

Computación Paralela y Distribuida

Práctica 1

Denis Alexander Rodríguez Venegas, Jonathan Ricardo Galvis Gálvez,
David Fernando Guerrero Álvarez
Universidad Nacional Bogotá, Colombia
dearodriguezve@unal.edu.co, jrgalvisg@unal.edu.co, dafguerreroal@unal.edu.co



Figura 1: Escudo de la Universidad.

Resumen – En este documento se presenta el informe relacionado a la práctica 2, en donde se realizó la difuminación o efecto borroso sobre una imagen por medio de una operación de convolución haciendo uso de un kernel gaussiano. Dichas operaciones se realizaron sobre 3 imágenes con tamaños diferentes cambiando el número de hilos usados para realizar las dichas operaciones. Los resultados obtenidos en términos de tiempo y eficiencia se pueden observar en las gráficas que se presentarán más adelante.

Índice de Términos – Hilos, Kernel, Paralela, Convolución

I. INTRODUCCIÓN

El efecto borroso sobre una imagen, o también llamado desenfoque, es el resultado de aplicar operaciones matemáticas sobre una imagen, más exactamente sobre sus píxeles, modificando así sus niveles de intensidad.

Para la realización de esta práctica se implementó en C++ el algoritmo de desenfoque gaussiano, alterando los niveles de intensidad de cada píxel haciendo operaciones sobre los valores RGB de cada uno de los píxeles de la imagen en cuestión.

Como segunda medida de desarrollo para la práctica, se implementaron condiciones de paralelización con hilos Posix y también por medio de openMP.

Una vez ejecutadas todas las pruebas pertinentes por medio de un script de ejecución del programa, se anotaron los resultados con el fin de graficar y poder analizar con mayor precisión el rendimiento de la máquina virtual de Google sobre la cual se ejecutó el programa.

II. IMPLEMENTACIÓN

A. Desenfoque Gaussiano

Se hizo uso del efecto de desenfoque gaussiano el cual consiste en convolucionar la imagen original con una función gaussiana, esto es, operar la matriz de píxeles referentes a la imagen con una matriz o kernel gaussiano con el fin de modificar los niveles de intensidad de cada uno de los píxeles de la imagen.

La función gaussiana utilizada para esta práctica se expresa a continuación:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

Haciendo uso de dicha función, se creó el kernel o matriz gaussiana que operaría sobre la imagen original. Se tuvo en cuenta que la matriz resultante debería ser cuadrada de tamaño impar y, además, ésta debería ser simétrica. Dicha matriz sería sometida a una normalización para obtener así el kernel final.

B. Convolución

Una vez obtenido el kernel, se procede a desenfocar la imagen por medio de la convolución entre la matriz de la imagen original y el kernel obtenido.

Para obtener la matriz de píxeles de intensidad de la imagen se hizo uso de la librería Open CV2 la cual proporciona un método que permite obtener la intensidad del píxel representado por su RGB correspondiente a la imagen.

Seguido a esto, la lógica sobre la cual opera el desenfoque gaussiano nos lleva a multiplicar cada píxel por el kernel gaussiano. Esto se hace colocando el píxel central del kernel en el píxel de la imagen y multiplicando los valores de la imagen original con los píxeles del kernel que se superponen. Los valores resultantes de estas multiplicaciones se sumarán y ese resultado se utiliza para el valor en el píxel de destino.

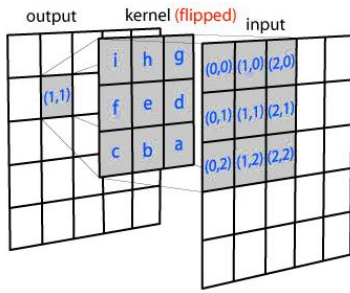


Figura 2. Convolución para desenfoque gaussiano.[2]

El resultado de estas operaciones se guarda sobre una imagen clonada de la imagen original, lo que significa que por medio de la librería OpenCV2, se modifican los valores originales de los píxeles de la matriz de la imagen clonada.

