

Trabajo práctico № II - OpenMP

CÁTEDRA DE SISTEMAS OPERATIVOS II

CASABELLA MARTIN, 39694763
martin.casabella@alumnos.unc.edu.ar

19 de mayo de 2019

Índice

Introducción	3
OpenMP	3
Propósito	3
Ámbito del sistema	3
Definiciones, acrónimos y abreviaturas	3
netCDF	3
Datos netCDF	3
Convolucion bidimensional y filtro	4
Bordes	4
Descripción general del documento	5
Descripción general	5
Perspectiva del Producto	5
Funciones del Producto	5
Características de los Usuarios	5
Restricciones, Suposiciones y Dependencias	6
Requisitos específicos	6
Interfaces externas	6
Requisitos funcionales	6
Requisitos de rendimiento	6
Requisitos de diseño	6
Atributos del sistema	7
Diseño de la solución	7
Implementación y resultados	7
Profiling	9
Valgrind/Callgrind	9
Perf	12
Imágenes procesadas	13
Análisis de tiempo de ejecución	17
Conclusiones	18

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 mas utilizados para sistemas de memoria compartida y memoria distribuida, se encuentra OpenMP y MPI respectivamente.

OpenMP

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas.

Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Propósito

En el trabajo se pretende desarrollar la solución de un problema de una convolución bidimensional que presenta cierto tipo de paralelismo. Primero debe proponerse una solución procedural, y luego una que explote el paralelismo inherente del problema.

Para ello, deben madurarse y entenderse los conceptos del funcionamiento del estándar OpenMP, sus aplicaciones, ventajas y desventajas.

Ámbito del sistema

Se requiere realizar una convolución bidimensional a partir de los datos extraídos de un satélite, que se reciben en forma de archivo con el formato .nc del estándar netCDF4. La convolución debe ser monocore y multicore, de tal manera que se puedan comparar los resultados.

Definiciones, acrónimos y abreviaturas

netCDF

Son 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 transforma en estándar abierto. La librería tiene versiones en Fortran, Python, Java y C, siendo estas últimas las que vamos a utilizar en este trabajo.

Datos netCDF

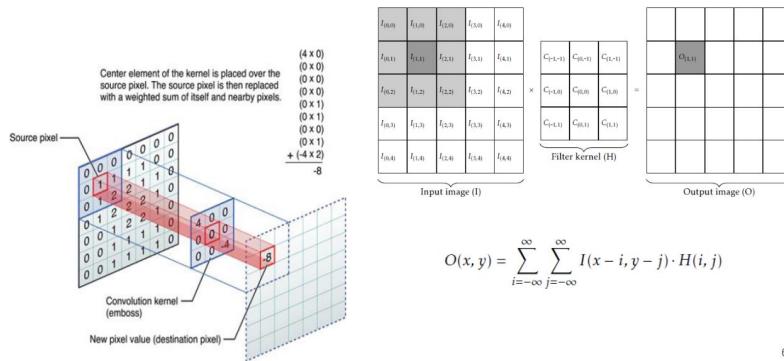
El netCDF (formulario de datos comunes en red) es un formato de archivo destinado a almacenar datos científicos multidimensionales (variables) como la temperatura, la humedad, la presión, la velo-

ciudad del viento y la dirección. Cada una de estas variables se puede mostrar mediante una dimensión (por ejemplo, tiempo) en ArcGIS creando una vista de tabla o de capa a partir del archivo netCDF.

Convolucion bidimensional y filtro

Las imágenes digitales son consideradas como señales discretas bidimensionales, las cuales son procesadas por sistemas discretos o filtros.

La operación fundamental en lo que abarca el procesamiento de imágenes, es la convolución en dos dimensiones, entre la imagen (o señal de entrada) con el filtro mencionado, denominado kernel.



6

Figura 1: Operación de convolucion 2D

Al variar los coeficientes del filtro discreto (o kernel) empleado, se producen diferentes efectos en la imagen procesada:

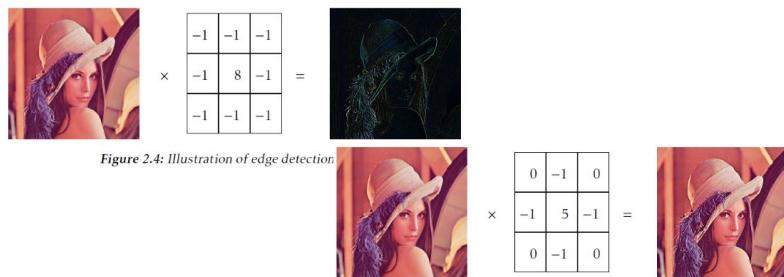


Figure 2.2: Illustration of sharpen filter.

Figura 2: Influencia del filtro utilizado

Bordes

Cuando la ventana de convolución se centra en el primer píxel de una imagen en (0,0), la máscara del kernel sobresale de la imagen en el borde superior y en el izquierdo.

Hay mas de una manera de resolver el tema. Se optó por la solución en que el pixel que va a ser procesado tiene que tener una ventana de convolución dentro de los límites de la imagen, de tal manera que los pixeles de los bordes no se procesarán.

Descripción general del documento

Este documento esta organizado en 6 secciones principales:

1. Describe las generalidades del documento y del proyecto
2. Describe la perspectiva del producto, funciones del mismo, características de usuario y restricciones
3. Referencia a los requerimientos del producto, interfaces y casos de uso. Incluye diagramas y gráficos que facilitan su entendimiento
4. Abarca un diseño de la solución
5. La última parte incluye la implementación de la solución y resultados
6. Por último, se encuentran las conclusiones

Descripción general

En la siguiente sección se presenta una breve descripción de los requisitos del sistema, sin entrar en detalle se explica las principales características del software.

Perspectiva del Producto

Como objetivo esta el diseño e implementación de un software desarrollado en lenguaje C. El componente principal es un archivo *.nc* que alberga los datos de una imagen satelital que consiste en una matriz de tipo de dato *float*. Al mismo se lo debe importar y leer, para así procesarlo en la convolución y guardarlos en un nuevo archivo.

Se generan archivos de salida tanto binarios, como en formato *.nc*, y se desarrolló un script en Python 2 que se encarga de leer dichos archivos y generar las correspondientes imágenes en formatos *png* y *svg*.

