismo tamaño (ver Fig. 2). El problema con este modelo de tablas hash utilizado por Voxgraph es que sufren de un rendimiento deficiente inducido por patrones de acceso a memoria irregulares. Los avances recientes [25], [26] han demostrado que la adopción de SoA para las tablas de hash y sus claves, mejora sustancialmente el rendimiento de las operaciones de inserción y búsqueda. La disposición AoS proporciona una localidad de caché relativamente alta si se accede tanto a la clave como al valor simultáneamente. Sin embargo, si sólo se accede a la clave, la efectividad de la caché se reduce. Esto es especialmente crítico si el tipo de valor es grande en comparación con el tipo de clave, como es en el caso de Voxgraph. En este trabajo, adoptamos estas ideas junto con el módulo SoA para implementar la tabla de hash.

III-C. *Integración de nubes de puntos*

El front-end de Voxgraph es responsable de generar los voxels del mapa TSDF utilizando los datos crudos del sensor. Dado que el mapa TSDF utiliza una representación menos detallada para almacenar la información de la nube de puntos, un solo voxel intersectará la informacion de varios puntos del sensor. Por lo tanto, la información almacenada en el voxel necesita ser actualizada a partir de los nuevos puntos que lo intersectan en cualquier orden. Al introducir la ejecución concurrente, estas actualizaciones necesitan sincronizarse cada vez que se actualice el voxel en cuestión, como se ve en la Fig. 4, donde se representan las primitivas de sincronización en rojo. Tomando en cuenta que el método de actualización de cada voxel es una operación conmutativa y asociativa, se puede utilizar la técnica de reducción en GPU introducida por

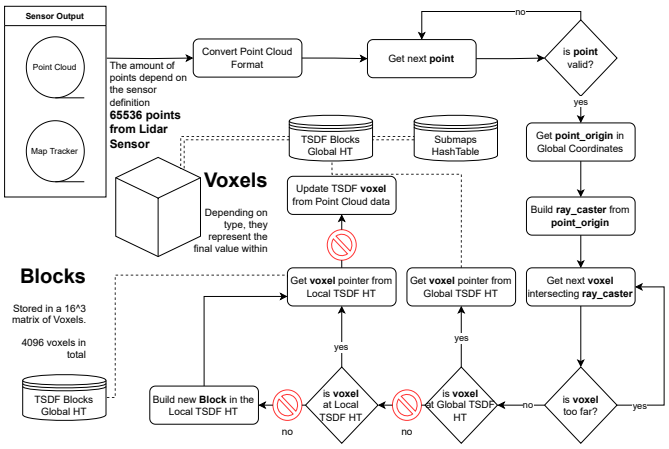

Figura 4: Diagrama de flujo de la integracion de las nubes de puntos del sensor. Los puntos de sincronizacion estan señalizados en rojo.

[27], donde cada paso en este algoritmo reduce a la mitad el número de voxels que se actualizan, resultando en un orden de complejidad $O(log(n))$, donde n es el número de voxels a actualizar. Proponemos un flujo de datos representado en la Fig. 1 que comienza en la transformación de los datos crudos del sensor de la nube de puntos hasta que todos los voxels son generados y actualizados dentro del sistema utilizando la técnica de Reduccion por Segmentos en GPU para acelerar el proceso. Los pasos intermedios del flujo de datos son responsables de agrupar el voxel de tal manera que estén disponibles para realizar el paso de reducción de manera óptima, como se propone en [28]. Finalmente, cada nuevo voxel único se asigna en la tabla hash de manera sincronizada. La asignación o actualización de voxels en la tabla de hash consume mucho tiempo ya que depende de la sincronización cada vez que escribimos un nuevo valor, por lo que retrasar esta operación hasta el último momento permite una reducción drástica en el número de elementos que requieren ser sincronizados y por ende en su tiempo de ejecución.

III-D. *Aproximación por cuadrados minimos*

Voxgraph realiza una aproximación por cuadrados mínimos no lineal para el grafo de poses cada vez que se finaliza un submapa nuevo. Este proceso es el primer paso del back-end y utiliza un modelo esparso por definición, dado que los submapas creados son el resultado de la intersección entre la trayectoria arbitraria del robot y la representación interna del mapa global.

