ANÁLISIS DEL RENDIMIENTO DE PROCESADORES MULTINÚCLEO EMBEBIDOS EN DISPOSITIVOS DIGITALES AL EJECUTAR APLICACIONES PARALELAS

Jimmy Josué Peña Koo

Instituto Tecnológico Superior del Sur del Estado de Yucatán

jimjpk@hotmail.com

Luis Fernando Curi Quintal

Universidad Autónoma de Yucatán, Facultad de Matemáticas

cquintal@uady.mx

Orlando Adrian Chan May

Instituto Tecnológico Superior Sur del Estado de Yucatán

orlandochan@itsyucatan.edu.mx

Resumen

Esta investigación ha sido posible gracias a las facilidades brindadas por el Instituto Tecnológico Superior del Sur del Estado de Yucatán y la Universidad Autónoma de Yucatán. El presente trabajo analiza el rendimiento de procesadores multinúcleo embebidos en dispositivos digitales al ejecutar aplicaciones paralelas desarrolladas con Python, C++, OpenMP y Boost. El rendimiento es analizado a partir de la medición y cálculo de tres indicadores: tiempo de ejecución, aceleración y eficiencia. Los procesadores multinúcleo reúnen varias unidades de procesamiento energéticamente eficientes en un solo microprocesador, pudiendo ser aprovechados al máximo si las aplicaciones se diseñan bajo el paradigma de computación paralela. Se aplicó la metodología de programación paralela en espiral para la implementación de dos aplicaciones: multiplicación de matrices y convolución de imágenes. Se ejecutaron en diversos dispositivos digitales con procesador multinúcleo embebido registrando los tiempos de ejecución para su

análisis. Los resultados demostraron la mejora del rendimiento al reducir el 73% del tiempo total de ejecución, alcanzando eficiencia de hasta 94% con cuatro núcleos.

Palabras Claves: Boost, OpenMP, procesador multinúcleo, programación paralela.

Abstract

This research has been possible thanks to the support made by the Instituto Tecnológico Superior del Sur del Estado de Yucatán and Universidad Autónoma de Yucatán facilities. In this document the performance of embedded multicore processors in digital devices is analyzed when running parallel applications developed using Python, C ++, OpenMP and Boost. Analysis of performance is based on three indicators: execution time, speed-up and efficiency. Multicore processors cluster several energy-efficient processing units in a single microprocessor and its best performance can be achieved if the applications are designed using the parallel computing. A spiral methodology for parallel programming was applied to implement two applications: matrix multiplication and image convolution. A variety of digital devices with embedded multicore processors were used to execute the programs and record execution times for analysis. The results showed an improvement in performance by reducing 73% of the total execution time, reaching up to 94% efficiency with four cores.

Keywords: *Boost, OpenMP, multicore processor, parallel computing.*

1. Introducción

Dada la necesidad de procesar una amplia variedad de problemas complejos en un tiempo razonable, los desarrolladores de sistemas de computación comenzaron a unir sus computadoras para trabajar como una sola. Este fue uno de los inicios de la computación paralela, enfatizando el tratamiento concurrente de un conjunto de datos usando más de un procesador con el objetivo de resolver un mismo problema.

La programación paralela es una técnica de programación basada en la ejecución simultánea de varias tareas, la cual solo era aplicada a la computación de altas prestaciones pero últimamente el interés ha crecido debido a las restricciones físicas que impiden el escalado en frecuencia de reloj (Chichizola, 2013).

El desarrollo tecnológico enfocado a la integración de múltiples transistores en un mismo procesador implica el aumento de escalado en frecuencia, operaciones ejecutadas simultáneamente y segmentación, lo cual permite continuar el paradigma de programación secuencial. Sin embargo, este desarrollo llegó a su límite, dando lugar a una nueva generación de procesadores al incrementar el número de núcleos de procesamiento secuencial en el mismo chip para obtener incrementos del rendimiento, en vez de diseñar un procesador con una única unidad de proceso más rápida y con mayor capacidad de cómputo (Amdahl, 2013). Debido a este cambio en el diseño de procesadores implementado en computadoras, en sistemas de cómputo móvil y embebido, es necesario un cambio de paradigma de programación.