Funciones del Producto

El producto no posee una interfaz interactiva con el usuario, si no que solo se ejecuta lee el archivo, realiza la convolución y genera los correspondientes archivos de salida.

- **Lectura:** Lectura del archivo *.nc* y almacenamiento en memoria de los datos extraídos
- **Convolución:** Calculo de la convolución con 3 implementaciones diferentes (procedural, y dos formas donde se explota el paralelismo)
- **Escritura de salida:** Almacenamiento de la imagen filtrada resultante como archivos binarios (procedural, primer versión paralela), y archivo *.nc* (segunda versión paralela)

Características de los Usuarios

El producto se destina a usuarios con los conocimientos básicos de manejo de sistemas operativos Linux. Se incluyeron un script para la versión local, y otro diferente para el cluster, que automatizan la compilación, ejecución y copia de archivos de salida.

Restricciones, Suposiciones y Dependencias

Se necesitará de un compilador GCC, junto con las librerías netCDF, netlib, glibc y lib instaladas. En caso que se desee, se puede compilar a mano y cambiar los directorios. Caso contrario, los scripts mencionados incluyen todas las etapas.

Requisitos específicos

Se detallan a continuación los requisitos específicos del sistema, especificados por el cliente.

Interfaces externas

Para el desarrollo del software se utiliza GNU/Linux, herramientas de compilación y librerías en su última versión estable, específicamente las versiones que cumplen los criterios son las rolling release como ArchLinux, Antergos, Manjaro y derivados:

- Kernel >= 5.0.7 (5.0.7-1-ARCH aarch64 GNU/Linux)
- GCC >= version 8.2
- GLibC >= 2.26
- Make >= 4.2.1
- libnet >= 1.1.6-3
- netcdf >= 4.7
- python >= 2.7.16-1
- python2-netcdf4 >= 1.5.1.2
- python2-matplotlib >= 2.2.4

Requisitos funcionales

El sistema no es interactivo. Debe ejecutarse, escribir resultados, terminar y salir del mismo. El archivo será ejecutado de la forma conveniente para el usuario, por consola con los respectivos permisos necesarios, o por interfaz gráfica con los mismos permisos.

Requisitos de rendimiento

El sistema debe ser capaz de correr en un SoC o una computadora personal (x86/AMD64), con al menos 4gb de ram y 8gb de espacio libre, un procesador con una frecuencia de 1GHz contando que solo compite en los recursos con los procesos del sistema operativo base. El sistema operativo base puede ser GNU/Linux o derivados de BSD.

Requisitos de diseño

Las restricciones incluyen el desarrollo de ambas soluciones en lenguaje C, utilizando la librería OpenMP.

Atributos del sistema

- Capacidad de la utilización de sistemas operativos tipo UNIX, con escasos recursos necesarios
- Escalabilidad en cuanto al grado de paralelismo

Diseño de la solución

La solución está documentada en el repositorio del proyecto. Se trabaja con tipo de dato *float*, y se generaron tres tipos de operación de filtrado de bordes (convolucion 2D):

1. **Procedural:** Forma típica de realizar esta operación: realiza la acumulación de los productos de los coeficientes del kernel con la sección correspondiente de la imagen antes de desplazarse horizontal o verticalmente para generar un nuevo pixel de salida.
2. **Paralela v1 (parallel collapse):** Algoritmo anterior pero paralelizado. Ya que consta de cuatro bucles anidados, se aplica el pragma *parallel collapse*. La característica de esta directiva para parallelizar el bucle es que colapsa los bucles formando un solo bucle largo (sea primer bucle el límite *N* y el límite del otro *M*) formando un solo bucle de *N x M*, que luego paraleliza.
3. **Paralela v2 (parallel for):** Esta solución está basada en el pragma *parallel for* donde puede setearse el numero de hilos a utilizar según los recursos que se dispongan. Básicamente se divide el bucle para que cada thread del conjunto creado, maneje una porción diferente de dos bucles anidados, que se encargan de ir moviendo el kernel a través de la imagen (horizontal o verticalmente, ya que en este caso es una matriz con la misma cantidad de filas que de columnas).

Implementación y resultados

En cuanto a medida de tiempo, se utilizo la función *omp_get_wtime()* brindada por la librería. Esta función devuelve un valor de tiempo de doble precisión, que equivale al numero de segundos del clock de tiempo real del sistema operativo. Se garantiza que este valor no cambia durante la ejecución del programa.

Iniciando este clock previo a llamar a la función, y luego al volver (habiendo terminado la operación de convolución) restando el correspondiente tiempo de inicio, se puede obtener el tiempo que demoro la operación en si.

Se ejecuto 50 veces el programa, variando el numero de threads, en un rango de 2 a 24 para la computadora local.

En cuanto al cluster, también se ejecuto 50 veces el programa, variando el numero de threads, hasta llegar a 32 threads instanciados. Esto se debe a las prestaciones que ofrecía el mismo.

Se ejecuto en una computadora local, con las siguientes prestaciones en cuanto al procesador:

Processor Name	<i>Intel Core i7-7700HQ</i>
CPU Family	<i>7th Generation Intel Core "Kaby Lake"</i>
Number of Cores	<i>Quad-core / 2 threads per core</i>
CPU Clock Speed	<i>2.8-3.8GHz</i>
Cache Size	<i>6MB</i>
Memory Support	<i>DDR3 1600MHz DDR4 2133MHz DDR4 2400MHz</i>
Integrated Graphics	<i>Intel HD 630</i>
Power Consumption	<i>45 Watts</i>
Production Technology	<i>14-nanometer</i>
Typical Use	<i>Gaming & high-performance laptops</i>
Technologies	<i>Intel HyperThreading Intel QuickSync Intel TurboBoost VT-d virtualization VT-x virtualization</i>
User Benchmark Average CPU Score	<i>69.8</i>

Figura 3: Prestaciones del procesador de la computadora local

En cuanto al cluster, el mismo dispone de Posee dos pastillas Intel Xeon Gold 6130 con 64Gb de ram (4 canales):

Performance	
# of Cores	16
# of Threads	32
Processor Base Frequency	2.10 GHz
Max Turbo Frequency	3.70 GHz
Cache	22 MB L3
# of UPI Links	3
TDP	125 W

Supplemental Information	
Embedded Options Available	Yes
Datasheet	View now
Product Brief	View now
Additional Information URL	View now

Memory Specifications	
Max Memory Size (dependent on memory type)	768 GB
Memory Types	DDR4-2666
Maximum Memory Speed	2666 MHz
Max # of Memory Channels	6
ECC Memory Supported	Yes

Figura 4: Prestaciones del procesador del cluster

Profiling

Valgrind/Callgrind

Valgrind es un conjunto de herramientas libres que ayuda en la depuración de problemas de memoria y rendimiento de programas. La herramienta más usada es Memcheck. Memcheck introduce código de instrumentación en el programa a depurar, lo que le permite realizar un seguimiento del uso de la memoria y detectar problemas.

