ANÁLISIS Y EVALUACIÓN DEL PROCESAMIENTO PARALELO DE SEÑALES ACÚSTICAS

ACOUSTIC SIGNAL MODIFICATION SYSTEM THROUGH PARALLEL PROCESSING

Francisco Javier Enríquez Aguilera
Universidad Autónoma de Ciudad Juárez, México
fenrique@uacj.mx

Jesús Martin Silva Aceves
Universidad Autónoma de Ciudad Juárez, México
jesilva@uacj.mx

Salvador Noriega Morales
Universidad Autónoma de Ciudad Juárez, México
snoriega@uacj.mx

Gabriel Bravo Martínez
Universidad Autónoma de Ciudad Juárez, México
gbravo@uacj.mx

Alfredo Bueno López
Universidad Autónoma de Ciudad Juárez, México
al140929@alumnos.uacj.mx

Recepción: 3/noviembre/2021 **Aceptación:** 7/marzo/2022

Resumen

Las GPU son unidades especializadas para el procesamiento de gráficos, éstas tienen un gran número de núcleos los cuales trabajan en paralelo. En MATLAB existen funciones asociadas a la arquitectura CUDA, creada por NVIDIA®, que permiten ser utilizadas directamente, sin tener que utilizar bibliotecas de bajo nivel. En este proyecto se implementó un sistema de modificación de señales acústicas, en donde se procesa una entrada de audio agregando un efecto de reverberación por convolución de una respuesta al impulso infinita (FIR). Se utilizaron dos GPU (una GeForce y otra Quadro), para comprobar la mejora en velocidad de procesamiento. Se obtuvo mejora de velocidad en cada una, sin embargo, la GPU para diseño resultó más eficiente en el tiempo de procesamiento.

Palabras Clave: GPU, NVIDIA CUDA, procesamiento de gráficos, procesamiento paralelo.

Abstract

GPU are specialized units for graphics processing, they have many cores which work in parallel. In MATLAB there are functions associated with the CUDA architecture, created by NVIDIA®, which allow them to be used directly, without having to use low-level libraries. In this project, an acoustic signal modification system was implemented, where an audio input is processed by adding a convolution reverberation effect of an infinite impulse response (FIR). Two GPU (one GeForce and one Quadro) were used to check the improvement in processing speed. Speed improvement was obtained in each one, however, the GPU for design was more efficient in processing time.

Keywords**:** *GPU, NVIDIA CUDA, graphics processing, parallel processing.*

1. Introducción

El sonido es un componente importante de los entornos virtuales y es utilizado en la forma de habla, música, sonido de objetos (Ej. un tren que pasa), efectos ambientales no visuales (Ej. viento), efectos de sonido (Ej. tiroteos), sonidos de fondo no asociado con efectos específicos (Ej. un patio de juegos), y señales de evento (Ej. el sonido de "¡kaching*!*") después de un movimiento exitoso en un videojuego) [Orellana, 2016]. El procesamiento de señales acústicas es utilizado por creadores de música y contenido multimedia tal como la producción de sonido en industrias como la de los videojuegos o el cine, y cuando este procesamiento requiere de un tiempo de cómputo corto, tal como el procesamiento en tiempo real, el procesamiento serial proporcionado por la Unidad de Procesamiento Central (CPU del inglés Central Processing Unit) de una computadora personal, puede llegar a ser insuficiente. Al implementar un sistema de modificación de señales acústicas mediante procesamiento paralelo se busca beneficiar a músicos y productores de audio ofreciendo alternativas para el procesamiento de señales acústicas cuando este requiera tener una alta velocidad de procesamiento.

Ha sido demostrado que el audio mejora la experiencia narrativa y puede ser utilizado para guiar al usuario a través de un entorno virtual inmersivo [Orellana, 2016]. Para lograr esto, los productores modifican el audio mediante la utilización de efectos. Tal como mencionan en [Liévano, 2013], los efectos de audio son utilizados por todo aquel involucrado en la producción de música, empieza desde las técnicas musicales ejecutadas por un músico y su instrumento, seguido por técnicas de grabación con micrófonos especiales y termina pasando por procesadores de efectos que la sintetizan, grabación, producción y difusión de estas señales de audio. Debido a que los músicos pretenden generar un sonido diferente y único en el momento de interpretar un instrumento, los efectos de audio son una alternativa para lograr este propósito. Actualmente, estos efectos se generan usando sistemas de procesamiento analógico o digital y se pueden clasificar según el dominio de procesamiento, en dinámico, frecuencia y usando retardos [CKSFE, 2021].

En [Nelson, 2018] se menciona que hoy en día existe software de edición de audio para computadoras multipropósito que requieren de señales pregrabadas y dan una apreciación post edición del efecto aplicado. Sin embargo, en un sistema de procesamiento que se encarga de procesar múltiples efectos puede llegar a tener una gran demanda de cómputo, esto se debe a que para poder implementar algunos efectos es necesario transformar la señal del dominio del tiempo al de la frecuencia, utilizando la FFT (Transformada Rápida de Fourier, del inglés *Fast Fourier Transform*). Como resultado disminuye el desempeño de un sistema de procesamiento y, además, la capacidad y/o velocidad de procesamiento se restringe a la cantidad de efectos que se van a procesar, esto es un problema en aplicaciones donde el tiempo de procesamiento debe ser lo menor posible. En este proyecto se desarrolla un sistema de modificación de señales acústicas mediante la utilización de filtros y efectos tales como eco y reverberación. Se busca implementar este sistema utilizando procesamiento paralelo para aumentar la velocidad de procesamiento.

