Cátedra de Sistemas Operativos II

Trabajo Práctico Nº II

Soriano, Juan

16 de mayo del 2019

Introducción

En esta sección se encuentra la especificación de los requerimientos de la aplicación de lectura de un archivo .nc y aplicación de un filtro de bordes de forma procedural y paralela correspondiente al trabajo N° 2 de la Sistemas Operativos II.

Propósito

El propósito del Trabajo Práctico n° 2 de Sistemas Operativos II es el de diseñar e implementar un software que sea capaz de leer un archivo .nc conteniendo una imagen con valores del tipo float que representan el brillo de cada pixel, luego realice una convolución aplicando un filtro de bordes y se exporte la imagen resultante empaquetada en un archivo binario. Se ofrecen dos implementaciones. La primera es procedural, sin el uso de paralelismo, luego paralelizar la misma implementación y observar los cambios en la performance.

Referencias

1. Linux man pages: https://linux.die.net/man/

2. GNU, “5 Making The Best Use of C”, https://www.gnu.org/prep/standards/html_node/Writing-C.html

3. W. Richard Stevens, Stephen A. Rago, Advanced Programming in the UNIX Environment, 3rd Edition, Adison-Wesley

Descripción General

En la presente sección se explicarán los distintos aspectos del trabajo. Las funciones que ofrece el producto, la perspectiva del mismo, las características de los usuarios, requisitos para futuras iteraciones, entre otras cosas.

Perspectiva del Producto

El producto es una implementación de una solución a un problema de aplicación real científica. Teniendo los datos de una imagen satélital almacenada en un archivo con el formato .nc del estándar netcdf4, se importa dicha imagen que contiene un arreglo de tipo de dato “float”, se realiza una convolución para detección de bordes y se exporta un archivo binario de floats con la matriz de la imagen resultante.

Funciones del Producto

Las funciones del producto se pueden dividir en 3 partes, y a su vez, se ofrecen dos implementaciones distintas en cuanto a performance de las mismas funcionalidades. El archivo realiza principalmente 3 tareas:

1. Lectura del archivo .nc y almacenamiento en memoria de los datos recopilados.
2. Cálculo de la convolución para cada punto de la imagen.
3. Almacenamiento de la matriz resultante como un archivo binario de floats.

Características de los Usuarios

El producto se destina a usuarios con conocimientos básicos del manejo de sistemas operativos Linux.

Restricciones

Las restricciones son muy bajas. Se necesita de una computadora con requerimientos de hardware mínimos capaz de ejecutar el programa. Para computadoras con menor capacidad de cómputo, el programa se ejecutará de la misma manera pero tomará más tiempo. Se necesitará de un compilador GCC para compilar el código. Se debe tener instalada la librería netCDF.

Requisitos Futuros

Queda un gran abanico de posibles funcionalidades para ser agregadas, a continuación se listan algunas de ellas:

- Mejorar el rendimiento del programa cambiando la forma de realizar el cálculo del algoritmo.
- Programar la búsqueda del archivo .nc de forma recursiva en el directorio donde se ejecuta el archivo para no necesitar ubicar el archivo en el directorio top level.

Requisitos Específicos

En la presente sección se explicará con mayor lujo de detalle las entradas y salidas del sistema como así también una descripción de las funciones y requerimientos funcionales del producto.

Interfaces Externas

A continuación se ofrece una breve descripción de las interfaces implementadas.

Interfaz de usuario

No se ofrece una interfaz de usuario. El archivo será ejecutado de la forma conveniente para el usuario, por consola con permisos de ejecución o por interfaz gráfica del sistema operativo Linux, con los mismos permisos.

Funciones

En esta sección se listan los requerimientos funcionales de la aplicación para la versión 1 del release del producto con fecha 14 de Mayo de 2018:

1. Parseo de archivo .nc con la estructura detallada en el enunciado del trabajo práctico.
2. Cálculo de la convolución de todos los puntos de la imagen.
3. Generación de un archivo de salida binario cuya estructura se detalla más adelante.
4. Implementación tanto procedural como paralelizada (utilizando la librería openmp) de la solución del problema.
5. Ejecuciones tanto procedurales como paralelas en una computadora de uso personal y en un Clúster puesto a disposición por la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba.
6. Comparación de performance con herramientas de profiling y generación de gráficos comparativos.

Requisitos de Rendimiento

En esta sección se listan los requerimientos funcionales de la aplicación para la versión 1 del release del producto con fecha 16 de Mayo de 2019:

1. Uso de memoria mínimo y apropiado para que el programa pueda ser corrido en cualquier computadora de uso personal con múltiples threads de procesamiento en el procesador con al menos 1000mb de memoria RAM libres.

