Explorando tres niveles de paralelismo empleando software libre y freeware en computadoras personales multicore. El caso de Linux versus Windows

J. Jesús Arellano Pimentel*, Guadalupe Toledo Toledo**

Resumen

El advenimiento de las computadoras personales multicore con capacidades SIMD hace muy importante el uso del procesamiento paralelo a fin de lograr el mejor desempeño del hardware instalado. El presente proyecto de clase, correspondiente a la asignatura de Sistemas Distribuidos y Paralelos del séptimo semestre de la carrera de Ingeniería en Computación de la Universidad del Istmo, se enfocó a explorar tres niveles de granularidad del procesamiento paralelo: muy fino, fino y medio; además de algunas combinaciones de éstos. En dicho proyecto se emplearon herramientas y API's de software libre, en el caso de Linux (nasm con instrucciones SIMD, hilos Posix y OpenMP), y freeware en el caso de Windows (Microsoft Visual Studio Express 2012 con soporte para ensamblador en línea, hebras y OpenMP). El objetivo de este proyecto de clase es identificar hasta qué punto es conveniente incrementar el número de hilos o hebras en función del número de núcleos instalados en el sistema y cuál sistema operativo (Linux o Windows) reporta los tiempos de ejecución más consistentes.

Es importante mencionar que este tipo de proyectos de clase permiten concretar aspectos teóricos propios de las asignaturas. Para este caso en particular, se abordaron los temas de: especificación de procesos y tareas, plataformas de programación paralela, y como un caso de estudio de sistemas paralelos. Todos estos temas forman parte del programa de estudios oficial de la asignatura de Sistemas Distribuidos y Paralelos.

1. Introducción

A decir de [1] existen diversas razones para usar cómputo paralelo, entre ellas están:

- La computación paralela en comparación con la computación secuencial ofrece mejores ventajas para modelar, simular y entender fenómenos complejos del mundo real.
- Se hace un mejor uso del hardware ya que los procesadores actualmente cuentan con múltiples núcleos.
- Ayuda a disminuir tiempo de cómputo y dinero.
- Facilidad para resolver problemas que requieren grandes cantidades de procesamiento, debido a que se puede dividir un proceso en tareas muy pequeñas.

* jjap@sandunga.unistmo.edu.mx. Universidad del Istmo.
** gtoledo@sandunga.unistmo.edu.mx. Universidad del Istmo.

La paralelización se puede realizar a distintos niveles de granularidad [2] tal como se puede apreciar en la Figura 1. Para el caso particular de este proyecto se empleará la granularidad muy fina a través de instrucciones SIMD (acrónimo del inglés: Single Instruction Multiple Data), específicamente usando instrucciones MMX a bajo nivel (ensamblador). Para el nivel fino se hará uso del OpenMP y para el nivel medio se emplearán hilos o hebras según sea el caso en Linux o Windows, respectivamente.

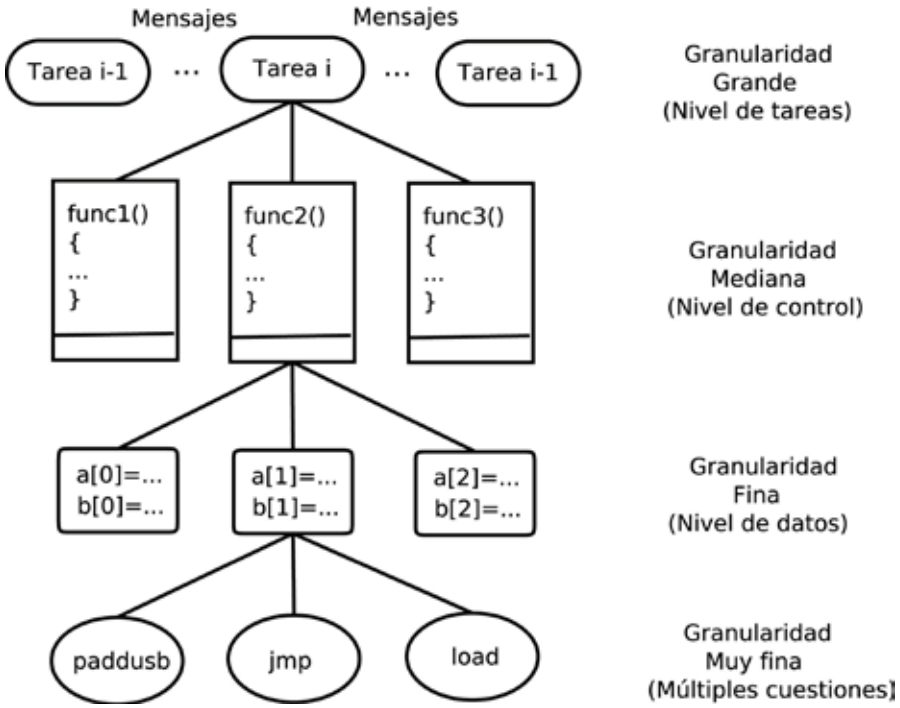

Figura 1. Niveles de granularidad.