Para evaluar el desempeño de un sistema de computación y compararlo respecto a otro se necesita definir y medir su rendimiento; esto implica determinar los factores que influyen en el desempeño del equipo de cómputo y así definir una expresión que lo caracterice.

El elemento común empleado para determinar la medida del rendimiento de un equipo de cómputo es el tiempo de ejecución (Cardinale, 2016). Para una aplicación secuencial, el rendimiento es evaluado por su tiempo de ejecución como función del tamaño del problema, cuyo comportamiento es idéntico en cualquier plataforma secuencial. Por su parte, el tiempo de ejecución de una aplicación programada en paralelo depende del tamaño del problema, del número de procesadores y de ciertos parámetros de comunicación de la plataforma. Es por ello que las aplicaciones paralelas deben ser evaluadas y analizadas teniendo en cuenta también la plataforma, tomando en cuenta adicional al tiempo de ejecución como medida de rendimiento también la aceleración y la eficiencia.

Para analizar el rendimiento de las plataformas con procesadores multinúcleo embebidos al ejecutar aplicaciones desarrolladas bajo el paradigma de

programación paralela, se desarrollaron dos aplicaciones utilizando C++, OpenMP y Boost en la parte central de las aplicaciones y Python para el desarrollo de la interfaz de usuario. Dichas aplicaciones fueron ejecutadas y evaluadas en secuencial y en paralelo, tomando como métricas del rendimiento: el tiempo de ejecución, la aceleración y la eficiencia.

2. Método

Metodología de la Investigación

Con base en la clasificación de Hernández (2010) esta investigación tiene un diseño experimental de clase cuasi-experimental con tratamientos múltiples, porque se manipulan deliberadamente las dos variables independientes (procesos secuenciales y paralelos) para observar su efecto y relación con la variable dependiente (rendimiento). Adicionalmente, los grupos de estudio no se asignaron al azar ni se emparejaron, pues son los dispositivos con procesadores multinúcleo embebidos que se tenían disponibles para pruebas y experimentación. El diseño de la investigación está representado con la siguiente simbología:

G X1 O X2 O

Donde:

- G Es el conjunto de dispositivos digitales con procesadores multinúcleo embebidos empleados para las pruebas y experimentación durante la recolección de información.
- X1 Tratamiento que consiste en la ejecución de aplicaciones secuenciales.
- X2 Tratamiento que consiste en la ejecución de aplicaciones paralelas.
- O Medición del rendimiento de las aplicaciones en el dispositivo ejecutado, basado en tres indicadores: tiempo de ejecución, cálculo de la aceleración y eficiencia.

Metodología para el desarrollo de la aplicación paralela

Como implementación de la aplicación paralela se desarrolló la multiplicación de matrices para medir el rendimiento de acuerdo a las dimensiones de las estructuras; posteriormente, se desarrolló una implementación del algoritmo de

convolución aplicado a la transformación de imágenes con filtros de matrices cuadradas de orden 3, 5 y 7.

La metodología de desarrollo empleada fue la propuesta por Aguilar y Leiss (2004), siendo específica para aplicaciones paralelas. Esta metodología, conocida como en espiral, se basa en dos premisas: primero, explotar el conocimiento que se va adquiriendo durante el desarrollo y diseñar inicialmente las partes claves de la aplicación, y segundo, poco a poco, añadirle las otras partes.

La metodología consta de cuatro fases:

a) **Descomposición.** El propósito fue identificar las oportunidades de paralelismo. Esta fase consistió en dividir o descomponer un programa en componentes que pueden ser ejecutados concurrentemente, asegurando un eficiente uso de los recursos por parte de la aplicación, reduciendo la latencia.

   La descomposición se aplicó por dominio, ya que se dividieron los datos en subgrupos y se asociaron los cálculos relacionados a cada grupo. La idea central es dividir en componentes que pueden ser manipulados independientemente. Los datos se dividen de igual tamaño a condición de que los procesadores tengan cargas de trabajo balanceadas. Dado que el algoritmo de convolución está compuesto por una sucesión de operaciones con matrices (Giménez et al., 2016), con salidas almacenadas en una matriz resultado como se observa en la figura 1, la descomposición por dominios se basó en el comportamiento del algoritmo al identificarse un total de (m-h+1) x (n-h+1) operaciones matriciales, donde: (m x n) es el tamaño de la matriz a transformar y h es el tamaño del filtro de convolución a aplicar. Luego de conocer el total de operaciones, se identificaron las cargas a aplicar por hilo de ejecución, teniendo en cuenta que a cada núcleo del procesador se le asignó un hilo.

b) **Comunicación**. Definir la estructura de comunicación en la aplicación es clave en el futuro desempeño de una aplicación. Es indispensable cuando los cálculos a realizar en una tarea, requieren datos de otras, lo que implica que se deben transferir esos datos entre las tareas.