Restricciones de Diseño

En esta sección se listan las restricciones de diseño de la aplicación para la versión 1 del release del producto con fecha 16 de Mayo de 2019:

1. El código tanto procedural como paralelizado será desarrollado en lenguaje C.
2. Para la implementación paralelizada se utilizará la librería openmp
3. Sintaxis de código del tipo GNU o Linux Kernel.

Atributos del Sistema

En esta sección se listan los atributos del sistema de la aplicación para la versión 1 del release del producto con fecha 16 de Mayo de 2019:

1. Sistema portable, es decir, deberá ser capaz de correr en cualquier sistema operativo.
2. Sistema compilado con GCC y flags -Werror -Wall -pedantic
3. Se correrá la herramienta de análisis de código Cppcheck.

Estructura del archivo binario de salida generado

El archivo binario de salida está formado por (NX-NKERNEL+1)*(NY-NKERNEL+1) de floats. En donde NX y NY representan la cantidad de pixels de la imagen original en el eje X e Y respectivamente, NKERNEL representa la dimensión del Kernel de convolución. El archivo está escrito por filas, por ejemplo, los primeros NY-NKERNEL+1 valores del archivo representan la primer fila de la imagen de salida.

Profiling

Perf es una herramienta de “profiling” para sistemas basados en Linux 2.6+ que permite realizar mediciones de rendimiento basado en “perf_events”, una interfaz que se ofrece en las versiones del kernel de Linux más recientes. Perf ofrece una interfaz de comandos sencilla. Utilizando el comando “perf --repeat <n>” en dónde n es la cantidad de veces que se desea ejecutar el programa, se generan estadísticas. Para más información, dirigirse a la manpage de

Linux: http://man7.org/linux/man-pages/man1/perf-stat.1.html.

Fuente: https://perf.wiki.kernel.org/index.php/Tutorial.

A continuación se muestra una salida a modo de ejemplo de una corrida en la computadora local con 8 threads:

```
Performance counter stats for './main':

        10,062.96 msec task-clock:u              #    1.043 CPUs
utilized
   25,320,113,321      cycles:u                  #    2.516 GHz
   57,119,089,478      instructions:u            #    2.26  insn per
cycle
       49,847,178      cache-references:u        #    4.954 M/sec
        7,403,087      cache-misses:u            #   14.852 % of all
cache refs

      9.645649322 seconds time elapsed

      6.604758000 seconds user
      3.278238000 seconds sys
```

Se debe considerar que este tiempo tiene en cuenta la carga de la imagen a memoria y el guardado de la imagen de salida en memoria.

Implementación y Resultados

La imagen original con la que se trabaja se presenta a continuación:

