



UNC

Universidad
Nacional

Cátedra de Sistemas Operativos II

Trabajo Práctico

Nº II

Illesca Martín Andrés
14 de mayo del 2020

Índice

Introducción	3
Propósito	3
Ámbito del Sistema	3
Definiciones, Acrónimos y Abreviaturas	4
Referencias	4
Descripción General	5
Perspectiva del Producto	5
Funciones del Producto	5
Características de los Usuarios	5
Restricciones	6
Suposiciones y Dependencias	6
Requisitos Futuros	6
Requisitos Específicos	6
Interfaces Externas	6
Requisitos de Rendimiento	6
Restricciones de Diseño	7
Atributos del Sistema	7
Otros Requisitos	7
Diseño de solución	7
Implementación y Resultados	9
Profilers	17
Conclusiones	22

Introducción

El objetivo de este trabajo es el procesamiento de una imagen bmp no comprimida de 24 bits y sin paleta de colores, el proceso consta de la separación en dos regiones siendo una región la parte exterior de un círculo de radio R ingresado por parámetro y la segunda el interior de dicho círculo, a estas regiones se les aplicará un filtro convolucional en dos dimensiones y un filtro lineal respectivamente. Se hará también la comparación entre el funcionamiento serial y paralelo de la aplicación usando para este último el estándar de memoria compartida openmp .

Propósito

Como se indicó en el punto anterior el propósito de este práctico es la resolución de un mismo problema usando la programación en serie y paralela y comparando los rendimientos en ambos casos haciendo mediciones de tiempos y uso de profilers para tal tarea. Se hizo la medición de tiempos tanto localmente como en cluster.

Ámbito del Sistema

No hay tantos requisitos a nivel software, en las plataformas(cluster o local) debe haber disponible el compilador gcc la librería math y openmp para poder hacer el uso correcto del producto, la ejecución tanto procedural como paralela se hizo en el nodo del cluster otorgado por la facultad y en la computadora local

A nivel de hardware se cuenta con los siguientes sistemas:

Local:

- Intel core i5-9400 coffee lake 2,90 Ghz
- 6 cores, 6 threads sin hyperthreading
- 16 Gb de memoria RAM

Cluster

- Nodo con microprocesador Intel Xeon E5-2630 v4 2,20GHz
- 10 cores, 19 threads con hiperThreading
- memoria RAM del sistema 62 gb

Definiciones, Acrónimos y Abreviaturas

kernel: se refiere a la matriz ([41][41] para las pruebas) que se utilizó para realizar la convolución

k: valor que representa el contraste del filtro lineal

l: valor que representa el brillo del filtro lineal.

Referencias

1. filminas de clase.
2. https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2_fig3_324165524
3. man de linux
4. <https://www.openmp.org/specifications/>
5. <https://github.com/Sistemas-Operativos-II-2020/simple bmp>
6. <http://manpages.ubuntu.com/manpages/bionic/es/man1/gprof.1.html>
7. <https://perf.wiki.kernel.org/index.php/Tutorial>
8. <https://valgrind.org/>

Descripción General

Perspectiva del Producto

Se requirio el analisis de una imagen bmp con la ayuda de la librería simple_bmp la cual provee herramientas para la lectura,inicializacion y guardado de una imagen bmp de 24 bits sin paleta de colores, la cual al leer una imagen nos da sus campos, incluyendo ancho, alto,tamaño,etc ademas nos da su mapa de pixeles a traves de una variable puntero a puntero y cada puntero de esta representa un pixel con sus 3 colores de 8 bits cada uno sobre los cuales se operara para aplicar los filtros correspondientes dependiendo de la hubiacacion de cada uno. Se realizan analisis de la ejecucion tanto de manera secuencial como de manera paralela.

Funciones del Producto

A partir de sus parametros de entrada ($R, l, k, nThreads$), el producto contiene las siguientes funciones:

- lectura de un archivo bmp de 24 bits sin paleta de colores.
- Creacion de estructura de imagen de salida
- Filtro lineal dentro de la region definida dentro del radio r de el archivo de salida, bajo los parametros k y l .
- Filtro convolucional en 2D fuera del radio R con un kernel de dimension 41×41 (se omiten los bordes del archivo para esta operación).
- Guardado de la imagen procesada en un nuevo archivo bmp.

Características de los Usuarios

El usuario solo debe tener el directorio del proyecto y contar con la librería simple_bmp, ademas se espera que los usuarios ingresen los 4 parametros necesarios en caso contrario se les mostrar un mensaje de error.