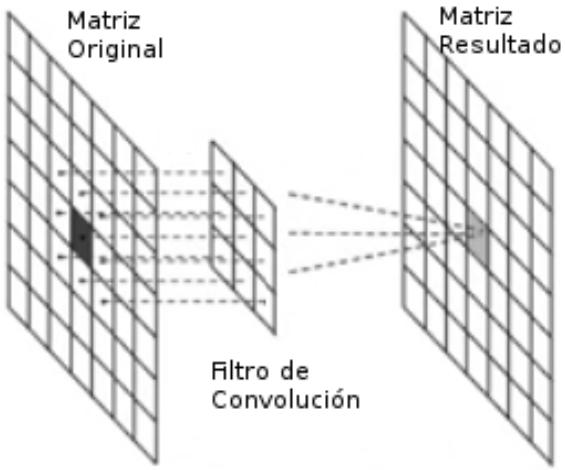

Figura 1 Descomposición por dominio (Autor).

En nuestro caso, se definió una comunicación asíncrona, ya que cada tarea resultante de la descomposición del algoritmo de convolución puede ejecutarse al obtener los datos que requiere, sin tener que esperar por otras tareas; para ello se identificaron y definieron variables por su alcance de tipo privadas o compartidas para las tareas resultantes de la descomposición.

c) **Agrupamiento**. En esta fase se revisaron las decisiones de descomposición y comunicación, para obtener un algoritmo que se ejecute eficientemente sobre un dispositivo con procesador multinúcleo embebido. Los objetivos fueron incrementar la cantidad de cálculos por núcleo, reducir los costos de comunicación, aumentar la flexibilidad con respecto a la escalabilidad y proceso de decisión de las asignaciones.

d) **Asignación**. Las tareas son asignadas al procesador de manera que maximice el uso de los núcleos y minimice los costos comunicacionales. El objetivo es minimizar el tiempo total de ejecución de la aplicación al asignar tareas concurrentes a diferentes núcleos del procesador y asignar tareas que se comunican frecuentemente en el mismo núcleo. Se definió una asignación dinámica, usando funciones de equilibro de carga de trabajo, de acuerdo a la arquitectura de hardware en la cual se ejecuta la aplicación, pudiendo tomar el máximo número de núcleos del procesador o dejar la tarea para que el usuario defina el total de núcleos durante la ejecución

(Chapman et al., 2008). El resultado de estas fases se puede observar en la figura 2, que presenta la rutina principal del algoritmo de convolución.

```
#pragma omp parallel num_threads(hilos) shared(original,resultado,i,alto,ancho,filtro) private(ini,fin,m,j,k,ind,suma)
{
    #pragma omp for
    for (i=0; i < hilos; i++){
        if (i == 0)
            ini = 1;
        else
            ini = i*alto/hilos;
        if (i == (hilos-1))
            fin = alto - 2;
        else
            fin = ((i+1)*alto/hilos)-1;

        for(m=ini; m<=fin; m++){
            for(j=1; j<ancho-2; j++){
                ind = m * ancho + j;
                suma = 255;
                for(k=0; k<(orden*orden); k++)
                    suma -= original[ind + ((k / orden) * ancho) + (k % orden) - 1]*filtro[k];
                if(suma<0) suma = 0;
                if(suma>255) suma = 255;
                resultado[ind] = suma;
            }
        }

    }
}
```

Figura 2 Código de la rutina de convolución con OpenMP (Autor).

3. Resultados

Aplicaciones desarrolladas

Para la construcción de las aplicaciones se consideraron los aspectos específicos de la arquitectura y software disponible para acoplar el diseño a las especificidades de la plataforma donde se ejecutará; esto permite aprovechar todo el conocimiento y experiencias obtenidas en los ciclos previos.