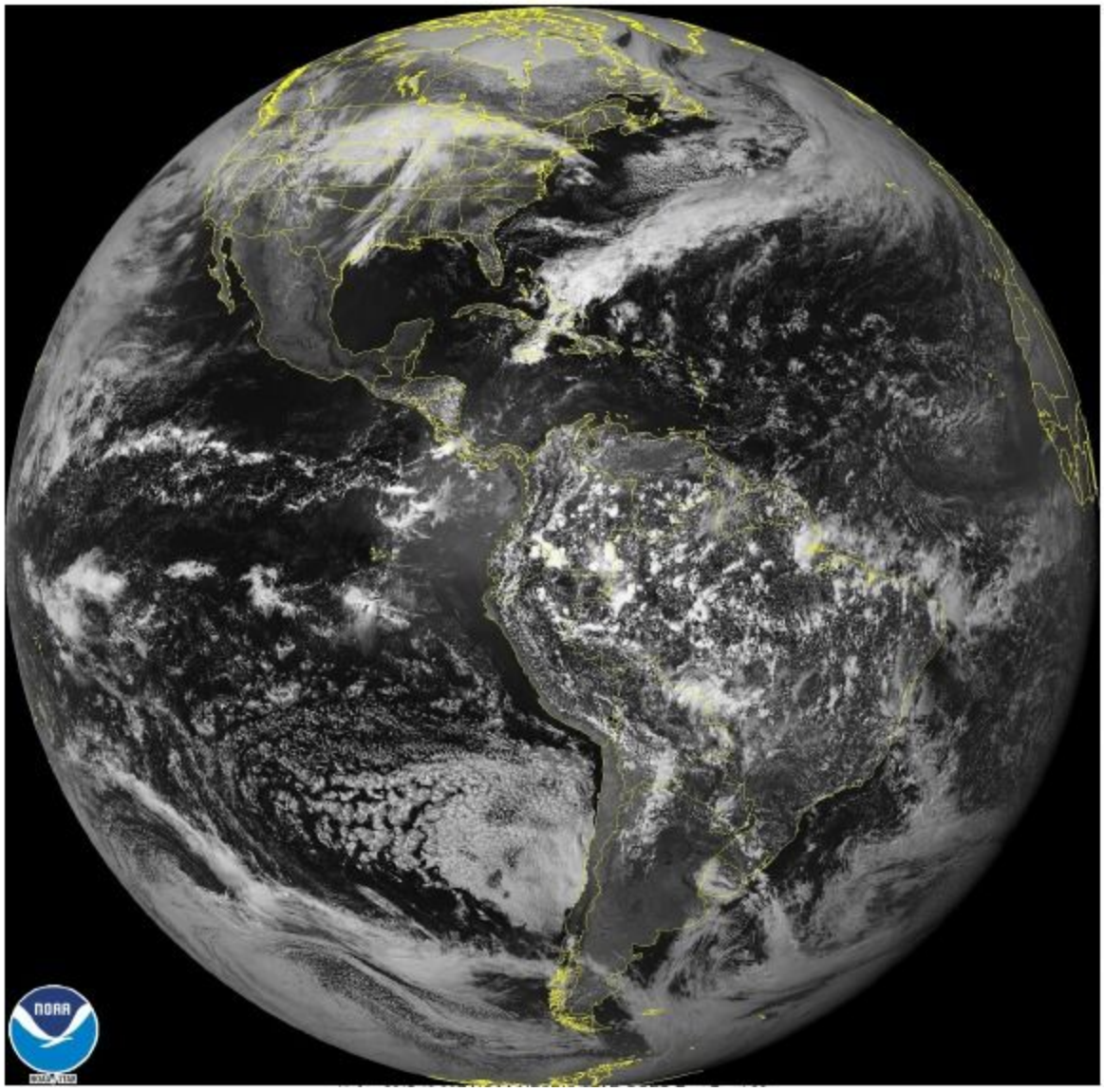

Figura 1. Imagen original

A continuación se procede a ejecutar el programa y se plotea en Python con matplotlib. La imagen obtenida con saturación en Vmin=0, Vmax=500 es:

