Universidad
Nacional
de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº II

NAVARRO, Matias Alejandro

17 de mayo del 2019

Índice

Introducción

Los niveles de integración electrónica han permitido la generación de procesadores de arquitecturas multiprocesos, multicore y ahora many integrated core (MIC). Este avance hace necesario que los programadores cuenten con un profundo conocimiento del hardware sobre el que se ejecutan sus programas, y que dichos programas ya no pueden ser monoproceso.

Entre las técnicas y estándares más utilizados para sistemas de memoria compartida y memoria distribuida, se encuentra OpenMP y MPI respectiva mente.

Propósito

El objetivo del presente trabajo práctico es que el estudiante sea capaz diseñar una solución de manera procedural de manera local y luego modificar la misma para que utilice el paradigma de memoria distribuida, utilizando OpenMP.

Ámbito del Sistema

Para realizar el presente trabajo práctico, es necesario instalar la librerías NetCDF4 en el sistema sobre el cual se diseñe, desarrolle y pruebe la solución al problema. Estas librerías permiten compartir información tecnológica y científica en un formato auto definido, independiente de la arquitectura del sistema y también definen un formato de datos que se transformó en estándar abierto. La librería tiene versiones en Fortran, Python, Java y C, siendo estas últimas las que vamos a utilizar en este trabajo.

Referencias

1. Filminas provistas por la cátedra de Sistemas Operativos II (UNC - Facultad de Ciencias Exactas Físicas y Naturales).
2. https://www.unidata.ucar.edu/software/netcdf/examples/programs/
3. http://supercomputingblog.com/openmp/tutorial-parallel-for-loops-with-openmp/
4. https://stackoverflow.com/questions/10811439/malloc-an-array-of-struct-pointers
5. https://www.ncdc.noaa.gov/data-access/satellite-data/goes-r-series-satellites

Descripción General

Perspectiva del Producto

El objetivo es diseñar, implementar y testear un software que permita aplicar un filtro detector de borde a una imagen tomada por el satélite GOES16. El filtro es implementado mediante una convolución de la imagen tomada por dicho satélite y una matriz 3x3 determinada.

Funciones del Producto

El producto ofrece las siguientes funciones:

- Carga de la matriz inicial: lectura de un archivo con información sobre una imagen de la Tierra.
- Setear valores NAN: pone en NAN (i,e (float)(0.0/0.0)) los valores -1 que encuentra en la matriz que se carga inicialmente.
- Convolución: función que realiza la convolución y de esta forma aplicar el filtro detector de bordes a la imagen cargada en la matriz inicial.
- Escritura de los datos: se crea un archivo .nc donde se proceden a guardar los datos del resultado de la convolución.

Características de los Usuarios

El usuario debe contar con conocimientos básicos de programación y contar con el archivo .nc donde se encuentra la imagen del planeta Tierra.

Restricciones

- El sistema donde se ejecute el programa debe contar con al menos 4Gb de memoria RAM disponibles.

Suposiciones y Dependencias

- El sistema donde correrá el programa será una distribución de Linux.
- El sistema debe tener instaladas las librerías NetCDF4 y OpenMP.

Requisitos Específicos

Interfaces Externas

Para ejecutar el programa se utiliza el terminal del sistema operativo, tanto en la ejecución del procedural como el OpenMp que corre en el cluster.

Diseño de solución

Para el diseño de la solución al problema planteado se lo abordó dividiéndolo en partes más pequeñas, primero se comenzó por realizar el procedural y luego la solución implementando múltiples hilos.
Se da inicio al programa leyendo el archivo a cargar. La lectura se realiza mediante la función *get_vara_float()*, indicando un inicio y fin.

La información obtenida es almacenada en un arreglo de punteros para su posterior manipulación. A dichos datos se los filtra buscando los valores -1 que deben ser tratados como “NAN”.