La interfaz de las aplicaciones se desarrolló en el lenguaje de programación Python, sin embargo la parte central de la aplicación fue desarrollada con C++ con el propósito de maximizar el rendimiento de la aplicación, tal como se puede observar en la figura 3. Adicionalmente, se empleó la interfaz de programación de aplicaciones paralelas de memoria compartida OpenMP para la división de tareas por hilos para cada núcleo del procesador; se empleó la librería de código abierto FreeImage para la manipulación de imágenes en el algoritmo de convolución y el repositorio de bibliotecas libre Boost para la comunicación y paso de parámetros entre las funciones definidas en Python y C++.

Con esta arquitectura se desarrollaron dos aplicaciones en los paradigmas secuencial y paralelo, ambas empleadas como tratamientos para el análisis del rendimiento de los dispositivos con procesador multinúcleo embebido.

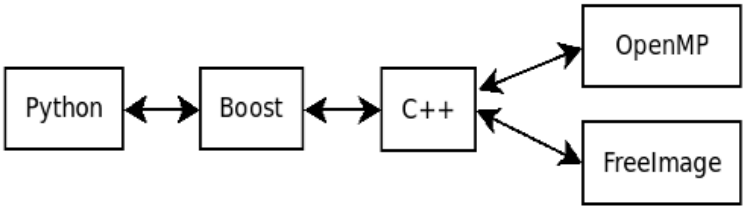

Figura 3 Arquitectura de la aplicación paralela (Autor).

La primera aplicación realiza la multiplicación de matrices (figura 4). Se ejecutó en sus versiones secuencial y paralela hasta con cuatro núcleos para matrices cuadradas de números enteros de dimensiones de 300, 600, 900, 1200 y 1500 elementos por dimensión, empleando memoria dinámica de tamaño 1.03 Mb, 4.12 Mb, 9.27 Mb, 16.48 Mb y 25.75 Mb respectivamente.

Figura 4 Aplicación para multiplicación de Matrices (Autor).

La segunda aplicación fue una implementación del algoritmo de convolución para tratamiento de imágenes (figura 5). Se ejecutó en sus versiones secuencial y paralela hasta con cuatro núcleos para fotos de 3 Megapixeles (Mpx), 5 Mpx y 8 Mpx, empleando filtros de convolución de 3x3, 5x5 y 7x7 pixeles, los cuales emplearon memoria dinámica de hasta 36 Mb, 57.67 Mb y 91.44 Mb respectivamente.

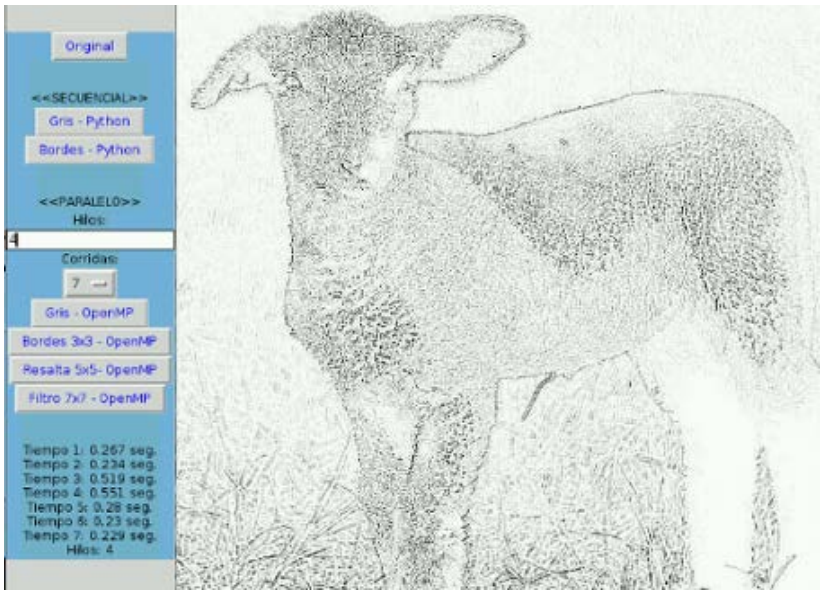

Figura 5 Aplicación para convolución de imágenes (Autor).