Además de la granularidad, el balanceo de carga es un aspecto muy importante a considerar para asegurar la eficiencia de la ejecución en paralelo, ya que si la carga de trabajo no está correctamente balanceada, se podría estar esperando por la culminación de una tarea mientras las otras ya han finalizado, por tal motivo, se pretende que cada hilo o hebra procese la misma cantidad de trabajo. Los datos que se van a procesar son los elementos de una imagen digital, mejor conocidos como pixels, abreviatura de su denominación inglesa "picture elements" [3]. En la siguiente sección, se aborda con mayor detalle la metodología para procesar los pixels en cuatro diferentes tamaños de imagen, se mide el tiempo de ejecución durante su procesamiento con 1, 2, 4 y 8 hilos o hebras, así como el tratamiento estadístico de los tiempos de ejecución obtenidos.

2. Metodología

La metodología empleada en el presente proyecto de clase consistió en seis fases: 1) Selección del formato y tamaño de las cuatro imágenes a procesar, 2) Puesta a punto de las herramientas

de software libre y freeware a emplear en el procesamiento de las imágenes, 3) Codificar en cada Sistema Operativo los diferentes niveles de granularidad con 1, 2, 4 y 8 hilos, 4) Medir los tiempos de ejecución para 100 corridas en cada codificación, 5) Tratamiento estadístico de los tiempos de ejecución obtenidos mediante el cálculo del tiempo más probable y diagramas de cajas y bigotes, y 6) Análisis y discusión de los resultados obtenidos del tratamiento estadístico. A continuación se describen de la primera a la quinta fase; la sexta fase se abordará en el apartado 3 correspondiente a las Pruebas y resultados.

2.1 Tipos y tamaños de las imágenes

El tipo de imagen utilizado para ser procesado fue "bmp", es decir imágenes mapa de bits de 24 bits, ésto para garantizar la compatibilidad entre Windows y Linux. De [4] se recuperó código fuente libre escrito en el lenguaje C++ para abrir, guardar, y en general manipular las imágenes bmp de 24 bits. Este código secuencial se tomó como base para su posterior paralelización.

En lo que respecta al tamaño de las imágenes se partió de una imagen pequeña de 320x240 pixels, y se incrementó su tamaño en exactamente el doble, es decir, la segunda imagen quedó de 640x480 pixels, de igual manera se generaron las siguientes dos imágenes, de tal forma que sus tamaños fueron de 1280x960 y 2560x1920 pixels, respectivamente.

2.2 Herramientas de Software

Con excepción del sistema operativo Windows, todas las herramientas de software utilizadas para este proyecto de clase no tienen costo para el usuario final, siendo en su gran mayoría herramientas de software libre. Dichas herramientas se pueden clasificar en: las utilizadas en la programación y compilación del código para procesamiento de las imágenes (desarrollo), las utilizadas en el tratamiento estadístico (estadística) de los tiempos obtenidos, y las utilizadas para generar el presente documento (edición). El Cuadro 1 enlista las herramientas empleadas en Linux, todas ellas son software libre. Cabe mencionar que la distribución de Linux instalada en todos los equipos de cómputo es Fedora 23. En lo que respecta a Windows la única herramienta utilizada es el Microsoft Visual Studio Express 2012 con soporte para ensamblador en línea, hebras y OpenMP, esta versión de Visual Studio es freeware.

Cuadro 1. Herramientas de software libre utilizadas en Linux (Fedora 23)

Herramienta	Desarrollo	Estadística	Edición
geany	•		
g++	•		
nasm	•		
OpenMP	•		
RStudio		•	
OpenOffice Calc		•	
OpenOffice Writer			•
Dia			•

Una vez instaladas las herramientas listadas en el Cuadro 1 se procedió a codificar la granularidad en cada sistema operativo.