La imagen filtrada se obtiene a partir de la convolución:

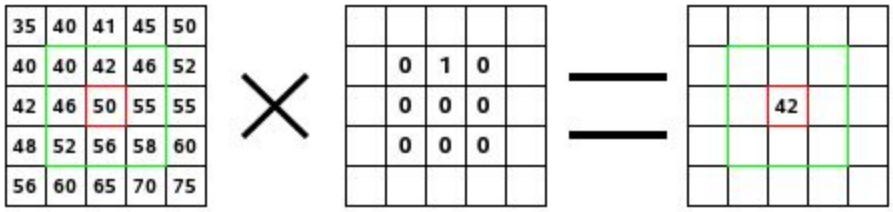
$$g(y,z) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

Donde *g(x,y)* es la imagen filtrada, *f(x,y)* es la imagen original y ω es la matriz con la que se aplica el filtro de bordes:

$$\omega = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Se realiza la convolución a partir de los datos almacenados en el arreglo inicial y el filtro mencionado. La convolución consiste en posicionarse en el pixel deseado filtrar, se recorren los pixeles que lo rodean y se multiplica por el filtro, los resultados obtenidos son sumados y se almacenan en la posición del píxel seleccionado pero de un arreglo nuevo, es decir, la información original (imagen) no es modificada, y de esta forma se realiza punto a punto en el arreglo original.

Un ejemplo simple para de como esta convolución es implementada:



Una vez realizada toda la convolución, el resultado se guarda en un arreglo nuevo y luego se escribe dicha información en un archivo tipo .nc, el cual luego es graficado a partir de un script de Python.

Para la implementación de forma paralela utilizando OpenMP, se implementaron secciones paralelas en todos los bucles que tenían un número elevado de iteraciones, logrando así poder dividir el trabajo entre varios hilos y logrando una mayor velocidad de ejecución.

Implementación y Resultados

La implementación se llevó a cabo en dos partes, la primera se centró en la ejecución del programa sin llevar a cabo ningún tipo de paralelismo, es decir, con solo un hilo (procedural). La imagen a ser filtrada es la siguiente:

