Práctica 3 – Caché Blocking

9. El siguiente ejemplo permite comprender mejor donde se encuentran los fallos de memoria caché cuando se usa la versión naïve del producto de matrices. Sea el producto de dos matrices de tamaño 3x3, donde la matriz A se encuentra almacenada a partir de la dirección 1000h, la matriz B de la dirección 2000h y la matriz C de 3000h. Además, el sistema de memoria donde se ejecuta el código tiene una memoria cache para datos, con 4 bloques de 2 palabras cada uno de ellos, y organización asociativa por conjuntos de 2 vías. Contesta las siguientes preguntas:

a) Obtén la traza de referencias de memoria generadas al ejecutar el código usando el Listado 2, considerando que los accesos de escritura se comportan igual que los de lectura, y completa la siguiente tabla con la evolución de la memoria caché (añade todas las líneas que sean necesarias):

REFERENCIAS	DIRECCION	VIA	CONJUNTO	¿ACIERTO?
C [0]	3000h	0	0	NO
A [0]	1000h	1	0	NO
B [0]	2000h	0	0	NO
A [1]	1004h	1	0	SI
B [3]	200Ch	0	1	NO
A [2]	1008h	1	1	NO
B [6]	2018h	0	1	NO
C [0]	3000h	0	0	NO
C [1]	3004h	0	0	SI
A [0]	1000h	1	0	SI
B [1]	2004h	0	0	NO
A [1]	1004h	1	0	SI
B [4]	2010h	0	0	NO
A [2]	1008h	1	1	SI
B [7]	201Ch	0	1	SI
C [1]	3004h	0	0	NO
C [2]	3008h	0	1	NO
A [0]	1000h	1	0	SI
B [2]	2008h	0	1	NO
A [1]	1004h	1	0	SI
B [5]	2014h	0	0	NO
A [2]	1008h	1	1	NO
B [8]	2020h	1	0	NO
C [2]	3008h	0	1	NO
C [3]	300Ch	0	1	SI
A [3]	100Ch	1	1	SI
B [0]	2000h	0	0	NO
A [4]	1010h	1	0	NO
B [3]	200Ch	0	1	NO
A [5]	1014h	1	0	SI
B [6]	2018h	1	1	NO
C [3]	300Ch	0	1	NO
C [4]	3010h	0	0	NO
A [3]	100Ch	0	1	NO
B [1]	2004h	1	0	NO

A [4]	1010h	0	0	NO
B [4]	2010h	1	0	NO
A [5]	1014h	0	0	SI
B [7]	201Ch	1	1	SI
C [4]	3010h	1	0	NO
C [5]	3014h	1	0	SI
A [3]	100Ch	0	1	SI
B [2]	2008h	1	1	NO
A [4]	1010h	0	0	SI
B [5]	2014h	1	0	NO
A [5]	1014h	0	0	SI
B [8]	2020h	1	0	NO
C [5]	3014h	0	0	NO
C [6]	3018h	0	1	NO
A [6]	1018h	1	1	NO
B [0]	2000h	1	0	NO
A [7]	101Ch	1	1	SI
B [3]	200Ch	0	1	NO
A [8]	1020h	0	0	NO
B [6]	2018h	1	1	NO
C [6]	3018h	0	1	NO
C [7]	301Ch	0	1	SI
A [6]	1018h	1	1	NO
B [1]	2004h	1	0	SI
A [7]	101Ch	1	1	SI
B [4]	2010h	0	0	NO
A [8]	1020h	1	0	NO
B [7]	201Ch	0	1	NO
C [7]	301Ch	1	1	NO
C [8]	3020h	0	0	NO
A [6]	1018h	0	1	NO
B [2]	2008h	1	1	NO
A [7]	101Ch	0	1	SI
B [5]	2014h	0	0	NO
A [8]	1020h	1	0	SI
B [8]	2020h	0	0	NO
C [8]	3020h	1	0	NO

b) Calcula la tasa de fallos al acceder a las tres matrices, y las tasas de fallo particulares para A, B y C. Explica razonadamente porque son distintas.

$$\textbf{Tasa de Fallos } (A,B,C) = \frac{\textbf{Número de fallos}}{\textbf{Total de Instrucciones}} = \frac{49}{72} = 0.68 * 100 = 68\%$$

$$\textbf{Tasa de Fallos } (A) = \frac{\textbf{Número de fallos}}{\textbf{Total de Instrucciones}} = \frac{11}{27} = 0.40 * 100 = 40\%$$

