Revista Elektron
ISSN: 2525-0159
revista.elektron@fi.uba.ar
Universidad de Buenos Aires
Argentina

Representación en tiempo real de señales de radar empleando Odroid XU4
Revista Elektron, vol. 4, núm. 2, 2020, Julio-, pp. 87-92
Universidad de Buenos Aires
Argentina

elektron ámbar

Revista elektron, Vol. 4, No. 2, pp. 87-92 (2020)
https://doi.org/10.37537/rev.elektron.4.2.104.2020

Original Article

Representación en tiempo real de señales de radar empleando Odroid XU4

Real-time representation of radar signals using Odroid XU4

Lisvan Guevara Trujillo, Alian E. Matos Rodríguez, Leandro Zambrano Méndez

Facultad de Ingeniería Informática, Universidad Tecnológica de la Habana ¨José Antonio Echeverría¨
Calle 114 No 11901, Marianao, La Habana, Cuba

gladysyanil.trujillo@nauta.cu
aematos@nauta.cu
lzambrano@ceis.cujae.edu.cu

Recibido: 05/10/20; Aceptado: 27/11/20

***Abstract*— The representation system in modern radars is carried out by obtaining digital signals. In the case of tracking radars, the representation is updated in the order of tenths of a millisecond. In order to achieve to update the display in real time, the acquisition, processing and representation process must comply with the time requirement imposed by the radar. The objective of this work is the visualization of the information for a medium of this type in real time using the Odroid XU4 board. For their solution, the parallel programming method was used, through the creation of threads in the Qt Creator Integrated Development Environment and the use of the ¨cauce segmentation parallel programming pattern¨. The result allowed an efficient use of computing resources, obtaining a decrease of the execution time and a greater acceleration with respect to the sequential variant.**

Keywords: digital data processing; parallel programming; cauce segmentation; Odroid XU4.

***Resumen*— El sistema de representación en los radares modernos se realiza a partir de la obtención de señales digitales. En el caso de los radares de seguimiento la actualización de la representación se realiza en el orden de las décimas de milisegundos. Para lograr la actualización del display en tiempo real el proceso de adquisición, procesamiento de la señal y representación tiene que cumplir con la exigencia de tiempo de exploración del radar. El objetivo del trabajo es lograr la visualización de la información para un radar en tiempo real empleando la placa Odroid XU4. Se utilizó el método de programación paralela, mediante la creación de hilos en el Entorno de Desarrollo Integrado Qt Creator y el empleo del patrón de programación paralela segmentación de cauce. Esto permitió el uso eficiente de los recursos de cómputo, obteniéndose una disminución del tiempo de ejecución y una mayor aceleración con respecto a la variante secuencial.**

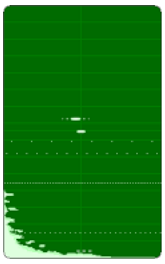
Palabras clave: procesamiento digital de datos; programación paralela; segmentación de cauce; Odroid XU4.

I. INTRODUCCIÓN

Los radares son los elementos fundamentales de los presentes y futuros sistemas para la seguridad y la defensa por su papel en la obtención de los datos básicos que permiten el funcionamiento de los sistemas de control de tráfico terrestre, aéreo y marítimo [1]. Entre los tipos de radares más comunes se encuentran los de exploración y los de seguimiento. Una de las diferencias entre estos es el período de actualización de la información, los radares de exploración lo realizan en el orden de los segundos y los de seguimiento en décimas de milisegundos. Conceptualmente, un sistema de radar consta de cinco componentes: generador, receptor, amplificador, analizador o procesador y visualizador [2]. En el presente trabajo se describe el desarrollo de un software para reemplazar los últimos dos componentes; lográndose actualizar y modernizar las prestaciones actuales de los radares de control de tráfico aéreo analógicos.

El visualizador de un radar tiene por objetivo presentar en una pantalla la información procesada por el analizador. Esta presentación puede realizarse de diferentes formas, según sean las necesidades del sistema. Por ejemplo, los radares de búsqueda o vigilancia que efectúan coberturas de 360º suelen utilizar presentaciones de tipo Indicador de Posición en el Plano (Plan Position Indicator, PPI), los radares empleados en el aterrizaje de precisión suelen utilizar presentaciones de tipo Indicador Ángulo Distancia (B-Scope) [3], las cuales ubican en el display los distintos objetivos según la información de distancia (línea vertical) y azimut (línea horizontal). Véase figura 1.