La opción `--leak-check` activa el “memory leak detector”. El programa se ejecutará con una velocidad inferior a la normal(en un orden de 20-30 %), y usará una cantidad de memoria mayor. Memcheck emitirá mensajes sobre las “fisuras” y resto de errores de memoria que detecte.

Aspectos a tener en cuenta:

- En cada mensaje de error aparece mucha información. Sería conveniente leerlos con detalle.
- El número entre ==number== es el identificador de proceso (PID). Normalmente carece de importancia.
- La primera línea (invalid write...) indica cual es el error. Aquí indica que el programa escribió en alguna zona de memoria en la que no tendría que haberlo hecho debido a un “heap block overrun”.
- A continuación de la primera línea se puede ver una traza de la pila que te dice donde ocurrió el problema. Las trazas de pila pueden tener un tamaño considerable, y ser confusas, especialmente si estás usando el C++ STL. Puede ser útil leerla desde abajo hacia arriba. Si la traza de la pila no es lo suficientemente grande, usa una opción `-num-callers` mayor.
- Las direcciones (ej. 0x0804838F) no son importantes normalmente, pero ocasionalmente son cruciales para seguir la traza de los errores más extraños.
- Algunos mensajes de error tienen un segundo componente que describe las direcciones de memoria involucradas

Haciendo uso de memcheck, se obtuvo:

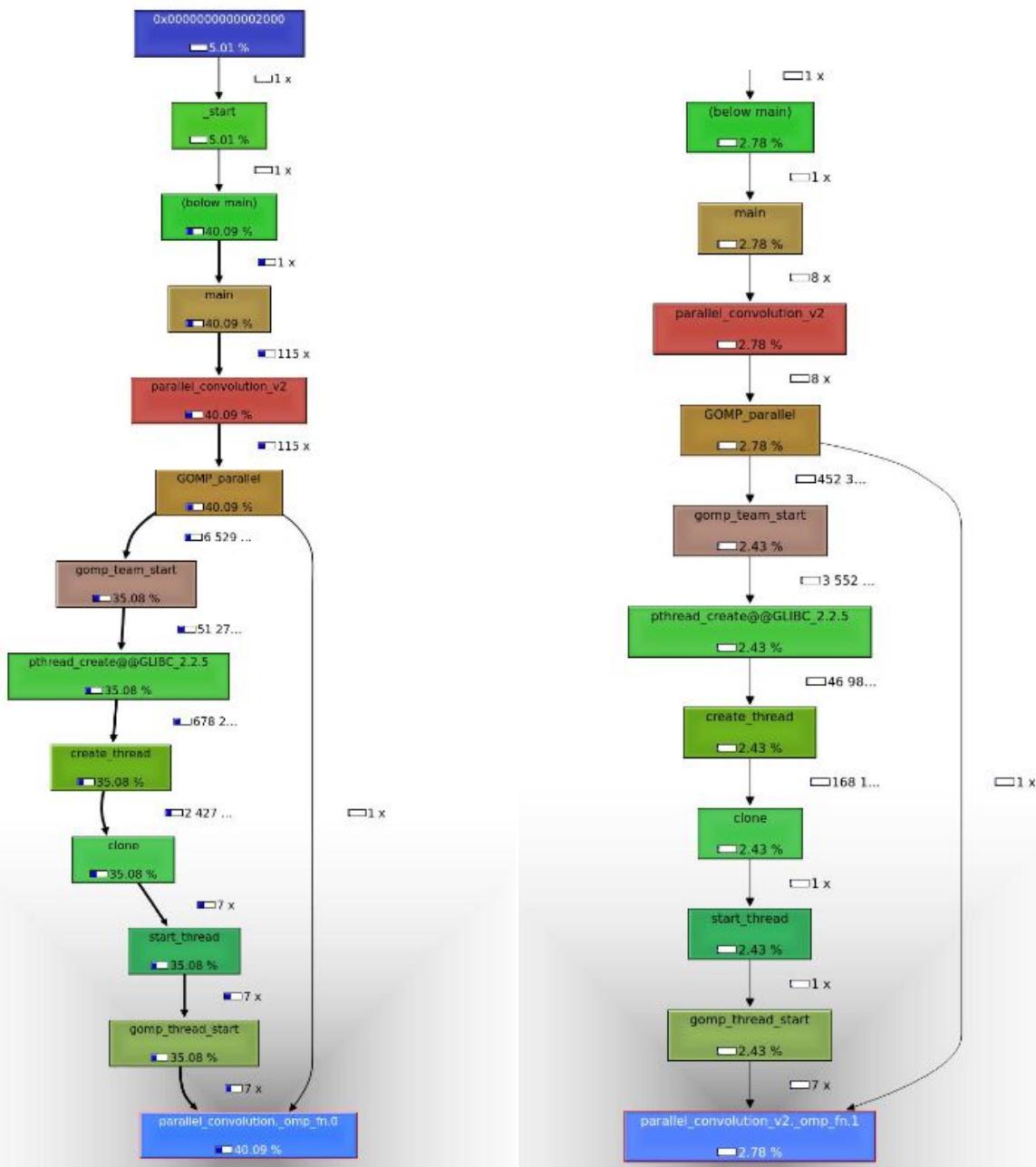
```
==3307==  
==3307== HEAP SUMMARY:  
==3307==     in use at exit: 6,518 bytes in 14 blocks  
==3307==   total heap usage: 220,392 allocs, 220,378 frees, 11,262,999,401 bytes allocated  
==3307==  
==3307== 2,128 bytes in 7 blocks are possibly lost in loss record 7 of 8  
==3307==    at 0x483AB65: calloc (vg_replace_malloc.c:752)  
==3307==    by 0x4012A32: allocate_dtv (in /usr/lib/ld-2.29.so)  
==3307==    by 0x40133E1: __dl_allocate_tls (in /usr/lib/ld-2.29.so)  
==3307==    by 0x49FD698: pthread_create@@GLIBC_2.2.5 (in /usr/lib/libpthread-2.29.so)  
==3307==    by 0x49DC09A: gomp_team_start (team.c:817)  
==3307==    by 0x49D2F9D: GOMP_parallel (parallel.c:167)  
==3307==    by 0x1093EB: main (in /home/mcasabella/Desktop/TP2_2Dconv/bin/main)  
==3307==  
==3307== LEAK SUMMARY:  
==3307==    definitely lost: 0 bytes in 0 blocks  
==3307==    indirectly lost: 0 bytes in 0 blocks  
==3307==    possibly lost: 2,128 bytes in 7 blocks  
==3307==    still reachable: 4,390 bytes in 7 blocks  
==3307==          suppressed: 0 bytes in 0 blocks  
==3307== Reachable blocks (those to which a pointer was found) are not shown.  
==3307== To see them, rerun with: --leak-check=full --show-leak-kinds=all  
==3307==  
==3307== For counts of detected and suppressed errors, rerun with: -v  
==3307== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Figura 5: Chequeo de perdida de memoria ('memory leak')

A su vez, otra herramienta es *Callgrind*, la cual genera un informe detallado sobre cada rutina del programa, entregando el porcentaje de tiempo que éstas ocupan respecto al tiempo total de ejecución, el número de veces que éstas son ejecutadas y con cuáles otras rutinas interaccionan.