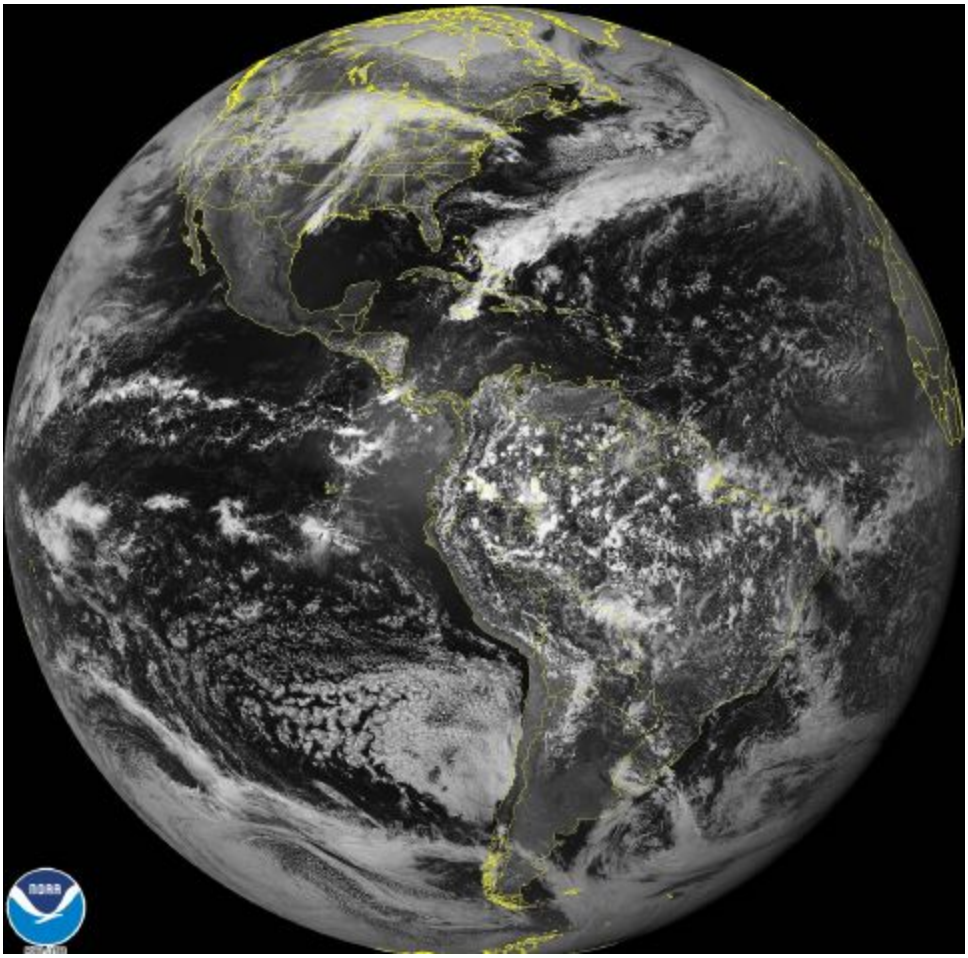


Imagen 1 - Imagen original obtenida del satélite GOES16.

Mientras que el resultado obtenido luego de realizar la convolución se puede ver a continuación:

