Parámetros de compilación

Hay diferentes tipos de optimizaciones, en general mejoran el espacio y tiempo, sacrificando otros aspectos. Algunas optimizaciones mejoran ambos aspectos a expensas de generar un fichero binario más largo o reduciendo el binario obtiendo un mayor tiempo de ejecución.

Por ejemplo, el espacio se puede optimizar "empaquetando" juntos los datos y funciones, y el tiempo se puede optimizar alineando los datos para satisfacer los requisitos de la arquitectura y hacer un mejor uso de la caché, pero se aumenta el espacio ocupado.

Otros ejemplos en los que se sacrifica el espacio en contra de un mejor tiempo son:

**Loop unrolling**: usado en bucles pequeños, transformando estos a instrucciones individuales reduciendo / eliminando las comprobaciones en el bucle.

**Function inlining**: en lugar de hacer una llamada a una función se "pega" el código de esta donde se haría la llamada, evitando saltos innecesarios en la pila.

**Code reordening:** reordenar el código para ejecutar antes el código no dependiente de cálculos anteriores.

En cuanto al compilador g++ ofrece cuatro niveles de optimización:

**O0**: por defecto, reduce el tiempo de compilación, no da problemas al depurar.

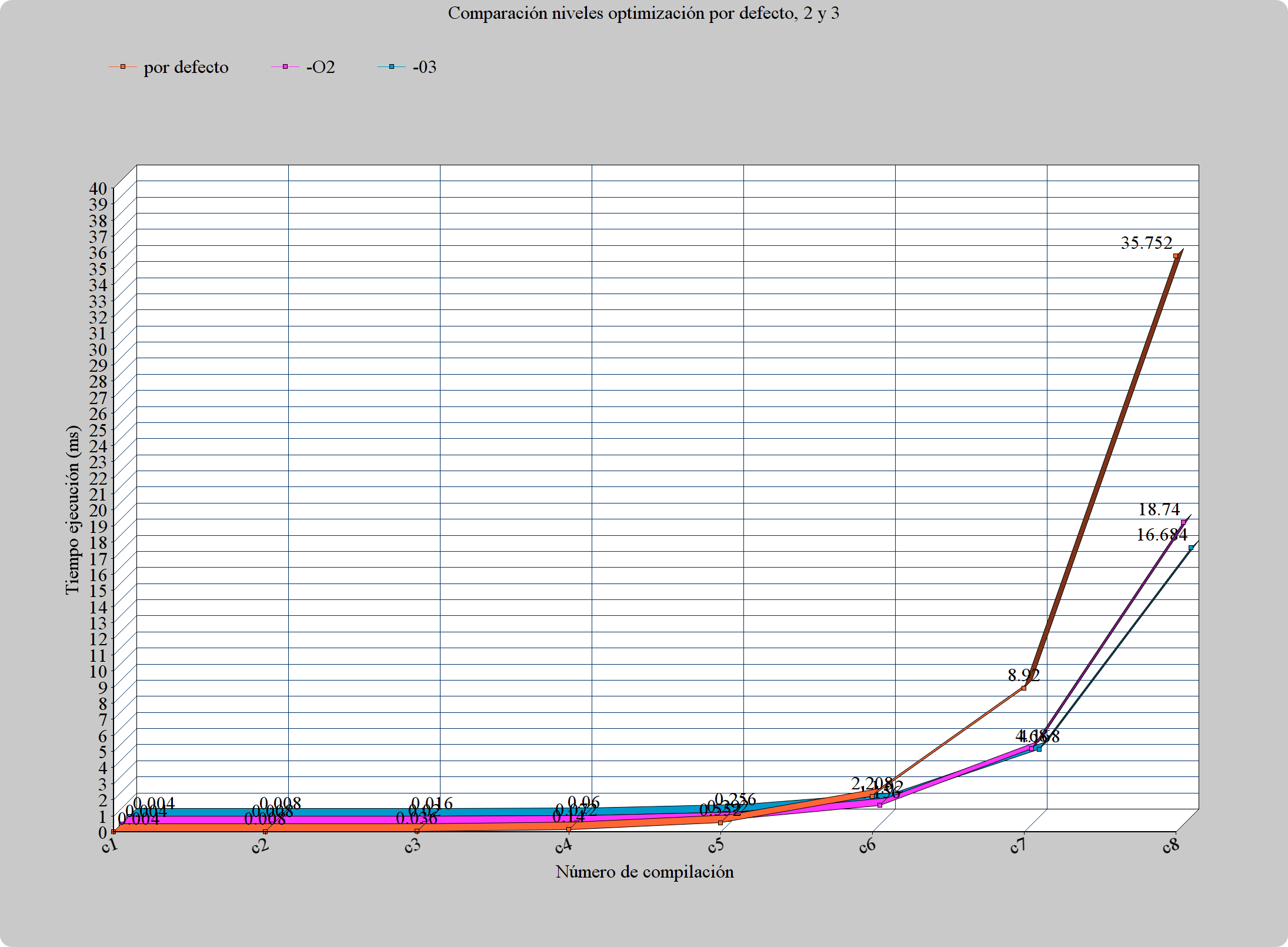
**O1**: el compilador intenta reducir el tamaño del código, con pocas optimizaciones que mejoren el tiempo de ejecución

**O2:** Optimiza mucho más, mejorando tanto el espacio como el tiempo, a este se le incluyen odas las del nivel O1 además de otras nuevas.

**O3:** mejora aun más la optimización.

Cada uno de los niveles descritos, implican una bateria de flags tras de sí, siendo en general los de nivel superior la suma de los anteriores más otros nuevos.

