COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2,5 puntos)

En un procesador Q1 con direcciones de 32 bits, el camino crítico, y por tanto el tiempo de ciclo, está limitado por la memoria cache de datos. El tiempo de retardo de los componentes de la memoria cache de datos se desglosa de la siguiente forma:

Componente	Tiempo
Memoria de etiquetas	0,32 ns
Comparación de etiquetas y (en caso necesario) **selección de vía**	0,18 ns
Memoria de datos y selección de byte de la linea	0,45 ns
Mux de vía de datos: selecciona el dato de la vía correspondiente (cuando sea necesario)	0,15 ns
Registro de desacoplo (cuando sea necesario)	0,05 ns

Queremos analizar 2 configuraciones para la cache de datos, todas ellas con 32KB de capacidad y lineas de 16 bytes:

- C1: Cache de mapeo directo con acceso PARALELO a etiquetas y datos.
- C2: Cache asociativa por conjuntos de dos vías SEGMENTADA en 2 etapas (el tiempo de acceso a la cache son 2 ciclos de procesador).

a) **Calcula** el tiempo de ciclo de la cache de datos y el tiempo total de un acceso para las diferentes versiones del procesador Q1 con las caches C1 y C2, usando la distribución más adecuada de los componentes por etapas.

C1: Tc= MAX(0,32+0,18; 0,45)= 0,5ns Tsa= 1 ciclo

C2: Tc= MAX(0,32+0,18+0,05; 0,45+0,15+0,05)= 0,65ns Tsa= 2 ciclos

b) **Calcula** la frecuencia de reloj para las diferentes versiones del procesador Q1 con las caches C1y C2.

F= 1/Tc

C1: 1/0,5ns= 2GHz

C2: 1/0,65ns= 1,538GHz

Un programa P que ejecuta $2{,}5x10^9$ instrucciones tiene un 50% de instrucciones aritméticas, un 20% de instrucciones de salto y un 30% de instrucciones de acceso a memoria (Load/Store). Las instrucciones aritméticas tardan 4 ciclos, las de salto 3 y las de memoria 5 ciclos + los ciclos del acceso a la cache.

c) **Calcula** el CPI del programa P para los procesadores con C1 y C2 suponiendo que nunca hay fallos en la cache de datos.

CPI_C1= 0,5*4 + 0,2*3 + 0,3*(5+1)= 4,4

CPI_C2= 0,5*4 + 0,2*3 + 0,3*(5+2)= 4,7

Sabemos que el programa P tiene un 10% de fallos con la cache de datos C1 y un 6% con la C2. Además, el tiempo de penalización medio por fallo en ambos casos es de 60 ciclos.

d) **Calcula** el speedup en tiempo de ejecución de C2 sobre C1 en % teniendo en cuenta la jerarquía de memoria completa.

$T=I*(CPI_{id}+CPI_{mem})*Tc$

T_C1= 2,5*10^9*(4,4 + 0,3*0,1*60)*0,5*10^(-9)= 7,75s
T_C2= 2,5*10^9*(4,7 + 0,3*0,06*60)*0,65*10^(-9)= 9,39s

SpeedUp= 7,75/9,39= 0,825 ==> Slowdown de 1/0,825 ==> 21,16% Slowdown o -17,5% de Speedup

Se hace una implementación multibanco de la cache C2, organizada en 4 bancos 2-asociativos, con entrelazado a nivel de bloque.

e) **Indica** cómo se desglosarían los bits de una dirección entre bits de Etiqueta, selección de Conjunto, Banco y Byte.

Etiqueta <31:14>	Conjunto <13:6>	Banco <5:4>	Byte <3:0>

4 bancos ==> 2 bits para seleccionar banco.
Entrelazado a nivel de bloque ==> 2 bits de menos peso del conjunto para seleccionar el banco.