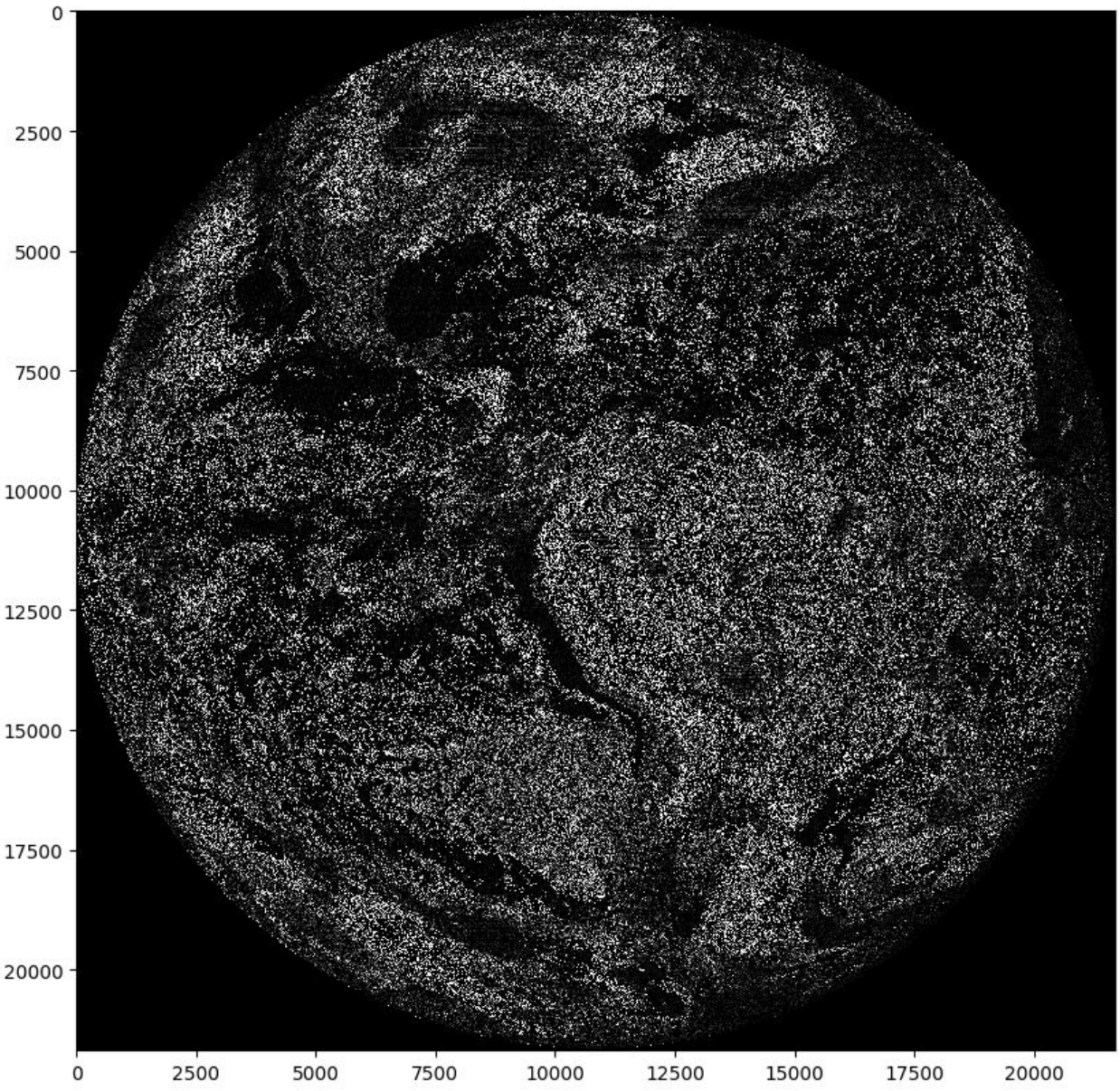

Figura 2 Imagen obtenida luego de la detección de bordes

Por ssh con las credenciales para “Estudiante35”, se accedió al Cluster de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba, se pasaron los archivos correspondientes por SCP y se corrió el programa sobre “yaku05”. En el caso de la computadora local en la se ejecutó el programa disponía de un procesador Intel I5 4690K overclockeado a 4.2 GHZ. A continuación se muestra la salida de lscpu del clúster:

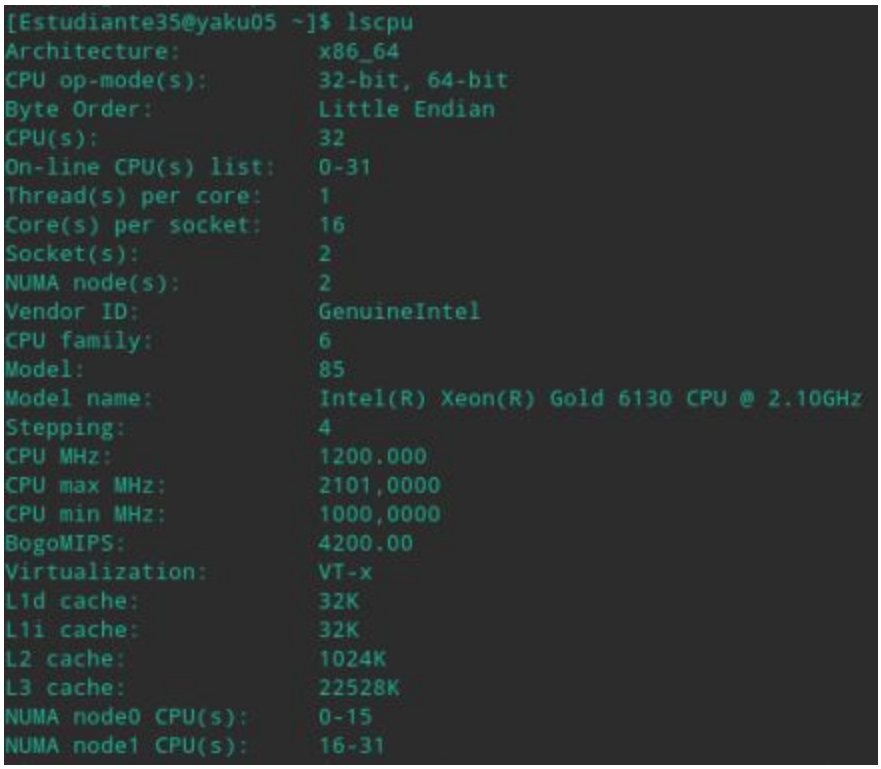

Figura 3 Procesador del cluster

Para compilar en el Cluster se utilizó ICC en lugar de GCC para explotar al máximo el potencial de cómputo.

A continuación se presenta el gráfico de tiempos obtenidos, para mayor resolución, en la carpeta data/graphs del repositorio se encuentra el mismo gráfico.