Análisis de rendimiento de aplicaciones paralelas

El rendimiento de acuerdo con Cardinale (2016), es la medida de qué tan bien un sistema, o los componentes que lo constituyen, lleva a cabo las tareas asignadas. En computación paralela es el tiempo de procesamiento medido con los siguientes indicadores: tiempo de ejecución, aceleración y eficiencia.

En el análisis de rendimiento de las aplicaciones paralelas desarrolladas para dispositivos con procesadores multinúcleo embebidos se tomó en cuenta como tiempo de ejecución la medida de tendencia central mediana de cinco corridas con los mismos parámetros, resultando los valores observados en tabla 1 para la multiplicación de matrices y los valores observados en tabla 2 para la aplicación que implementa el algoritmo de convolución.

Como resultado de la observación del indicador tiempo de ejecución, obtenido de la medición del tiempo transcurrido entre el comienzo y finalización de una tarea, se observó que mejoró el rendimiento en la aplicación multiplicación de matrices reduciendo en el mejor de los casos de 118.3 segundos a 31.8 segundos al emplear los cuatro núcleos del procesador, reduciendo el 73% del tiempo de ejecución, tal como se puede ver en la gráfica presentada en la figura 6.

Tabla 1 Tiempos de ejecución de la multiplicación de matrices (Autor).

Núcleos	Corrida	Tamaño de la matriz				
		300	600	900	1200	1500
1	1	0.244	4.276	20.923	58.347	119.158
	2	0.307	4.335	20.925	59.463	118.336
	3	0.301	4.311	20.703	59.326	118.291
	4	0.251	4.317	20.875	61.534	118.385
	5	0.251	4.344	20.857	60.209	118.347
	Mediana	0.251	4.317	20.875	59.463	118.347
2	1	0.179	2.255	10.586	30.921	59.194
	2	0.137	2.273	10.642	30.292	59.060
	3	0.098	2.288	10.585	29.088	59.082
	4	0.190	2.282	10.588	31.548	59.110
	5	0.152	2.239	10.591	31.344	59.062
	Mediana	0.152	2.273	10.588	30.921	59.082
3	1	0.109	1.597	7.399	20.837	40.602
	2	0.148	1.601	7.443	20.543	40.432
	3	0.150	1.602	7.482	20.451	40.268
	4	0.126	1.548	7.365	20.231	41.012
	5	0.137	1.570	7.335	20.239	41.808
	Mediana	0.137	1.597	7.399	20.451	40.602
4	1	0.105	1.232	5.855	15.799	32.363
	2	0.113	1.241	5.856	15.813	31.827
	3	0.062	1.221	5.852	15.532	31.792
	4	0.120	1.262	5.739	16.096	31.960
	5	0.117	1.251	5.696	16.073	31.823
	Mediana	0.113	1.241	5.852	15.813	31.827

Tabla 2 Tiempos de ejecución de la convolución (Autor).

Corrida	Filtro de Convolución											
	3 x 3				5 x 5				7 x 7			
	Núcleos				Núcleos				Núcleos			
	1	2	3	4	1	2	3	4	1	2	3	4
1	3.389	2.395	2.013	1.878	7.514	4.471	3.447	2.998	13.923	7.747	5.633	4.663
2	3.326	2.322	1.979	1.848	7.435	4.365	3.401	2.850	13.862	7.682	6.132	4.588
3	3.330	2.352	1.979	1.828	7.431	4.365	3.380	2.855	13.849	7.692	5.620	4.602
4	3.330	2.346	2.247	1.889	7.459	4.367	3.363	2.852	13.846	7.690	5.640	4.602
5	4.060	2.341	1.989	1.830	7.434	4.368	3.372	2.887	13.852	7.705	5.624	4.606
Mediana	3.330	2.346	1.989	1.848	7.435	4.367	3.380	2.855	13.852	7.692	5.633	4.602
1	5.449	3.787	3.297	3.028	11.805	6.895	5.387	4.648	21.966	12.275	8.989	7.414
2	5.370	3.725	3.248	2.983	11.695	6.941	5.677	4.578	21.853	12.159	8.914	7.235
3	5.392	3.707	3.196	3.227	11.650	6.860	5.348	4.560	21.992	12.138	8.863	7.266
4	5.364	3.708	3.209	2.935	11.596	6.869	5.359	4.560	22.025	12.218	8.909	7.223
5	5.366	3.718	3.205	2.960	11.592	6.869	5.322	4.513	21.906	12.192	8.930	7.223
Mediana	5.370	3.718	3.209	2.983	11.650	6.869	5.359	4.560	21.966	12.192	8.914	7.235
1	8.597	6.040	5.187	4.705	18.469	11.023	8.506	7.250	34.985	19.321	14.195	11.611
2	8.642	5.939	5.173	4.709	18.486	10.851	8.414	7.296	34.902	19.356	14.114	11.476
3	8.553	5.979	5.074	4.651	18.443	10.913	8.472	7.172	34.933	19.297	14.109	11.521
4	8.598	5.930	5.188	4.705	18.559	11.168	8.405	7.216	34.879	19.264	14.111	11.467
5	8.541	5.978	5.098	4.651	18.423	10.969	8.458	7.199	34.929	19.312	14.105	11.576
Mediana	8.597	5.978	5.173	4.705	18.469	10.969	8.458	7.216	34.929	19.312	14.111	11.521