Para resolver el problema de aproximación por cuadrados mínimos resultante, Voxgraph utiliza la biblioteca Ceres, la cual solo ofrece una version acelerada en GPU para problemas de aproximación de sistemas lineales densos. Por este motivo, presentamos en este trabajo una implementación del método iterativo de gradiente conjugado para GPU fuertemente basada en implementación en CPU de Ceres, pero limitada al problema de actualización del grafo de poses que se presenta en Voxgraph.

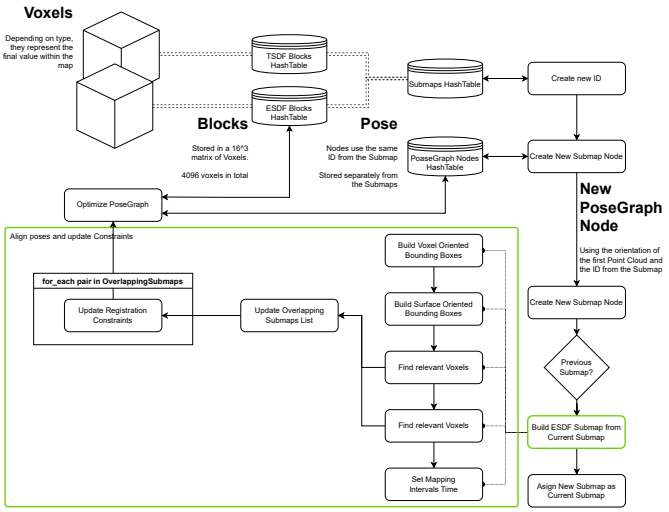

Figura 5: Diagrama de flujo de ejecución del back-end en Voxgraph.

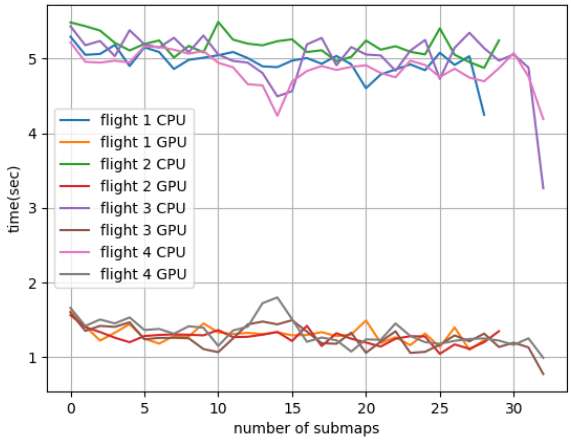

Figura 6: Tiempo de ejecución del modulo de integración de la nube de puntos del sensor (front-end) para cuatro vuelos distintos.

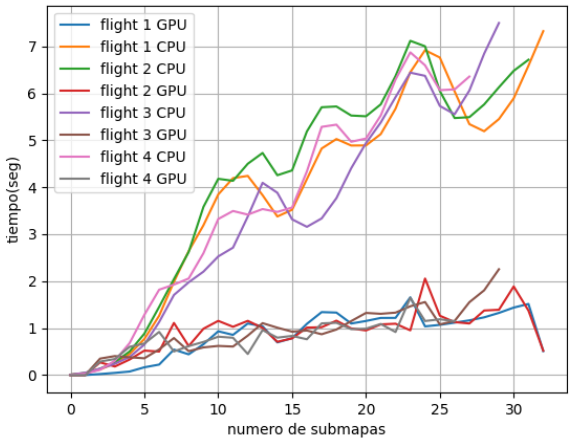

Figura 7: Tiempo de ejecución de modulo de aproximación por cuadrados mínimos (back-end) para cuatro vuelos distintos.