El uso de Unidades de Procesamiento Gráfico (GPU del inglés Graphics Processing Unit) para diversas áreas del procesamiento digital de imágenes y diversos países,

se detalla en [Enríquez, 2018], se menciona que con esta herramienta es posible mejorar los tiempos de procesamiento y que puede ser desde un aumento en la velocidad de procesamiento de 5x hasta 1360x, dependiendo el tipo de procesamiento que se aplique.

Otra área en la que es fundamental el tiempo de procesamiento es la cancelación de ruido en imágenes, a pesar de existir hardware especializado para esta tarea el tiempo de procesamiento sigue siendo un problema, por lo que, al implementar un algoritmo acelerado mediante GPU, se pueden conseguir tiempos menores a 300 µs [Bhaskar, 2021]. También el ruido acústico en forma de eco es un problema en donde se ha propuesto experimentar con el procesamiento digital mediante el uso de GPU [Yeongseok, 2020].

En este trabajo se implementó un sistema de modificación de señales acústicas, en el cual se procesa una entrada de audio agregando un efecto de reverberación por convolución de una respuesta al impulso infinita (FIR). Se utilizaron dos GPU para comprobar la mejora de tiempo de procesamiento, una para uso en videojuegos y la otra con la finalidad de diseño profesional.

2. Métodos

Para el desarrollo de este trabajo se utilizaron dos equipos de prueba con capacidades distintas. Esto, para poder observar cómo afecta procesar con GPU o CPU de gama más alta:

- **Equipo de prueba 1**
  - ✓ GPU: NVIDIA GeForce 940Mx, especializada para videojuegos.
  - ✓ CPU: Intel Core i7 7500u, @ 2.7 GHz, 4 MB Cache, 4 núcleos (2 físicos; 2 lógicos), RAM: 12 GB DDR4.
- **Equipo de prueba 2**
  - ✓ GPU: NVIDIA Quadro K2100M, especializada para diseño asistido en computadora.
  - ✓ CPU: Intel Core i7 4810MQ, @ 2.8 GHz, 6 MB Cache, 8 núcleos (4 físicos, 4 lógicos), RAM: 16 GB.

En la tabla 1, se muestra una comparativa de las dos GPU que se utilizaron en el proyecto.

Tabla 1 Comparación entre NVIDIA GeForce 940Mx y NVIDIA Quadro K2100M.

Características	**Nvidia GeForce 940MX**	**Nvidia Quadro K2100M**
Núcleos	384	576
Unidades de mapeo de textura (TMUs)	24	48
Unidades de salida de renderizado (ROP)	8	16
Tamaño de memoria (GB)	2	2
Tipo de memoria	DDR3	GDDR5
Ancho de bus (bits)	64	128
Frecuencia (MHz)	1242	3000

Se realizaron pruebas en MATLAB con uno de los algoritmos de reverberación digital artificial más básicos, creado por Manfred Schroeder de Laboratorios Bell en los años 60s. Este consta de un retraso basado en un filtro *"comb" A(z)*, una retroalimentación, un puente y nodos de ganancia, tal como se muestra en la figura 1. Este tipo de reverberación resultó computacionalmente pesada, a pesar de ser el algoritmo "más simple". Como alternativa se utilizaron las cajas de herramienta para cómputo paralelo, *gpuArray*, para llevar a cabo el algoritmo de convolución, utilizando respuestas al impulso pregrabadas (en la convolución se puede dividir el procesamiento en secciones, lo cual es idóneo para trabajo paralelo). Ya que las GPU se desempeñan muy bien al realizar este tipo de operaciones, sobre todo si se utilizan métodos como la convolución en frecuencia, como se verá más adelante.

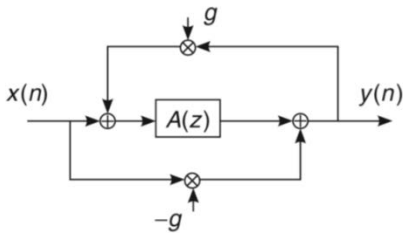

Figura 1 Estructura básica de reverberación [Arfib, 2011].

La técnica para realizar una convolución rápida es seccionar la señal de entrada en varios fragmentos de tamaños iguales para posteriormente procesar cada uno de

forma individual. Esto es especialmente útil cuando la señal de audio que se quiere procesar es muy larga. Existen dos técnicas de convolución rápida de las cuales se hizo uso en este proyecto: el método *overlap-add* y *overlap-save*. Ambas pueden realizar convolución tanto en el tiempo como en la frecuencia, sin embargo, para sacar provecho de la GPU, se utilizaron para convolucionar en el dominio de la frecuencia. Con el fin de comprobar el rendimiento de ambos métodos y además observar la diferencia en el desempeño al utilizar o no la GPU, se realizaron cuatro programas diferentes, dos por cada método de convolución, de los cuales uno hará uso de la GPU y el otro no.