COGNOMS:

NOM:

Problema 2. (2.5 puntos)

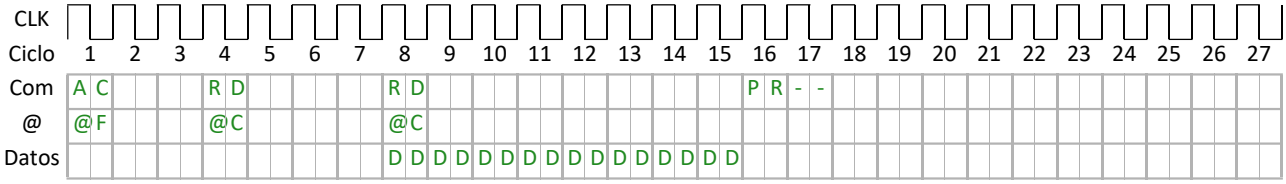
Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 2 bancos. El DIMM esta configurado para leer/escribir ráfagas de 64 bytes (justo el tamaño de bloque de las caches). La latencia de fila es de 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Es posible que el conjunto **CPU**+**$I**+**$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultáneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda.

La siguiente tabla muestra en qué banco y qué página de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D.

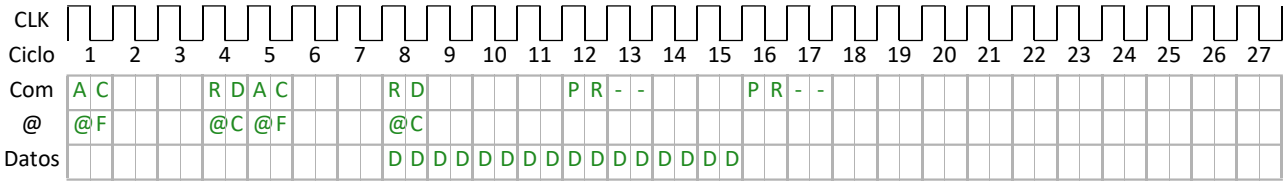
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes (en el orden que se indica), en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. Indica la ocupación de los distintos recursos de la memoria DDR: bus de datos, bus de direcciones y bus de comandos. En todos los cronogramas supondremos que no hay ninguna página de DRAM abierta previamente.

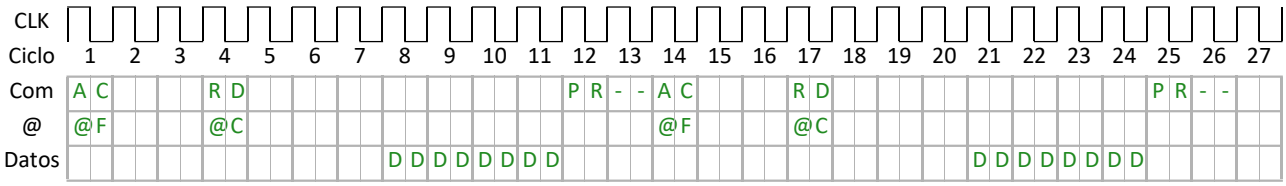
a) **Rellena** el siguiente cronograma para la lectura de los bloques AB.



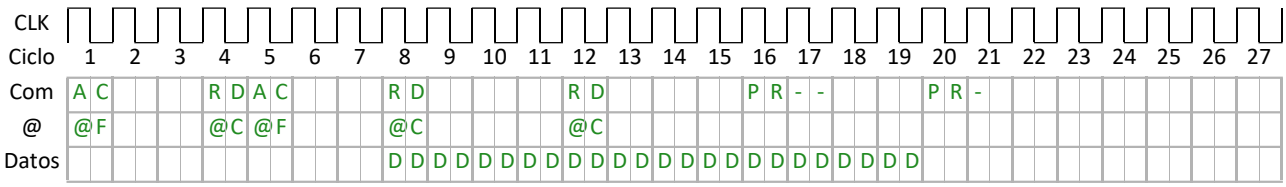
b) **Rellena** el siguiente cronograma para la lectura de los bloques AC.



c) **Rellena** el siguiente cronograma para la lectura de los bloques CD



d) **Rellena** el siguiente cronograma para la lectura de los bloques ADB.



En esta CPU ejecutamos un programa, en el que se detecta que el bucle **`for`** del siguiente fragmento de código consume la mayor parte de el tiempo de ejecución:

```
/* Variables globales */
float A[1024*1024]; float B[1024*1024]; /* un float ocupa 4 bytes */
.....
/* codigo */
for (i=0;i<1024*1024; i++)
   A[i] = A[i] + B[i];
```

Un análisis detallado muestra que los fallos en la cache de instrucciones son despreciables, pero que los fallos en la cache de datos son excesivos (mas del 60%). Sabemos que la cache de datos es de mapeo directo, con bloques de 64 bytes y política de escritura copy back + write allocate. Mediante el uso del debugger hemos averiguado que el compilador almacena las variables A y B consecutivas en memoria y que se encuentran respectivamente en las direcciones 0x00400000 y 0x00800000

e) **Indica** a qué son debidos los fallos. **Realiza** una optimización de código, de las vistas en clase, que minimice los fallos en la cache de datos.

