Práctica 2

Equipo

La máquina en la que se ejecutan las siguientes pruebas consta de un procesador **Intel Core 2 Duo P8600** a 2,4 GHz con **3 MB de caché L2**. También cuenta con **8GB de DDR3 a 1066MHz**. El Sistema Operativo es **Arch Linux**.

Ejercicio 11

En este ejercicio se usa la técnica de optimización **inlining**, que consiste en sustituir la llamada a una función por el propio cuerpo de la función, y así ahorrarse el tiempo de llamada a la rutina. En este caso tenemos una función de suma:

```
for(j=0; j<ITER; j++)
      for(i=0; i<N; i++)
            add(x[i],y[i],&z[i]); /* Llamada a la función */
```

Que sustituimos por el cuerpo de la misma:

```
for(j=0; j<ITER; j++)
      for(i=0; i<N; i++)
            z[i]=x[i]*y[i]; /* Cuerpo de la función */
```

Medimos diferentes ejecuciones del código para 100 iteraciones y diferentes valores de N (1.000, 10.000, 100.000 y 1.000.000). También ejecutamos con diferentes opciones de optimización del compilador (sin optimización, O0, O1, O2 y O3).

Hay que destacar que estos resultados se obtuvieron teniendo en cuenta que la **caché podría afectar a las mediciones** consecutivas, así que antes de medir los tiempos nos aseguramos de que esta no afecte, **añadiendo el código a medir una segunda vez** (justo antes de medirlo) para que en el caso de que la caché afecte a las primeras iteraciones ocurra en el primer código (que no medimos) y no en el segundo (que sí medimos).

A continuación se presentan los **resultados** de las mediciones tanto para el código original como para el optimizado con inlining, así como para los diferentes valores de N.

Resultados sin inlining

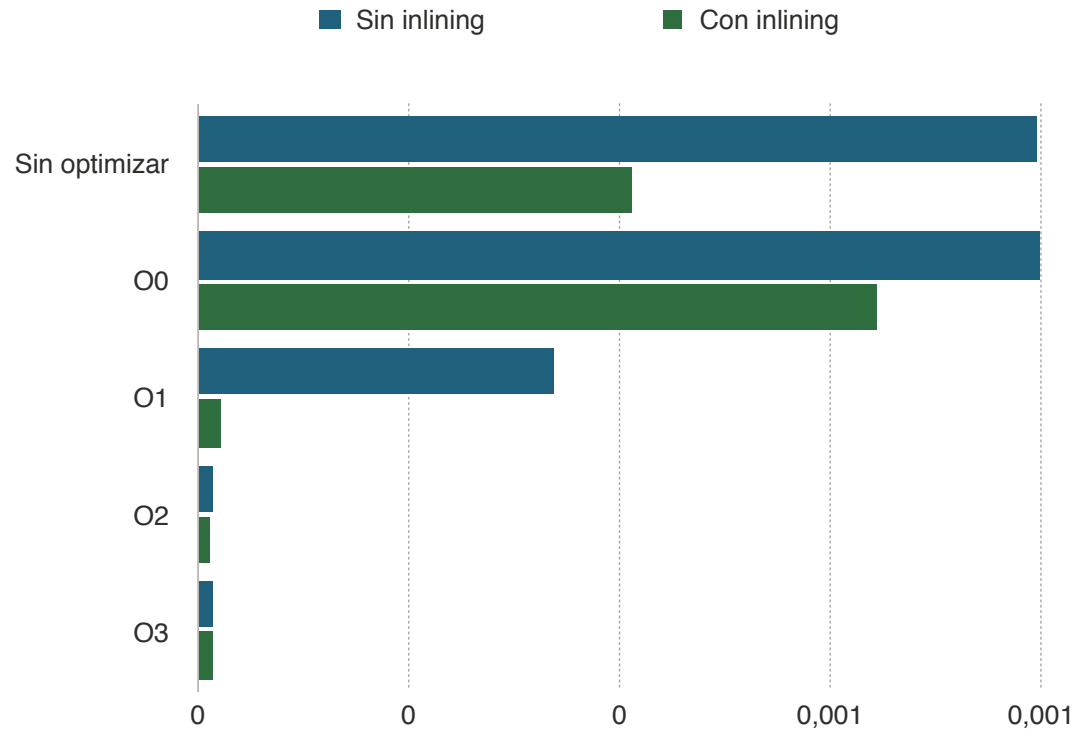
	Sin optim.	O0	O1	O2	O3
N=1000	0,000697	0,000699	0,000296	0,000013	0,000013
N=10000	0,00793	0,010221	0,00302	0,000318	0,000295
N=100000	0,074951	0,073485	0,031397	0,003033	0,003151
N=1000000	0,815885	0,758727	0,331522	0,169363	0,162531