2.3 Codificación de la granularidad

La codificación de la granularidad, es decir, del paralelismo, se encuentra esquematizada en la Figura 2. En una primera instancia, los cuatro tamaños de imagen son procesados con código secuencial en tres variantes: codificación en C++ (.cpp), codificación en ensamblador (.asm), y codificación utilizando instrucciones SIMD (mmx). En éste último caso ya se tiene un procesamiento paralelo de granularidad muy fina al emplear instrucciones mmx. Esto se logra ya que con una sola instrucción mmx (paddusb) se procesan simultáneamente 8 bytes de cualquiera de las imágenes, en lugar de un solo byte por instrucción secuencial. Para los tres casos anteriores se levantan 1, 2, 4 y 8 hilos Posix, repartiendo equitativamente una porción de cada una de las imágenes para garantizar un balanceo de carga adecuado. Por otro lado, cada una de las cuatro imágenes también se procesan utilizando 2, 4 y 8 hilos con la librería de OpenMP. Todas estas codificaciones se realizaron en los sistemas operativos Linux en su distribución de Fedora 23, y Windows en su versión 7.

2.4 Tiempo de ejecución

El procedimiento para tomar el tiempo sigue un esquema muy conocido de tres pasos: 1) se toma el tiempo del sistema previo a la ejecución del código, 2) se ejecuta el código, y 3) se toma el tiempo tras la ejecución. Para el caso particular de este proyecto también se consideró el reparto de la carga de trabajo dentro del tiempo medible. En cada sistema operativo se emplearon funciones de librerías propias. Para el caso de Linux, se utilizó la función getttimeofday() y para el caso de Windows QueryPerformanceCounter().

2.5 Tratamiento estadístico de los tiempos de ejecución

Con el propósito de observar cómo se comporta cada sistema operativo en cada uno de los niveles de granularidad, con los diferentes tamaños de imagen, realizando 100 ejecuciones de prueba por cada caso, es posible utilizar el cálculo del tiempo más probable y los diagramas de cajas y bigotes.

El cálculo del tiempo más probable permitirá conocer con cierta certeza qué tiempo se lleva cada ejecución. Este cálculo involucra ciertas fórmulas de probabilidad y estadística tales como el promedio, la desviación estándar y una consideración del error en un 50% como el sugerido en [5]. Siendo σ la desviación estándar de la media y considerando un error del 50%, el error probable se calcula como:

$$Ep = 0.6745 \times \sigma$$

Una vez calculado el Ep (error probable) el valor más probable se calcula como:

$$X = \mu \pm Ep$$

donde μ representa el promedio.

Por su parte, el diagrama de cajas y bigotes es una presentación visual que describe al mismo tiempo varias características importantes de un conjunto de datos, tales como el centro, la dispersión, la simetría o asimetría y la identificación de datos atípicos [6]. Una gráfica de este tipo (ver Figura 3) consiste en una caja rectangular, donde los lados más largos muestran el recorrido intercuartílico. Este rectángulo está dividido por un segmento vertical que indica dónde se posiciona la mediana y por lo tanto su relación con los cuartiles primero y tercero.

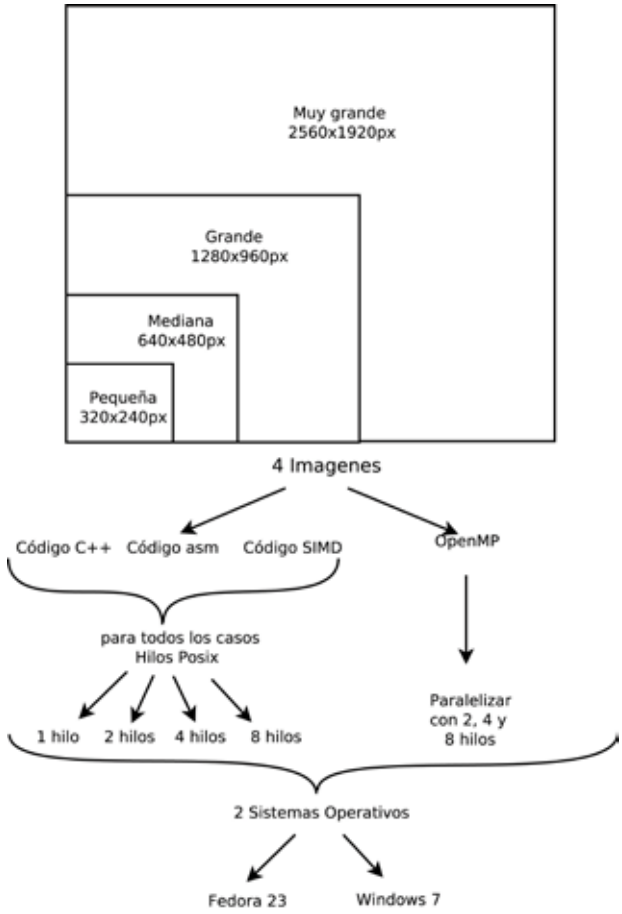

Figura 2. Esquema del paralelismo realizado.