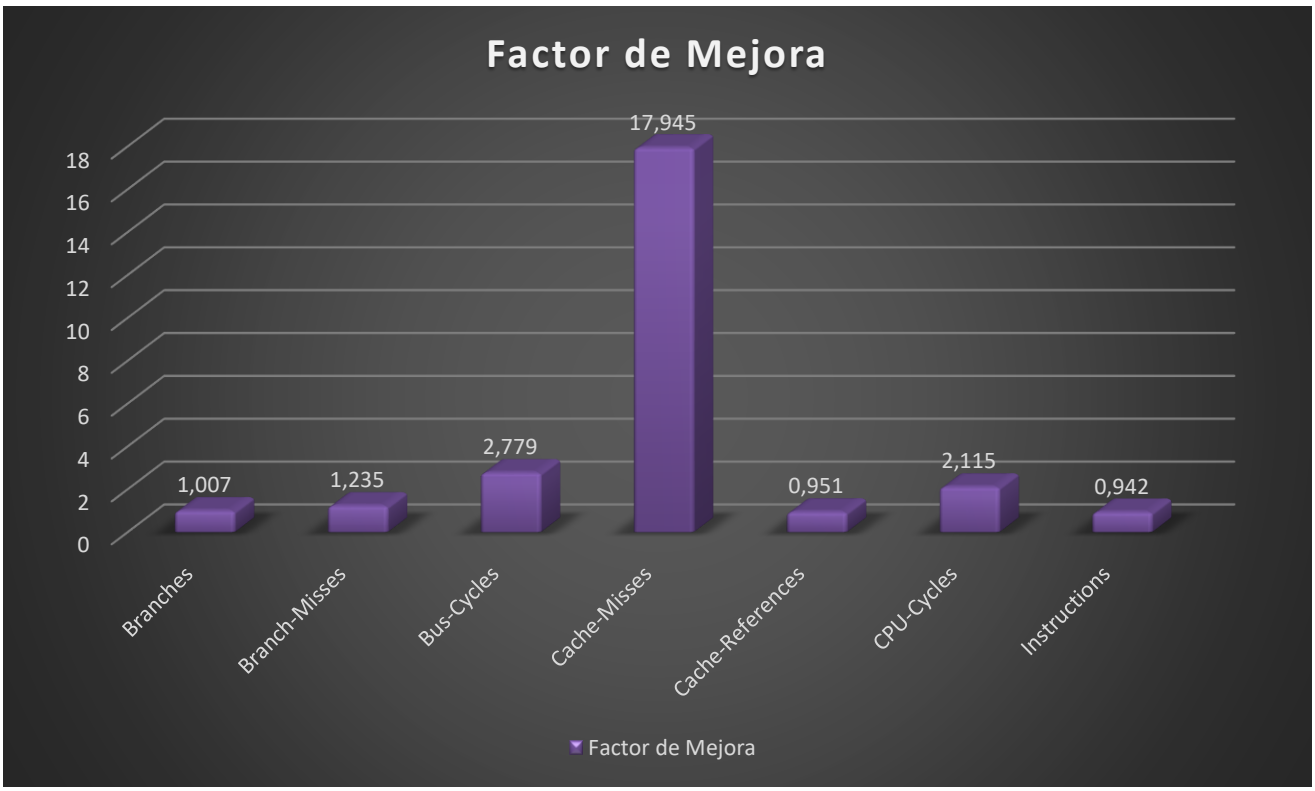
$$\textbf{Tasa de Fallos } (B) = \frac{\textbf{Número de fallos}}{\textbf{Total de Instrucciones}} = \frac{24}{27} = 0.88 * 100 = 88\%$$

$$\textbf{Tasa de Fallos } (C) = \frac{\textbf{Número de fallos}}{\textbf{Total de Instrucciones}} = \frac{14}{18} = 0.77 * 100 = 77\%$$

10. Compila el código incluido en las carpetas Naive y Blocking, con la opción –O3 dentro del Makefile para que el compilador optimice el código al máximo. A continuación, usa perf para rellenar la tabla de eventos hardware y calcula los factores de mejora. Anota también el tiempo que tarda cada versión.

EVENTO	NAIVE	BLOCKING	FACTOR
Branches	18.852.072	18.718.028	1.007%
Branch-misses	141.265	114.296	1.235%
Bus-cycles	203.793.476	73.319.054	2.779%
Cache-misses	17.001.604	947.382	17.945%
Cache-references	40.479.704	42.410.513	0.951%
Cpu-cycles	304.728.841	144.020.222	2.115%
Instructions	118.769.868	126.052.704	0.942%
Tiempo	3.662358e-01	1.3380255e-01	2.737%

11. ¿Cuáles son los eventos más representativos de la mejora?