Restricciones

De momento solo es posible el uso de imagenes de 24 bits sin paleta de colores, ademas el path a la imagen estara definido como una constante en el codigo.

Suposiciones y Dependencias

- En principio el binario se creo y probó usando gcc se espera que el uso se dé en un dispositivo con este compilador
- Se supone que el producto se ejecutara en un sistema con alguna distribución de linux
- deben estar instaladas las bibliotecas math y openmp, caso contrario no se podrá compilar.

Requisitos Futuros

A futuro se espera:

1. Poder agregar como parámetro el path de la imagen.
2. Poder versatilizar el producto modificando la biblioteca simple_bmp para que acepte imágenes con paleta de colores.

Requisitos Específicos

Interfaces Externas

La ejecución y posterior lectura de los datos se hace a través de la terminal de linux por ende se espera que el usuario cuente tanto con monitor como con teclado.

Requisitos de Rendimiento

Se espera aprovechar la flexibilidad de openMp ya que este proyecto es para comprobar los cambios de rendimiento usando paralelismo, se harán ejecuciones graduales cambiando el número de threads del producto consiguiendo así el objetivo deseado.

Restricciones de Diseño

- Se debe realizar primero una implementacion secuencial y una vez realizada pasarla a paralela
- se debe poder ingresar el radio separador por parametro
- debe incluirse mecanismo de control y manejo de errores.
- Se debe realizar la ejecucion de ambos tipos 30 veces para obtener estadistica suficiente sobre los tiempos de ejecucion
- se deben colocar tablas y graficos sobre el analisis de los datos
- se debe usar alguna herramienta de profiling.

Atributos del Sistema

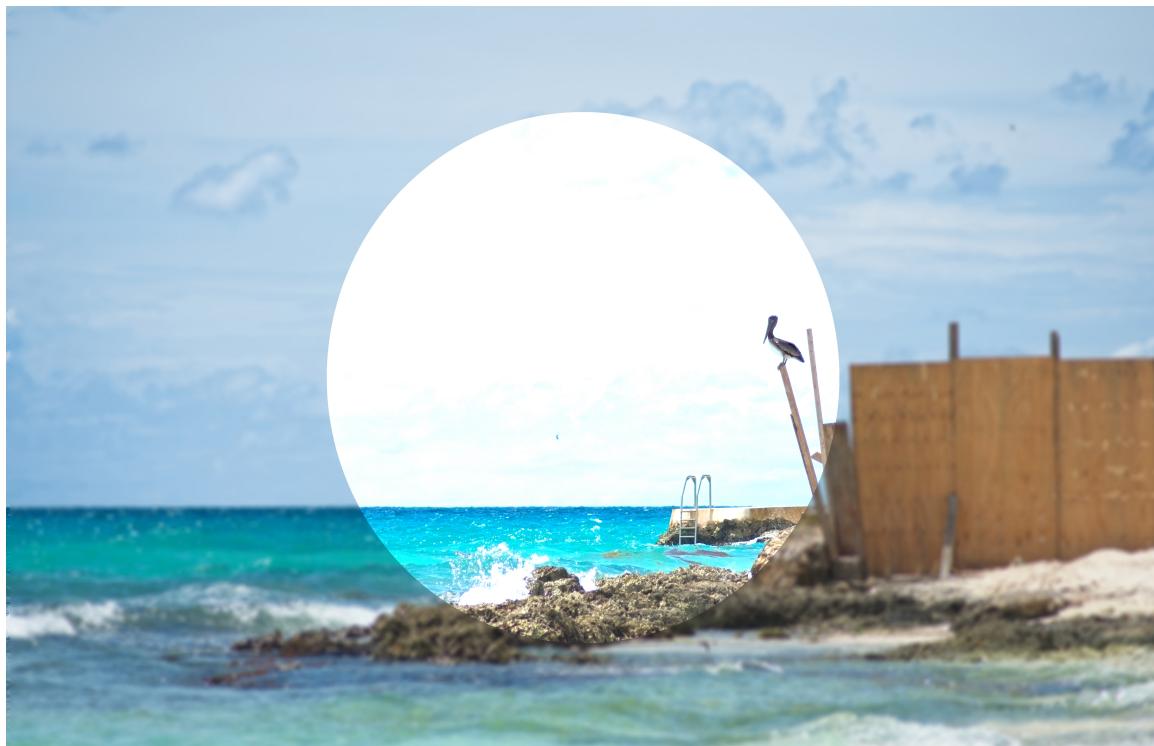
Otros Requisitos

Diseño de solución

Se inicia la solucion haciendo la lectura dos veces de la imagen origen(base.bmp) se lee dos veces para guardar la misma en dos estructuras del tipo sbmp_image, luego sobre una estructura escribiremos los datos filtrados y de la otra leeremos los datos originales. La imagen a procesar es la siguiente :