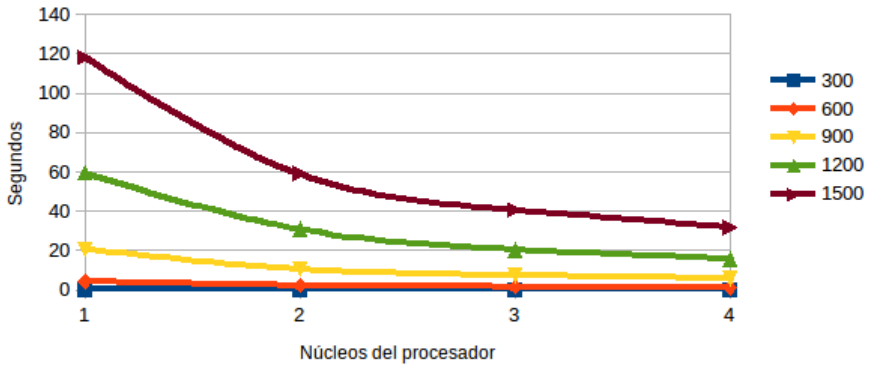

Figura 6 Indicador tiempo de ejecución multiplicación de matrices (Autor).

En el algoritmo de convolución de imágenes aplicado a la transformación de fotos de 8 Mpx con filtros de 7x7, se observa que el tiempo de ejecución se reduce de 34.9 segundos a 11.5 segundos, reduciendo el 67% del tiempo de ejecución, tal como se observa en la gráfica de la figura 7.

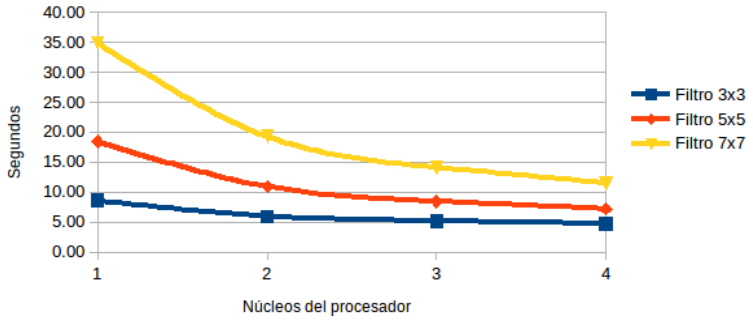

Figura 7 Indicador tiempo de ejecución del algoritmo de convolución (Autor).

La aceleración de un código paralelo se define como la razón del tiempo de ejecución en secuencial, entre el tiempo de ejecución en paralelo (Cruz, 2009). Este segundo indicador del rendimiento aplicado a la multiplicación de matrices en el mejor de los casos empleando dos núcleos es 2.0, empleando tres núcleos en 2.91 y empleando cuatro núcleos en 3.76, lo cual se aproxima a la aceleración ideal, presentado en la gráfica de la figura 8. Al calcular la aceleración al algoritmo de convolución para fotos de 8 Mpx ejecutado con cuatro núcleos, con el filtro de 3x3 la aceleración fue 1.83, para el filtro de 5x5 fue 2.56 y para el filtro de 7x7 fue 3.03, observable en la figura 9.