Podemos ver que el evento con mayor mejora respecto a los demás es el Cache-misses, que tiene un factor de mejora en el de Blocking con respecto a Naive de un 17.945%. Esto se debe ya que la mejora en el Blocking está centrada en disminuir el número de fallos en caché.
Al fallar en el nivel 1 de la caché, el dato tiene que ser traído del nivel 2 de caché (un nivel superior), entonces, al tener menos fallos, también afecta al tiempo de ejecución, mejorándolo, lo que significa tener que ejecutarse en menos ciclos.
Pero también podemos ver que hay unas buenas mejoras en los eventos Bus-cycles y Cpu-cycles, ambos con valores parecidos.

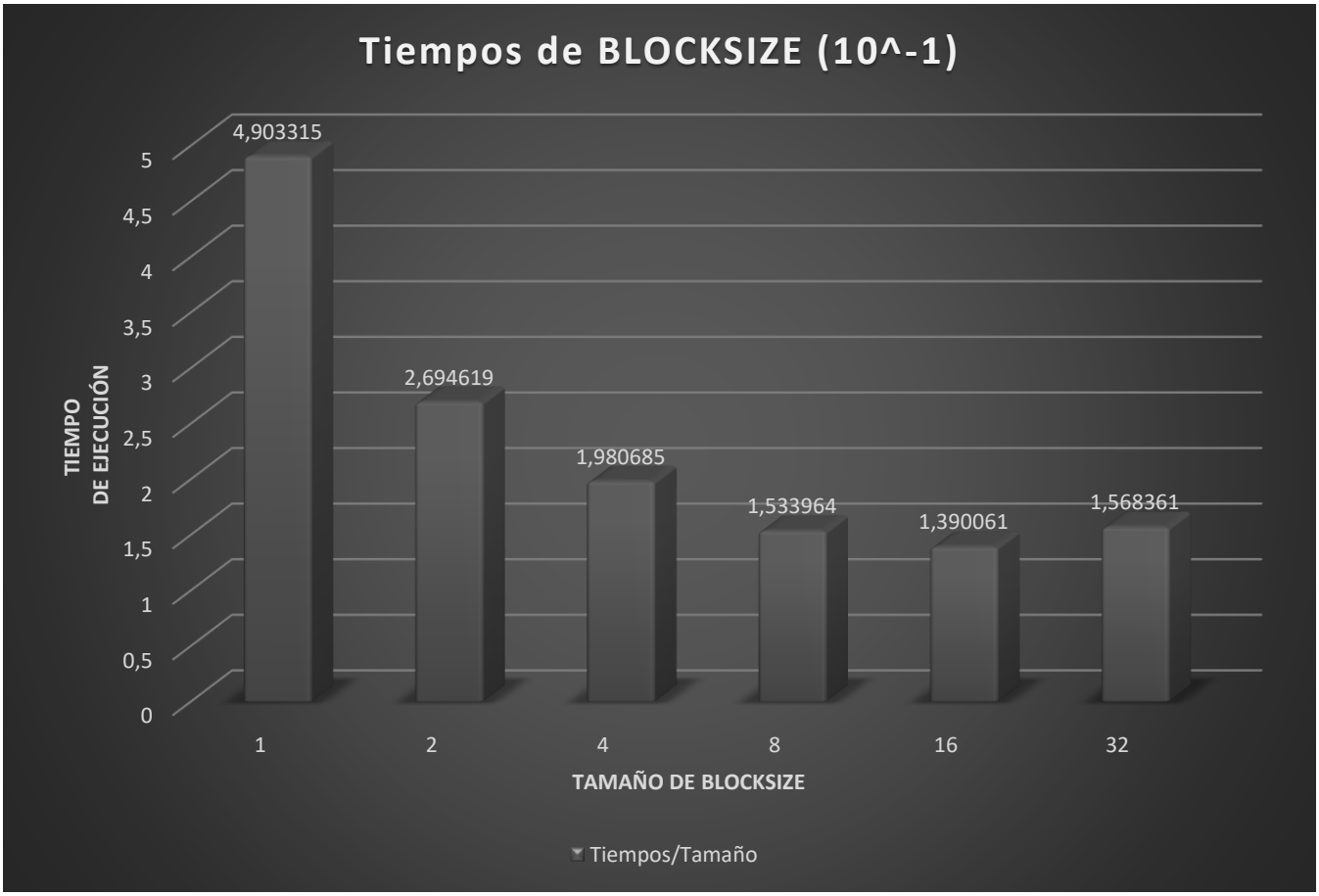
12. Aparte de los eventos anteriores, hay otros relacionados con la caché como por ejemplo L1-dcache-loads, LLC-loads, dTLB-load-misses, etc. Estos eventos permiten determinar con más exactitud que está ocurriendo en los diferentes niveles del sistema de memoria. Anota los valores de todos los eventos relacionados con la caché de datos, para las versiones Naive y cache Blocking, y justifica razonadamente su variación.