INDICADOR PPI



INDICADOR B-SCOPE

Fig. 1. Indicadores empleados para la representación de señales de radar.

Entre las ventajas que posee la recepción de la señal digital es la posibilidad del procesamiento automático de la información, además de aumentar la cantidad de información a representar en pantalla, facilitando la

comprensión y toma de decisiones a los operadores, evidenciándose mejoras respecto a la limitada representación de simples plots en las antiguas pantallas de fósforo. También posibilita agregar nuevas prestaciones sin necesidad de cambios electrónicos.

El empleo de las señales digitales de radar para mostrar la situación aérea de forma precisa y en tiempo real ha sido abordado por varios autores debido a las grandes exigencias de tiempo y procesamiento que conlleva el desarrollo del sistema, implementadas en diversas plataformas hardware. En [4] se describe un sistema generador de blancos de radar para evaluar un procesador de datos y visualizar diferentes escenarios, en [5] se describe la implementación de un visualizador de señales de radar en tiempo real con el objetivo de representar el seguimiento realizado a múltiples objetivos. En [5] se empleó la clase QPainter para la representación. En ambos trabajos se empleó el framework Qt, lo que permitió obtener un programa que puede ser corrido en diferentes plataformas software y hardware.

En [6] se desarrolló un componente del sistema de radar denominado Extractor Digital de Datos de Radar permitiendo actualizar y modernizar radares analógicos. Una de las funciones que permite este componente es la visualización de la situación aérea. Se empleó como arquitectura hardware ordenadores de placa reducida (Single Board Computer, SBC) y como sistema operativo la distribución de GNU/Linux Debian. Cuatro años después se le dio continuidad a este trabajo en [7], en el que se abordó el Generador de Pistas (tracker). En ambos trabajos se hizo uso de la programación paralela mediante la creación de hilos de ejecución en diferentes puntos del código fuente, lo cual permitió ejecutar varios procesos concurrentemente lográndose un mejor aprovechamiento de los recursos de cómputo. En el epígrafe 4.3 de [7] se abordó el rendimiento computacional, en el que se obtuvo de forma breve el desempeño del Generador de Pistas en el SBC. En el análisis bibliográfico se apreció que los autores para dar solución emplearon el lenguaje de programación C++ [4-7], al poseer mejor rendimiento respecto a otros lenguajes de alto nivel. Además del uso de la interfaz Ethernet para el envío y recepción de los datos [4-7], y el empleo del protocolo UDP (User Datagram Protocol, en lengua inglesa) [5-7]. Todos los trabajos anteriores carecen de un experimento detallado que demuestre el uso de los recursos de cómputo en el dispositivo seleccionado empleando medidas de evaluación del rendimiento del programa, como es el caso de la aceleración y la eficiencia, y la selección del dispositivo hardware en el que se implementará la solución.

En el entorno de los radares destinados al seguimiento con exigencias de actualización de la información en el orden de las décimas de milisegundos son escasas las publicaciones. La representación de las señales de un radar de seguimiento cumpliendo con el período de actualización de 40 milisegundos constituye un reto. En este trabajo los autores se plantearon dar solución a partir del uso de un sistema embebido. El objetivo de este trabajo es representar en tiempo real las señales de radar a partir del empleo del Odroid XU4. Además, a diferencia de los trabajos consultados, los autores pretenden dejar plasmado de forma clara los aspectos que se tuvieron en cuenta en el proceso de selección del dispositivo entre varios disponibles y los experimentos que permitieron evaluar el rendimiento del programa en el dispositivo seleccionado, quedando documentado los tiempos de ejecución del software en su versión secuencial y paralela.

II. DESARROLLO

La aplicación se enfocó en optimizar la carga computacional y minimizar los tiempos de representación, para acelerar el programa y aprovechar al máximo los recursos computacionales con un bajo consumo eléctrico. Además del empleo del Entorno de Desarrollo Integrado (Integrated Development Environment, IDE) Qt Creator se empleó la API (Application Programming Interface, en lengua inglesa) OpenMP, por ser esta una de las mejores opciones dentro de las tecnologías de programación en paralelo debido a que es soportado por la mayoría de los sistemas operativos, herramientas de compilación y otros dispositivos hardware, como los dispositivos móviles.