Algoritmo *Overlap-Add*

En la figura 2, se muestra el diagrama de flujo para el método *Overlap-Add*. Al iniciar este algoritmo, primero, se leen los archivos de entrada, es decir, se lee el audio de entrada que se quiere procesar seguido de la respuesta al impulso con el cual se va a convolucionar para lograr el efecto de reverberación. Si el programa determina que la frecuencia de muestreo de la respuesta al impulso difiere de la del audio de entrada, entonces el programa realizará un remuestreo de la respuesta al impulso al calcular el factor de compresión/expansión a la cual se debe someter la señal, seguido de la compresión o expansión de esta.

Una vez que las frecuencias de muestreo de ambas señales son iguales, el programa continúa. Se descompone la entrada de audio en varios segmentos del mismo tamaño, esto con la finalidad de procesar cada uno de forma individual. El tamaño de éstos deberá ser determinado según la siguiente regla establecida en MATLAB: el segmento no debe exceder un tamaño equivalente al doble de la frecuencia de muestreo de la señal que se va a seccionar, esto es una restricción que es impuesta al buffer de salida de la tarjeta de sonido una vez que es configurada a una frecuencia de muestreo determinada. Es decir, por ejemplo, si la señal de entrada tiene una frecuencia de muestreo de 48 kHz, el tamaño por cada segmento no debe exceder de 96000 muestras. Enseguida, se inicializan las variables del programa, además de transformar al dominio de la frecuencia la respuesta al impulso. Esto se realiza en este punto ya que no es necesario que se

esté calculando repetidamente, por esta razón se computa antes de que inicie el ciclo donde se convolucionarán los fragmentos de la entrada de audio. Seguido de esto, se genera un archivo de texto en el cual se guardan posteriormente los resultados de las pruebas. Además, se guarda como encabezado los datos de los archivos que se están convolucionando.

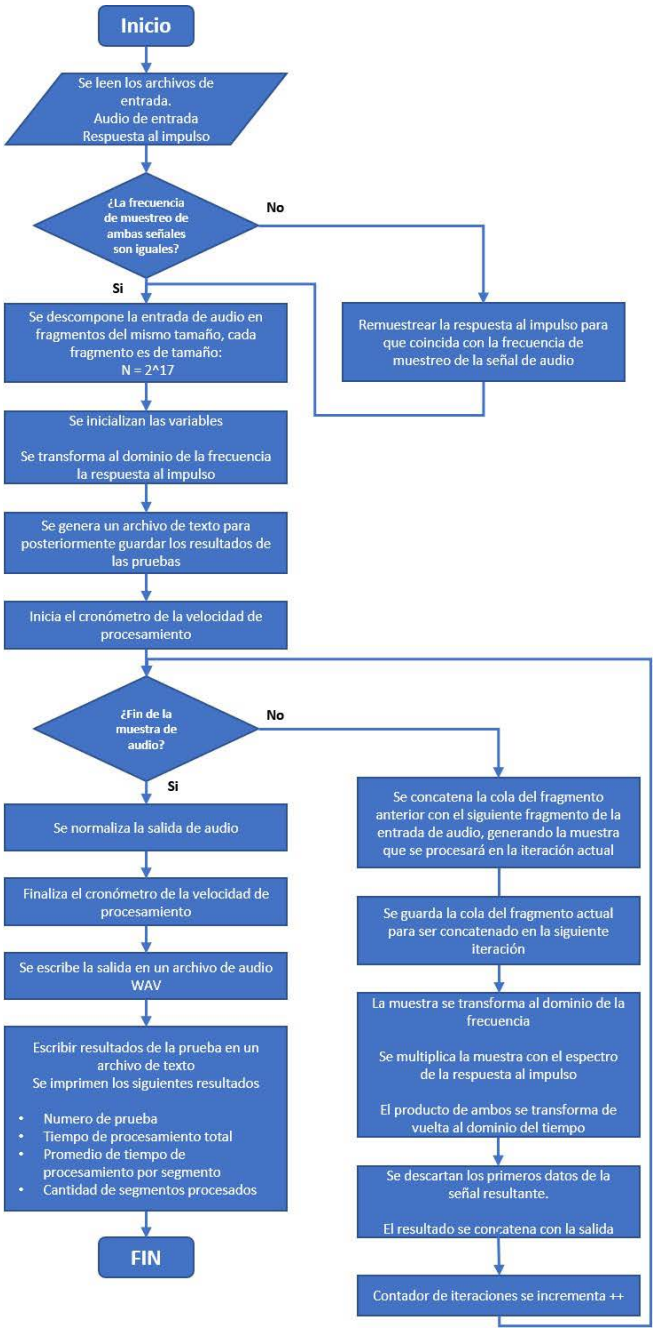

Figura 2 Diagramas de flujo para comprobar el rendimiento de Overlap-Add.

A partir de este punto se comienza a cronometrar la velocidad del algoritmo. Se inicia un ciclo WHILE el cual se repetirá hasta que se haya procesado completamente la entrada de audio. Dentro del ciclo WHILE, la muestra de audio se transforma al dominio de la frecuencia para posteriormente ser multiplicado con el espectro en frecuencia de la señal impulso.