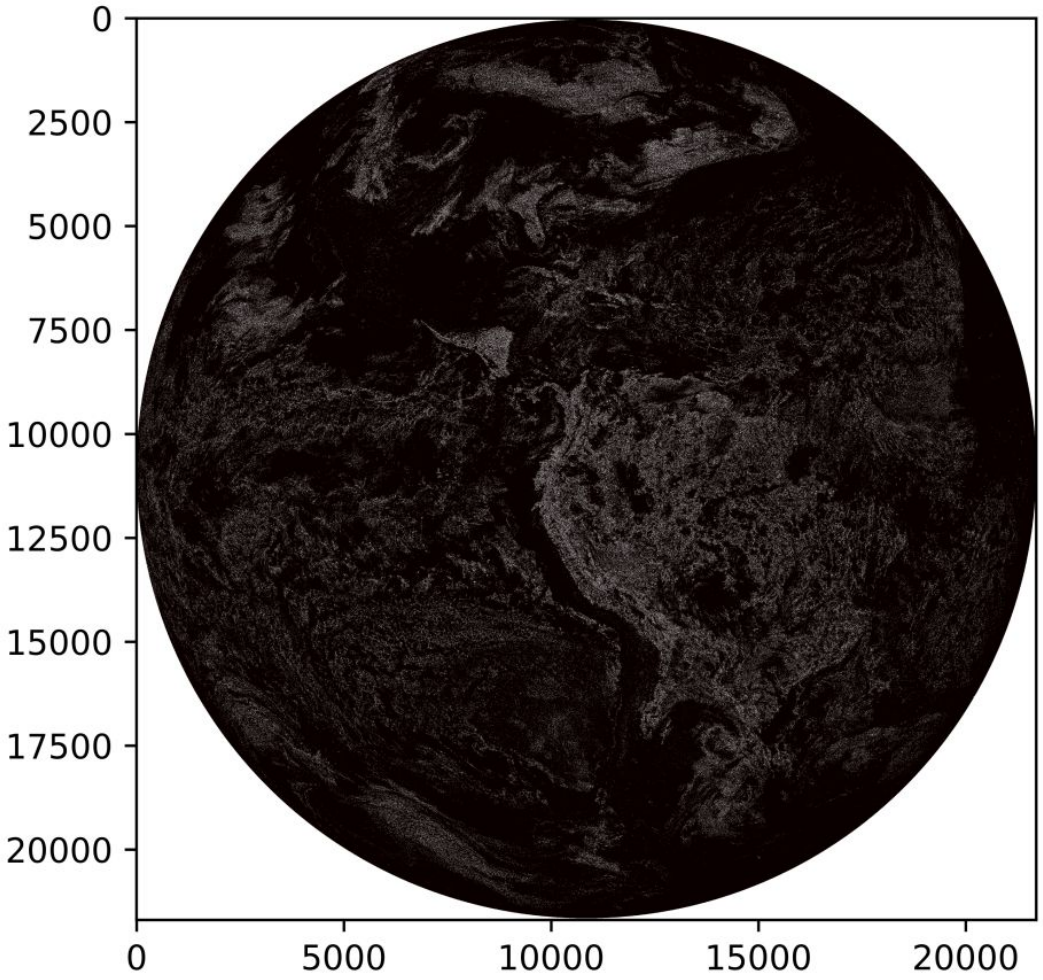
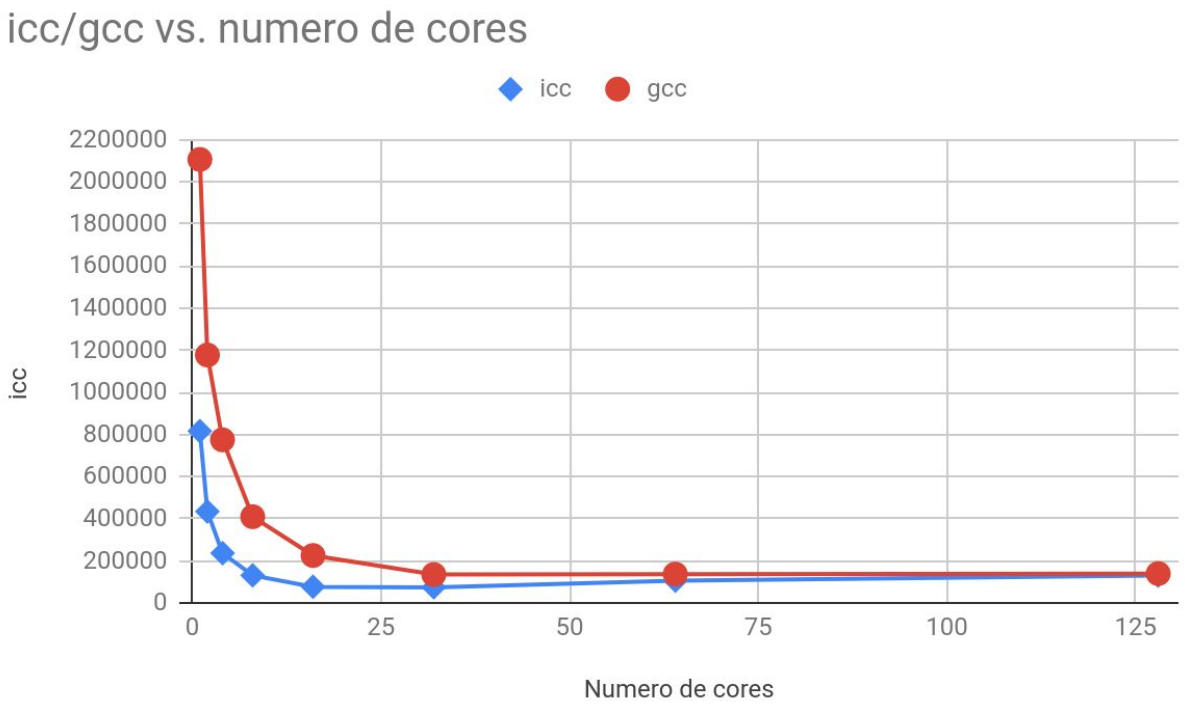


Imagen 2 - Imagen con el filtro de bordes aplicado.

El programa se ejecutó 30 veces para poder tener estadísticas del rendimiento con diferente números de hilos para poder observar cómo afecta el paralelismo. Además, el programa también fue compilado con diferentes herramientas, como lo son *gcc* e *icc*.