sus dimensiones son: 6000X4000 px (24 millones de pixeles) se espera que el resultado sea similar a la siguiente imagen :



una vez que se tiene la imagen dentro del contexto de la aplicación se procedio

a recorrer la imagen por filas es decis a lo ancho una fila a la vez siendo el limite de filas el alto de la imagen y el limite de columnas el ancho.

Habiendo establecido el centro de la imagen se procedio a dividir si el la cordenada actual estaba dentro o no del radio establecido por parametro y una vez hecho se procedio a aplicar el filtro correspondiente sobre la imagen de salida.

Para los algoritmos de los filtros se tuvieron en cuenta las funciones matematicas

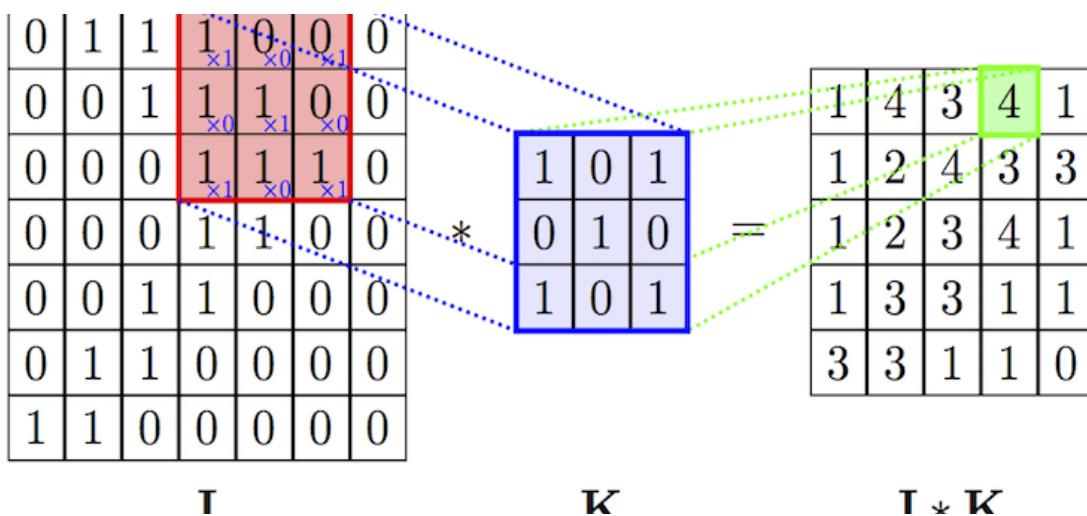
$$p(x,y) = p(x,y) * K + L$$

para el filtro lineal, en donde $p(x,y)$ se refiere al un color determinado del pixel actual y K y L han sido definidos previamente.

Mientras que para la convolucion en dos dimensiones se uso:

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

donde $w(s,t)$ es el kernel previamente mencionado que es una matriz simetrica 41×41 , la convolucion seria la accion de realizar la multiplicacion y acumulacion de los valores simultaneos de dos matrices de entrada una de las cuales deberia invertirse en nuestro caso la matriz es simetrica asi que ese ultimo paso es innecesario. Una explicacion grafica de la convolucion seria la siguiente:



Implementación y Resultados

Para la implementacion el filtro lineal se realizo literalmente como la formula arriba expresada, una vez por cada color del pixel, luego se limitan los valores para que estos no puedan ser mayores de 255 ni menores que cero(debidos a las propias limitaciones de los colores definidos en la estructura sbmp_rawdata).

Para la implementacion de la convolucion sin embargo se hizo el uso de dos ciclos for para poder realizar la sumatoria la operando con una matriz obtenida de un pedazo de la imagen el cual se calcula tomando el valor medio de la dimension del kernel y estableciendo limites para ambos lados y en ambas direcciones del punto

medio, que es tambien donde se guardara el resultado, si alguno de estos limites salen del los limites de la imagen entonces el procesamiento no se realiza, en caso contrario se realiza sobre cada color y se guarda en resultado en el punto medio en la nueva imagen.