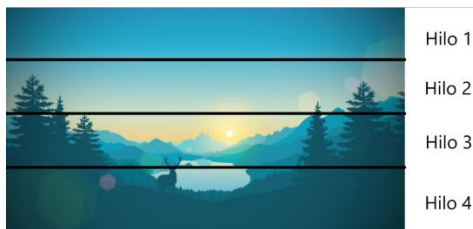
C. Hilos POSIX, openMP y balanceo de carga

La implementación de hilos POSIX tuvo en cuenta la entrada del atributo THREADS para definir la cantidad de hilos que se deben crear para operar la matriz, esto es, dividir la matriz referente a la imagen original, según la cantidad de hilos deseados en cada uno de los casos.

La implementación de openMP se realizó pasando como sección paralela el llamado a la función de convolución con su respectivo número de hilo.

De esta manera, cada uno de los hilos trabajará sobre una sección de la imagen, para ello, a cada hilo se le asigna como trabajo realizar la convolución antes mencionada, no de toda la matriz de la imagen, sino de su sección correspondiente dado el número de dicho hilo.

Para ser precisos, en la distribución de cargas se utilizó el método block – wise, por el cual se dividió cada región de la matriz en secciones por filas, en las cuales el inicio y el final están determinados por la cantidad total de hilos en las que la matriz es distribuida, en caso tal que fuesen más hilos que el tamaño total K de la matriz, los hilos operarían píxeles que también corresponderían a secciones de trabajo de otros hilos, por lo que habría píxeles doblemente operados en la imagen. Esto se explica en la siguiente imagen:



$$Y = (\text{Rows}/\text{threads}) * \text{idthreads}$$

Figura 3: Escudo de la Universidad.

D. Cálculos de tiempos de respuesta

Para realizar la medición del tiempo de respuesta o de proceso de desenfoque gaussiano de cada imagen bajo los respectivos parámetros (número de hilos, tamaño de kernel) se hizo uso de

clock_gettime. Para ello, el primer tiempo *ts1* se toma justo al iniciar el programa y el segundo tiempo *ts2* se toma justo después de sobrescribir los datos de la matriz referente a la imagen clonada, dado que en este punto todos los hilos ya han terminado su trabajo.

E. Script

El script de ejecución se realizó con el fin de ejecutar el programa para 3 imágenes diferentes. La primera de un tamaño de 720p, la segunda un tamaño de 1080p y la tercera un tamaño de 2160p (4k).

Para cada una de las imágenes, el programa se ejecuta una vez por cada tamaño de kernel de 3 a 15, como ya se mencionó, el tamaño del kernel debe ser impar, por lo tanto, los kernel que se toman son 3, 5, 7, 9, 11, 13 y 15.

Ahora bien, para cada uno de estos tamaños de kernel, el script ejecuta el programa con 1, 2, 4, 8 y 16 hilos.

Para obtener resultados más precisos, se ejecuta 10 veces el programa con los mismos atributos para lograr sacar un promedio de dichos datos, y así obtener los tiempos más puntuales.

Por lo tanto, lo que se está obteniendo son 350 ejecuciones del programa por cada una de las imágenes, lo que al final resulta en 1050 ejecuciones, éste es el número de datos de tiempos de ejecución que se obtienen al finalizar el script, pero como ya se mencionó, se toman los promedios de los datos para analizar los resultados generales.

Como bien se mencionaba anteriormente, cada ejecución arroja el tiempo que duró programa en realizar el desenfoque de la imagen con sus respectivos parámetros de entrada (hilos y tamaño de kernel).

Estos resultados de tiempos son guardados en un archivo de texto plano (*results.txt*) con el fin de graficar dichos datos en tablas que permitirán ver las comparaciones de rendimiento.

F. Hardware utilizado

La ejecución del programa se realizó sobre una máquina virtual de Google Cloud con las siguientes características:

- Núcleos: 16
- RAM: 16GB
- Disco Duro: SSD 50 GB
- Frecuencias: 3.4 GHz
- Procesador: Intel Xeon Skylake

III. RESULTADOS

A. Tablas de tiempos

A continuación, se presentan las tablas de resultados referentes a los tiempos (medido en segundos) de ejecución para cada una de las imágenes con sus respectivos parámetros.