Los fallos son debidos a conflictos entre el vector A y el B. A y B se reemplazan mutuamente en cada iteración por lo que tendremos fallo tanto en A como en B en todas las iteraciones. Aprovechamos la localidad temporal de A pero no la espacial (ni en A ni en B).

Se pueden minimizar los fallos haciendo padding, el padding tiene que ser como mínimo del tamaño de bloque:

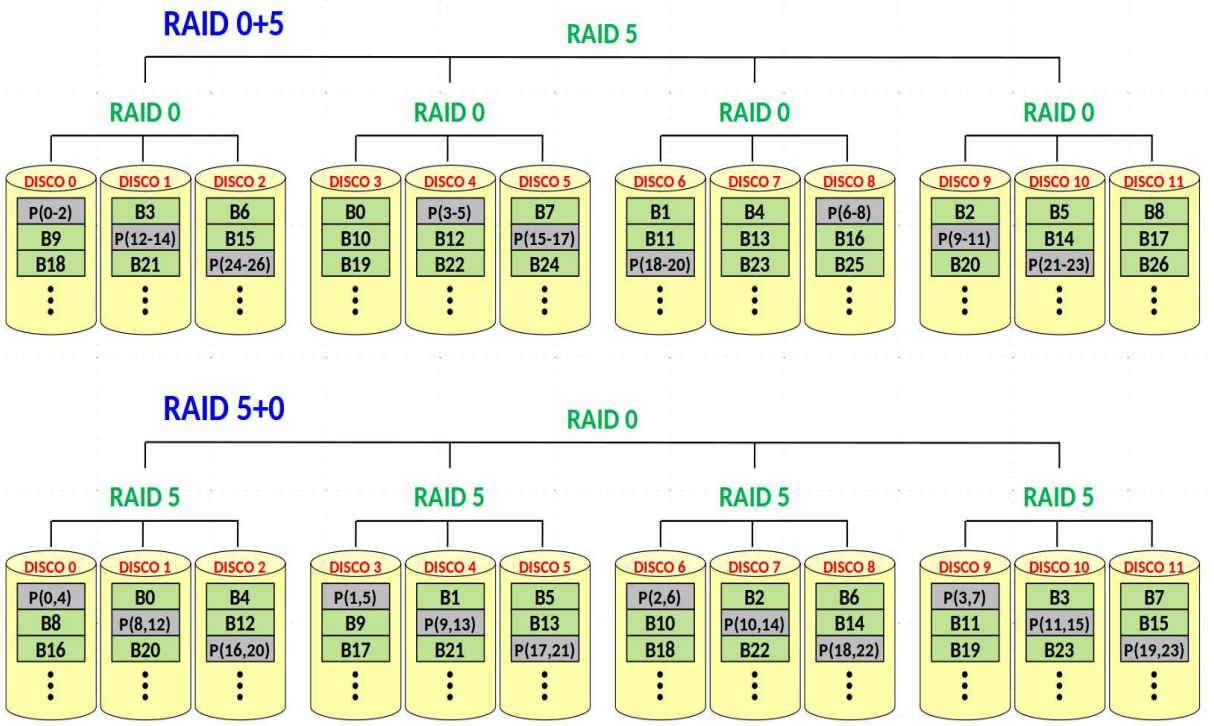
```
float A[1024*1024]; float pad[16]; float B[1024*1024];
```

COGNOMS:

NOM:

Problema 3. (2.5 puntos)

La figura muestra las 2 posibles combinaciones vistas en clase para montar un disco lógico multi-RAID combinando RAID 0 y RAID 5. Se tratan del RAID 05 y RAID 50 que, para un caso general, se forman con N discos y G grupos de dG discos por grupo. De la figura observamos cómo la paridad que introduce RAID 5 se calcula de diferente forma en RAID 05 que en RAID 50. Esto afecta a la capacidad útil y a la fiabilidad que ofrece cada sistema.



a) **Indica** la expresión general simplificada para calcular la capacidad útil de información que se puede almacenar en RAID 50 y en RAID 05. Debes utilizar para ello las variables N, G, dG y llamar T_d al tamaño en GBytes de cada disco. Indica el resultado en GBytes.