Resultados con inlining

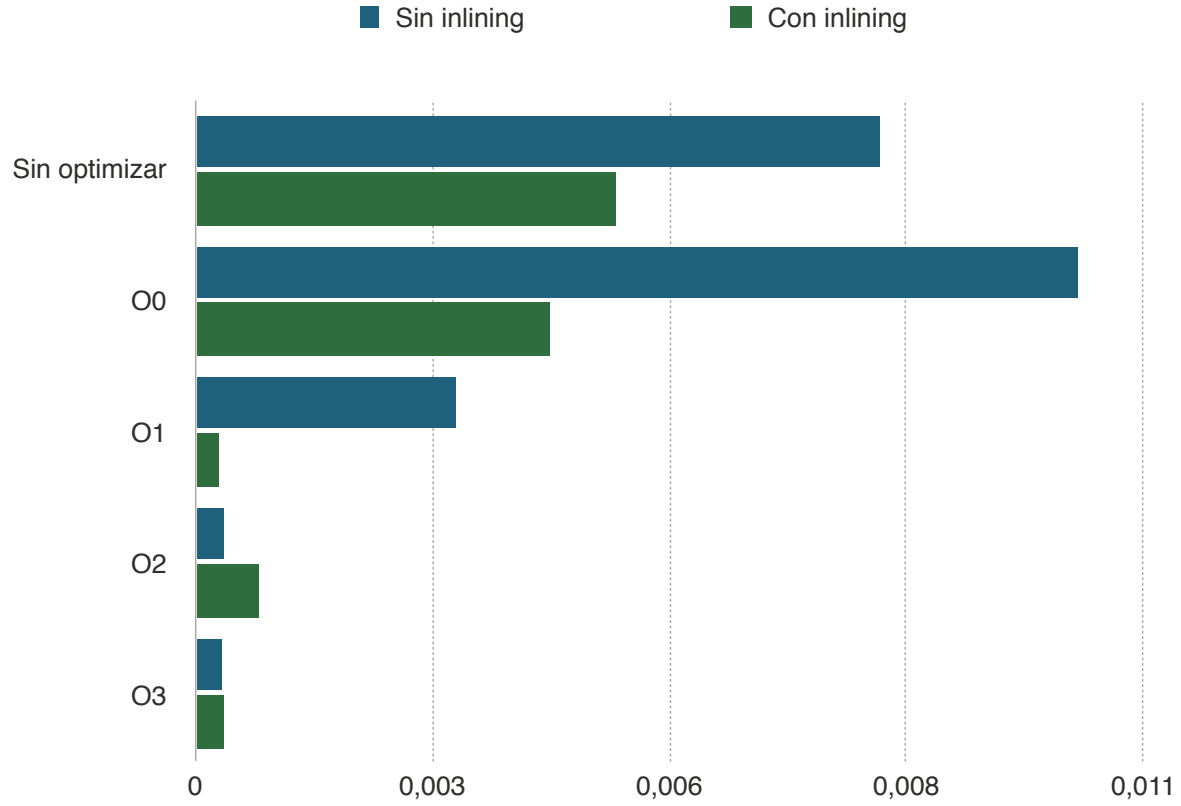
	Sin optim.	O0	O1	O2	O3
N=1000	0,000361	0,000563	0,000018	0,000011	0,000012
N=10000	0,004884	0,004117	0,000268	0,000738	0,000343
N=100000	0,045852	0,047693	0,002955	0,003137	0,002801
N=1000000	0,494612	0,485808	0,159465	0,160878	0,174938