Se muestran los resultados tanto con la implementación POSIX como con la implementación openMP:

Tabla 1. Tiempos Imagen 720p Posix

Imagen 720p.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	0.79537006	0.45516541	0.42128839	0.41840627	0.40446216
5	1.10410454	0.69588122	0.33153517	0.29755243	0.2678294
7	1.45481283	1.04335162	0.81373375	0.49718603	0.62151386
9	2.10119206	0.9725309	0.95361775	0.81138341	0.52253645
11	3.05517546	2.04797855	1.14906008	0.95170157	0.8263617
13	4.15623241	2.06325777	1.02450437	1.03920658	0.95158395
15	5.80506535	3.03041979	1.7062175	1.07140588	1.03718425

Tabla 2. Tiempos Imagen 1080p Posix

Imagen 1080p.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	1.03804573	0.82208491	0.64612788	0.48375794	0.46528313
5	1.83916745	1.06166083	0.94886871	0.68138477	0.65859676
7	3.18440834	1.93437864	1.17324843	1.05191604	0.94488902
9	4.47758144	2.45739911	1.44038054	1.03254209	1.11796397
11	7.15791862	3.71910127	2.11556348	1.05240735	1.10792527
13	9.04923476	4.80925139	2.49216083	1.62567253	1.23269415
15	12.0442097	6.02070805	3.23792851	2.10415614	1.84509579

Tabla 3. Tiempos Imagen 4k Posix

Imagen 4k.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	2.50146483	1.80235889	1.09797257	1.1176532	1.12862533
5	6.1159927	3.04468219	2.17239489	1.01863669	1.06271843
7	10.8127816	5.69743369	3.12111707	2.11669174	2.05281155
9	17.1787313	8.95230673	4.94818852	2.94692163	2.30178812
11	26.9534667	13.7452855	7.10982709	4.03641682	3.64342188
13	34.055723	17.2892263	9.11262191	5.04039514	4.31980367
15	48.1363742	24.2248161	12.6345973	6.66510323	5.96851282

Tabla 4. Tiempos Imagen 720p openMP

Imagen 720p.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	0.65915861	0.45497021	0.25594464	0.21542460	0.23956141
5	1.10136157	0.69436252	0.47998872	0.43415452	0.27450381
7	1.46467037	1.03698330	0.68962945	0.32948757	0.32295472
9	2.11418975	1.04443335	1.04678233	0.53384621	0.66240840
11	3.06667853	1.95395269	1.15191697	0.83863970	0.71483472
13	4.11334196	2.08277432	1.27347152	0.95832518	0.95058154
15	5.77580079	3.16180134	1.69769632	1.13926125	1.09995665

Tabla 5. Tiempos Imagen 1080p OpenMP

Imagen 1080p.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	1.114388527	0.583288546	0.649007972	0.502531402	0.620812736
5	1.945834192	1.153537284	1.052869376	0.534035949	0.527036038
7	3.111147878	1.825312135	1.077860048	0.744153919	0.837459994
9	4.785945474	2.2929754	1.279392859	1.178149922	1.042692535
11	6.963655063	3.825447193	2.172305936	1.064861523	1.094472743
13	8.888617281	4.679508786	2.503987112	1.340849571	1.239790212
15	12.13281448	6.00914084	3.319900696	2.039118182	2.05195707

Tabla 6. Tiempos Imagen 4k OpenMP

Imagen 4k.jpg					
kernel	1 hilos	2 hilos	4 hilos	8 hilos	16 hilos
3	2.502905337	1.424510924	1.137538132	1.041131904	1.045288518
5	6.07721304	3.064728337	1.961870174	1.044392225	1.10212772
7	10.82908487	5.807886986	3.179885014	2.042599593	1.854616153
9	17.05162057	9.23156129	4.949137749	2.826407796	2.47478627
11	27.61955051	14.12087804	7.09366319	4.100890501	3.825450439
13	34.23412147	17.64040075	9.111558628	5.118061511	4.458873638
15	48.94172166	24.21718202	12.57877046	6.517481347	6.12636181

B. Gráficas Tiempo

A continuación, se presentan las gráficas de tiempo vs número de hilos para cada una de las imágenes con cada uno de los kernel de operación, tomando los datos de las tablas anteriormente presentadas. En estas gráficas se podrá comparar el rendimiento con los diferentes números de hilos para las ejecuciones.