Una vez terminado el procesamiento se guarda el resultado en una nueva imagen, y se imprime en un prompt el tiempo de procesamiento de la region paralela siendo para el caso monocore.

A continuacion se procede a realizar el analisis de los tiempos para los cuatro diferentes tipos de schedulers que ofrece open mp se observa que para 1 solo thread estos tiempos son siempre iguales ya que al usar un solo core la planificacion impacta. Para todos los casos se realizaron 30 pruebas tanto en el cluster como localmente.

Para todas las pruebas se usaron los parametros

1. radio=1200
2. l=20
3. k=2

Ademas se utilizo el parametro O3 para optimizacion.

Las siguientes tablas muestran el tiempo medido en segundos, tratandose del tiempo de ejecucion de la zona paralela.

Scheduler static

- Cluster

metrica/threads	1	2	4	8	16	32	64
promedio	42,2	22,0	13,9	7,6	6,1	5,7	5,3
desvio	0,3	0,2	0,1	0,0	0,1	0,8	0,0
max	43,6	22,4	14,1	7,7	6,3	8,9	5,5
min	41,9	21,7	13,6	7,6	5,9	5,4	5,3
speedup	1,0	1,9	3,0	5,5	6,9	7,4	7,9

- Local

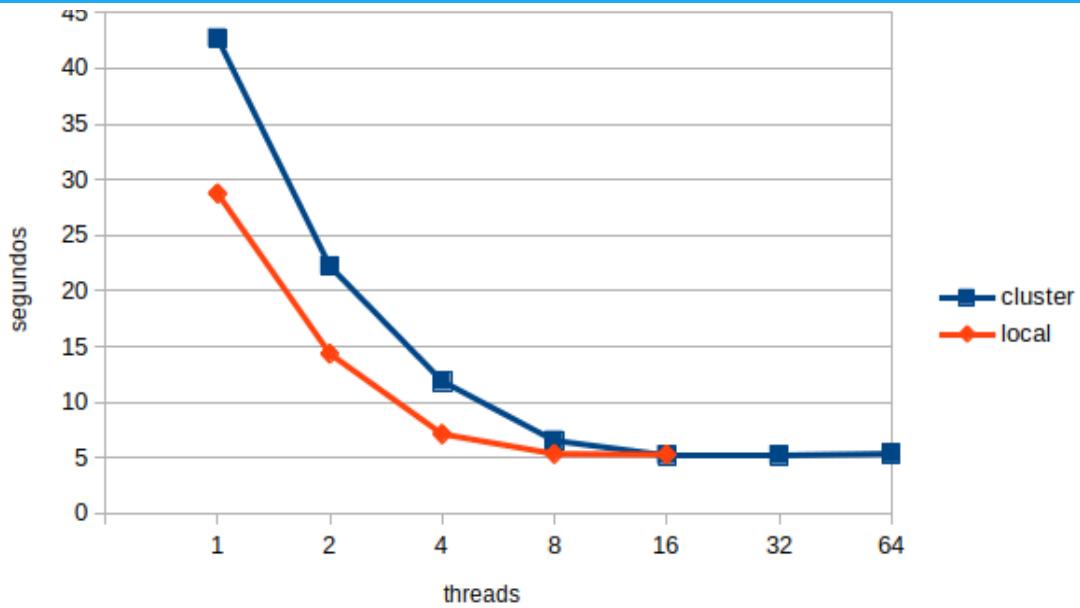
metrica/threads	1	2	4	8	16
promedio	28,3	14,7	8,7	5,9	5,5
desvio	1,3	1,1	0,6	0,3	0,2
max	32,0	16,8	9,9	6,4	6,2
min	27,4	14,0	8,3	5,5	5,3
speedup	1,0	2,0	3,3	5,0	5,2

Para este caso se vio que si bien en cluster a mayor cantidad de hilos se logro un mayor speed up en general los tiempos de la maquina local fueron menores en estas pruebas no se utilizo el parametro `mach=native` lo que pudo haber incrementado la performance en ambos casos, ademas se supone que esta diferencia puede tener que ver con las diferencias en la frecuencia de trabajo de ambos procesadores.

Se nota que a pesar de correr la ejecucion con cantidades mayores de hilos a los cores fisicos de los procesadores el speedup siguio aumentando marginalmente luego de sobrepassar dicho limite.