El producto de ambos espectros se transforma de regreso al dominio del tiempo, dando como resultado la convolución de ambas señales. Se suma la cola reverberante del fragmento anterior con las primeras muestras de la señal convolucionada. Posteriormente, se guarda la cola reverberante de la señal procesada para ser sumada al siguiente fragmento en la próxima iteración. Para finalizar el procesado del segmento, se concatena la porción útil de la señal convolucionada a la salida y el contador de iteraciones se incrementa en uno. Una vez que se hayan procesado todos los fragmentos, se terminará con el ciclo y el programa continuará su operación normal.

El programa finaliza después de normalizar la matriz resultante de salida y detiene el cronómetro de la velocidad de procesamiento. Los resultados de la prueba se reportan en un archivo de texto creado anteriormente, el cual reporta el número de prueba, tiempo de procesamiento total el cual es obtenido mediante el uso de las funciones tic y toc (cuando se llama a la función tic almacena el tiempo actual, la función toc devuelve el tiempo transcurrido una vez que se ejecutó la función tic), cantidad de segmentos procesados y promedio de tiempo de procesamiento por cada segmento.

Algoritmo *Overlap-Save*

En la figura 3, se muestra el diagrama de flujo para este método. La operación de este programa es idéntica a la del algoritmo *Overlap-Add*, con excepción del bloque de procesamiento de la señal. Dentro del ciclo de procesamiento, primero se concatena la cola del fragmento anterior con el fragmento de entrada actual.

Al mismo tiempo se guarda la cola del fragmento actual para ser concatenado con el fragmento de la siguiente iteración. Después, la muestra se transforma al dominio de la frecuencia, se multiplica con el espectro en frecuencia de la respuesta al

impulso y el producto de ambos se transforma de regreso al dominio del tiempo para dar como resultado la convolución de ambas señales.

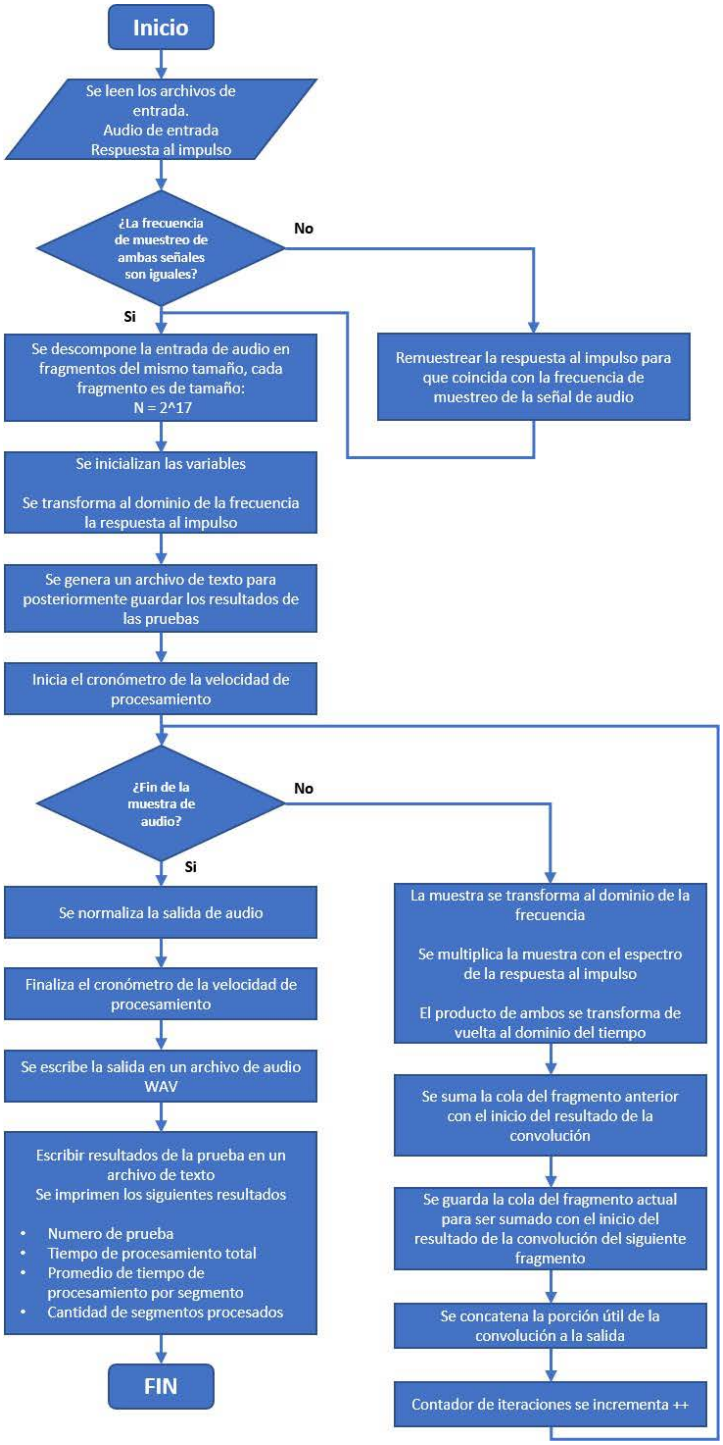

Figura 3 Diagramas de flujo para comprobar el rendimiento de Overlap-Save.