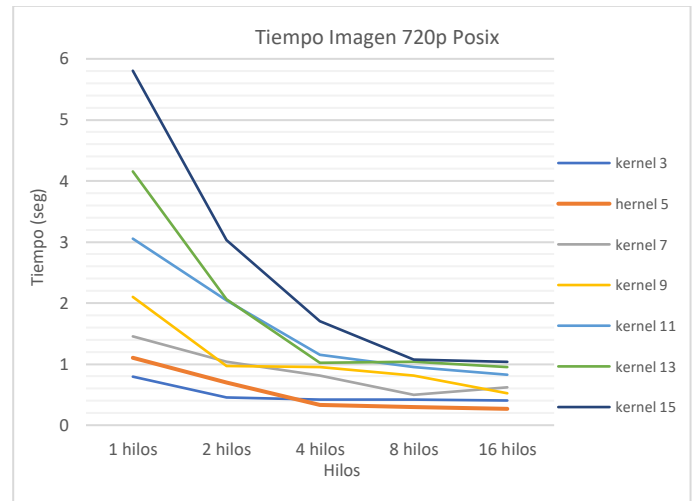


Figura 4. Imagen 720p. t vs hilos - Posix

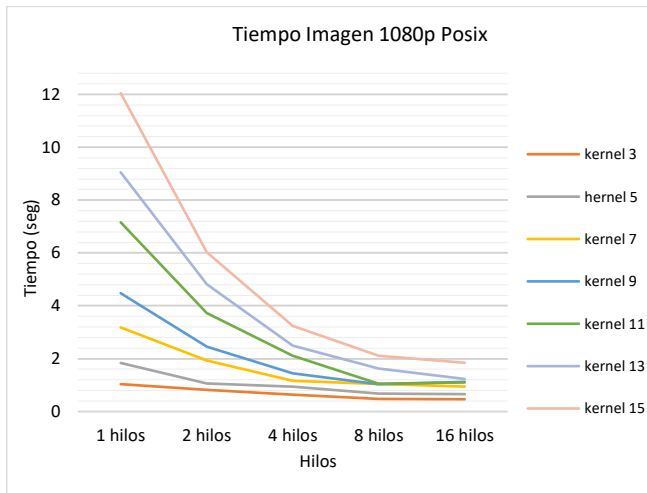


Figura 5. Imagen 1080p. t vs hilos – Posix

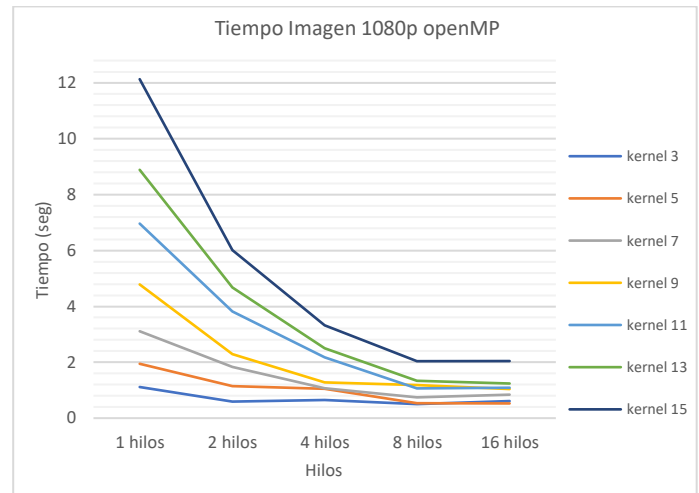


Figura 8. Imagen 1080p. t vs hilos – openMP

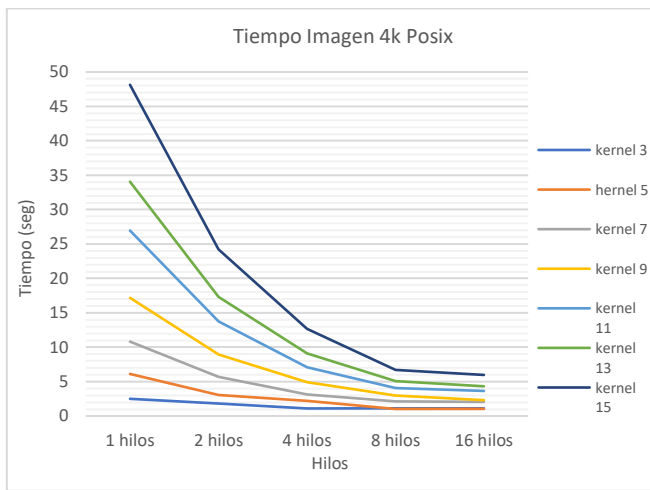


Figura 6. Imagen 4k. t vs hilos – Posix

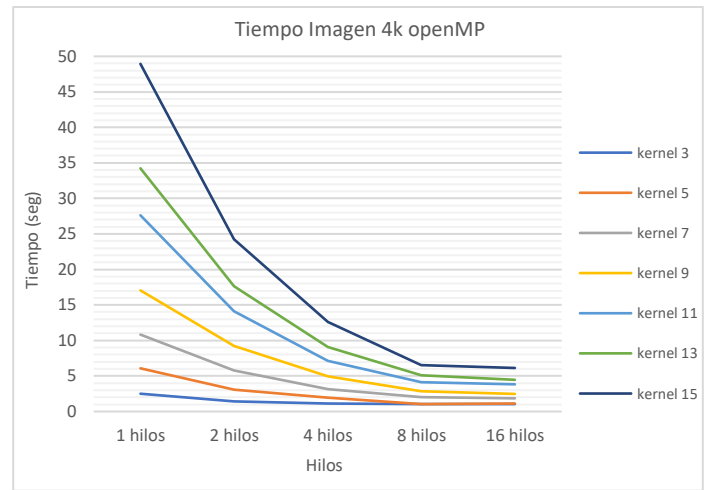


Figura 8. Imagen 4k. t vs hilos - openMP

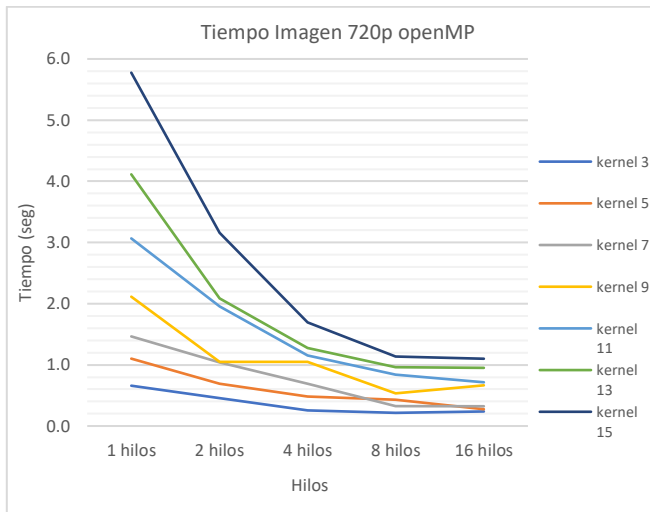


Figura 7. Imagen 720p. t vs hilos - openMP

IV. ANÁLISIS

A. SpeedUp

El SpeedUp representa la ganancia que se obtiene en la versión paralela del programa respecto a la versión secuencial del mismo. Por lo tanto, se presentarán las tablas respectivas al speedup de cada una de las imágenes para cada una de sus ejecuciones con los diferentes parámetros, esto es, el speedup referente a la ejecución secuencial (1 hilo) con cada ejecución para cada número de hilos y estos a su vez con cada tamaño de kernel.