3. Pruebas y resultados

Los equipos de cómputo personal utilizados para la ejecución de las pruebas con sus correspondientes características se muestran en el Cuadro 2.

Cuadro 2. Equipos de cómputo personal utilizados en las pruebas.

Equipo de cómputo	***CPU***	***Número de núcleos***	***RAM***
Laptop Hp	Celeron a 1.86 Ghz de 64 bits	2	4 GB
Dell Vostro	Intel i5 a 2.67Ghz de 32 bits	4	4 GB
Dell Precision T1700	i5 4a Gen. a 3.5 Ghz de 64 bits	4	8 GB

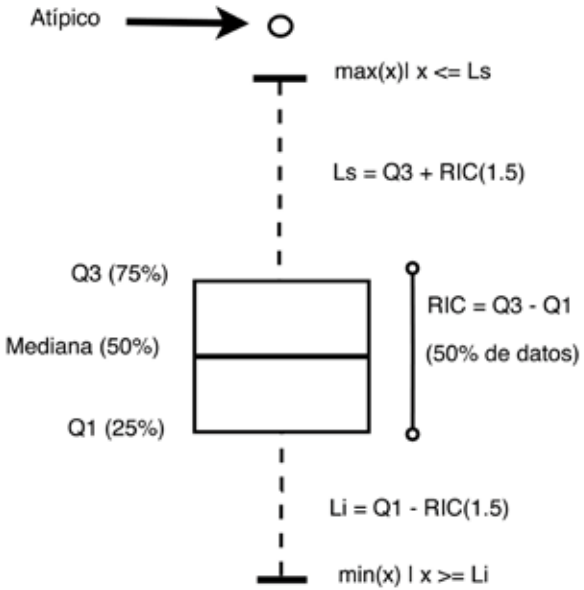

Figura 3. Diagrama de cajas y bigotes.

Cabe mencionar que para la compilación de todos los códigos tanto en Linux como en Windows no se emplearon directivas de optimización en la generación de código para los compiladores g++ o Visual Studio. Además, tampoco se alteró la prioridad de los procesos, dejando completamente el control de la planificación a los algoritmos que se emplean en cada sistema operativo de forma automática [7]. De esta forma se asegura una comparativa más justa entre ambos sistemas operativos.

El número de pruebas realizadas para cada tipo de granularidad (ver Figura 2) fue de 100 ejecuciones. Si se consideran los 4 tamaños de imagen, por 3 tipos de codificación (cpp, asm, mmx), por 4 configuraciones de hilos Posix por 2 sistemas operativos, se obtienen 9600 pruebas. Además, considerando 3 codificaciones distintas con OpenMP por los 4 tamaños de imagen para los 2 sistemas operativos realizando 100 ejecuciones, se obtienen otras 2400 pruebas más. Así la suma total ejecuciones es de 12000 por equipo de cómputo, y son 3 equipos, así que finalmente se consiguieron 36000 ejecuciones.

Dada la gran cantidad de pruebas realizadas sólo se mostrarán los resultados más representativos mediante gráficas de dispersión con puntos y líneas suavizadas de los tiempos más probables, así como diagramas de cajas y bigotes para los casos más representativos de la comparativa entre los tiempos reportados en Linux y Windows.

3.1 Resultados del tiempo más probable

Los resultados arrojados por el equipo de cómputo HP Laptop ejecutándose las pruebas en Linux muestran una mayor consistencia en cuanto a la proximidad del tiempo más probable positivo y el tiempo más probable negativo respecto al tiempo promedio, esto es prácticamente consistente para todos los casos de prueba. La gráfica de la Figura 4 muestra un caso representativo utilizando codificación en ensamblador (asm) con 4 hilos en ambos sistemas para los diferentes tamaños

de imágenes, considerándose como indicador para el tamaño pequeño (1), mediano (2), grande (3) y muy grande (4) en el eje x.

Para el caso de la implementación secuencial, es decir con un hilo en cpp y asm, ambos sistemas operativos muestran consistencia con las imágenes pequeña, mediana y grande; sin embargo, en Windows para la imagen muy grande presenta cierta dispersión de los tiempos más probable positivo y negativo respecto al tiempo promedio, además, para estos mismos casos, el tiempo en Linux siempre es menor. Un aspecto interesante observado con la combinación de instrucciones mmx e hilos Posix es que los tiempos más probables se comportan más dispersos respecto al tiempo promedio cuando el número de hilos se incrementa, no obstante, los tiempos reportados respecto al resto de las implementaciones (cpp, asm, y OpenMP) son menores, algo similar pasa en Windows pero con tiempos relativamente mayores.