A continuación presentamos las gráficas de **comparación** entre ambos métodos (junto con las diferentes optimizaciones) para los diferentes valores de N

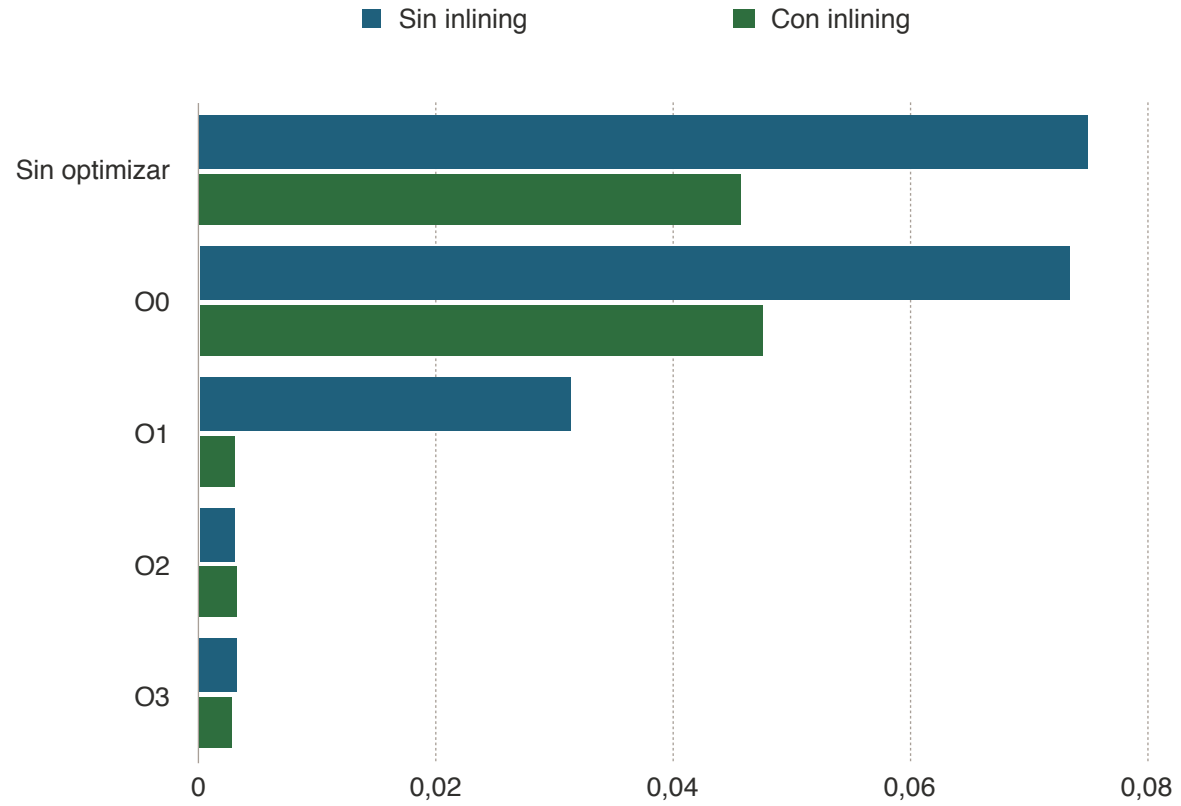
Comparación N = 1.000



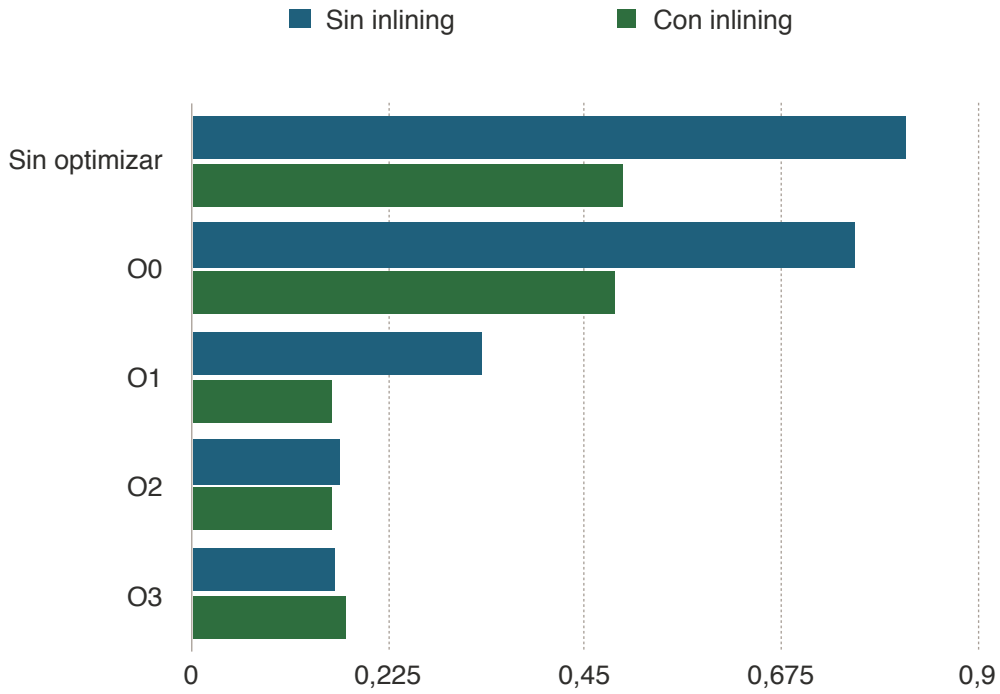
Comparación N = 10.000



Comparación N = 100.000



Comparación N = 1.000.000



En primer lugar podemos observar que **la sustitución de la función por su propio cuerpo ya provoca una reducción del tiempo considerable,** sobretodo notable cuando no se realizan optimizaciones.

Esta reducción se hace más notable valores grandes de N y para situaciones en las que no se producen optimizaciones considerables (sin optimizaciones y O0) y para O1.

- **O0**: Este nivel prácticamente no optimiza el código, de hecho podemos ver que los tiempos para todos los valores de N y para ambos códigos es relativamente igual.
- **O1**: Este nivel es el más básico en optimización. El compilador intenta reducir el tamaño del código produciendo una compilación y un programa más rápidos. Con ello vemos una reducción considerable en el tiempo de ejecución. Sin embargo la optimización inline + O1 reduce muchísimo más el tiempo, llegando a niveles comparables con O2 y O3.
- **O2**: En este nivel el compilador intentará aumentar el rendimiento del código sin comprometer el tamaño y sin tomar mucho más tiempo de compilación, consiguiendo mejores tiempos que con O1. Sin embargo vemos que este nivel ya implementa por sí sólo la optimización inline, dado que los tiempos sin ella son casi iguales que con ella.
- **O3**: Este es el nivel más alto de optimización posible, arriesga tiempo de compilación y uso de memoria, aunque puede que no se requiera tal nivel para ciertos códigos, como por ejemplo el que estamos analizando.

Conclusiones

Para el código a medir vemos que no necesitamos niveles de optimización profunda como O3, sino que basta con el nivel básico de optimización (O1) más la optimización inlining. Esto es justo lo que hace O2, ya que conseguimos prácticamente los mismos tiempos en ambos códigos.