La ecuación de speedup está definida por:

$$SpeedUp = \frac{Tiempo\ ejecución\ secuencial}{Tiempo\ ejecución\ paralela}$$

Tabla 7. SpeedUp Imagen 720p Posix

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.74743081	1.8879468	1.90095158	1.96648821
5	1.58662788	3.33027875	3.71062178	4.12241732
7	1.39436485	1.78782412	2.92609352	2.34075687
9	2.16054015	2.20339026	2.5896414	4.02113969
11	1.49180052	2.65884745	3.21022426	3.69714068
13	2.01440289	4.05682251	3.9994285	4.36769915
15	1.91559776	3.40230091	5.41817575	5.59694705

Tabla 8. SpeedUp Imagen 1080p Posix

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.26269892	1.60656391	2.14579574	2.23099801
5	1.73234936	1.93827389	2.69916136	2.79255467
7	1.64621769	2.71418078	3.02724573	3.37014005
9	2.45739911	1.44038054	1.03254209	1.11796397
11	1.92463665	3.38345727	6.80147156	6.46065113
13	1.88163064	3.63107977	5.56645611	7.3410219
15	2.00046399	3.71972687	5.72400947	6.52768801

Tabla 9. SpeedUp Imagen 4k Posix

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.38788387	2.27825804	2.23814044	2.21638198
5	2.00874585	2.81532273	6.00409624	5.75504531
7	1.89783369	3.46439474	5.10834022	5.26730356
9	1.91891675	3.47172126	5.82938181	7.46321138
11	1.96092447	3.79101578	6.67757267	7.39784399
13	1.96976558	3.73720355	6.75655816	7.88362749
15	1.98706871	3.8098859	7.22214984	8.06505333

Tabla 10. SpeedUp Imagen 720p openMP

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.448795	2.575395	3.059811	2.751522
5	1.586148	2.294557	2.536796	4.012191
7	1.412434	2.123851	4.445298	4.535219
9	2.024246	2.019703	3.960297	3.191671
11	1.569474	2.662239	3.656729	4.290053
13	1.974934	3.230023	4.292219	4.327185
15	1.826744	3.402140	5.069777	5.250935

Tabla 11. SpeedUp Imagen 1080p openMP

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.910527	1.717064	2.217550	1.795048
5	1.686841	1.848125	3.643639	3.692033
7	1.704447	2.886412	4.180785	3.714981
9	2.087221	3.740794	4.062255	4.589987
11	1.820351	3.205651	6.539494	6.362566
13	1.899477	3.549786	6.629094	7.169453
15	2.019060	3.654572	5.950030	5.912801

Tabla 12. SpeedUp Imagen 4k openMP

kernel	2 hilos	4 hilos	8 hilos	16 hilos
3	1.757028	2.200283	2.404023	2.394464
5	1.982953	3.097663	5.818899	5.514073
7	1.864548	3.405496	5.301619	5.838990
9	1.847100	3.445372	6.032965	6.890139
11	1.955937	3.893553	6.735013	7.219947
13	1.940666	3.757219	6.688884	7.677751
15	2.020950	3.890819	7.509300	7.988709

Dados los datos resultantes donde se observa la ganancia de paralelización al trabajar con 2, 4, 8 y 16 hilos frente a los tiempos secuenciales (1 hilo) para cada una de las imágenes operada con cada uno de los kernel, se pueden obtener las gráficas de rendimiento para los datos mencionados.