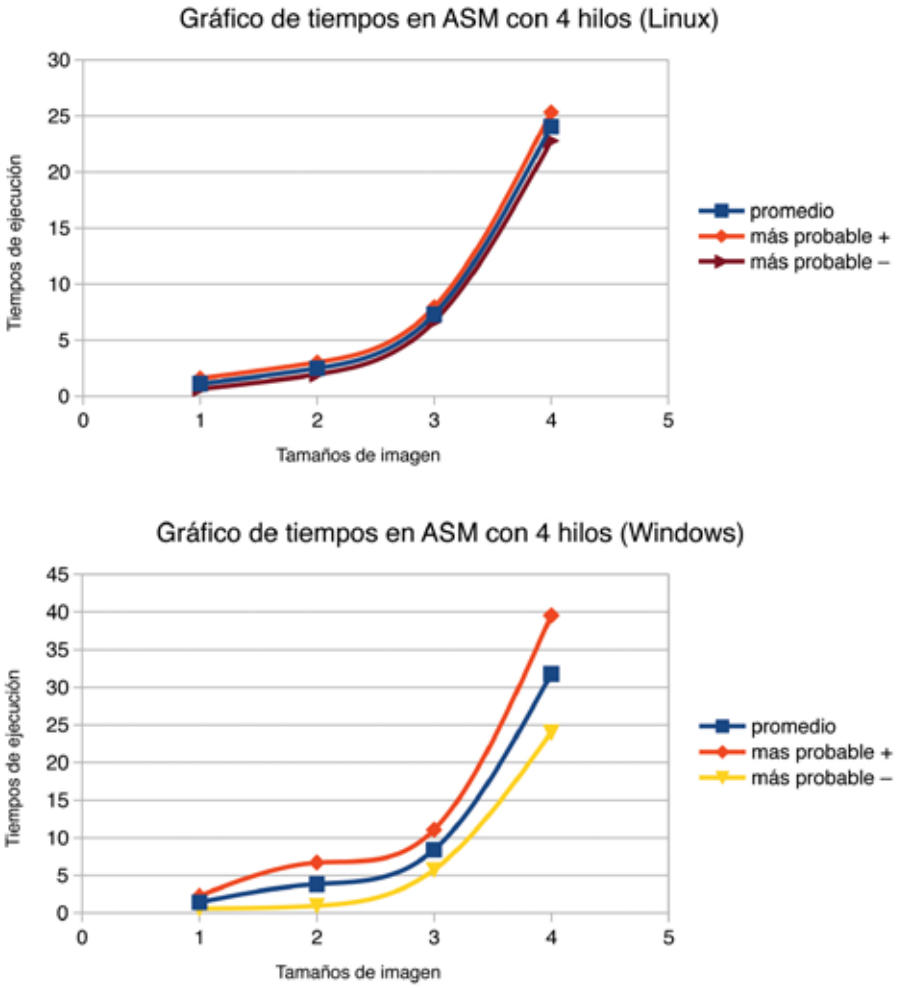

Figura 4. Comparativa del valor más probable entre Linux y Windows para la implementación en ensamblador con 4 hilos.

Los resultados obtenidos de las pruebas realizadas en el equipo de cómputo Dell Vostro muestran que en la mayoría de los casos cuando se emplean 2 hilos, sin importar la implementación cpp, asm o mmx, los mejores tiempos probables se encuentran en Windows, mientras que a medida que se incrementa el número de hilos (a partir de 4 hilos), los mejores tiempos probables en la gran mayoría de las veces ocurren en Linux (ver Figura 5). Además, se puede apreciar una diferencia considerable en las ejecuciones de OpenMP siendo Linux el sistema operativo con los mejores tiempos, sin importar el número de hilos.

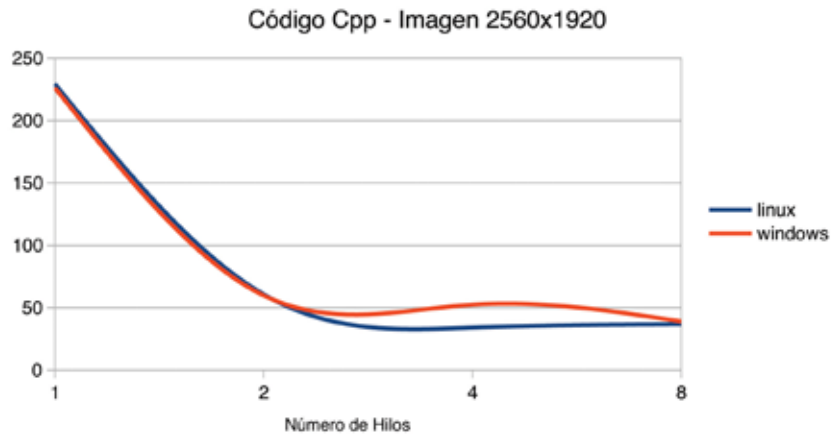

Figura 5. Comparativa del valor más probable positivo entre Linux y Windows para la implementación en cpp con 1, 2, 4 y 8 hilos.