La estrategia general al resolver problemas de optimización no lineales es a través de una secuencia de aproximaciones al problema original [29]. En cada iteración, se resuelve la aproximación para determinar una corrección Δx al vector x. Para determinar el paso siguiente Δx en cada iteración se pueden diferenciar dos grandes estrategias: Line Search y Trust Region. Para este trabajo decidimos utilizar la estrategia de Trust Region, en particular implementamos el algoritmo de Levenberg-Marquardt (LMA), que puede verse como una interpolación entre el algoritmo de Gauss-Newton (GNA) y el método de Descenso de Gradiente. LMA por su parte es más robusto que GNA ya que no requiere proveerle de una solución inicial que este demasiado cercana a la solución final. Por su parte, el sistema lineal se representa utilizando el Complemento Schur, lo que simplifica y acelera las operaciones a realizarse durante las iteraciones del algoritmo y a su vez facilita la paralelización por bloques en GPU. Para la representación interna de las matrices utilizamos la Biblioteca de Nvidia cuSparse, que provee la mayoría de las funcionalidades que se requieren para el manejo de matrices esparsas en memoria de GPU, también utilizamos la Biblioteca de Nvidia cuBlas que ofrece el conjunto de funcionalidad estandar de algebra lineal implementadas en GPU que utilizamos para los algoritmos mencionados.

IV. Resultados

Para la evaluación experimental, se utilizó el mismo conjunto de datos de dominio público[2] presentado junto a Voxgraph. Este conjunto de datos corresponde a cuatro vuelos realizados por un hexacóptero Micro Aerial Vehicle (MAV) equipado con un LiDAR Ouster OS1, cada uno con una trayectoria aproximada de 400m, alrededor de un área de desastre diseñada para entrenar rescatistas. El ground-truth de campo se generó a partir de un sistema RTK-GNSS.

[2]Disponible en: http://robotics.ethz.ch/∼asl-datasets/2020_voxgraph_arche

En primer lugar se analizaron los tiempos de ejecución del front-end y back-end en ambas versiones, la original basada en CPU y la acelerada en GPU. Las unidades de procesamiento utilizadas para realizar los experimentos fueron una PC de escritorio estándar (CPU con un procesador AMD Ryzen 9 5950x y una GPU Nvidia GeForce RTX 2060). Como puede verse en la Fig. 6 el tiempo de ejecución del front-end (el integrador de nubes de puntos) se mantiene relativamente estable porque el tamaño de la entrada no varía de una instancia de ejecución a otra, por otro lado la mejora introducida por coVoxgraph supera a la versión original por 4x. Como se puede ver en la Fig 7 el tiempo de ejecución para el proceso de back-end aumenta junto con el número de submapas superpuestos. Esto ocurre porque el número de bloques por submapa se distribuye homogéneamente, y por lo tanto también el número de vóxeles o píxeles relevantes precalculados cada vez que se termina un submapa. En este

caso, la version acelerada por GPU supera a la version original hasta en $8\times$ al final del vuelo.

Para evaluar la estimación de la trayectoria de Voxgraph y coVoxgraph, se realizó una comparación contra el ground-truth utilizando el error cuadrático medio (RMSE) del Error Absoluto de Trayectoria (ATE). Las mediciones de RMSE calculadas siguiendo [30] se pueden comparar en la Tabla I. Como se esperaba, tanto la versión original como la versión basada en GPU resultan en un RMSE muy similar.

Flight Version	Voxgraph	coVoxgraph
1	**1.01**	1.26
2	0.78	**0.42**
3	1.16	**0.92**
4	**0.54**	1.95

Cuadro I: Raíz del error cuadrático medio (metros) del Error Absoluto de Trayectoria (ATE) para Voxgraph y coVoxgraph.

V. Conclusiones y trabajo futuro

En este trabajo presentamos un novedoso sistema acelerado por GPU denominado coVoxgraph para construir mapas volumétricos globalmente consistentes en tiempo real basados en Voxgraph. Tanto el front-end como el back-end fueron rediseñados para aprovechar al máximo el poder de procesamiento paralelo de la GPU. Los resultados experimentales, llevados a cabo utilizando los mismos conjuntos de datos de Voxgraph, demuestran que la implementación presentada es hasta $8\times$ veces más rápida que la original al ejecutarse en una PC de escritorio. En todos los experimentos, la precisión obtenida fue la misma que los resultados generados por Voxgraph. Como trabajo futuro llevaremos a cabo un estudio de rendimiento del nuevo sistema coVoxgraph en placas GPU embebidas tales como la Jetson Xavier AGX que pueden ser montadas a bordo de pequeños robots móviles aéreos y terrestres. De esta forma buscamos analizar la factibilidad de alcazar un sistema de SLAM para mapeo volumétrico globalmente consistente, que pueda ser ejecutado en en tiempo real bordo de estos robots.