B. Gráficas SpeedUp

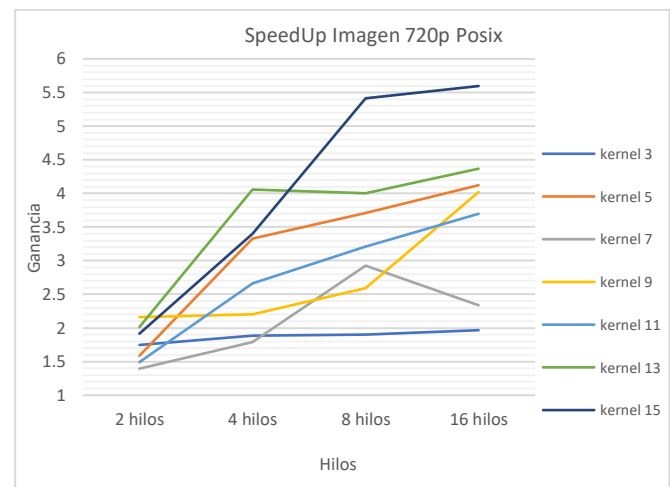


Figura 9. Imagen 720p. g vs hilos - Posix

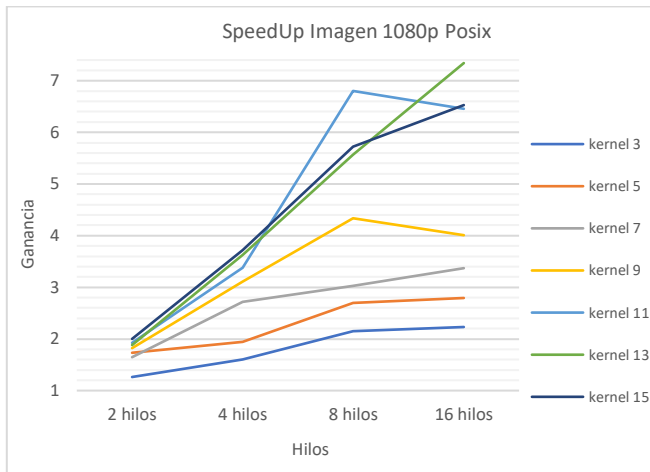


Figura 10. Imagen 1080p. g vs hilos - Posix

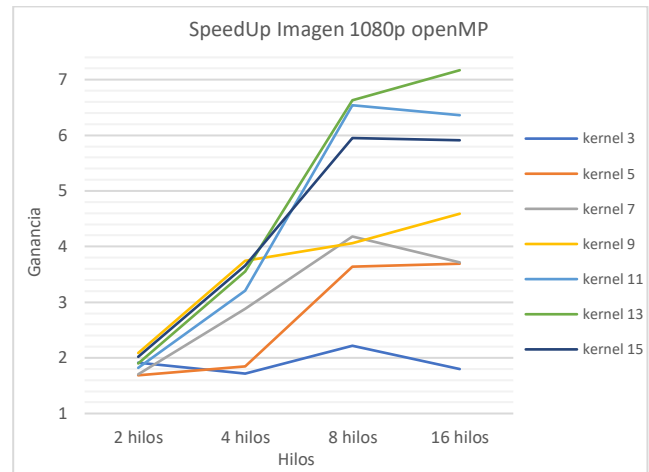


Figura13. Imagen1080p. g vs hilos - openMP

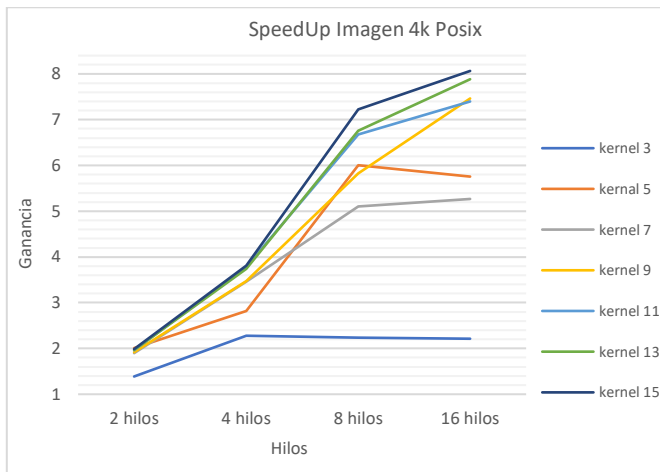


Figura 11. Imagen 4k. g vs hilos - Posix

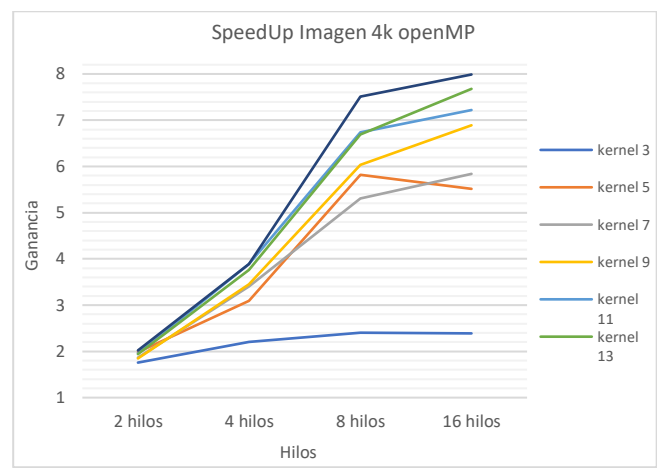


Figura14. Imagen4k. g vs hilos - openMP

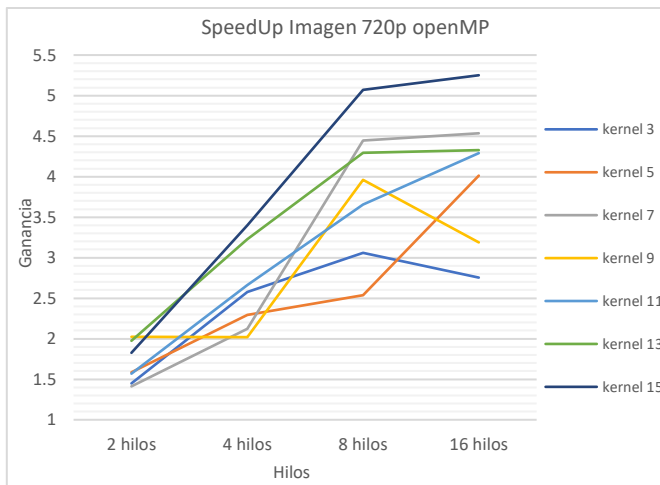


Figura12. Imagen 720p. g vs hilos - openMP

V. CONCLUSIÓN

- Dados los resultados presentados anteriormente, se puede afirmar que para el problema de desenfoque de imagen es pertinente realizar paralelización como lo muestran las gráficas de tiempo de ejecución y speedup, en donde a mayor cantidad de hilos se aprecia un mejor rendimiento.
- Se puede apreciar que la máquina virtual al ser dedicada mejora los tiempos de ejecución de los procesos.
- Las gráficas de speedup nos permiten afirmar que la escogencia del número de hilos para el procesamiento de la imagen también depende del tamaño del kernel.
- Para la imagen 720p, los parámetros óptimos son un tamaño de kernel de 15 y 8 hilos.