NAIVE	BLOCKING
L1-dcache-load-misses → 16.988.011	L1-dcache-load-misses → 942.309
L1-dcache-loads → 40.485.173	L1-dcache-loads → 42.423.643
L1-dcache-store-misses → 16.988.011	L1-dcache-store-misses → 942.309
L1-dcache-loads → 40.696.679	L1-dcache-loads → 42.410.030

Principalmente vemos que, al aplicar la mejora de Blocking, las mayores diferencias se encuentran en el número de fallos que se producen en la memoria caché de Nivel 1.
Esto también tiene referencias con la organización de los datos en el programa Blocking, siguiendo los principios de Localidad Espacial y Localidad Temporal.

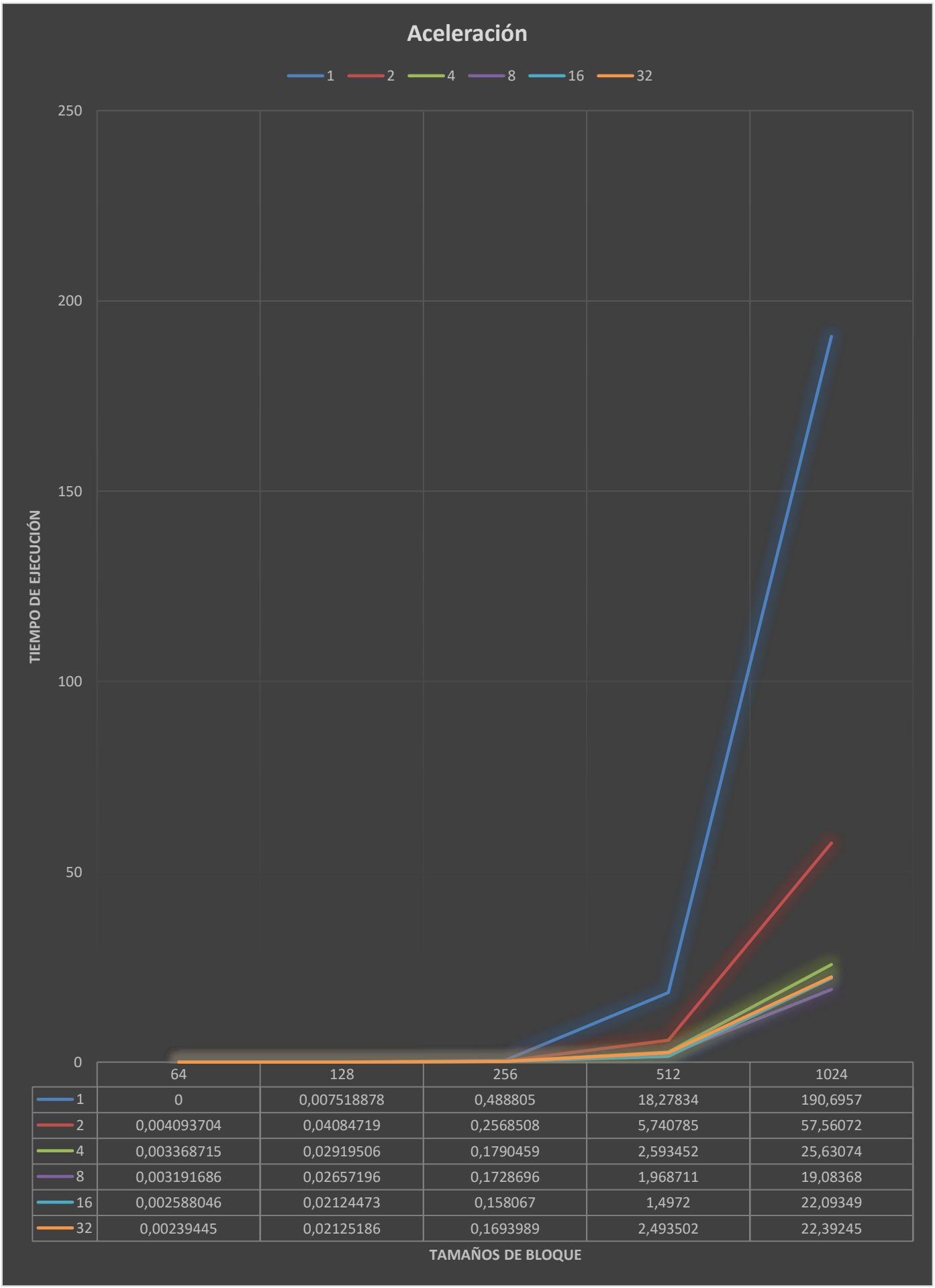
Si dividimos en bloques el array a multiplicar, cargaríamos en cache los datos más cercanos, siguiendo los principios mencionados antes.
Así, conseguiremos disminuir el número de Fallos por Conflicto, quedándonos con los Fallos de Capacidad y los Fallos Inevitables.