Callgrind corre en el framework Valgrind, y cuando se utiliza para hacer profiling para cierta aplicación, la aplicación se transforma en un lenguaje intermedio y luego corre en un procesador virtual emulado por Valgrind. Esto tiene un gran overhead de tiempo de ejecución (de 10 a 50 veces más lento), pero la precisión del profiling es realmente buena.

Adicionalmente, para este análisis se utiliza la herramienta de visualización denominada KCachegrind la cual permite obtener una mejor visión general de la información generada por Cachegrind y Callgrind. KCachegrind transforma la información en tablas y grafos para su fácil interpretación:



(a) Grafo de la función de convolución paralela v1

(b) Grafo de la función de convolución paralela v1

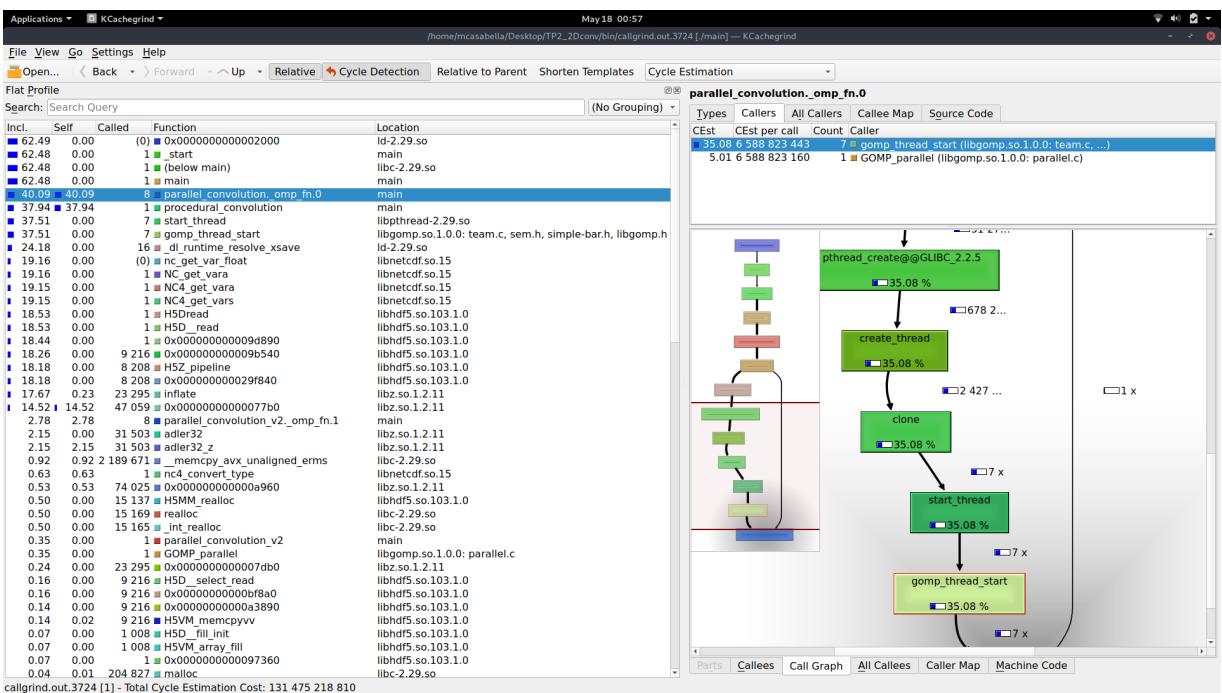
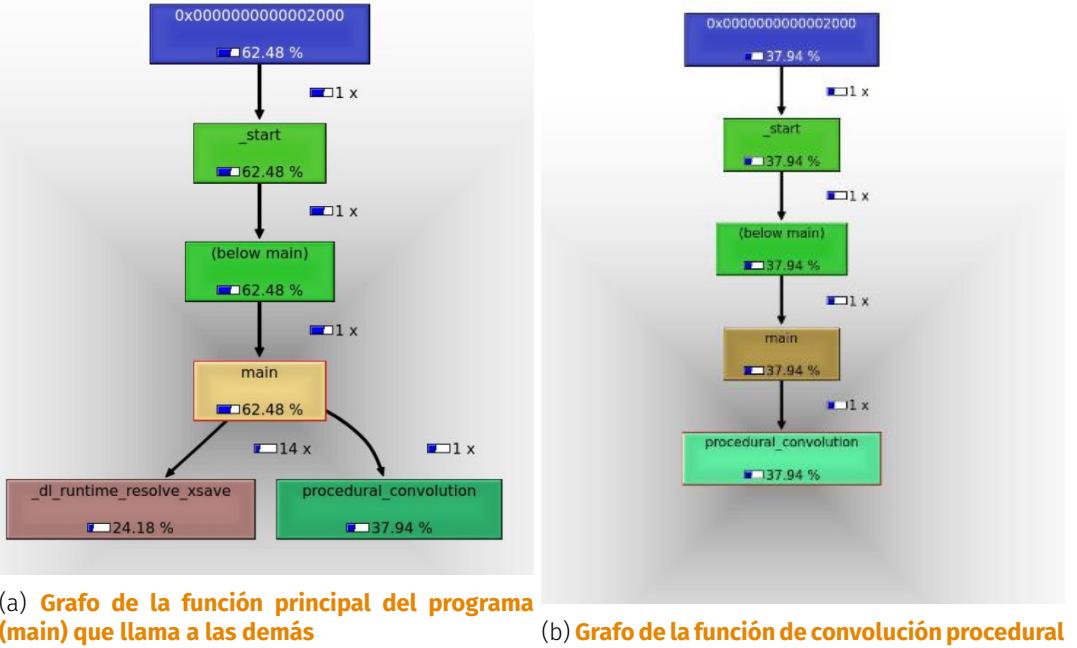


Figura 8: Apertura del archivo generado por la ejecución de Callgrind, mediante el software KCachegrind

Perf

Perf es una herramienta de análisis de performance que se accede como cualquier comando linux y provee de una serie de subcomandos.

Perf puede obtener varios datos desde aquellos que son parte del sistema o bien procesos realizados por el usuario, proporciona contadores de hardware, puntos de rastreo (tracepoints), contadores de perfomance de software, pruebas dinámicas, entre otros. La mayoría de la funcionalidad de Perf se encuentra integrada desde el kernel.

Es posible usar el comando *perf stat n* y ejecutar la misma carga de trabajo de prueba *n* veces , y obtener por cada cuenta, la desviación estándar de la media.