En la actualidad, con la aparición de los ordenadores de placa reducida, de bajo coste y alto rendimiento como la Raspberry Pi, Odroid XU4, etc; se logra contar con una alternativa viable como plataforma hardware para la implementación del algoritmo de representación de señales de radares de seguimiento. Se optó por emplear un ordenador de placa reducida frente a otras alternativas como las del uso de hardware reconfigurable, primero por el costo del dispositivo, ya que desarrollar una placa con FPGA que cuente con salida de video e interfaz Ethernet para la recepción de datos o la compra de un kit con FPGA los costos serían mayores respecto a los SBC disponibles. Otra cuestión es el proceso de síntesis del hardware en los FPGAs, para lo cual hay que emplear las herramientas propias del fabricante las cuales son privativas y costosas, al igual que sus licencias y al día de hoy no existen alternativas libres. En el caso del IDE multiplataforma Qt Creator soportado también por Linux embebido está bajo licencias libres (GNU GPL, GNU LGPL) las cuales garantizan la libertad de compartir y modificar el software, asegurando que este es libre para todos los usuarios. Además los SBC son fáciles de programar, ya que es posible utilizar las mismas herramientas y bibliotecas que se usan para las aplicaciones de computadoras, lo cual acorta significativamente la curva de aprendizaje necesaria para dominar una nueva arquitectura de hardware y lenguaje de programación. Entre las ventajas del empleo de estas placas respecto a un computador estándar se encuentran el bajo consumo energético, tamaño reducido y menor costo.

Para la selección del ordenador de placa reducida se le realizaron pruebas de rendimiento a los tres dispositivos con los que contamos: Jetson Nano, Raspberry Pi 4 Modelo B de 2 GB de RAM y el Odroid XU4, véase la figura 2.

1-JETSON NANO 2-RASPBERRY PI4 MODELO B 3-ODROID XU4

Fig. 2. Ordenadores de placa reducida

Los sistemas operativos de estos tres dispositivos son distribuciones de Linux las pruebas se realizaron empleando el Sysbench. Esta es una herramienta de pruebas de

rendimiento modular y con ejecución multi-hilo, diseñada para extraer una visión rápida de la velocidad de un equipo. A estos dispositivos se le realizaron tres pruebas una a la memoria, otra de lectura/escritura al dispositivo de almacenamiento y la más importante la de la unidad central de procesamiento por la alta demanda que este tipo de aplicaciones suele realizarle. La prueba del procesador consistió en determinar 20000 números primos, Sysbench verificará los números primos haciendo la división estándar del número por todos los números entre 2 y la raíz cuadrada del número. Por lo que esto pondrá algo de estrés en el procesador. Para lanzar un test sobre un núcleo del procesador, se escribió el siguiente comando en el terminal:

sysbench --test=cpu --cpu-max-prime=20000 run

Para lanzar el test sobre los 4 núcleos se escribió lo siguiente:

sysbench --test=cpu --cpu-max-prime=20000 run --num-threads=4

Los resultados de las pruebas se muestran en la figura 3, en la que se representan el tiempo total en segundos al determinar 20000 números primos, el rendimiento del dispositivo es mejor mientras menor sea el tiempo.

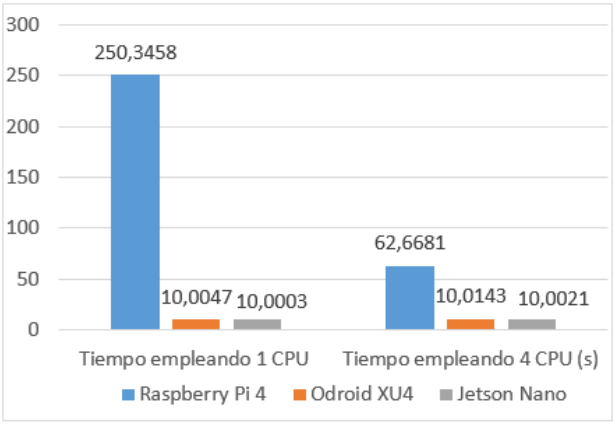


Fig. 3. Resultados de las pruebas de rendimiento a las unidades centrales de procesamiento.

Del análisis de la figura se concluye que el Jetson Nano y el Odroid XU4 obtuvieron los mejores resultados. Al ser los tiempos de estos similares se seleccionó este último atendiendo a que el precio de mercado es menor.