Las pruebas realizadas en el equipo Dell Precision muestran un comportamiento similar a las realizadas en los otros dos equipos de cómputo. Presentan en su gran mayoría mejores tiempos más probables en las ejecuciones de Linux que en las ejecuciones de Windows. Además, la proximidad de los tiempos más probables positivos y negativos son más próximos al tiempo promedio en los resultados de las pruebas en Linux que en Windows, pero a diferencia de los otros dos equipos de cómputo en éste, en particular se observó que entre mayor era el tamaño de la imagen, los tiempos más probables en las pruebas de codificación cpp para ambos sistemas operativos se alejaban más. En la Figura 6 se puede apreciar un caso representativo (1 indica pequeña, 2 mediana, 3 grande, y 4 muy grande).

3.2 Resultados en formato de gráficas de cajas y bigotes

La construcción de las gráficas de cajas y bigotes se realizó empleando un sencillo script escrito en el lenguaje R [8]. A continuación se muestran las más representativas en cada uno de los equipos de cómputo utilizados empleando la imagen de mayor tamaño.

El gráfico de cajas y bigotes obtenido de los tiempos de ejecución para el equipo Laptop HP con la implementación en ensamblador (asm) se muestra en la Figura 7.

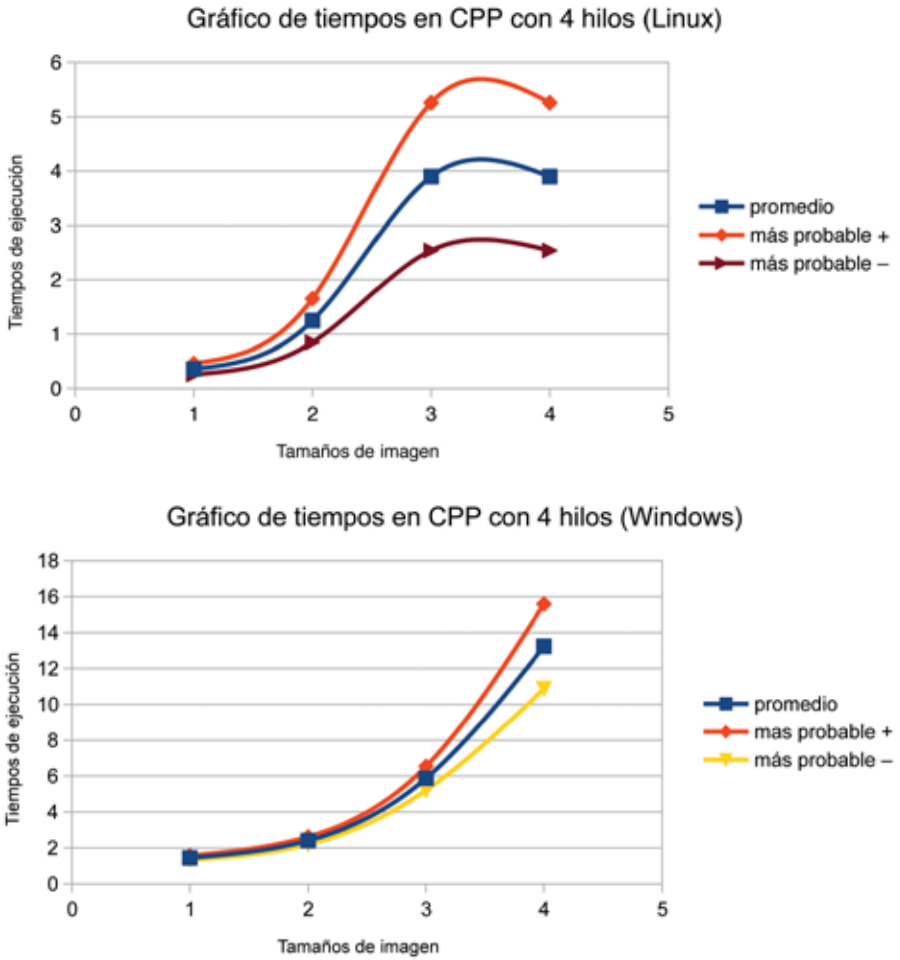

Figura 6. Comparativa del valor más probable entre Linux y Windows para la implementación en cpp con 4 hilos.