RAID 05: $N * T_d - dG * T_d = (N - dG) * T_d$ GBytes

RAID 50: $N * T_d - 1 * G * T_d = (N - G) * T_d$ GBytes

Para las dos configuraciones del caso particular de la figura se cumple que **N = 12, G = 4 y dG = 3**. Sin embargo, las dos poseen distinta capacidad útil, y RAID 50 es el que proporciona mejor tolerancia a fallos.

b) El RAID 50 de la figura puede seguir funcionando aunque fallen 4 discos. **Marca** con una cruz en la tabla siguiente 4 discos que podrían fallar y el RAID 50 seguir funcionando. Utiliza la fila de la tabla etiquetada con **(b)**.

c) El mismo RAID 50 de la figura puede dejar de funcionar si fallan 2 discos. **Marca** con una cruz en la tabla siguiente 2 discos que deberían fallar para que el RAID 50 no funcione. Utiliza la fila de la tabla etiquetada con **(c)**.

	RAID 0											
	RAID 5			**RAID 5**			**RAID 5**			**RAID 5**		
	disco 0	**disco 1**	**disco 2**	**disco 3**	**disco 4**	**disco 5**	**disco 6**	**disco 7**	**disco 8**	**disco 9**	**disco 10**	**disco 11**
(b)	X			X				X			X	
(c)	X	X										

Montamos un disco lógico con la configuración RAID 50 con valores **N = 15, G = 3 y dG = 5**. El $MTTF_{\_d}$ de un disco es de 60.000 horas y el tiempo de reemplazar un disco y reconstruir la información es MTTR = 30 horas. Sabemos que $MTTF_{\_RAID\ 50}$ coincide con el MTTF de uno de sus grupos ($MTTF_{\_grupo}$) dividido por el número de grupos (G).

d) **Escribe** la expresión general del $MTTF_{\_RAID\ 50}$ suponiendo que los discos son el único componente que puede fallar. Debes utilizar para ello las variables N, G, dG, MTTF_d y MTTR. **Calcula** el valor de este $MTTF_{\_RAID\ 50}$ para los valores proporcionados.

$MTTF_{\_RAID\ 50} = MTTF_{\_grupo} / G$ y 1 grupo = RAID 5 con 5 discos (dG = 5)

$MTTF_{\_RAID\ 50} = (MTTF_{\_d} / dG) * (MTTF_{\_d} / ((dG - 1) * MTTR)) * (1 / G) = MTTF_{\_d}\ ^2 / (dG * (dG -1) * G * MTTR)$

$MTTF_{\_RAID\ 50} = 60.000\ ^2 / (5 * (5 - 1) * 3 * 30) = 2.000.000$ horas

Queremos evaluar el rendimiento de utilizar esta misma configuración RAID 50 con **N = 15, G = 3 y dG = 5**. Utilizamos para ello discos de 1 TByte, tamaño de sector de 512 Bytes, y un mismo ancho de banda de 256 MB/s por disco tanto para leer como para escribir un bloque de datos. Un bloque de datos está formado por 50000 sectores consecutivos. Ejecutamos una aplicación formada por 4 fases (en las fases 1, 3 y 4 sólo consideramos el tiempo de la transferencia):

- La fase 1 lee de disco los datos de entrada formados por 100 bloques de datos distribuidos entre todos los discos.
- La fase 2 realiza los cálculos, con un tiempo de ejecución de 4 segundos.
- La fase 3 escribe a disco 50 bloques de datos realizando **escrituras secuenciales**.
- La fase 4 escribe a disco 50 bloques de datos realizando **escrituras aleatorias**, distribuidas uniformemente entre todos los discos.

e) **Calcula** el tiempo de ejecución de nuestra aplicación si utilizamos un único disco. **Calcula** también el tiempo de ejecución de la aplicación cuando usamos el RAID 50 de **N = 15, G = 3 y dG = 5**.