A. Odroid XU4

El Odroid XU4 es un potente y económico ordenador de placa reducida, la cual posee una arquitectura multi-núcleo asimétrica basada en ARM (Advanced RISC Machine, en lengua inglesa). Está provista de un SoC (System on chip, en lengua inglesa) Samsung Exynos 5422 big.LITTLE de ocho núcleos [8], formado por dos clúster de 4 núcleos cada uno, el Big, formado por 4 núcleos Cortex-A15 a 2GHz y el Little por 4 núcleos Cortex-A7 a 1.4 GHz [9, 10]. Además posee 2GB de RAM, 2 puertos USB 3.0, interfaz Ethernet a 1 Gbps, un consumo eléctrico entre 10 y 20 W [11].

Un procesador multi-núcleo asimétrico integra núcleos con distintas características en un mismo chip. Por un lado integran un grupo de núcleos rápidos, de alto rendimiento y consumo, que trabajan a alta frecuencia e implementan complejas técnicas como la ejecución fuera de orden o el lanzamiento múltiple de instrucciones. Estos procesadores también incluyen un grupo de núcleos más lentos y sencillos, los cuales operan a una frecuencia de trabajo más baja e implementan un pipeline más sencillo y de bajo consumo. Estos aportan una mayor flexibilidad que los procesadores convencionales, ya que tienen la capacidad de mantener un alto rendimiento para aplicaciones intensivas en CPU, sin sacrificar la eficiencia energética [12].

B. Implementación del software

La implementación del sistema se realizó en el sistema operativo Ubuntu por varios aspectos: es una distribución de GNU/Linux muy conocida y utilizada a nivel mundial por su simplicidad y fácil uso; disponible para teléfonos móviles, tabletas, SBC, televisores inteligentes y computadoras. Es fácilmente configurable y puesta a punto para la ejecución de la aplicación. Al contarse con experiencia de trabajo sobre la plataforma, se ahorra tiempo de configuración y se aborda rápidamente el problema a resolver. Es uno de los sistemas operativos recomendados por el fabricante de la SBC (Hardkernel) a emplear en el Odroid XU4, disponible en https://odroid.com/dokuwiki/doku.php?id=en:odroid-xu4#software_os_release. Además carece de restricciones comerciales, por lo que se puede adquirir de forma gratuita. Posee licencia GPL lo que posibilita acceder al código fuente, analizarlo y aportar ideas a la comunidad de desarrolladores. También es un sistema operativo multitarea y multihilo, el cual es muy eficiente en términos de uso de recursos del sistema. Es un sistema operativo estable, característica necesaria para aplicaciones de este tipo, donde el mantenimiento del sistema es esporádico y el tiempo de ejecución continua predomina. Es menos propenso a malware y virus.

La aplicación fue desarrollada en el IDE Qt Creator, en lenguaje C++ y en el sistema operativo Ubuntu 18.04. Este algoritmo puede ser visto de manera más general como un proceso de tres etapas, primeramente, los valores son recibidos para luego ser procesados y por último representados en el indicador. Este proceso se muestra en la figura 4.

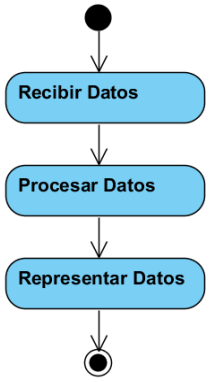


Fig. 4. Diagrama de actividades que describe el algoritmo para la representación de señales de radar.

La recepción de datos se realiza por interfaz Ethernet empleando el protocolo de comunicación UDP donde los valores recibidos corresponden con la amplitud de la señal del radar, de 8 bits. Se reciben 6660 valores cada 280 microsegundos, debido a que este es el período de repetición del impulso de sondeo del transmisor; estos valores serán recibidos durante 36 milisegundos, siendo este el tiempo de duración de la exploración del radar.

Teniendo en cuenta que en cada recepción se obtienen 6660 muestras, es necesario realizar un procesamiento a las mismas para determinar el valor que va a tomar cada píxel, ya que son más valores que píxeles disponibles para la representación. Teniendo en cuenta que 10 valores representan un píxel se hace necesario realizar la integración por píxel de las muestras. Para realizar la integración empleamos el método de selección del valor máximo de promedios, ya que si solamente se promedian los valores tiende a disminuir la amplitud de la señal correspondiente al medio aéreo y si solo se selecciona el mayor valor entonces resaltan los picos de ruido. El método de selección del valor máximo de los promedios consiste en dividir las 10 muestras que representan un píxel, en dos subgrupos de 5 muestras y estos son promediados de manera independiente y luego se escoge el mayor de esos dos valores, el cual sería el valor de intensidad del píxel.