```
Performance counter stats for './main' (6 runs):

      23,404.89 msec task-clock:u          #    2.045 CPUs utilized          ( +-  0.71% )
          0      context-switches:u        #    0.000 K/sec
          0      cpu-migrations:u        #    0.000 K/sec
      3,038,031    page-faults:u          #    0.130 M/sec          ( +-  0.00% )
  56,594,857,414    cycles:u            #    2.418 GHz          ( +-  0.26% )
130,482,942,938    instructions:u       #    2.31  insn per cycle          ( +-  0.00% )
 17,063,660,793    branches:u          #   729.064 M/sec          ( +-  0.00% )
     226,138,687    branch-misses:u      #    1.33% of all branches          ( +-  0.06% )

      11.447 +- 0.163 seconds time elapsed  ( +-  1.42% )

[mcasabella@goleta ~/Desktop/TP2_2Dconv/bin]$
```

Figura 9: Profiling con perf para 6 ejecuciones con un número fijo de threads (8)

Imágenes procesadas

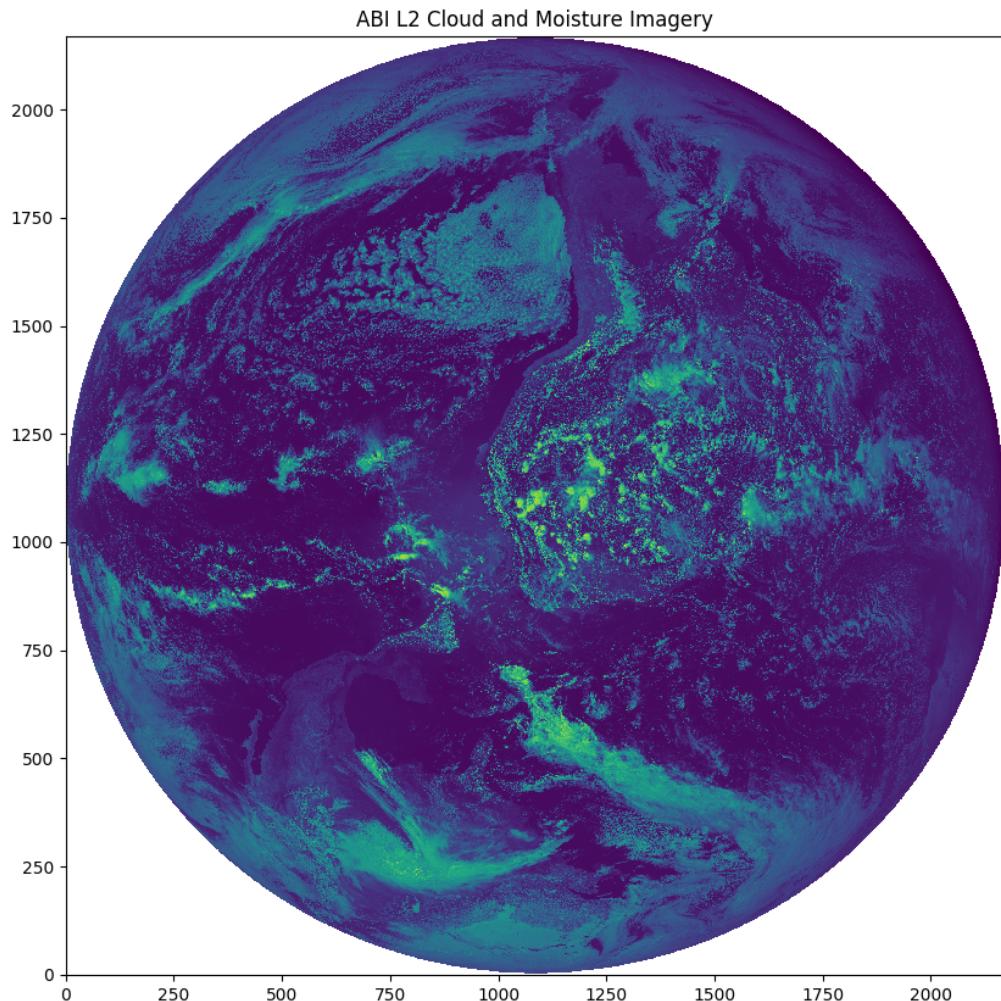


Figura 10: Imagen original ploteada desde Python, desde archivo .nc

Respecto a la convolucion procedural, su salida se almacena en un archivo binario. Se realizo lo mismo con los datos del archivo .nc para generar un binario con la imagen original.

Luego, utilizando Python, se graficron las mismas donde se varia el rango de valores de cada pixel utilizando la `matplotlib.pyplot.imshow`.

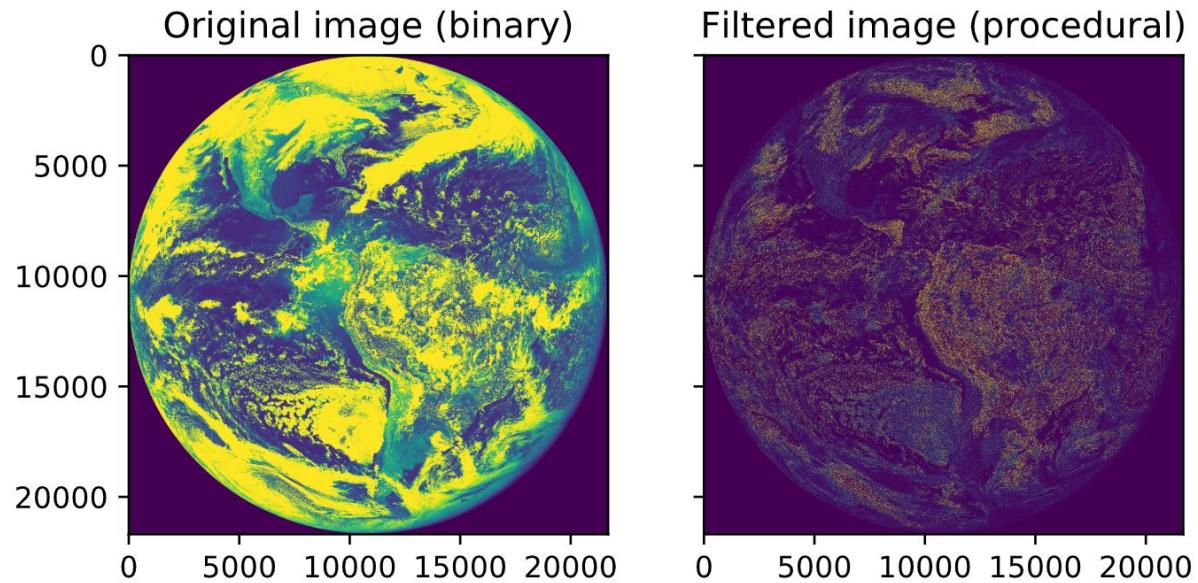


Figura 11: Imágenes binarias ploteadas, sin especificar mapeo de colores

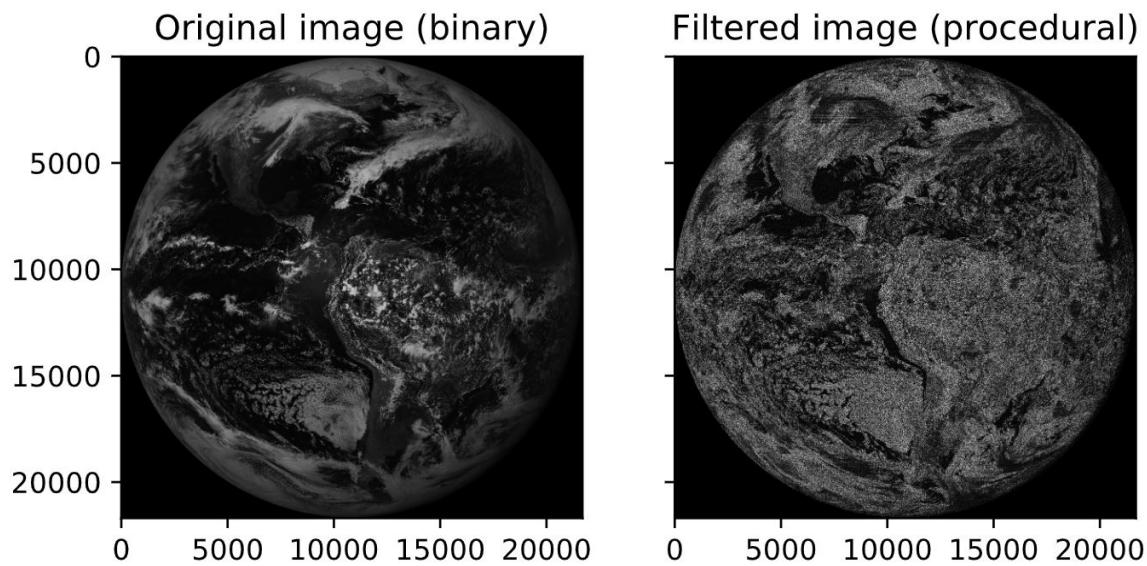


Figura 12: Imágenes en escala de grises

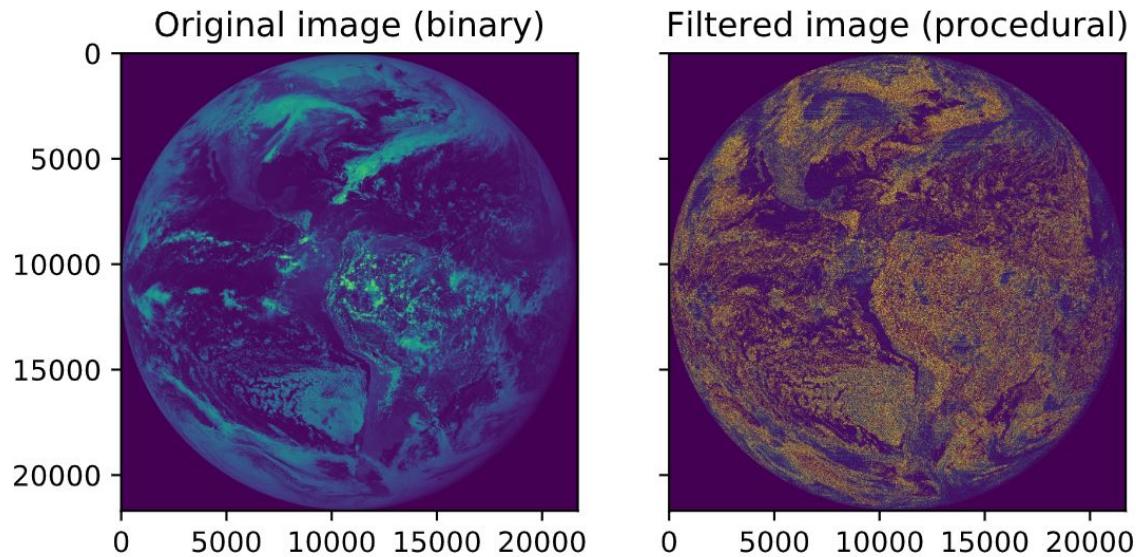