13. Cambia el factor de BLOCKSIZE y vuelve a compilar y ejecutar el programa. Prueba con todos los múltiplos de 2 entre 1 y 32 y representa gráficamente el tiempo de ejecución (o el factor de mejora) con respecto al factor de Blocking. Explica razonadamente qué ocurre.



Podemos ver que, aumentando el tamaño del bloque, disminuye el tiempo de ejecución, aunque si aumentamos demasiado el tamaño (como en nuestro caso (32)), vemos que el tiempo aumenta.
También podemos decir que cuanto mayor sea el tamaño de las submatrices, menor será el tiempo de ejecución, ya que disminuiremos el número de fallos, y tendremos menos accesos a la Caché de Nivel 2, que son los que ralentizan el tiempo global de ejecución.
Sin embargo, vemos que para un tamaño de bloque de 32, sufrimos un pequeño aumento de tiempo de ejecución. Esto ocurre ya que la matriz no cabe en la caché y empezamos a tener fallos por conflicto.

14. Ejecuta ahora cada versión con diferentes tamaños de matrices y representa gráficamente la mejora (aceleración) obtenida con respecto al tamaño de la matriz. Puedes probar con tamaños de 64, 128, 256, 512 y 1024.



	64	128	256	512	1024
1	0	0,007518878	0,488805	18,27834	190,6957
2	0,004093704	0,04084719	0,2568508	5,740785	57,56072
4	0,003368715	0,02919506	0,1790459	2,593452	25,63074
8	0,003191686	0,02657196	0,1728696	1,968711	19,08368
16	0,002588046	0,02124473	0,158067	1,4972	22,09349
32	0,00239445	0,02125186	0,1693989	2,493502	22,39245

15. Para los distintos tamaños de matrices y factores de Blocking, anota los eventos hardware relacionados con la caché y comprueba como varían.