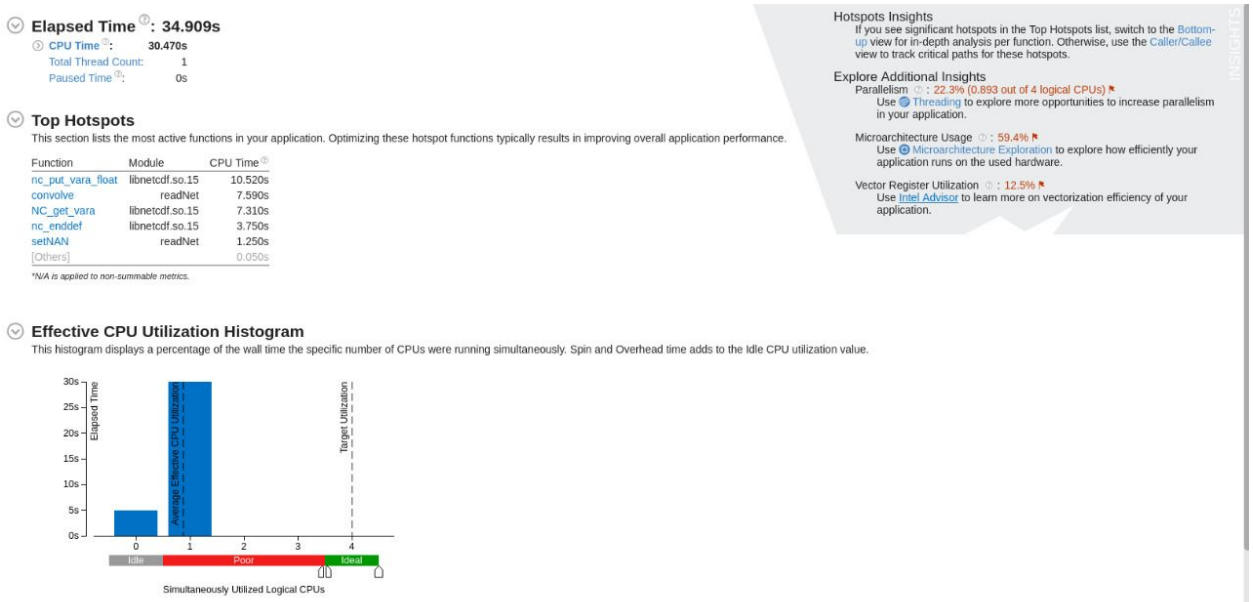
Se adjunta la siguiente tabla en comparativa de las herramientas de compilación utilizadas que contiene el promedio de las 30 ejecuciones para distintos números de hilos:

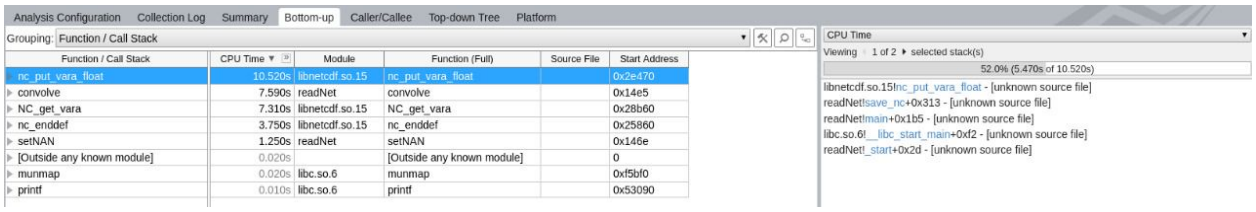
número de cores utilizados	1	2	4	8	16	32	64	128
gcc [us]	2106213	1177221	774393	409322	225050	**134593**	136620	139232
icc [us]	815834	433709	236095	130291	76092	**73988**	105605	129747