bloque de datos = 50000 sectores = 25,6 MB

fase 1: 100 bloques * 25,6 MB = 2,56 GB ---> 2,56 GB / 256 MB/s = 10 s

fase 3: 50 bloques * 25,6 MB = 1,28 GB ---> 1,28 GB / 256 MB/s = 5 s

fase 4: igual que fase 3 = 5 s

Total con un único disco: 10 + 4 + 5 + 5 = 24 s

fase 1: 2,56 GB / (15 * 256 MB/s) = 10 s / 15 = 0,66 s

fase 3: 1,28 GB / ((15 - 3) * 256 MB/s) = 5 s / 12 = 0,41 s

fase 4: 1,28 GB / ((15 / 4) * 256 MB/s) = 5 s / 3,75 = 1,33 s

Total con RAID 50: 0,66 + 4 + 0,41 + 1,33 = 6,4 s

COGNOMS:

NOM:

Problema 4. (2.5 puntos)

Queremos evaluar un servidor web multi-thread en un sistema con un multiprocesador. El programa, que se ejecuta de forma permanente, se compone de dos threads de servicio y dos threads multimedia. Los threads de servicio se dedicarán a atender las peticiones HTTP y, de forma inherente, estarán siempre activos el 100% del tiempo. Los threads multimedia estarán dedicados al tratamiento de imágenes, necesario para atender las peticiones de los usuarios, y estos estarán activos sólo cuando se requiera según la carga del servidor, que mediremos en una media de $3.6 \cdot 10^{13}$ operaciones de coma flotante por hora a repartir entre los dos threads multimedia. Esta carga será constante en todo el problema.

Dado que tenemos cuatro threads en total, evaluaremos configuraciones con cuatro cores, en el que asumiremos que siempre tenemos un único thread asignado a cada core. Por simplicidad, podremos hablar de cores de servicio y cores multimedia. Para un prodesador de 4 cores tendremos 2 cores de servivio y 2 cores multimedia.

a) **Calcula** en GFLOPS el rendimiento mínimo que debe tener un procesador para poder ejecutar toda la carga de trabajo del servidor. **Calcula** los GFLOPS mínimos por cada core multimedia .

GFLOPS total = $3.6*10^{13}$ flop * (1 gflop/ 10^9 flop) / 3600 s = 10 GFLOPS

GFLOPS por core = 10 GFLOPS / 2 = 5 GFLOPS

Evaluamos el programa en un multiprocesador CISC que llamaremos C1. Este multiprocesador se compone de cuatro cores idénticos. Cada core trabaja a una frecuencia de 1.6 GHz, tiene una corriente de fuga de 2 A, se alimenta a un voltaje de 1.5 V y tiene una carga capacitiva equivalente de 5 nF. El consumo debido a cortocircuito es despreciable.

b) **Calcula** la potencia disipada por el chip C1 al ejecutar el servicio web.

P_{fugas} = I*V = 2 A * 1.5 V = 3 W por core
P_{comm} = $C*V^2*F$ = 5 nF * $(1.5\ V)^2$ * 1.6 GHz = 18 W por core
P_{C1} = (3 W + 18 W) * 4 cores = 84 W

El chip C1 no implementa escalado de frecuencia, es decir que cada core mantiene el voltaje y la frecuencia independientemente de que el thread esté activo o no. La evaluación de este sistema indica que:

- Cada uno de los threads de servicio está activo el 100% del tiempo.
- Con la carga de trabajo que va a soportar el servidor, vemos que cada uno de los threads multimedia está activo sólo el 60% del tiempo.

Disponemos de un procesador más avanzado que llamaremos C2, con las mismas características que C1 pero que implementa voltaje dinámico y escalado de frecuencia. Este procesador puede configurar individualmente cada core para trabajar en modo bajo consumo, normal (como C1), o turbo. En bajo consumo, un core del chip C2 reduce su voltaje a 0.8 V y su frecuencia a 1.2 GHz. En modo turbo, un core incrementa su voltaje a 2 V y su frecuencia a 1.8 GHz.

Ejecutamos la aplicación en el chip C2 y vemos que los cores de servicio pueden estar todo el tiempo en modo bajo consumo, aunque sus threads asociados estén siempre activos. También observamos que los cores multimedia van cambiando entre bajo consumo y turbo en función de si su thread asociado está activo; y, debido al aumento de frecuencia del modo turbo, también vemos que el tiempo que los threads multimedia están activos es menor.

c) **Calcula**, en porcentaje, cuánto tiempo está en modo turbo cada uno de los cores multimedia en el chip C2 al ejecutar el servicio web (un core multimedia estará en modo turbo sólo cuando su thread asociado esté activo).