Aca se nota una mejora general de los tiempo tanto localmente como en el cluster si, dandose el caso de que localmente se lograron mejores tiempos que en el cluster con la misma cantidad de threads.

Este caso parece escalar mejor que el anterior ya que escala perfectamente hasta cierto punto para luego de pasado el limite de cores tener un escalado menor que el ideal



como se puede apreciar en el grafico en este caso los tiempos de ejecucion fueron mas similares en ambas plataformasya que las curvas tienen menos distancia entre si y hay puntos de entrecruzamiento antes que en el caso anterior.

Scheduler runtime

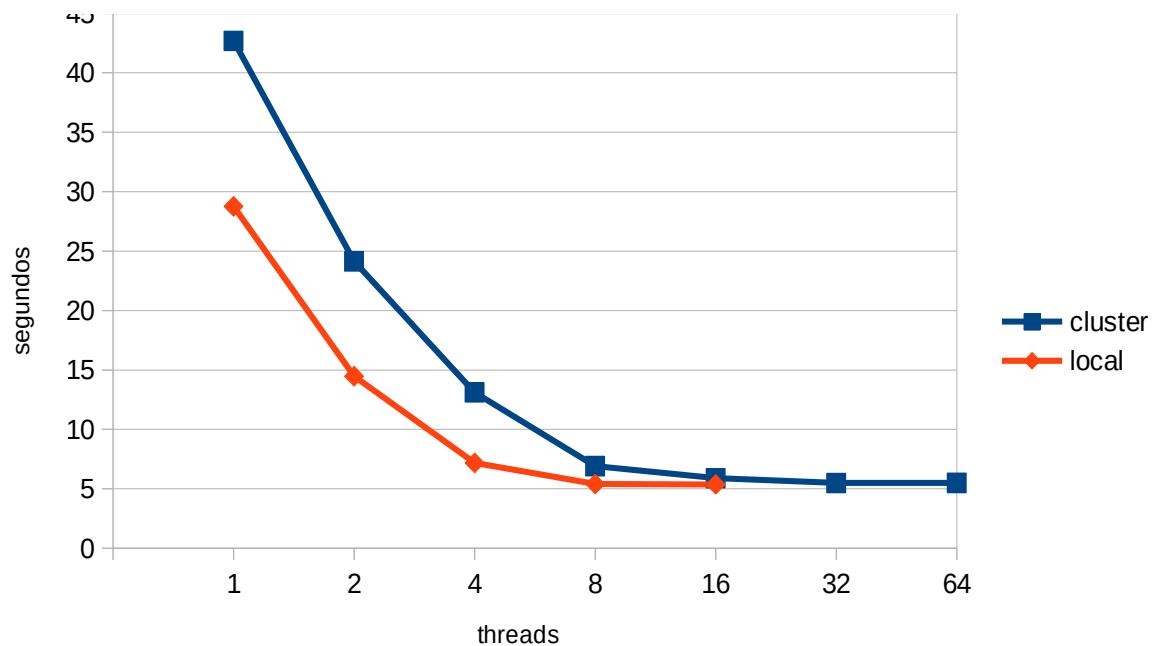
- Cluster

metrica/thread	1,0	2,0	4,0	8,0	16,0	32,0	64,0
promedio	42,7	24,1	13,1	6,9	5,9	5,5	5,5
desvio estandar	0,8	0,2	0,1	0,0	0,2	0,0	0,0
maximo	45,3	24,5	13,3	7,0	6,1	5,6	5,6
min	41,6	23,7	12,8	6,9	5,5	5,5	5,5
speedup	1,0	1,8	3,3	6,2	7,2	7,8	7,8

- local

metrica/threads	1	2	4	8	16
promedio	28,758768433	14,467824333	7,1768661333	5,3922372333	5,3704377333
desvio estandar	5,2460158376	2,2975850096	0,5809378137	0,4876045225	1,9130641448
maximo	32,794985	15,171769	7,32049	5,778883	5,599681
minimo	27,871645	13,949473	6,996219	5,180681	5,119517
speedup	1,0	2,0	4,0	5,3	5,4

este caso fue el de peor rendimiento en el cluster en cuanto a tiempo de ejecucion pero a la vez maneja speedups similares a los del caso static, pero dio resultados similares localmente, siendo tanto los tiempo como el speedup similar al caso dinamico.



En el grafico se nota que de igual manera las curvas son parecidas al caso dinamico pese a que hasta 8 los valores de tiempo difieren mas que en este ultimo.