- Para la imagen 1080p, los parámetros óptimos son un tamaño de kernel de 13 con 16 hilos
- Para la imagen 4k, el comportamiento permite deducir que el procesamiento de la imagen funciona de manera correcta y similar desde el tamaño de kernel 9 (9,11,13,15) con 16 hilos.

- Se puede estudiar la posibilidad de procesar la imagen viéndola de manera unidimensional con el fin de reducir la complejidad a la hora de hacer la convolución y de esta manera reducir el tiempo de ejecución.

REFERENCIAS

- [1] tutorialspoint. Pone CV tutorial. Recuperado en marzo de 2019 en: <https://www.tutorialspoint.com/opencv/index.htm>
- [2] Computer Graphics Beta. Recuperado en marzo de 2019 en: https://computergraphics.stackexchange.com/questions/39/how-is-gaussian-blur-implemented?fbclid=IwAR2qK5NJ4zVZ5lTyvutp2g6-hVM_Br4TCPBfjSfX_WmKj9MBsrA2seofm-M
- [3] Codebin. How to Install OpenCV in Ubuntu 16.04 LTS for C / C++. Recuperado en marzo de 2019 en: <http://www.codebind.com/cpp-tutorial/install-opencv-ubuntu-cpp/>
- [4] Parallel Computing. Pedraza Cesar. Recuperado en marzo de 2019 en: <https://github.com/capedrazab/pc-20201/wiki>