1. Análisis del programa:

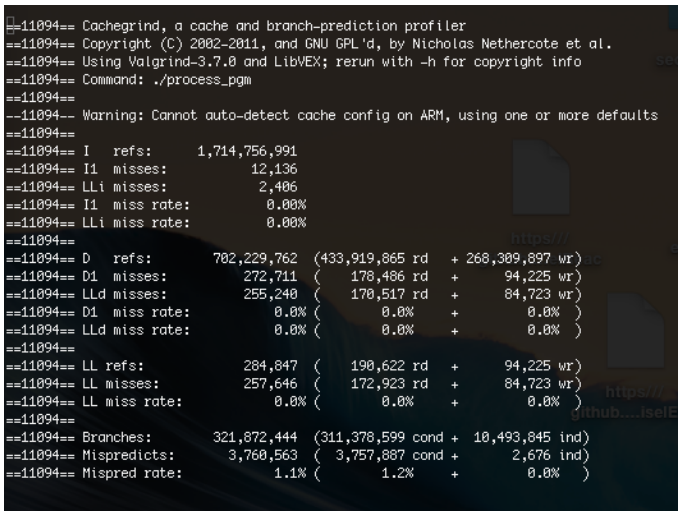
¿Dónde se accede a memoria?

Usaremos la herramienta Cachegrind para ver el número de lecturas y escrituras que realiza cada línea del código a los niveles de cache. El mayor número de instrucciones se leen y son ejecutadas en los bucles while onde se leen y escriben los ficheros imagen .pgm.
Además, con la herramienta Massif, podemos ver la ocupación de la pila así como los porcentajes de las funciones que acceden a ella.
En este caso, las funciones que más acceden a memoria son *load_from_file() y save_image_from()* que a su vez invocan a la función *fopen().*

¿A qué zona de memoria se accede? ¿En qué orden?

En primer lugar, se guardan en memoria las variables globales y las constantes.
Cuando se ejecuta el programa, se guardan los parámetros de las funciones y las variables locales en la pila. Después se guardan en la pila todos los accesos a memoria que sean necesitados y finalmente la pila deberá ser liberada.

¿Dónde están los fallos de caché?



Al ejecutar Cachegrind obtenemos un fichero de salida .output en el que vemos las referencias a cada tipo de caché con sus tasas de fallos y el número de saltos con las predicciones fallidas.

En este caso observamos que tanto para la caché de datos como para la caché de instrucciones, tanto de primer nivel como de último nivel,en todos los casos, la tasa de fallos es despreciable frente a las referencias hechas. Cabe mencionar que existe un gran número de fallos en datos, pero aún así sigue siendo despreciable frente al número de referencias

Por otro lado, usando cg_annotate para analizar el fichero .output de Cachegrind obtenemos un fichero .cachegrind que muestra la información de fallos de caché encada línea. Podemos observar un mayor número de fallos en caché de datos al ejecutarse los bucles de las funciones que tienen que acceder al array que contiene la imagen, por ejemplo "*invert_colours()*", "*apply_threshold()*", "*horizontal_edge_detect()*", "*vertical_edge_detect()*" y "*fprintf()*".

¿Se puede reducir la memoria caché de primer nivel? ¿Hasta dónde?

Sí, es posible, ya que tenemos una tasa de fallos despreciable, 0%. Esto supone que estamos desperdiciando recursos por lo que podemos reducir el tamaño de caché de primer nivel hasta obtener una tasa de fallos próxima al 1-2%

2. Sin cambiar el programa:

Optimizar la memoria, minima memoria caché necesaria (sin pérdida significativa de rendimiento), mínima RAM necesaria.

Se puede optimizar la memoria usando las opciones de optimización del compilador gcc, es decir, usando la variable –O al compilar.
Existen diferentes niveles de optimización: -O0, -O1, -O2, -O3, -Os, -Og y -Ofast. Se debe utilizar solo uno de ellos en /etc/portage/make.conf.

-**O0**: Este nivel (que consiste en la letra "O" seguida de un cero) desconecta por completo la optimización y es el predeterminado si no se especifica ningún nivel -O enCFLAGS o CXXFLAGS. El código no se optimizará. Esto, normalmente, no es lo que se desea.

-**O1**: El nivel de optimización más básico. El compilador intentará producir un código rápido y pequeño sin tomar mucho tiempo de compilación. Conseguirá realizar correctamente el trabajo.

-**O2**: Es el nivel recomendado de optimización. Activará algunas opciones añadidas a las que se activan con -O1, el compilador intentará aumentar el rendimiento del código sin comprometer el tamaño y sin tomar mucho más tiempo de compilación.

-**O3**: El nivel más alto de optimización posible. Activa optimizaciones que son caras en términos de tiempo de compilación y uso de memoria, no garantiza una forma de mejorar el rendimiento y, en muchos casos puede ralentizar un sistema debido al uso de binarios de gran tamaño y mucho uso de la memoria. También puede romper algunos paquetes. No se recomienda su uso.

-**Os**: Optimizará el tamaño del código. Activa todas las opciones de -O2 que no incrementan el tamaño del código generado. Es útil para máquinas con capacidad limitada de disco o con CPUs que tienen poca caché.

-**Og**: En GCC 4.8 aparece un nuevo nivel del optimización general. Trata de solucionar la necesidad de realizar compilaciones más rápidas y obtener una experiencia superior en la depuración a la vez que ofrece un nivel razonable de rendimiento en la ejecución. La experiencia global en el desarrollo debería ser mejor que para el nivel de optimización. Deshabilita optimizaciones que podrían interferir con la depuración.

-**Ofast**: Nuevo en GCC 4.7. Consiste en el ajuste -O3 más las opciones -ffast-math, -fno-protect-parens y -fstack-arrays. Esta opción rompe el cumplimiento de estándares estrictos y no se recomienda su utilización.

Proponer cambios arquitecturales para mejorar rendimiento/consumo. Estructura de jerarquía de memoria.

Habría que adaptar el tamaño de la caché de instrucciones y la de datos a los del punto 1. Además podríamos suprimir la caché de último nivel al tener un mayor tasa de fallos. Con esto reducimos el tiempo medio de acceso a memoria y por tanto un menor consumo.

3. Cambiar el programa para:

Mejorar la tasa de aciertos y/o reducir la memoria caché. Estimar la mejora en tamaño de memoria. Estimar el impacto en rendimiento.
Reducir el consumo. Estimar la mejora en consumo. Estimar impacto en rendimiento.

Los anteriores apartados nos indican que debemos reducir las memorias caché para reducir la tasa de fallos.La mayor tasa de fallos se produce en las funciones de tratamiento de la imagen por lo que tendríamos que simplificar estos procesos.

Una posible solución es reducir el número de veces que se carga la imagen, es decir cargarla una vez en lugar de cuatro, como lo hace actualmente ,y realizar cuatro procesados simultáneos.

Otra opción es procesar la imagen por zonas, usando bucles con menos iteraciones

También se podría mejorar los procesos de lectura y escritura. En vez de leer filas de una sola columna habría que poder leer filas de varias columas. De esta manera aumentamos la localidad y mejoramos el rendimiento del programa

Por otro lado podemos usar la herramienta eCACTI con el que podremos estimar tanto la disipación de potencia estática como dinámica en cachés al realizar lecturas y escrituras.
Además podemos estimar el tiempo de acceso de lectura a caché, su área y encontrar
la configuración óptima. De esta manera podremos ver las diferencias de eficiencia y de uso de recursos así como las configuraciones óptimas antes y después de realizar cambios de optimización en el código