Luego se procede a la representación de la señal empleando para ello la clase QPainter del IDE Qt Creator. Esta es visualizada en dos indicadores del tipo B-Scope, donde en el eje vertical de cada indicador tiene una longitud de 625 píxeles representando la escala de distancia. En el eje horizontal con una longitud de 256 píxeles se representa el sector de exploración del radar. Se emplearon dos indicadores debido a que estos pueden estar representando escalas de distancias diferentes en el mismo ángulo de exploración de la antena, véase la figura 5.

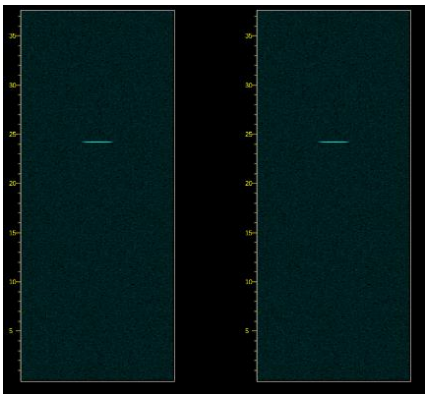


Fig. 5. Representación en indicadores B-Scope.

Al utilizar esta variante secuencial, para iniciar cada una de las etapas en las que se divide el algoritmo es necesario esperar a que termine la etapa anterior. Una vez implementada esta primera variante de solución, se verificó su funcionamiento, evidenciándose el incumplimiento del valor de tiempo exigido de 40 milisegundos para la actualización de la información, por lo que se tendrían que visualizar 25 pantallas por segundo.

Fue necesario un proceso de optimización, con el análisis del rendimiento del programa secuencial y la localización de los puntos de mayor consumo de tiempo. Después de una etapa experimental se detectó que las funciones más costosas temporalmente son en primer lugar la encargada de la recepción de datos representando el 53% del tiempo total y las encargadas de la representación, que entre las dos representan el 20% del tiempo total. Teniendo en cuenta que la placa empleada es multi-núcleo una forma de acelerar la ejecución del programa es mediante la paralelización del sistema, de forma que se haga una repartición óptima de tareas entre los núcleos de la misma.

C. Paralelización del algoritmo secuencial

La computación paralela es el uso simultáneo de múltiples recursos de cómputo para resolver un problema computacional [13]. Una manera de diseñar e implementar de manera eficiente los algoritmos paralelos es a través del uso de patrones de computación paralela y distribuida [14]. Los patrones de computación paralela son una combinación de distribución de tareas y accesos a datos que resuelven un problema específico en el diseño de algoritmos paralelos. Con frecuencia estos patrones son una generalización de patrones de computación secuencial. Estos pueden ser aplicados en cualquier sistema de programación paralela, pues no están restringidos a ninguna arquitectura de hardware, lenguaje de programación o sistema [14]. Ejemplo de patrón de computación paralela y distribuida es el patrón segmentación del cauce. Este patrón consiste en crear una secuencia lineal de etapas. Los datos son adquiridos de forma segmentada. Estos pasan por todas las etapas desde la primera hasta la última. Cada etapa realiza una transformación de los datos. Más de una etapa puede estar activa al mismo tiempo, por lo que puede ocurrir paralelismo de tareas [14]. Es importante resaltar que este patrón es utilizado en algoritmos donde los datos no se reciben de una sola vez, sino que se reciben por parte o de manera segmentada.

La figura 6 muestra un esquema del patrón utilizando estados en serie y estados en paralelo. En ambas figuras los elementos verdes representan los datos de entrada y salida del algoritmo. Los elementos azules representan las diferentes etapas en las que puede ser dividido un algoritmo que utilice este patrón.

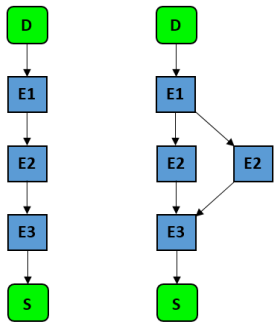


Fig. 6. Segmentación de Cauce con estados en serie y en paralelo.

El IDE multiplataforma Qt Creator posee un gran número de clases, entre ellas está la clase QThread. Esta clase proporciona una manera independiente de la plataforma para administrar hilos[15]. Es decir, un objeto de la clase QThread se ejecuta de manera independiente de los demás hilos.