Scheduler guided

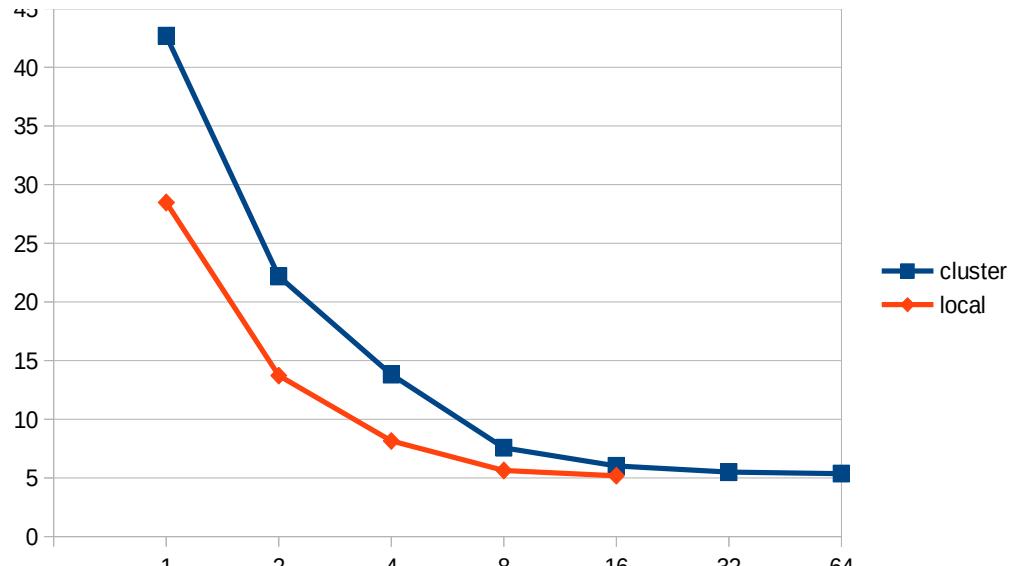
- Cluster

metrica/threads	1,0	2,0	4,0	8,0	16,0	32,0	64,0
promedio	42,7	22,2	13,8	7,6	6,0	5,5	5,4
desvio estandar	0,8	0,4	0,1	0,0	0,1	0,1	0,1
maximo	45,3	23,6	14,1	7,7	6,2	5,7	5,5
min	41,6	21,5	13,6	7,5	5,6	5,4	5,3
speedup	1,0	1,9	3,1	5,6	7,1	7,8	7,9

- Local

metrica/threads	1,0	2,0	4,0	8,0	16,0
promedio	28,5	13,7	8,1	5,6	5,2
desvio estandar	5,3	2,1	0,7	0,6	1,9
maximo	32,3	14,6	8,3	6,4	5,4
minimo	26,7	13,6	8,1	4,9	5,0
speedup	1,0	2,1	3,5	5,1	5,5

en este caso se obtienen los segundos mejores resultados despues de en el caso dinamico tanto para la pc local y el cluster



Profilers

Los profilers son herramientas de análisis de rendimiento de software usadas en gran parte para la optimización del mismo. Los mismos pueden medir por ejemplo

el tiempo o memoria relacionados a un programa, la cantidad de llamadas a una función o instrucción en particular,etc.

Algunos ejemplos

- **Perf**

Perf es una herramienta de profiling para linux 2.6+ que se abstracta de las diferencias de hardware en las medidas de rendimiento de linux y que presenta una interfaz por línea de comandos simple.

Modo de uso: `perf [--version] [--help] COMMAND [ARGS]`

- **Gprof**

gprof es una herramienta de análisis de rendimiento para aplicaciones de unix usa un híbrido de instrumentación y sampling y fue creado a partir de prof.

modo de uso: `gprof options [executable-file [profile-data-files...]] [> outfile]`