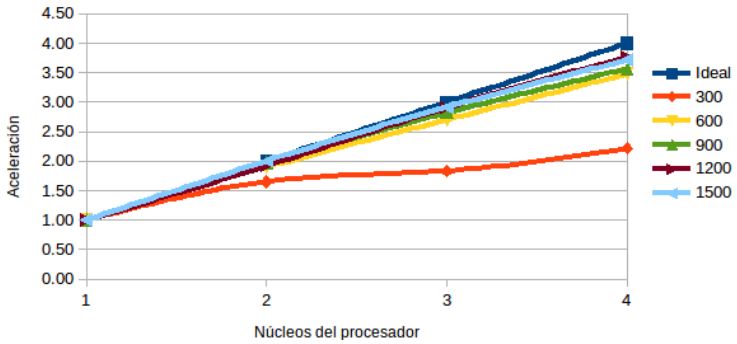

Figura 8 Indicador aceleración de la multiplicación de matrices (Autor).

El tercer indicador eficiencia, se define como la razón de la aceleración dividida por el número de núcleos empleados en la ejecución paralela (Kathavate &

Srinath, 2014). Esta medida calculada para la multiplicación de matrices, en el mejor de los casos para dos núcleos es 100%, para tres núcleos es 97% y para cuatro núcleos es 94%, observable en la figura 10. Mientras que la misma medida calculada para el algoritmo de convolución con filtro de 7x7 para fotos de 8Mpx, con dos núcleos fue 90%, con tres núcleos fue 83% y con cuatro núcleos fue 76%, observable en la figura 11.

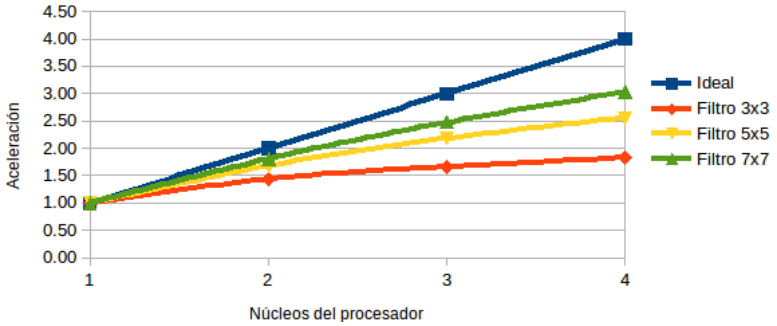

Figura 9 Indicador aceleración del algoritmo de convolución (Autor).

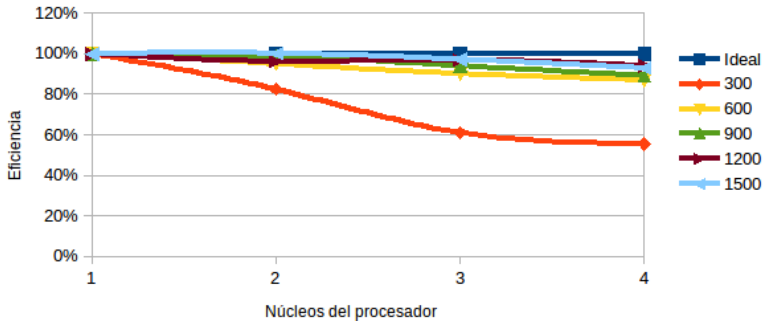

Figura 10 Indicador eficiencia de la multiplicación de matrices (Autor).

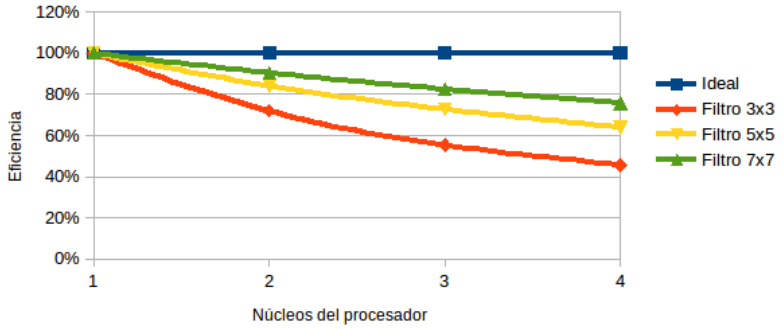

Figura 11 Indicador eficiencia del algoritmo de convolución (Autor).