Al contar el algoritmo secuencial con las siguientes 3 etapas: proceso de recepción de datos, procesamiento de datos y representación en los dos indicadores; y conociendo que la etapa de recepción de datos es la más costosa desde el punto de vista de tiempo, se le asignó un hilo de ejecución empleando la clase QThread. Se empleó el patrón segmentación de cauce para lograr la ejecución paralela utilizando OpenMP (Open MultiProcessing).

La ejecución del algoritmo consiste en un conjunto de iteraciones donde se reciben, se procesan y se representan los datos en paralelo. En la primera iteración sólo se reciben los datos, ya que no se pueden procesar porque no existía valor alguno previamente. En la siguiente se realiza en paralelo el procesamiento de la anterior y la recepción que serán procesados en la próxima iteración. Luego le siguen un conjunto de iteraciones donde se realizan en paralelo el proceso de recepción, procesamiento y representación de los mismos. En la penúltima iteración solamente se realiza el procesamiento y representación, y en la última solo se efectúa la representación. El paralelismo que se alcanza cuando se pone en práctica este patrón se muestra en la figura 7, en la que se aprecia como varias etapas pueden estar activas al mismo tiempo por lo que se logra disminuir el tiempo de ejecución respecto a la variante secuencial.

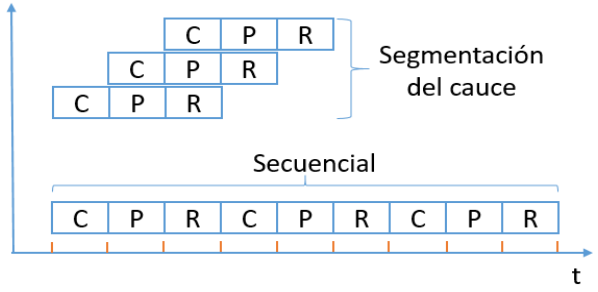


Fig. 7. Comportamiento en el tiempo del patrón segmentación de cauce.

OpenMP es una interfaz de programación de aplicaciones (API) para la programación paralela en el que se considera al sistema como una colección de núcleos o CPUs, teniendo todos acceso a la memoria principal[16]. El paralelismo se especifica a través de directivas que se insertan en el código [16]. En C y C++ estas directivas se añaden empleando instrucciones especiales de preprocesado conocidas como pragmas. Un compilador que soporte OpenMP reconocerá las directivas pragmas y generará el código ejecutable paralelizado, si el compilador no soporta OpenMP, específicamente los pragmas, simplemente los ignorarán como si fuesen comentarios y generarán el código ejecutable habitual[16]. Las directivas son especificadas usando la siguiente sintaxis: #pragma omp <directiva> [cláusulas], donde las cláusulas permiten modificar el comportamiento por defecto de una directiva. Esta API está basada en el modelo Fork-Join, en el que un hilo maestro o principal crea tantos hilos como necesite [17], como se muestra en la figura 8. El hilo 1 representa al hilo maestro y los hilos del 2 al 4 los hilos esclavos.

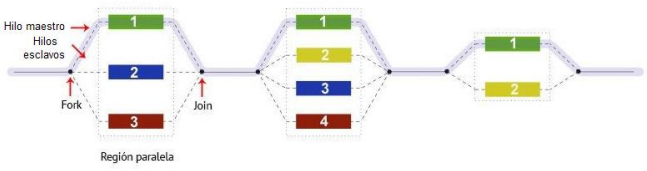


Fig. 8. Modelo Fork-Join.

Para activar OpenMP en Qt Creator se agregaron las siguientes líneas en el archivo .pro:

```
QMAKE_CXXFLAGS += -fopenmp
QMAKE_LFLAGS += -fopenmp
LIBS += -fopenmp
```

Para lograr el paralelismo deseado primero se incluyó la biblioteca omp.h y se añadieron algunas directivas OpenMP al algoritmo secuencial para garantizar la ejecución concurrente empleando el patrón de programación paralela segmentación de cauce. Estas directivas son "#pragma omp parallel sections" que permite ejecutar en paralelo las secciones definidas dentro del bloque de instrucciones definido por esta directiva. Para delimitar las secciones que se deben ejecutar en paralelo dentro de ese bloque, es utilizada la directiva "#pragma omp section" [18], lográndose ejecutar cada sección una vez por un hilo. Esencialmente, un programa con OpenMP es un programa secuencial al que se le añaden directivas en los puntos adecuados.