Profiling

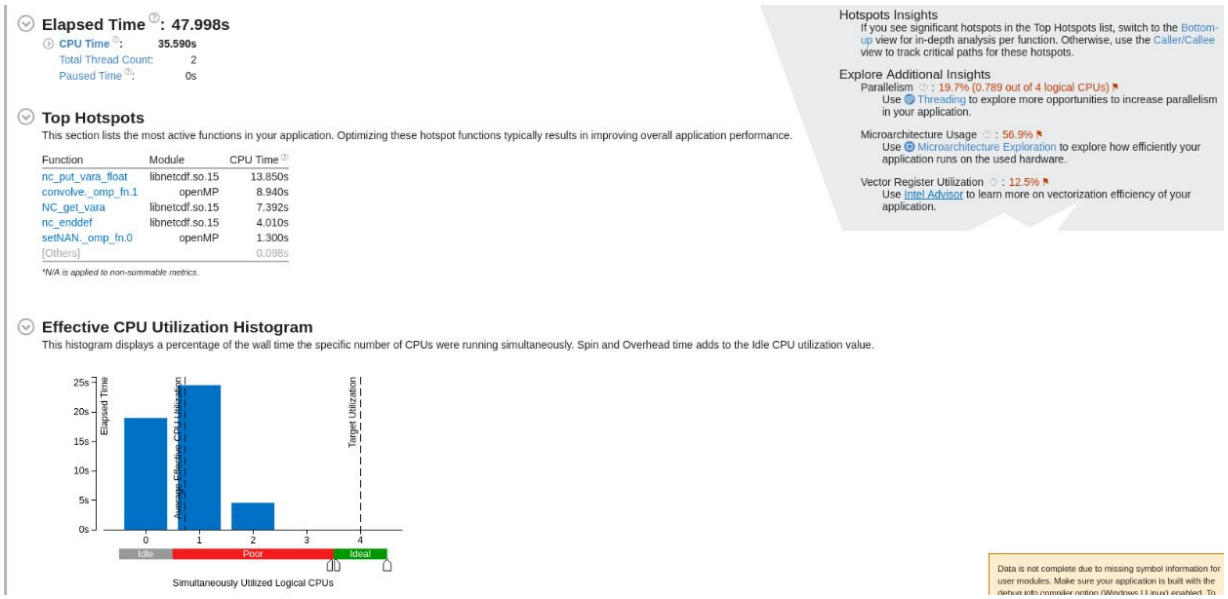
La herramienta utilizada para hacer el profiling viene en el paquete de System Studio de Intel y es el vTune Amplifier. Dicho análisis es realizado en la computadora del estudiante.

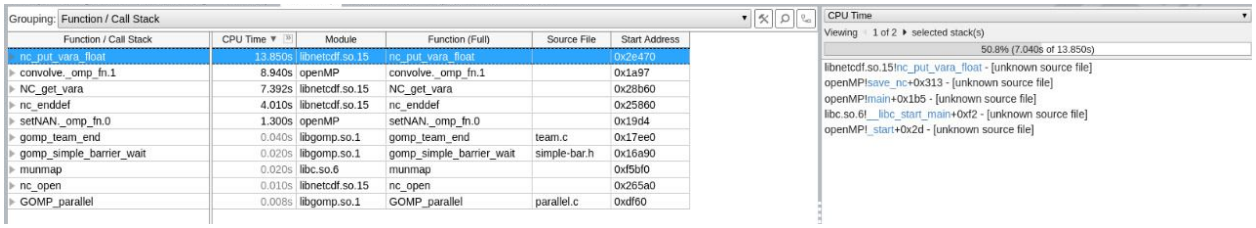
A continuación se muestra la ejecución del programa de manera procedural (un solo hilo):





Ahora se ejecuta el programa de forma paralela con 4 hilos y se analiza nuevamente con la herramienta:





Conclusiones

Al finalizar este trabajo, se pudo aprender sobre los beneficios de explotar el paralelismo de un programa sabiendo encontrar las secciones donde se puede hacer el mismo trabajo dividiendo las tareas en varios hilos.

Se obtuvo experiencia en el manejo de la librería OpenMP y las diferentes posibilidades de manejo de concurrencia y multihilos, pudiendo aplicar los distintos conocimientos adquiridos a lo largo de la carrera (Programación Concurrente, Sistemas Operativos, etc).

También se pudo ver la gran mejora de rendimiento en el programa al ejecutar con un cluster de alto rendimiento, aprovechando los recursos disponibles de la facultad.