ESCOLA TÈCNICA SUPERIOR
D'ENGINYERIA
Universitat Rovira i Virgili

Práctica 3

Estructura de computadores

Alumno/s: Dickinson Bedoya Perez
Anna Gracia Colmenarejo
Enseñamiento: Ingeniería Informática
Profesor/es: Carlos Aliagas
Carlos Molina
Fecha: 04/2021

ÍNDICE

FASE 1: Hit/Miss Ratio

Se quiere ver cómo se comportan las diferentes caches de un procesador, es decir, ver su miss ratio.

El **hit ratio** es el porcentaje de veces que se accede a memoria y se encuentra un dato.

$$hit\ ratio\ = \frac{\#aciertos}{\#accesos} * \ 100$$

El **miss ratio** es el porcentaje de veces que se accede a memoria y no se encuentra el dato.

$$miss\ ratio\ = \frac{\#fallos}{\#accesos} * \ 100$$

o

$$miss\ ratio\ = 1 - \ hit\ ratio$$

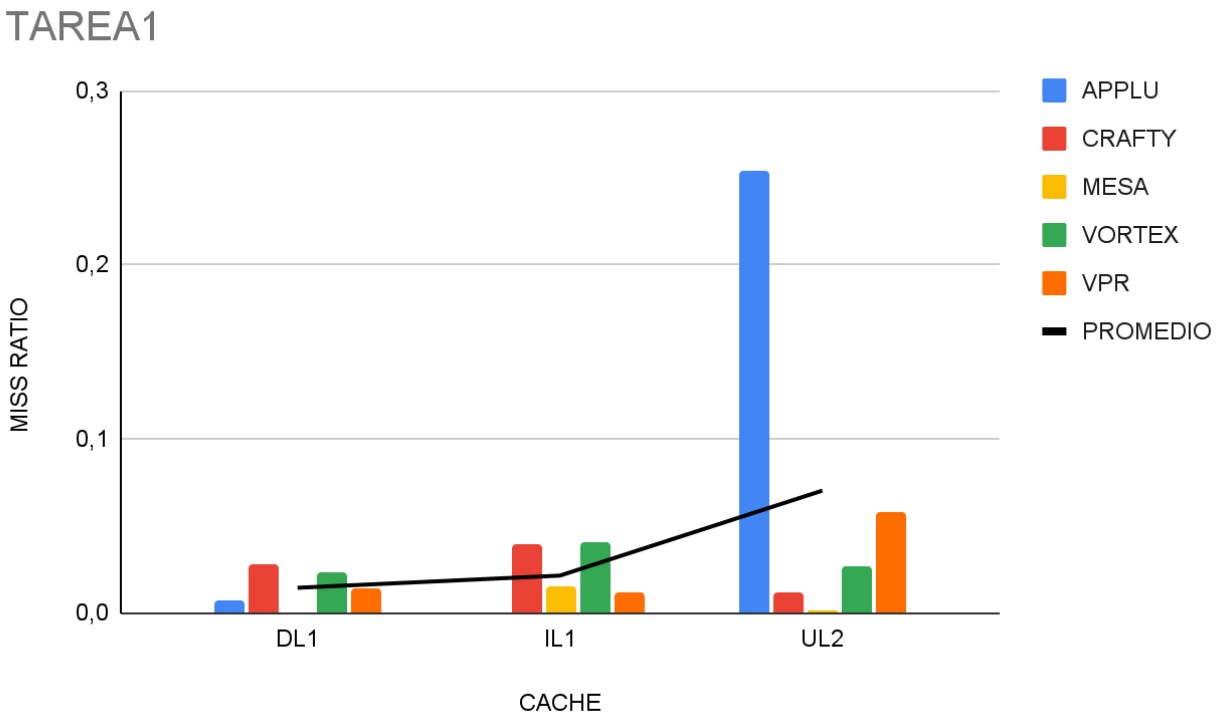
TAREA 1: Configuración básica

En esta tarea se muestra el comportamiento, *miss ratio*, de los valores base asumidos por el simulador SimpleScalar para cada caché. Para eso simularemos 5 benchmarks: applu, crafty, mesa, vortex y vpr.

La caché de datos de primer nivel, DL1, tiene como valores base un tamaño total de 16 KB, una asociatividad de 4, un tamaño de bloque de 32 bytes y una política de reemplazo LRU. La caché de instrucciones de primer nivel, IL1, tiene un tamaño total de 16KB, una asociatividad de 1, un tamaño de bloque de 32 bytes y una política de reemplazo de LRU. La caché de datos/instrucciones de segundo nivel, UL2, tiene un tamaño total de 256KB, una asociatividad de 4, un tamaño de bloque de 64 bytes y una política de reemplazo de LRU.

Para automatizar la ejecución hemos creado un script que ejecuta los 6 benchmarks para cada caché.

A continuación se muestran la gráfica de los resultados del miss ratio de la simulación de cada benchmark en cada caché, y también el promedio de cada una.



RESULTADOS

Para la caché de datos de nivel 1 hemos obtenido un promedio de 0,0144, para la de instrucciones de segundo nivel hemos obtenido un promedio de 0,0214 y para la de instrucciones/datos de tercer nivel hemos obtenido un promedio de 0,0702.

Como podemos ver la caché de instrucciones falla un poco más que la de datos y eso es porque la de instrucciones tiene una asociatividad de 1 (directa) por lo que cada dato se guarda en una línea lo que provoca es que los datos se peleen por las líneas y no quepan tantos.

Los resultados son los esperados ya que las cachés de primer nivel fallan menos que las de segundo nivel porque en las de primer nivel se guardan los datos más utilizados y si no lo encuentra en una L1 posiblemente tampoco la encuentre en la L2.