D. Resultados experimentales

Para la realización del experimento se hicieron 10 ejecuciones de la versión paralela y secuencial bajo el principio de repetición para minimizar el error experimental, con el objetivo de comprobar si se logró disminuir los tiempos del mismo. Se creó un escenario de experimentación bajo las mismas condiciones en cuanto a sistema operativo Ubuntu 18.04, tarjeta de almacenamiento eMMC de 16 GB de capacidad, placa Odroid XU4, así como la señal patrón a procesar.

Teniendo en cuenta que el software no posee más de 4 hilos de ejecución, solamente se emplearon los 4 núcleos Cortex-A15 del Odroid XU4, para lograr mayor eficiencia y menor consumo de energía que al utilizar los 8 núcleos del Odroid.

Los tiempos resultantes en milisegundos (ms) de las 10 ejecuciones se muestran en la figura 9.

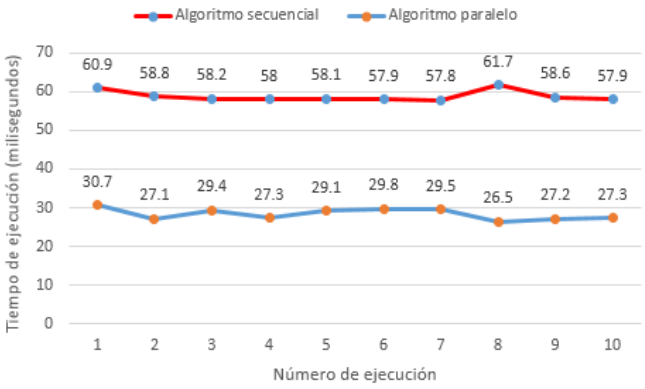


Fig. 9. Tiempo de ejecución del algoritmo secuencial y paralelo.

Como resultado de las 10 ejecuciones se obtuvo que el valor promedio para el algoritmo secuencial y paralelo fueron de 58.79 ms y 28.39 ms respectivamente. Al cumplirse el requisito de tiempo en la variante paralela de menos de 40 ms, no fue necesario emplear un Sistema Operativo de Tiempo Real.

Dentro de las medidas que permiten evaluar el rendimiento de un programa paralelo están la aceleración y la eficiencia. La aceleración es la relación que existe entre los tiempos de ejecución cuando se utiliza un procesador (serial) y cuando se utiliza P procesadores (paralelo) [19].

La aceleración representa cuantas veces es más rápido un programa paralelo con respecto a uno secuencial, resolviendo ambos el mismo problema. Matemáticamente se define como se muestra en la ecuación (1):

$$S_{(P)} = T_{(1)} / T_{(P)} = 58.79 / 28.39 = 2.0707 \quad (1)$$

Donde:

- $S_{(P)}$: aceleración usando P procesadores.
- $T_{(1)}$: tiempo requerido por el sistema con un procesador para resolver el problema en cuestión.
- $T_{(P)}$: tiempo requerido por el sistema con P procesadores para resolver el problema en cuestión.

La eficiencia es definida como la fracción del tiempo que el procesador consume haciendo trabajo útil. Esta muestra cuán bien se ha utilizado los procesadores en la solución de un problema. Se define matemáticamente como se muestra en la ecuación (2)[20]:

$$E_{(P)} = S_{(P)} / P = 2.0707 / 4 = 0.5176 \quad (2)$$

Donde:

- $E_{(P)}$: eficiencia.
- $S_{(P)}$: aceleración asociada a la ejecución paralela del algoritmo.
- P*:* número de hilos de ejecución del procesador.

La eficiencia obtenida durante los experimentos es de 0.5176 lo que implica que se aprovecha en un 51.76 % los recursos del procesador.

III. Conclusiones

Los experimentos realizados demostraron que es posible construir un sistema que garantice la representación de las señales de un radar de seguimiento en tiempo real empleando el Odroid XU4, aprovechando de forma eficiente los recursos de cómputos a través de la computación paralela. Al emplear el patrón de programación paralela segmentación de cauce conjuntamente con las facilidades que brinda Qt Creator para el trabajo con hilos se logró la reducción del tiempo de ejecución en un 51.70% y una aceleración de 2.07 veces con respecto a la variante secuencial. Además se obtuvo un código óptimo en tiempo de ejecución, implementado en un dispositivo de bajo consumo eléctrico y de bajo coste. Teniendo en cuenta que en la literatura consultada no existen antecedentes de la evaluación del rendimiento del programa, en este escrito queda plasmado la metodología empleada. También, a la hora de seleccionar uno de los dispositivos comparados, este documento puede constituir una guía para otros desarrolladores, evidenciándose la factibilidad del empleo del Odroid XU4 y el IDE Qt Creator en aplicaciones de radar.