- **Valgrind**

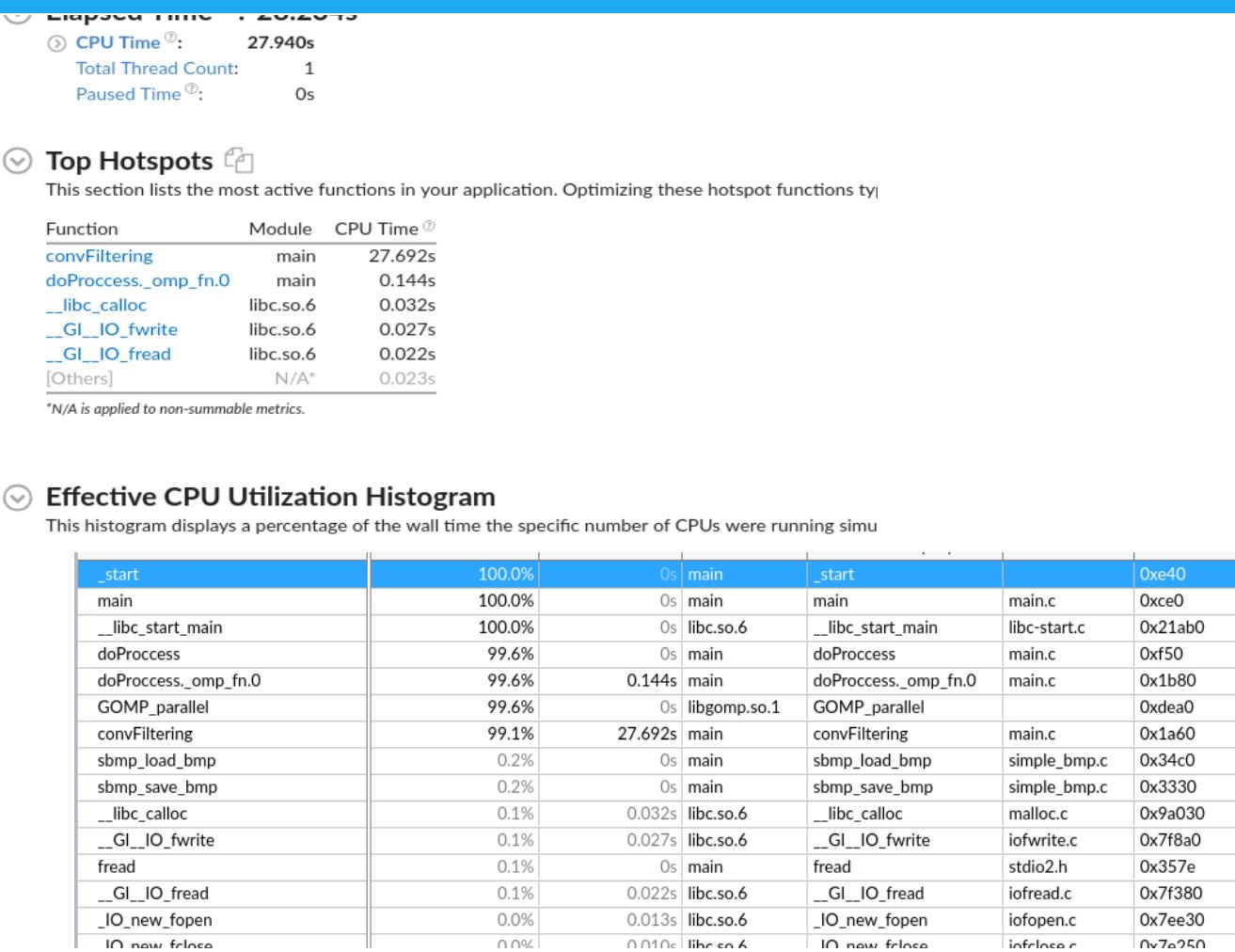
Valgrind es un conjunto de herramientas para el análisis de software entre los se destacan:

1. detector de errores de memoria
 2. detector de errores en dos threads
 3. profiler de cache y branch-prediction
 4. profiler de cache y branch prediction generador de grafo de llamadas a funciones
 5. heap profiler
- Intel Vtune
Al igual que valgrind es una herramienta de varias funcionalidades para el profiling de programas entre ellas
 1. análisis de hotspots(puntos de mucho uso de tiempo de ejecución)
 2. análisis de threading
 3. análisis de performance hpc

Uso de profiler

En el caso de este proyecto en particular se uso intel vtune, se realizo el análisis de hotspots para 1 y 6 threads con scheduler dinamico , y el análisis de threading para 6 threads.