Aumenta la frencuencia de 1.6 GHz a 1.8 GHz:
S = 1.8 / 1.6 = 1.125
60% / 1.125 = 53.3 %

d) **Calcula** la potencia media disipada por el chip C2 mientras se ejecuta el servicio web.

$P_{fugas\ bajo\ consumo}$ = 2 A * 0.8 V = 1.6 W
$P_{comm\ bajo\ consumo}$ = 5 nF * $(0.8\ V)^2$ * 1.2 GHz = 3.84 W
$P_{fugas\ turbo}$ = 2 A * 2 V = 4 W
$P_{comm\ turbo}$ = 5 nF * $(2\ V)^2$ * 1.8 GHz = 36 W

P_{C2} = (1.6+3.84) * (2 + 2*(1-0.533)) + (4+36)*2*0.533 = 58.6 W

Queremos analizar el rendimiento de la aplicación en un procesador RISC. Es común en estos procesadores tener cores heterogéneos y activar uno u otro en función de la carga de trabajo con el fin de reducir el consumo. ARM llamó a esta tecnología big.LITTLE, y en un principio el control era completamente hardware, y o bien se activaban los cores big, o bien los LITTLE. Ahora, los diseños más modernos ya permiten que el S.O. sea consciente de los cores heterogéneos y puede asociar un thread concreto a un core específico según una serie de análisis e inferencias. Creemos que esta tecnología nos puede ir muy bien porque los threads de servicio podrían ejecutarse perfectamente en cores sencillos, como los LITTLE, y los thread multimedia en los cores que ofrecen más rendimiento.

Estudiamos el mercado y podemos elegir entre estos chips RISC que implementan multi-procesamiento heterogéneo:

Chip	Número de cores	Consumo total	Rendimiendo máximo por core big
R1	2 big, 2 LITTLE	1.5 W	2 GFLOPS
R2	2 big, 2 LITTLE	2 W	2.5 GFLOPS
R3	4 big, 4 LITTLE	5 W	3 GFLOPS

Estos modelos nos ofrecen un consumo mucho menor en comparación a los anteriores, pero a cambio el rendimiento de los cores big (que irán destinados a ejecutar los thread multimedia) también se ve reducido. Asumiendo la misma carga de peticiones que hemos analizado anteriormente, hemos de asegurarnos que los threads multimedia podrán ejecutar la misma carga de trabajo. Debido a ello, consideramos también algún modelo con cuatro cores big y asumiremos que la tarea multimedia es perfectamente paralelizable y que no conlleva penalización de sincronización. Además, para estos chips asumiremos que siempre podemos llegar a los GFLOPS del rendimiento máximo y que no hay voltaje dinámico, y por tanto el consumo será siempre estable.

e) **Justifica** si alguno de los chips RISC podría servirnos para la aplicación, y elige cuál (R1, R2, R3, o ninguno). Si lo hubiera, el chip idóneo será aquel con mejor rendimiento por consumo. **Calcula** los GFLOPS/w del chip.

Mínimo necesitamos 10 GFLOPS, el único chip que nos servirá será R3, que nos ofrece 3 x 4 = 12 GFLOPS.

GFLOPS/w = 12 GFLOPS / 5 W = 2.4

Tras analizar el algoritmo multimedia, observamos que durante el 10% del tiempo se ejecuta una rutina que ejecuta las operaciones óptimas, pero durante el otro 90% del tiempo se ejecuta otra rutina que con otro algoritmo de cálculo podría optimizarse para hacer el mismo trabajo con menos operaciones. Queremos estudiar cuánto habría que mejorar el algoritmo de esta rutina para poder ejecutar nuestro programa en el chip R1, consiguiendo así un consumo mínimo.

f) **Calcula** la ganancia que debemos aplicar al algoritmo no optimizado para poder ejecutar la aplicación en el chip R1.

Si el algoritmo actual necesita 10 GFLOPS de rendimiento y queremos ejecutarlo en un chip que ofrece 4 GFLOPS, necesitamos una ganancia global de 10 / 4 = 2.5.

2.5 = 1 / (0.1 + 0.9/s) -> s = 3