Para finalizar la etapa de procesamiento, se descartan los primeros datos de la señal resultante, para después concatenar la porción útil de la señal con la salida. Por último, se incrementa el contador de iteraciones. El ciclo se repite hasta que todos los fragmentos de la señal de entrada hayan sido procesados.

Para procesar ambos algoritmos dentro de la GPU es necesario mover las variables que se van a procesar a la memoria de la GPU. La clave para poder mantener una alta velocidad de procesamiento es no mover innecesariamente los datos de la memoria de la GPU a la RAM de la computadora, ya que el adquirir los datos guardados en la GPU es una operación costosa en términos de tiempo. Para esto se utilizaron de tres funciones proporcionadas por la librería de "*Parallel Computing*" de MATLAB:

- *gpuArray*() - Función utilizada para mover o escribir datos a la memoria dedicada de la GPU.
- *gather*() - Función utilizada para adquirir datos guardados en la GPU y guardarlos en la RAM de la computadora.
- *gpuDevice*() - Función que retorna las características de la GPU, además sirve para reiniciar la GPU, borrando las variables y/o procesos guardados en la memoria de la GPU.

La función *gather*() consume mucho tiempo, por lo que se hace uso de ella hasta finalizar con el procesamiento de todos los fragmentos de entrada, justo antes de guardar la señal de salida resultante.

Desarrollo de las pruebas

Para poder adquirir un tiempo de procesamiento lo más preciso posible cada programa se repitió un total de cinco veces, para posteriormente generar un promedio de tiempo de las pruebas. Se realizó un programa más, cuya función es la de ejecutar los cuatro programas en serie cinco veces. Las pruebas se ejecutaron convolucionando tres respuestas al impulso diferentes, cada una con una cantidad de muestras diferente. Esto para poder observar la influencia que tienen distintas respuestas al impulso en el tiempo de ejecución de los programas.

La tabla 2 muestra los datos de tres señales que representan la respuesta al impulso unitario a ser utilizadas para llevar a cabo la convolución con las señales de audio, adquiridas de un base de datos de uso libre del sitio de internet [CKSDE, 2021].

Tabla 2 Datos de las tres respuestas al impulso utilizadas.

Nombre	BIG HALL E003 M2S.wav	GATED PLACE E001 M2S.wav	MEDIUM METAL ROOM E001 M2S.wav
Canales	2	2	2
Frecuencia de muestreo	48000	48000	48000
Bits por muestra	24	24	32
Total de muestras	784216	25184	338685
Duración	16.337833 s	0.524667 s	7.055937 s

La muestra de audio que se utilizó para las pruebas consta de dos canales de audio, a una frecuencia de muestreo de 96 kHz, tiene 24 bits de resolución y una duración total de 149.39 segundos o bien 2 minutos con 29 segundos.

3. Resultados

Todas las pruebas procesaron un total de 110 segmentos de audio y el número de segmentos procesados se calcula dividiendo la cantidad de muestras de entrada de audio entre el tamaño de cada segmento, en este caso 2^{17} o bien 131,072 muestras.

Resultados utilizando el equipo de prueba 1

Resultados con respuesta al Impulso Big Hall. En la tabla 3 se pueden observar el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso “BIG HALL E003 M2S.wav”, cuyas características ya fueron presentadas en la tabla 1. Sin el uso de GPU, se observa que, de los dos algoritmos de convolución rápida, el más eficiente es el de *Overlap-Save* teniendo una mejora del 105.22% con respecto al de *Overlap-Add*. Sin embargo, con el uso de la GPU, este mismo tiene un rendimiento de 108.21%.

En la figura 4 se presenta que, al utilizar una GPU, la entrada de audio se procesa hasta un 146.79% más rápido (utilizando el método *Overlap- Save*) con respecto al procesamiento normal de una CPU.

Tabla 3 Tiempo total de procesamiento - equipo 1 - Big Hall.

Método / # Prueba	Sin GPU (s)		Con GPU (s)	
	Overlap-Add	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	75.8909194	71.2230543	52.4442108	48.5171256
2	74.8396273	71.2589851	52.4443414	48.4612467
3	74.7276903	71.5838946	52.4413585	48.4417469
4	74.9814685	71.1205366	52.4404247	48.4438128
5	74.5551725	71.217445	52.441964	48.4596129
Promedio	74.9989756	71.28078312	52.44245988	48.46470898

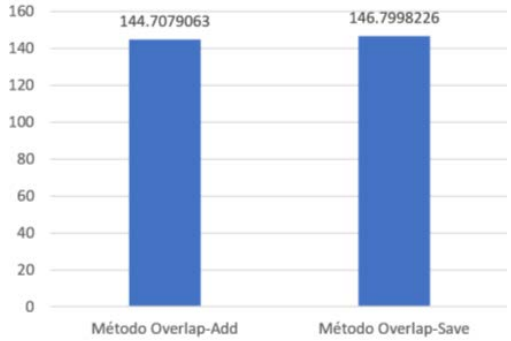

Figura 4 Gráfica del rendimiento entre algoritmos – equipo 1 - Big Hall.