Se observa cómo decrece el indicador eficiencia de forma paulatina al incrementarse más núcleos de procesamiento, lo cual demuestra el

comportamiento de la Ley de Amdahl que indica que la eficiencia obtenida en una implementación paralela viene limitada por la fracción del programa no paralelizable (Amdahl, 2013).

4. Discusión

Con base en la metodología en espiral para programación paralela, se desarrollaron librerías con C++ y OpenMP, y llamadas desde interfaces de usuario desarrolladas en Python. Estas implementaciones fueron compiladas y ejecutadas en dispositivos con procesadores multinúcleo embebidos, demostrando que el rendimiento en su ejecución en paralelo usando de dos hasta cuatro núcleos de procesamiento mejora significativamente respecto a la ejecución en secuencial (usando un sólo núcleo).

Las aplicaciones paralelas para dispositivos con procesadores multinúcleos embebidos desarrolladas con OpenMP, permiten mejorar el rendimiento frente a las aplicaciones secuenciales, en términos de tres indicadores: tiempo de ejecución, aceleración y eficiencia.

Las aplicaciones paralelas con interfaz Python, incrementan el rendimiento por medio de la construcción de librerías en C++ con OpenMP, las cuales se comunican y pasan parámetros por medio de la biblioteca libre Boost.

La construcción de aplicaciones para dispositivos con procesador embebido con múltiples núcleos, debe seguir una metodología bajo el paradigma de programación paralela para la implementación de la sección de código paralelizable, y con esto permitir un mejor aprovechamiento del hardware y óptimo rendimiento en la ejecución de las aplicaciones.

Se propone como trabajo futuro el análisis del rendimiento de aplicaciones paralelas en dispositivos con procesador embebido con múltiples núcleos, con otras tecnologías soportadas por la arquitectura de hardware como TBB (Threading Bulding Blocks), haciendo un análisis comparativo con respecto a OpenMP, así como la implementación de otros algoritmos paralelizables.

Estos resultados fueron obtenidos de la ejecución de aplicaciones paralelizadas con procesadores multinúcleo embebidos en dispositivos digitales, sin embargo,

también queda abierto el estudio para el análisis al aprovechar las capacidades de cómputo al implementar aplicaciones paralelas con las GPUs (Unidades de Procesamiento Gráfico) frente a las CPUs.

5. Bibliografía y Referencias

[1] Aguilar, J. & Leiss, E. (2004). Introducción a la Computación Paralela. Venezuela: Gráficas Quinteto.

[2] Amdahl, G. (2013, December). Computer Architecture and Amdahl`s Law. Computer. 46, pp. 38-46.

[3] Cardinale, Y. (2016). Evaluación del Rendimiento de Algoritmos Paralelos. 20/09/2016, de Universidad Simón Bolívar Dpto. de Computación y Tecnología de la Información. http://ldc.usb.ve/~yudith/docencia/ci-6842/.

[4] Chapman, B., Jost, G. & Pas, R. (2008). Using OpenMP Portable Shared Memory Parallel Programming. England: The MIT Press.

[5] Chichizola, F. (2013). Efecto de la distribución de trabajo en aplicaciones paralelas irregulares sobre clusters heterogéneos. Especialista en Cómputo de Altas Prestaciones y Tecnología GRID. Facultad de Informática - Universidad Nacional de La Plata.

[6] Cruz, L. (2009). Computación Científica en Paralelo. Agosto 20, 2016, de Universidad Nacional Autónoma de México, Unidad de Investigación en Cómputo Aplicado. http://www.dci.dgsca.unam.mx/lmcs

[7] Giménez, F., Monsoriu, J. & Alemany, E. (2016, febrero). Application to convolution of matrices to image filtering. Modelling in Science Education and Learning, 9, 97-108.

[8] Hernández, R., Fernández, C. y Baptista, P. (2010). Metodología de la Investigación. México: Mc Graw Hill.

[9] Kathavate, S. & Srinath, N. (2014, October). Efficiency of Parallel Algorithms on Multi Core Systems Using OpenMP. International Journal of Advanced Research in Computer and Communication Engineering, Vol. 3, pp. 8237-8241.