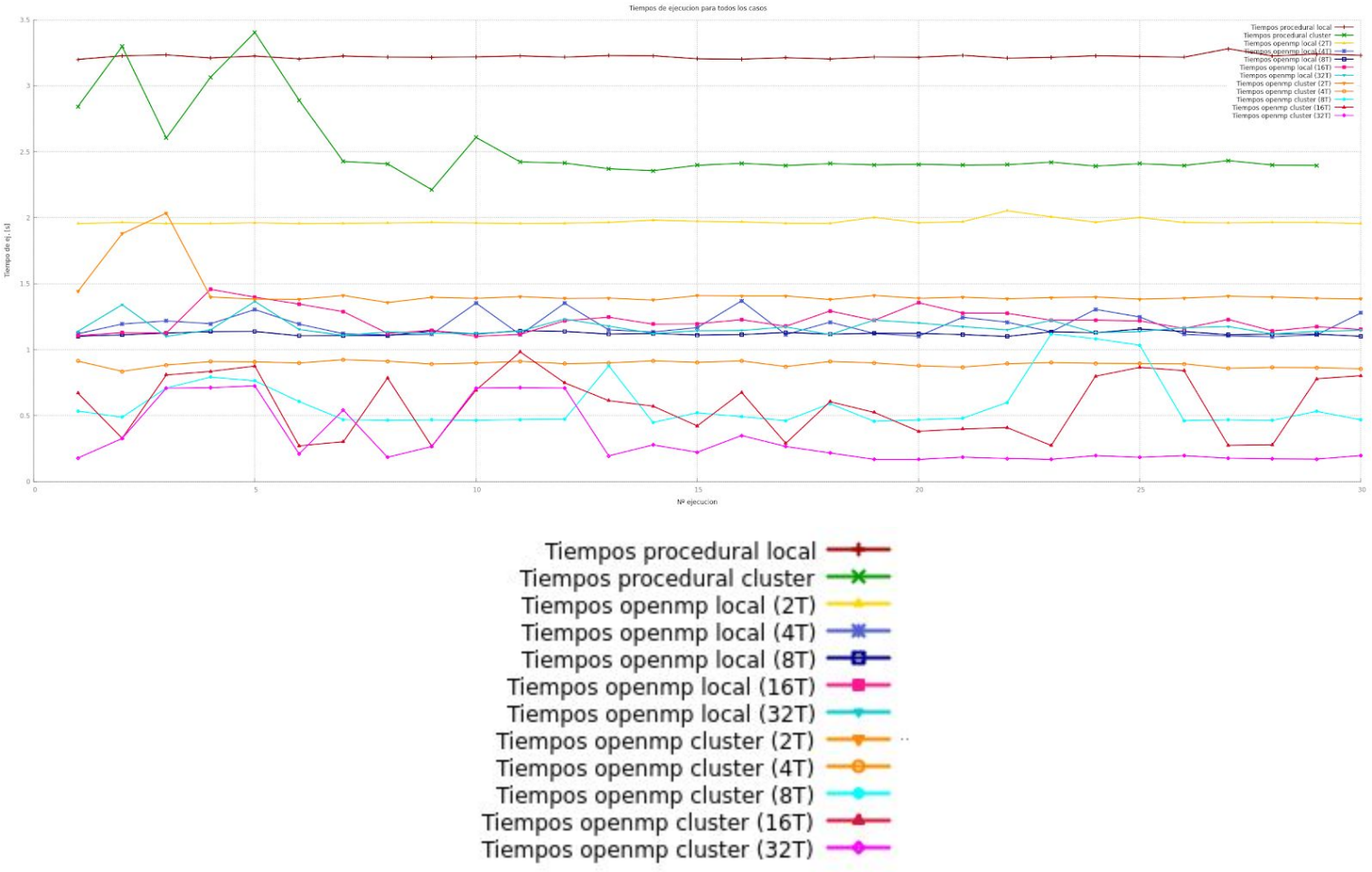

Figura 4: Tiempos de ejecución

Conclusiones

A modo de conclusión se puede ver que los resultados obtenidos en la computadora local son superados por los resultados obtenidos en el cluster. Esto se debe a que el cluster dispone de un intel Xeon que permite explotar la creación de hilos y ejecución en paralelo

Se puede ver en los gráficos que la implementación con OpenMP mejora ampliamente el rendimiento del programa, sobre todo en el caso de la ejecución en el Cluster. Esto se logra debido a que la implementación presenta múltiples ciclos “for” independientes uno de los otros. En el caso del cluster, con 32 hilos el tiempo de ejecucion es el más bajo, mientras que en el caso de la computadora local, esto no es tan claro, en 4-8 threads el rendimiento parece alcanzar su máximo debido a que el Intel i5-4690k tiene 4 cores.