En la siguiente gràfica se pueden ver los tiempos de ejecución del programa presentado en esta documentación, ejecutándolo en las máquinas de la eps:



Como se puede observar, en las tres línias, en las sucesivas compilaciones el tiempo de ejecución aumenta en gran cantidad tras llegar a la sexta compilación.

Además es notable la disminución del tiempo de ejecución en la última compilación usando los niveles de compilación por defecto del compilador g++.

Por último voy a explicar los unos de los flags escogidos, que major relación tienen con nuestro programa y una gràfica comparando con los nuevos resultados:

* **fipa-cp:** Esta optimización analiza el programa para determinar cuando los valores pasados a las funciones son constantes y luego se optimiza en consecuencia. Esta optimización puede mejorar sustancialmente el rendimiento cuando se tienen constantes pasadas a funciones.

Este flag serà de utilidad en la función de interpalción, a causa de la constantes llamadas a esta y la cantidad de parámetros pasados.

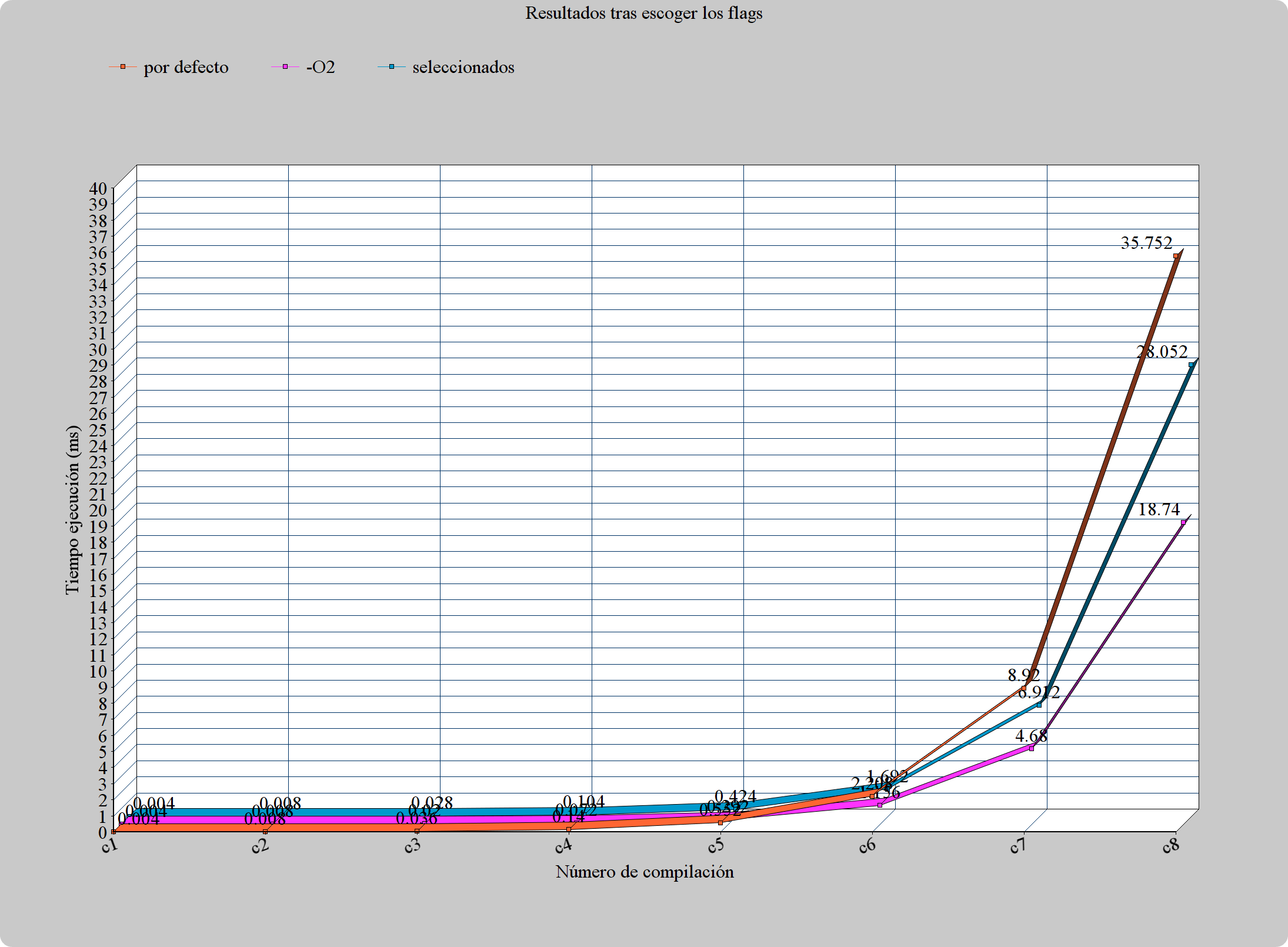
* **fpartial-inlining**: Esta optimización hace parte de la funciones en linea.

Este serà aplicable tanto a la función de interpolación, como la usada para calcular el tiempo de ejecución, cuyos beneficiós han sido explicados anteriormente.

* **fipa-icf:** realiza un plegado de codigo repetido en funciones y variables. Esta optimizacion reduce el tamaño del codigo.

En la función principal, donde se recorre la información de la imagen y se hacen las llamadas a la función de interpalación, se realizan muchas veces la misma operación para cada uno de las iteraciones de los bucles.

Los resultados, comparados con los niveles de optimización de g++, tras usar los flags explicados son:



Como se puede observar, hay una disminución en casi 8 segundos tan solo usando los 3 flags comentados, no siendo equiparable al nivel 2, pero si es notòria teniendo en cuenta que este usa una gran cantidad de optimizaciones.