Resultados con Respuesta al Impulso Gated Place

En la tabla 4 se puede observar el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso "GATED PLACE E001 M2S.wav", cuyas características ya fueron presentadas en la tabla 1.

Tabla 4 Tiempo total de procesamiento - equipo 1 - Gated Place.

Método / # Prueba	Sin GPU (s)		Con GPU (s)	
	Overlap-Add	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	13.318782	12.995768	10.285654	8.487259
2	13.235982	13.051056	9.341746	8.40628
3	13.646965	12.961998	9.339753	8.409163
4	13.287347	12.888554	9.345322	8.40426
5	13.886018	12.997716	9.3492	8.403366
Promedio	13.4750188	12.9790184	9.532335	8.4220656

Tal como se observó en la prueba pasada, el algoritmo de convolución más eficiente resulto ser el de *Overlap-Save* teniendo una mejora del 103.82% con respecto al de

Overlap-Add. Sin embargo, con el uso de la GPU, este mismo tiene un rendimiento de 113.18%. En la figura 5 se muestra que el rendimiento al utilizar la GPU la entrada de audio se procesa hasta un 153.12% más rápido (utilizando el método *Overlap-Save*) con respecto al procesamiento normal de una CPU.

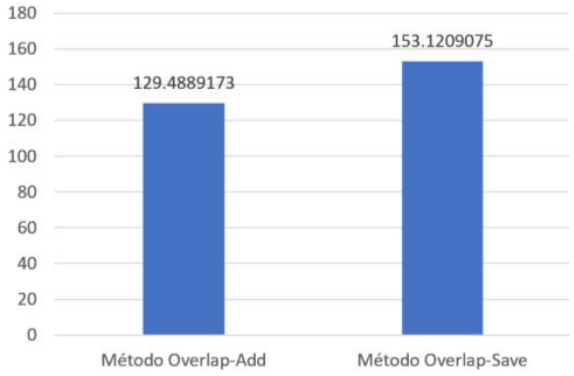

Figura 5 Gráfica del rendimiento entre algoritmos – equipo 1 - Gated Place.

Resultados con Respuesta al Impulso *Medium Metal Room*

Como última prueba, en la tabla 5 se puede observar el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso "MEDIUM METAL ROOM E001 M2S.wav", cuyas características ya fueron presentadas en la tabla 2.

Tabla 5 Tiempo total de procesamiento - equipo 1 - Medium Metal Room.

Método	Sin GPU (s)		Con GPU (s)	
# Prueba	*Overlap-Add*	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	120.520357	117.509432	27.486402	25.29513
2	120.058183	116.613715	27.445589	25.22074
3	118.318931	116.376678	27.450772	25.228579
4	118.30129	117.768816	27.477109	25.225867
5	118.671828	116.762422	27.449995	25.226475
Promedio	119.1741178	117.0062126	27.4619734	25.2393582

Al igual que en las otras dos pruebas pasadas, el algoritmo de convolución más eficiente resulto ser es el de *Overlap-Save* teniendo una mejora del 101.85% con respecto al de *Overlap-Add*. Sin embargo, con el uso de la GPU, este mismo tiene un rendimiento de 108.81%. La figura 6 muestra que, al utilizar una GPU, la entrada

de audio se procesa hasta un 464.55% más rápido (utilizando el método *Overlap-Save*) con respecto al procesamiento normal de una CPU.

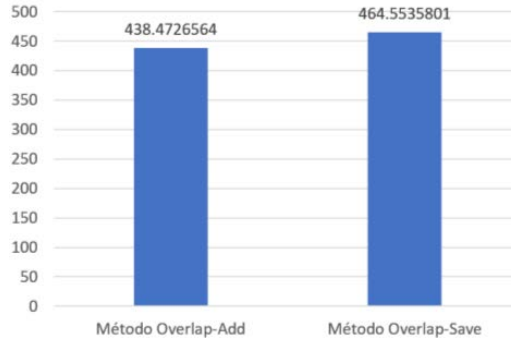

Figura 6 Gráfica del rendimiento entre algoritmos – equipo 1 - Medium Metal Room.

Resultados utilizando el equipo de prueba 2

En la tabla 6 se puede observar el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso "BIG HALL E003 M2S.wav". Sin el uso de GPU, se puede observar que, de los dos algoritmos de convolución rápida, el más eficiente es el de *Overlap-Save* teniendo una mejora del 105.54% con respecto al de *Overlap-Add*. Sin embargo, con el uso de la GPU, este mismo tiene un rendimiento de 107.431%. Como se puede ver, refiriéndose al desempeño entre ambos algoritmos, en ambos equipos se tiene un desempeño similar.

Tabla 6 Tiempo total de procesamiento - equipo 2 - Big Hall.