La Figura 7 presenta cómo Linux resulta más estable en cuanto a los tiempos de ejecución obtenidos, con datos atípicos menos alejados de la mayoría. Además, otro aspecto a resaltar de dicha figura es que se puede observar lo mencionado en la Ley de Amdahl [9] respecto a alcanzar el máximo de rendimiento en un sistema con n núcleos de procesamiento, en este caso el equipo en cuestión cuenta con sólo dos núcleos.

En todas las pruebas realizadas la combinación de hilos o hebras con instrucciones mmx generaron los menores tiempos que sus contrapartes implementadas en cpp, asm u OpenMP en cualquiera de los sistemas operativos, no obstante también fue en esta implementación (mmx) donde ocurrieron la mayor cantidad de datos atípicos, si se consideran todos los ocurridos en los tres equipos de cómputo. La Figura 8 muestra la gráfica de cajas y bigotes correspondiente a la ejecución de pruebas de la

implementación mmx con 2, 4 y 8 hilos o hebras en Linux y Windows realizadas en el equipo de cómputo Dell Precision, aquí también se puede observar que Linux es más consistente que Windows respecto a los tiempos obtenidos, inclusive es evidente que la ejecución con 8 hebras en Windows genera tiempos mayores que con 4 hebras.

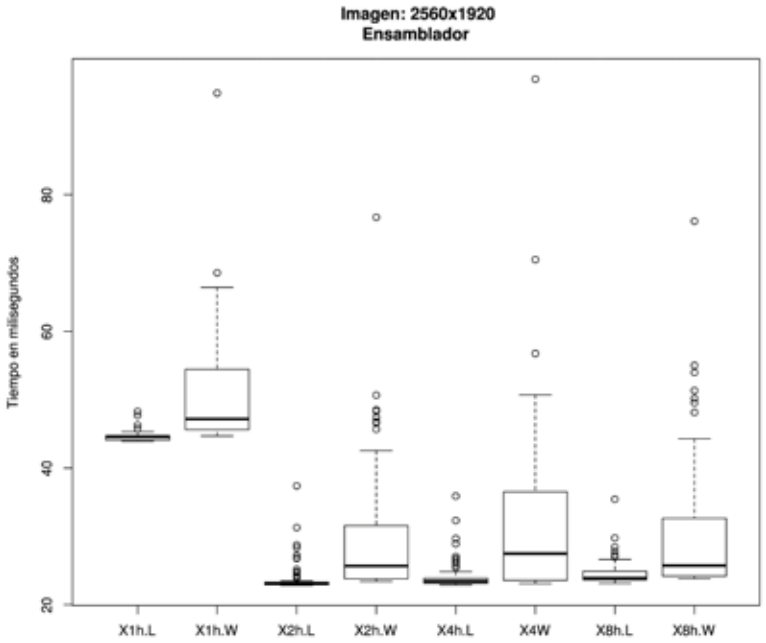

Figura 7. Diagrama de cajas y bigotes correspondiente a la implementación en ensamblador y pruebas ejecutadas en el equipo Laptop HP.

En lo que respecta a la implementación en OpenMP el comportamiento de los tiempos de ejecución son similares en los tres equipos. En este sentido es de resaltar que la implementación en Windows genera prácticamente los mismos tiempos, sin importar el número de hilos programados; sin embargo, esto no ocurre en Linux ya que los tiempos generados corresponden mejor a lo esperado al incrementar el número de hilos en cada equipo de cómputo. La Figura 9 muestra el diagrama de cajas y bigotes correspondiente a la implementación en OpenMP y pruebas ejecutadas en el equipo de cómputo Dell Vostro donde es posible apreciar lo antes mencionado.

Conclusiones

Con base en el análisis de los resultados obtenidos se puede concluir que Linux genera en la gran mayoría de las pruebas los tiempos más consistentes y menores que los generados en Windows. No obstante, vale la pena resaltar que Linux no está exento de generar datos atípicos, que en algunos casos (los menos) superan considerablemente los tiempos generados en su contraparte

Windows. Además, las gráficas de cajas y bigotes muestran que, una vez que el número de hilos es igual al número de núcleos instalados en el sistema, ya no se generan mejores tiempos, e inclusive pueden generarse mayores tiempos de ejecución.