Agradecimientos

Los autores agradecen los señalamientos del Dr. C Marcelino Sánchez Posada y el Ms. C Ariel Hernández Reyes en la parte del procesamiento de datos y el trabajo con los radares. Al Dr. C. Bárbaro Nicolás Socarras Hernández por la revisión de la metodología desarrollada.

Referencias

[1] J. A. G. Fominaya, "Nuevas técnicas de localización, clasificación e identificación para radares de vigilancia superficial y alta resolución en escenarios LPI," Tesis Doctoral, Departamento de sistemas, señales y radiocomunicaciones, Universidad Politécnica de Madrid, España, 2004.

[2] S. A. Hovanessian, *Radar Detection and Tracking Systems*: Artech House, 1973.

[3] P. Kaushik, "Radar Displays," International Journal of Innovative Research in Technology, vol. 1, p. 5, 2014.

[4] V. Kavyashree, P. N. Madhu Chaitra, A. Y. Prasad, and H. Vinutha, "A Radar Target Generator for Airborne Targets," *International Journal of Science Technology & Engineering*, vol. 3, 2017.

[5] C. Ravindra, S. Rajkumar, and M. Sreenivasa Babu, "Design and Implementation of Radar Console Displays for Multi Object Tracking Radar using Qt-IDE," *11th International Radar Symposium India*, 2017.

[6] I. A. Montamat, "Combinador de Información Primaria y Secundaria para Extractor Digital de Datos de Radar en Sistemas de Vigilancia," Tesis de Maestría, Universidad Nacional de Córdoba, Argentina, 2015.

[7] S. A. Rodríguez González, "Sistema de Seguimiento y Generación de Pistas para Radar Track While Scan," Tesis de Maestría, Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Argentina, 2019.

[8] E. Österberg, "Profiling memory accesses on the ODROID-XU4," Department of Information Technology, Uppsala Universitet, Suecia, 2017.

[9] D. Paul. (2019) ODROID-XU4 Tweaks: A Collection of Popular Modifications. *Odroid Magazine*. Available: https://magazine.odroid.com/wp-content/uploads/ODROID-Magazine-201911.pdf

[10] J. Rondx. (2019) Cryptocurrency Mining: Earning Verium Coins With Your ODROID. *Odroid Magazine*. Available: https://magazine.odroid.com/wp-content/uploads/ODROID-Magazine-201912.pdf

[11] V. B. R. Roy. (2017) User Manual Odroid XU4. *Odroid Magazine*. Available: https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf

[12] L. M. C. Valero, "Evaluación y optimización de rendimiento y consumo energético de aplicaciones paralelas a nivel de tareas sobre arquitecturas asimétricas," Tesis de Maestría, Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, 2016.

[13] L. L. N. L. Blaise Barney. (2020). *Introduction to Parallel Computing*. Available: https://computing.llnl.gov/tutorials/parallel_comp/

[14] A. D. R. Michael McCool, James Reinders, *Structured Parallel Programming Patterns for Efficient Computation*. USA: Morgan Kaufmann Publishers, 2012.

[15] The Qt Company Ltd. (2020). *QThread*. Available: https://doc.qt.io/qt-5/qthread.html#details

[16] L. D. R. Chandra, D. Kohr, D. Maydan, J. McDonald, R. Menon, *Parallel Programming in OpenMP*. USA: Morgan Kaufmann, 2000.

[17] L. L. N. L. Blaise Barney. (2020). *OpenMP*. Available: https://computing.llnl.gov/tutorials/openMP/

[18] A. R. B. R. Álvarez Pérez, "Desarrollo de versiones paralelas del algoritmo para el descubrimiento del modelo organizacional utilizando los patrones segmentación de cauce y reducción.," Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana José Antonio Echeverría, 2017.

[19] V. A. G. Samaniego, "Desarrollo de un algoritmo para romper por fuerza bruta al Simplified Data Encryption Standard (S-DES) mediante el uso de computación paralela.," Tesis de Ingeniería, Facultad de Ingeniería Eléctrica y Electrónica, Escuela Politécnica Nacional de Ecuador, 2019.

[20] J. L. Aguilar Castro and E. Leiss, *Introducción a la Computación Paralela*. Mérida, Venezuela: Universidad de los Andes, 2004.