Método	Sin GPU (s)		Con GPU (s)	
# Prueba	*Overlap-Add*	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	77.438215	73.188471	35.797983	33.07181
2	77.303786	73.101117	35.372439	32.974876
3	77.184129	73.405767	35.372439	32.973925
4	76.876546	72.973284	35.360293	32.975505
5	77.287573	73.13661	35.343818	32.980806
Promedio	77.2180498	73.1610498	35.4493944	32.9953844

En la figura 7, se muestra que al utilizar una GPU la entrada de audio se procesa hasta un 221.3% más rápido (utilizando el método *Overlap-Save*) con respecto al procesamiento normal de una CPU.

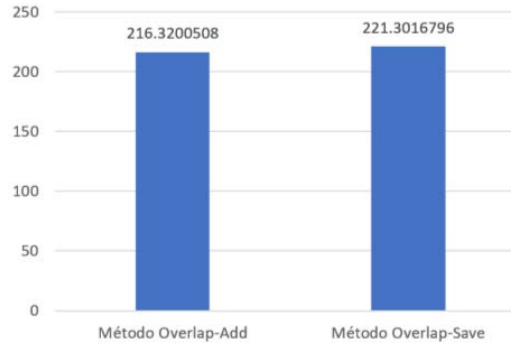

Figura 7 Gráfica del rendimiento entre algoritmos – equipo 2 - Big Hall.

En esta prueba, al comparar los resultados del equipo anterior, se puede observar una mejoría significativa en el tiempo de procesamiento. Recordando que se obtuvo un desempeño del 146.79% utilizando en equipo 1 contra el 221.3% logrado con este segundo equipo de prueba. En la tabla 7 se muestra el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso "GATED PLACE E001 M2S.wav". Siguiendo el mismo patrón, se puede observar que, de los dos algoritmos de convolución rápida, el más eficiente es el de *Overlap-Save* teniendo una mejora del 103.97% con respecto al de *Overlap-Add*. Utilizando una GPU para su procesamiento, este mismo tiene un rendimiento de 125.28%. En este caso, al utilizar la GPU, el equipo 2 tiene un desempeño mejor superando al equipo 1 por más del 10%.

Tabla 7 Tiempo total de procesamiento - equipo 2 - Gated Place.

Método	Sin GPU (s)		Con GPU (s)	
# Prueba	*Overlap-Add*	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	14.341408	12.995497	7.987326	5.07703
2	13.653348	12.886833	5.844114	4.993031
3	13.154547	12.942628	5.816557	4.989251
4	13.18385	13.243033	5.856971	4.987881
5	13.20993	12.955507	5.859161	4.986359
Promedio	13.508617	13.0046996	6.2728258	5.0067104

La figura 8, se muestra que al utilizar una GPU la entrada de audio se procesa hasta un 255.96% más rápido (utilizando el método *Overlap-Save*) con respecto al procesamiento normal de una CPU. En la figura 7 se puede observar gráficamente

el desempeño de estos algoritmos. Como se puede observar, al comparar los resultados del equipo anterior, se puede observar una mejoría significativa en el tiempo de procesamiento.

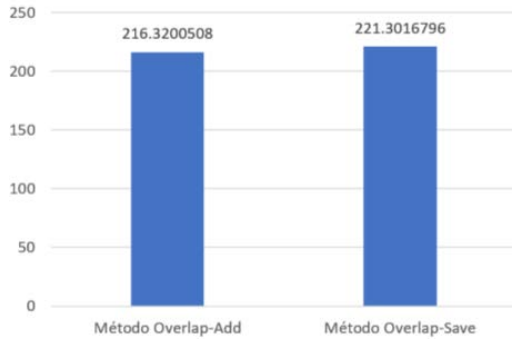

Figura 8 Gráfica del rendimiento entre algoritmos – equipo 2 - Gated Place.

Finalmente, la tabla 8 se puede observar el tiempo de procesamiento total en cada prueba utilizando la respuesta al impulso "MEDIUM METAL ROOM E001 M2S.wav". Al igual que en las pruebas anteriores, sin el uso de GPU, se puede observar que, de ambos algoritmos de convolución rápida, el más eficiente es el de *Overlap-Save* teniendo una mejora del 102.56% con respecto al de *Overlap-Add*. Sin embargo, con el uso de la GPU, este mismo tiene un rendimiento de 108.27%. Refiriéndose al desempeño entre ambos algoritmos, en ambos equipos se tiene un desempeño similar.

Tabla 8 Tiempo total de procesamiento - equipo 2 - Medium Metal Room.

Método	Sin GPU (s)		Con GPU (s)	
# Prueba	*Overlap-Add*	*Overlap-Save*	*Overlap-Add*	*Overlap-Save*
1	91.608063	89.224862	18.944287	17.533997
2	91.972667	89.403762	18.937831	17.488444
3	91.675359	89.165073	18.930728	17.483551
4	91.312311	89.455032	18.936387	17.459071
5	91.427032	89.307479	18.933922	17.482444
Promedio	91.5990864	89.3112416	18.936631	17.4895014

Por último, en la figura 9 se muestra que al utilizar una GPU la entrada de audio se procesa hasta un 508.86% más rápido (utilizando el método *Overlap-Save*) con respecto al procesamiento normal de una CPU.