Referencias

[1] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[2] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of field robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[3] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena, "maplab 2.0–a modular and multi-modal mapping framework," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 520–527, 2022.

[4] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*. IEEE, 2011, pp. 2320–2327.

[5] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[6] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "Elasticfusion: Dense slam without a pose graph." Robotics: Science and Systems, 2015.

[7] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, "Deepfactors: Real-time probabilistic dense monocular slam," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 721–728, 2020.

[8] L. Koestler, N. Yang, N. Zeller, and D. Cremers, "Tandem: Tracking and dense mapping in real-time using deep multi-view stereo," in *Conference on Robot Learning*. PMLR, 2022, pp. 34–45.

[9] X. Yang, H. Li, H. Zhai, Y. Ming, Y. Liu, and G. Zhang, "Vox-fusion: Dense tracking and mapping with voxel-based neural implicit representation," in *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2022, pp. 499–507.

[10] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 303–312.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 2011, pp. 127–136.

[12] N. Fioraio, J. Taylor, A. Fitzgibbon, L. Di Stefano, and S. Izadi, "Large-scale and drift-free surface reconstruction using online subvolume registration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4475–4483.

[13] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto, "Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps," *IEEE Robotics and Automation Letters*, vol. 5, no. 1, pp. 227–234, 2019.

[14] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1366–1373.

[15] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The Art of Multiprocessor Programming*. Elsevier Science, 2020. [Online]. Available: https://books.google.com.ar/books?id=7MqcBAAAQBAJ

[16] S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli, "Data flow orb-slam for real-time performance on embedded gpu boards," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 5370–5375.

[17] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ros-compatible stereo visual slam methods on a nvidia jetson tx2," *Measurement*, vol. 140, pp. 161–170, 2019.

[18] J. Song, J. Wang, L. Zhao, S. Huang, and G. Dissanayake, "Mis-slam: Real-time large-scale dense deformable slam system in minimal invasive surgery based on heterogeneous computing," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4068–4075, 2018.

[19] T. Schops, T. Sattler, and M. Pollefeys, "Bad slam: Bundle adjusted direct rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 134–144.

[20] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments." in *Robotics: Science and Systems*, vol. 2018, 2018, p. 59.

[21] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Globally consistent 3d lidar mapping with gpu-accelerated gicp matching cost factors," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8591–8598, 2021.

[22] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. Kelly, A. J. Davison, M. Luján, M. F. O'Boyle, G. Riley *et al.*, "Introducing slambench, a performance and accuracy benchmarking methodology for slam," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 5783–5790.

[23] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, pp. 1–11, 2013.

[24] L. Buckley, J. Byrne, and D. Moloney, "Investigating the impact of suboptimal hashing functions," in *2018 IEEE Games, Entertainment, Media Conference (GEM)*. IEEE, 2018, pp. 324–331.

[25] D. Jünger, R. Kobus, A. Müller, C. Hundt, K. Xu, W. Liu, and B. Schmidt, "Warpcore: A library for fast hash tables on gpus," in *2020 IEEE 27th international conference on high performance computing, data, and analytics (HiPC)*. IEEE, 2020, pp. 11–20.

[26] W. Dong, Y. Lao, M. Kaess, and V. Koltun, "Ash: A modern framework for parallel spatial hashing in 3d perception," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5417–5435, 2022.

[27] E. E. Santos, "Optimal and efficient algorithms for summing and prefix summing on parallel machines," *Journal of Parallel and Distributed Computing*, vol. 62, no. 4, pp. 517–543, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731500916988

[28] R. W. Larsen and T. Henriksen, "Strategies for regular segmented reductions on gpu," in *Proceedings of the 6th ACM SIGPLAN International Workshop on Functional High-Performance Computing*, ser. FHPC 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 42–52. [Online]. Available: https://doi.org/10.1145/3122948.3122952

[29] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. [Online]. Available: https://books.google.com.ar/books?id=VbHYoSyelFcC

[30] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7244–7251.