Figura 13: Imágenes variando rango de valores (min-max)

Se busca mostrar que la imagen filtrada tiene calidad aceptable, y que el cambio en el rango de valores es lo que afecta la escala de colores y tonalidades. Fijemos la misma escala, y veamos la imagen obtenida con la primer forma de convolución paralela:

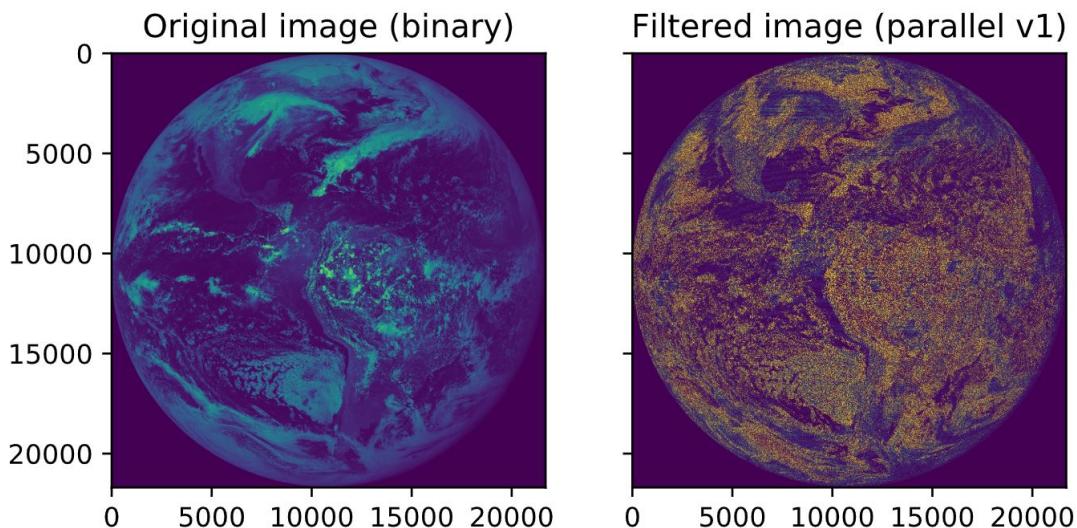


Figura 14: Imágenes, pero obtenidas de la explotación del paralelismo en la operación

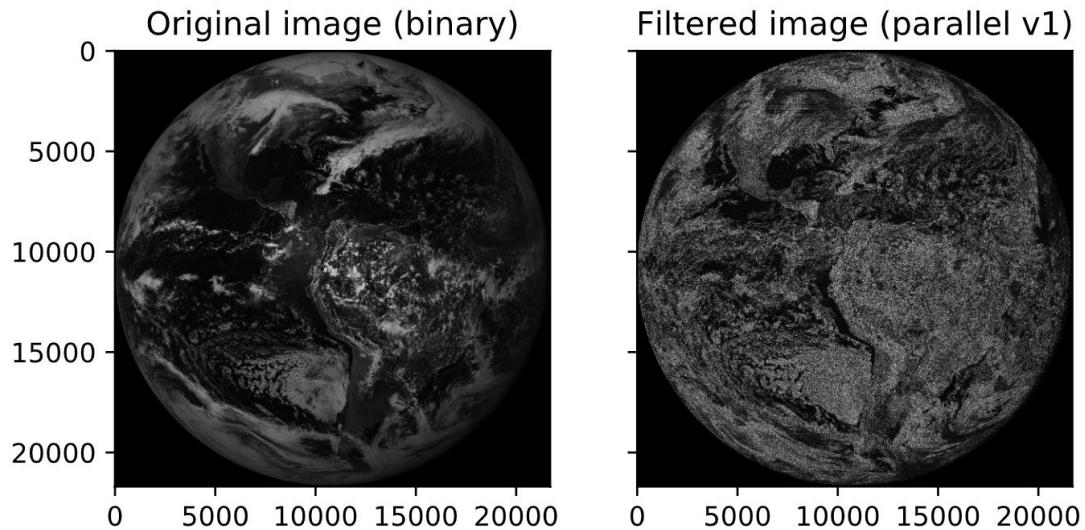


Figura 15: Imágenes, pero obtenidas de la explotación del paralelismo en la operación (escala de grises)

Recordando que se tratan de archivos binarios, veamos lo que obtenemos del segundo metodo de convolucion paralela, donde la salida se almaceno en un archivo .nc:

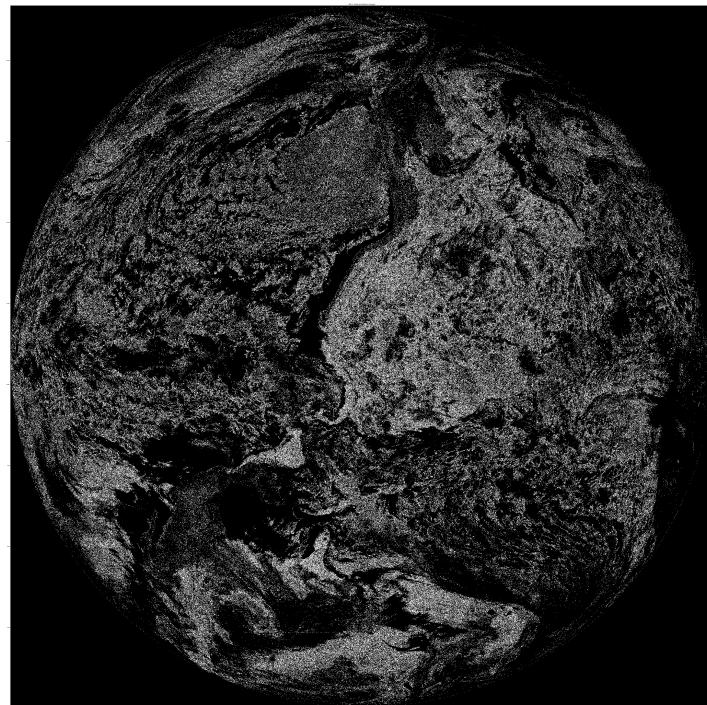


Figura 16: Imagen filtrada y pleteada desde archivo .nc (convolucion paralela v2)

Análisis de tiempo de ejecución

Al principio de la sección se anexaron las prestaciones del procesador de la máquina local *intel i7-7700hq 2.80ghz*. El mismo consta de 4 núcleos físicos, y 2 thread por núcleo. Para 50 corridas variando el número de hilos (2, 4, 8, 12, 16, 20, se obtuvieron los siguientes resultados:

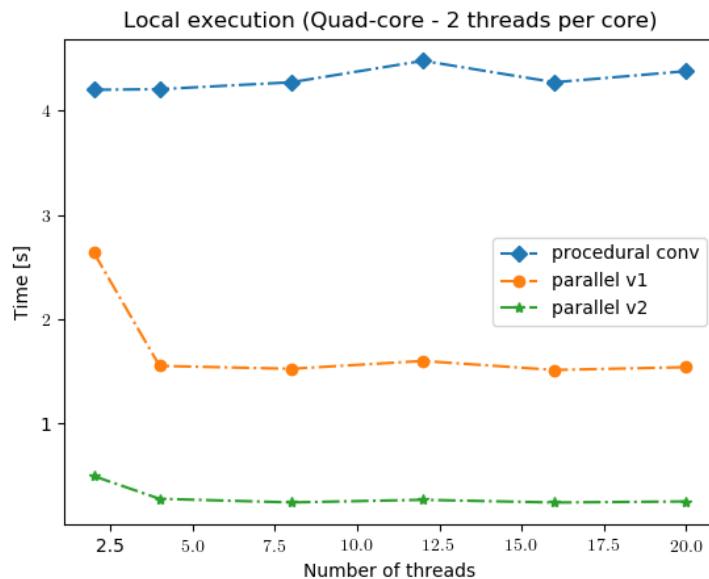


Figura 17: Ejecución de 50 corridas del programa para cada numero de hilos instanciados

En cuanto al cluster (Xeon Gold 6130 , de 16 cores y 32 threads), se obtuvieron las siguientes curvas:

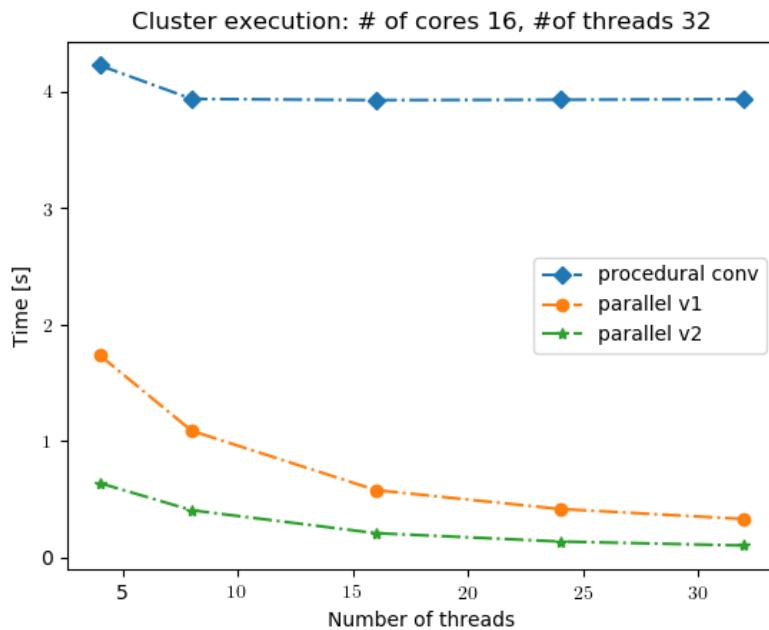


Figura 18: Ejecución de 50 corridas en el cluster del programa para cada numero de hilos instanciados