- Análisis de hotspots 1 thread



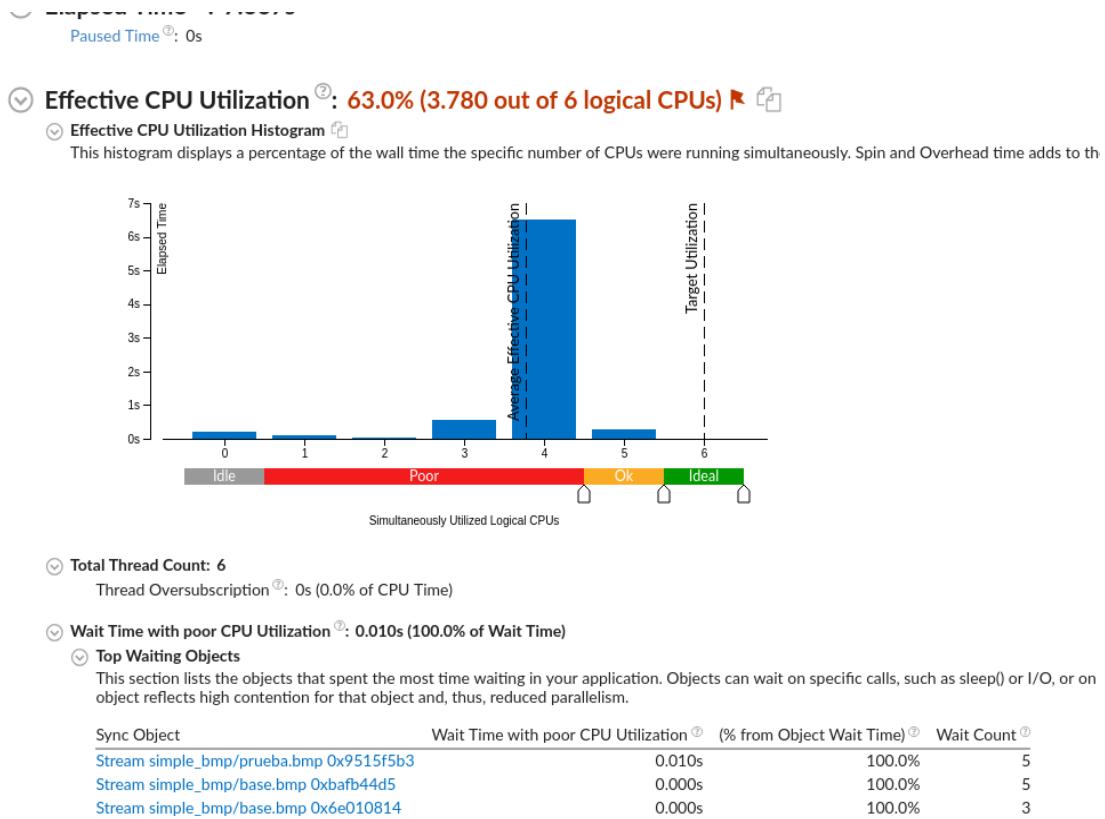
como era de esperarse la funcion que mas tiempo ocupa es la funcion de convolucion ocupando un 99,1% del tiempo de procesamiento en su tiempo total de ejecucion ,mientras que la funcion do process que se encarga de llamar a los respectivos filtros tiene muy poco tiempo de ejecucion neta en comparacion.

como era de esperarse el software marca un rendimiento pobre debido a la baja utilizacion de los procesadores disponibles.

- Análisis de hotspots 6 threads

En este caso se puede apreciar que el tiempo de cpu es practicamente el mismo pero al estar repartido entre todos los procesadores se reduce a aproximadamente un sexto del tiempo total se ve tambien que, como cabria esperar,no varia el tiempo de ejecucion de las funciones y que si bien la mayoria del tiempo estan trabajando los 6 procesadores existen momentos en que no lo que baja la media de uso efectivo del cpu.

- Análisis de threading con 6 hilos



Este análisis explica que en la ejecución se usan en promedio 3,8 núcleos al mismo tiempo. Este análisis considera la parte serial del problema que implica la lectura y escritura de archivos durante la cual hay un solo procesador habilitado, lo cual reduce el promedio de utilización en gran medida comparado con el gráfico de la sección anterior donde el enfoque estaba mayormente en la zona paralela.

Conclusiones

Se logro el uso correcto de las herramientas otorgadas tanto la librería simple_bmp como las librerías de openMp logrando así cumplir el objetivo de la consigna, logrando el filtrado de la imagen y luego el paralelismo sobre esa acción.

Se pudieron comprobar las ventajas del paralelismo ,aunque este no esté implementado a la perfección, viendo la gran diferencia de tiempo de ejecución que resultaba de su uso.

A su vez se consolidó la experiencia del uso de ssh para acceder a equipos remotos.

Se adquirió conocimiento y experiencia sobre el uso de profilers y banderas para la optimización del código.