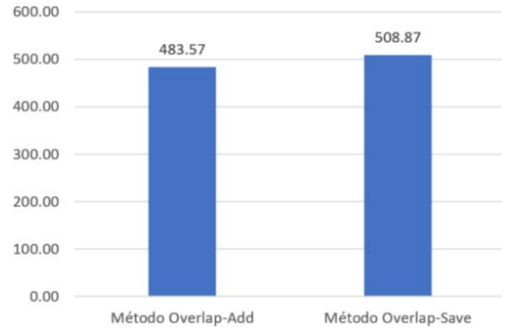

Figura 9 Gráfica del rendimiento entre algoritmos – equipo 2 - Medium Metal Room.

4. Discusión

Se puede observar en los resultados que el desempeño del sistema al utilizar una GPU se ve fuertemente afectado por las especificaciones de esta. En la figura 10 se muestra gráficamente la comparación del desempeño de ambos equipos a las distintas respuestas al impulso.

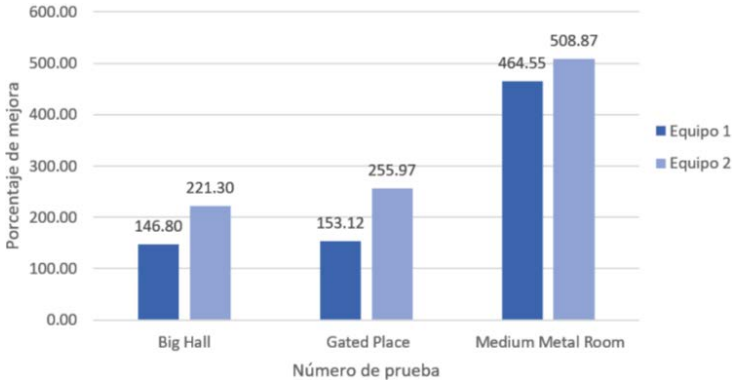

Figura 10 Gráfico comparativo del desempeño de ambos equipos de prueba.

En este caso, el equipo con mejor tarjeta gráfica, y por lo tanto con mejor desempeño, es el equipo 2. Sin embargo, el tener una tarjeta con el doble de ancho en el bus, más del doble de frecuencia de operación con la memoria, mayor número de núcleos, en general con mejores características, no garantiza que la mejor GPU trabajará al doble de la velocidad de procesamiento que la de menores prestaciones. La diferencia en velocidad de procesamiento es más notable al procesar información con mayor número de bits por muestra (32 bits para el audio Medium Metal y 24 para los otros dos).

5. Conclusiones

En este documento se presentaron los resultados obtenidos de la implementación de una reverberación por convolución en dos GPU. Se pudo comprobar que las señales de audio también obtienen un beneficio de velocidad al ser procesadas en una GPU, esto gracias a que CUDA permite implementar de una forma eficiente la FFT utilizada para convolución en frecuencia. Se comprobó que el uso de las GPU mejora el tiempo de procesamiento aplicado a la convolución y que con información que contenga mayor número de bits (32 bits) por muestra se mejora considerablemente el tiempo de procesamiento de 508.87% comparado con información que utiliza 24 bits en el que se obtuvo un 255.97% de mejora. Se recomienda el experimentar con respuestas al impulso de mayor tamaño, mayor frecuencia de muestreo y mayor resolución para observar el desempeño de la GPU al procesarlas. En este trabajo se procesaron señales de dos canales, como continuación de este proyecto, se podría procesar señales de aún más canales para medir el rendimiento del procesamiento de estas utilizando GPU. La mejora de la implementación de los métodos overlap-add y overlap-save se dio ya que son algoritmos que se pueden descomponer en múltiples convoluciones de segmentos cortos y por lo tanto se pueden calcular paralelamente.

6. Bibliografía y Referencias

[1] Arfib D., Keiler F, Zölzer U., Verfaille V., and Bonada J., DAFX: Digital Audio Effects. Second ed. 2011.

[2] Bhaskar J. and Chi-Kuang Sun, A GPU-Accelerated Modified Unsharp-Masking Method for High-Frequency Background- Noise Suppression, in IEEE Access, vol. 9, pp. 68746-68757, 2021.

[3] CKSDE, Cyber Kitchen Sound Design Enterprise, Impulse response audio files,: http://www.cksde.com/p_6_250.htm.

[4] Liévano Torres P. P., Espinosa Durán J. M. and Velasco Medina J., Implementación de algoritmos para efectos de audio digital con alta fidelidad usando hardware programable, Ingeniería y Universidad, vol. 17, no. 1, pp. 93-108, 2013.

[5] Enríquez F., Silva A., Torres A., Martínez A. y Bravo M., Utilización de GPU-CUDA en el Procesamiento Digital de Imágenes, Revista Cultura Científica y Tecnológica, Año 15, No. 66, ISSN 2007-0411, pp. 65-79, 2018.

[6] Nelson C., A Low-Spec Extendable GPU-Based Audio Library, International Conference on Image and Vision Computing New Zealand (IVCNZ), 2018.

[7] Yeongseok K., Youngjin P., Experimental Verification of CPU-GPU Architecture for ANC Algorithms, INTER-NOISE and NOISE-CON Congress and Conference Proceedings, Seoul, Korea, pages 1-989, pp. 607-613(7), 2020.