BLOCKING	EVENTO	64	128	256	512	1024
1	L1-dcache-load-misses	22.899	1.968.899	18.599.736	152.513.186	1.302.997.655
1	L1-cache-loads	1.471.376	8.196.669	57.518.206	434.196.210	3.489.476.913
1	L1-dcache-store-misses	22.899	1.968.899	18.599.736	152.513.186	1.302.997.655
1	L1-cache-stores	1.471.376	8.196.669	57.518.206	434.196.210	3.489.476.913
1	LLC-store-misses	8.957	14.081	275.833	116.083.422	1.145.946.139
1	LLC-stores	136.966	3.872.045	37.305.082	307.121.863	2.163.049.745
1	LLC-load-misses	8.957	14.081	275.833	116.083.422	1.145.946.139
1	LLC-loads	136.966	3.872.045	37.305.082	307.121.863	2.613.049.768
2	L1-dcache-load-misses	17.689	539.179	5.131.564	43.421.900	347.041.073
2	L1-cache-loads	1.465.987	8.203.762	57.320.836	432.094.598	3.342.275.142
2	L1-dcache-store-misses	17.689	539.179	5.131.564	43.421.900	347.041.073
2	L1-cache-stores	1.465.987	8.203.762	57.320.836	432.094.598	3.342.275.142
2	LLC-store-misses	8.311	13.516	156.884	30.678.433	304.006.977
2	LLC-stores	107.351	1.181.281	10.428.673	87.243.471	687.422.281
2	LLC-load-misses	8.311	13.516	156.884	30.678.433	304.006.977
2	LLC-loads	107.351	1.181.281	10.428.673	87.243.471	687.422.281
4	L1-dcache-load-misses	15.251	204.294	1.848.515	16.907.044	239.186.334
4	L1-cache-loads	1.337.983	7.150.903	48.755.342	364.077.776	2.795.748.653
4	L1-dcache-store-misses	15.251	204.294	1.848.515	16.907.044	239.186.334
4	L1-cache-stores	1.337.983	7.150.903	48.755.342	364.077.776	2.795.748.653
4	LLC-store-misses	8.500	13.668	126.319	10.398.742	102.531.368
4	LLC-stores	100.673	517.361	3.920.424	33.480.437	486.553.327
4	LLC-load-misses	8.500	13.668	126.319	10.398.742	102.531.368
4	LLC-loads	100.673	517.361	3.920.424	33.480.437	486.553.327
8	L1-dcache-load-misses	12.437	84.767	755.546	11.926.483	468.764.259
8	L1-cache-loads	1.318.358	6.975.128	47.460.388	351.122.289	2.706.443.906
8	L1-dcache-store-misses	12.437	84.767	755.546	11.926.483	468.764.259
8	L1-cache-stores	1.318.358	6.975.128	47.460.388	351.122.289	2.706.443.906
8	LLC-store-misses	8.227	12.907	87.736	3.768.242	35.823.207
8	LLC-stores	98.705	268.455	2.327.725	24.476.521	882.695.509
8	LLC-load-misses	8.227	12.907	87.736	3.768.242	35.823.207
8	LLC-loads	98.705	268.455	2.327.725	24.476.521	882.695.509
16	L1-dcache-load-misses	11.602	54.637	1.778.904	35.717.767	917.336.819
16	L1-cache-loads	1.232.679	6.347.959	42.377.623	312.552.325	2.404.478.290
16	L1-dcache-store-misses	11.602	54.637	1.778.904	35.717.767	917.336.819
16	L1-cache-stores	1.232.679	6.347.959	42.377.623	312.552.325	2.404.478.290
16	LLC-store-misses	8.174	13.296	73.258	1.627.942	14.564.444
16	LLC-stores	93.838	208.813	2.314.162	72.033.711	1.836.731.560
16	LLC-load-misses	8.174	13.296	73.258	1.627.942	14.564.444
16	LLC-loads	93.838	208.813	2.314.162	72.033.711	1.836.731.560

32	L1-dcache-load-misses	11.224	94.966	4.904.754	120.685.863	1.051.139.947
32	L1-cache-loads	1.223.417	6.241.053	41.522.754	305.423.743	2.340.643.233
32	L1-dcache-store-misses	11.224	94.966	4.904.754	120.685.863	1.051.139.947
32	L1-cache-stores	1.223.417	6.241.053	41.522.754	305.423.743	2.340.643.233
32	LLC-store-misses	8.272	13.391	62.191	834.630	7.816.684
32	LLC-stores	96.882	259.027	5.309.697	244.142.451	2.095.807.922
32	LLC-load-misses	8.272	13.391	62.191	834.630	7.816.684
32	LLC-loads	96.882	259.027	5.309.697	244.142.451	2.095.807.922

16. Recupera la gráfica del ejercicio 7 (el modelo Roofline) y sitúa esta nueva versión en ella. Ten en cuenta que la intensidad operacional ha podido variar por lo que tienes que volver a calcularla, aparte de su rendimiento.

Para medir nuestro programa primero necesitamos calcular el número de operaciones en punto flotante (Flops).

$$FLOPS = \frac{\left(\frac{n}{Tamaño\ Bloque\ L1}\right)^3 * Tamaño\ Bloque\ L1^3 * 2}{Tiempo\ de\ Ejecución}$$

$$FLOPS = \frac{\left(\frac{256}{16}\right)^3 * 16^3 * 2}{1.591386 * 10^{-1}} = 210850365.7$$

Nos salen aproximadamente 0.168 GFlops, que los encontraremos en el Eje Y de la gráfica.
Una vez que hemos calculado los Flops, los dividimos entre el Tamaño del Bloque * Tráfico para calcular la Intensidad Operacional (IO), tal que:

$$IO = \frac{210850365.7}{64 * 73258} = 44.971$$

La IO la encontraremos en el Eje X de la gráfica