El tiempo en el eje *y* es el tiempo promedio de las 50 ejecuciones por cada numero de threads instanciados. Se obtuvieron métricas temporales con la función mencionada, y se las almaceno en archivos. Dichos archivos luego, se leyeron efectuando un promedio de los 50 valores para cada tipo de convolución, por cada nivel de paralelismo (o cantidad de threads instanciados). Con esa recopilación de datos, se confeccionaron los gráficos anteriores.

En la ejecución local se ve como a partir de que se supera el numero máximo dadas sus prestaciones, directamente se estanca el tiempo, o mejor dicho, no hay mejora, oscila.

A nivel tiempo no parece una desventaja drástica, pero cabe destacar que a nivel recursos dichos threads son instanciados y alternados en ejecución, por ende que la ejecución torna en si a ser deficiente.

En cuanto al cluster, se ve la notoria diferencia en el promedio de tiempo de las 50 ejecuciones en las dos versiones paralelas, donde se llega a un tiempo casi un 45 % mas rápido que en la ejecución en la maquina local.

Conclusiones

Se puede ver que los resultados obtenidos en la computadora local son superados por los obtenidos del cluster. Esto se debe a las prestaciones de su procesador.

A su vez, paralelizar el problema planteado con OpenMP incrementa la performance del programa. Esto es por la paralelismo inherente de este tipo de operación, que si bien en la computadora local se llega a un mínimo rondando los 6 u 8 threads, en el cluster se ve claramente dicha disminucion notoria, llegando a un tiempo de 100[ms] en filtrar la imagen por completo.