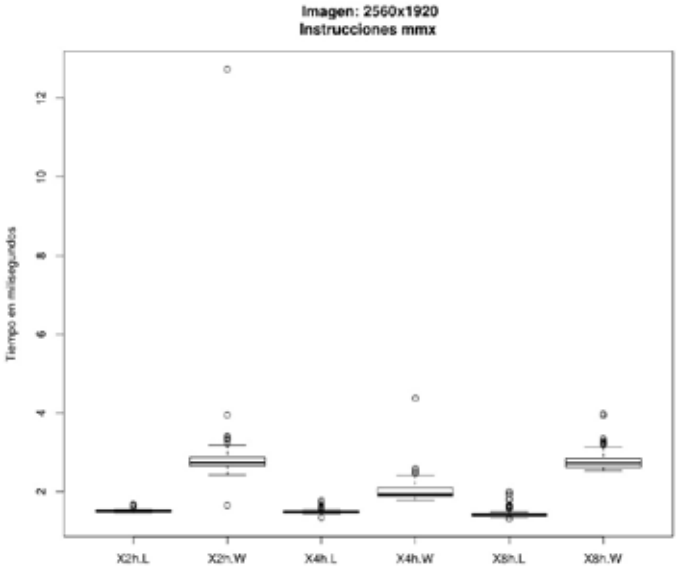

Figura 8. Diagrama de cajas y bigotes correspondiente a la implementación en mmx y pruebas ejecutadas en el equipo Dell Precision.

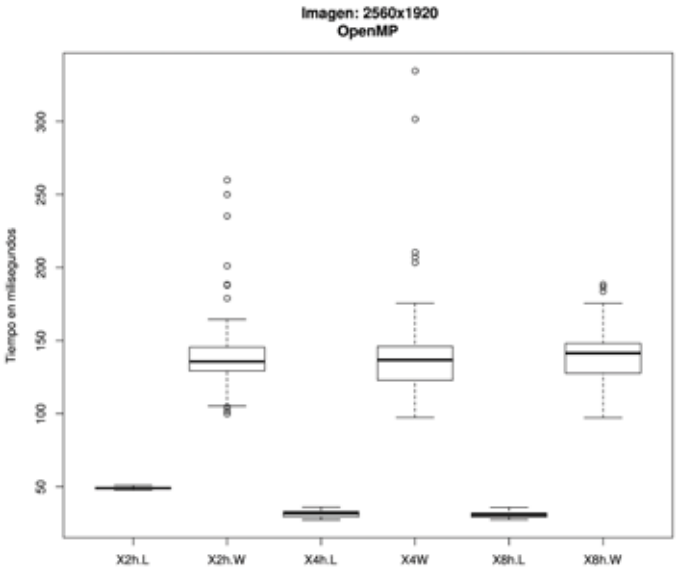

Figura 9. Diagrama de cajas y bigotes correspondiente a la implementación con OpenMP y pruebas ejecutadas en el equipo Dell Vostro.

Es de resaltar que las herramientas de software libre utilizadas durante el desarrollo (implementación), estadística y edición de este proyecto, proporcionaron prestaciones iguales e inclusive superiores respecto a sus contrapartes en software privativo. Además, como trabajo futuro se planean utilizar herramientas de software libre, tales como Score-P o Paraver a fin de mostrar el aprovechamiento de los recursos computacionales.

Referencias

[1] Barney, B. (2016). Introduction to parallel computing, 2014. Accedido: 18 de abril del 2017.
[2] Buyya, R. (2000). The design of paras microkernel. Parallel Computing at a Glance.
[3] González, R. C., & Woods, R. E. (1996).Tratamiento digital de imágenes.
[4] Mena Christiam. Cargar y procesar imágenes BMP de 24bits con C++. 2013. Recuperado de: http://www.unistmo.edu.mx/~jjap/BMP24bits.zip
[5] Miguel, J. (1993). Teoría de errores de observación. Departamento de Física de la Tierra, Astronomía y Astrofísica, 133.
[6] Minaard, C., Condesse, V., & Rabino, C. (2005). Los gráficos de Caja: un recurso innovador. Revista Iberoamericana de Educación,35(8), 237.
[7] Anthu P. & Rohini V. Performance Evaluation of Linux and Windows Operating System. 2017. International Journal Of Innovative Research in Advanced Engineering. Issue 03, Vol 4.
[8] Ahumada, J. A. (2003). R para Principiantes. University of Hawaii.
[9] Molero, X., Juiz, C., & Rodeño, M. (2004). Evaluación y modelado del rendimiento de los sistemas informáticos. Prentice Hall.