TAREA 2: Tamaño DL1

En esta tarea hemos comprobado la tasa de fallos de la caché de datos de primer nivel, DL1, a medida que su tamaño total aumenta, que tiene la configuración de asociatividad 1 (directa), con un tamaño de línea de 16 bytes y una política de reemplazo LRU.
Hemos creado un script que ejecuta los 6 benchmarks para cada tamaño total de la caché y así automatizar el proceso de simulación.
Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber para introducir los datos en el comando del SimpleScalar es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para un tamaño de 1KB:

numSets = 1*1024 / (16 * 1) = 64

Para un tamaño de 2KB:

numSets = 2*1024 / 16 = 128

Para un tamaño de 4KB:

numSets = 4*1024 / 16 = 256

Para un tamaño de 8KB:

numSets = 8*1048 / 16 = 512

Para un tamaño de 16KB:

numSets = 16*1048 / 16 = 1024

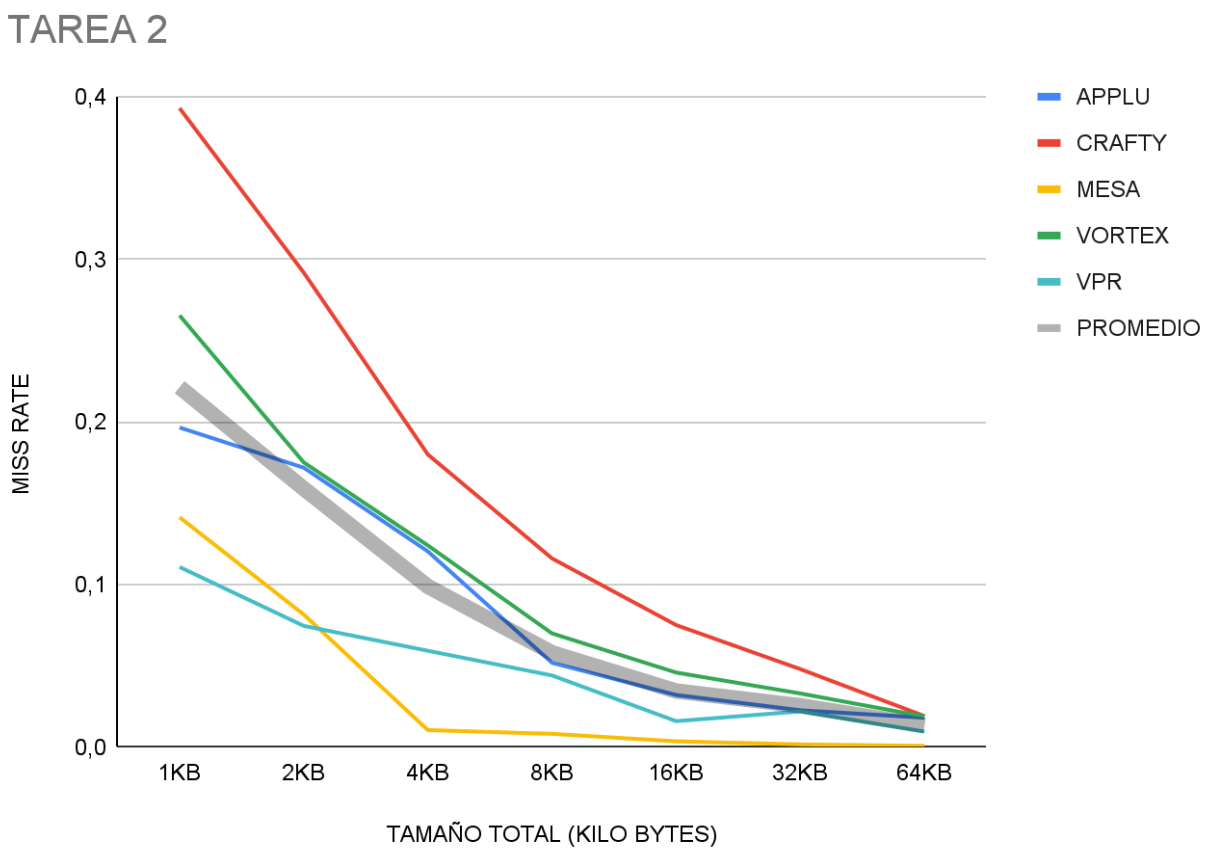
Para un tamaño de 32KB:

numSets = 32*1024 / 16 = 2048

Para un tamaño de 64KB:

numSets = 64*1024 / 16 = 4096

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Los resultados son los esperados ya que al aumentar el tamaño de la caché baja el miss ratio porque caben más datos en la caché de primer nivel DL1. Con un tamaño más grande nos estaríamos aprovechando de la localidad temporal, es decir, más datos se podrán almacenar en esta caché. La localidad temporal significa guardar los datos recientemente accedidos cerca del procesador.

TAREA 3: Tamaño IL1

En esta tarea hemos comprobado la tasa de fallos de la caché de instrucciones de primer nivel, IL1, a medida que su tamaño total aumenta que tiene la configuración de asociatividad 1 (directa), con un tamaño de línea de 16 bytes y una política de reemplazo LRU.

Hemos creado un script que ejecuta los 6 benchmarks para cada tamaño total de la caché y así automatizar el proceso de simulación.

Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para un tamaño de 1KB:

numSets = 1*1024 / (16 * 1) = 64

Para un tamaño de 2KB:

numSets = 2*1024 / 16 = 128

Para un tamaño de 4KB:

numSets = 4*1024 / 16 = 256

Para un tamaño de 8KB:

numSets = 8*1024 / 16 = 512

Para un tamaño de 16KB:

numSets = 16*1024 / 16 = 1024

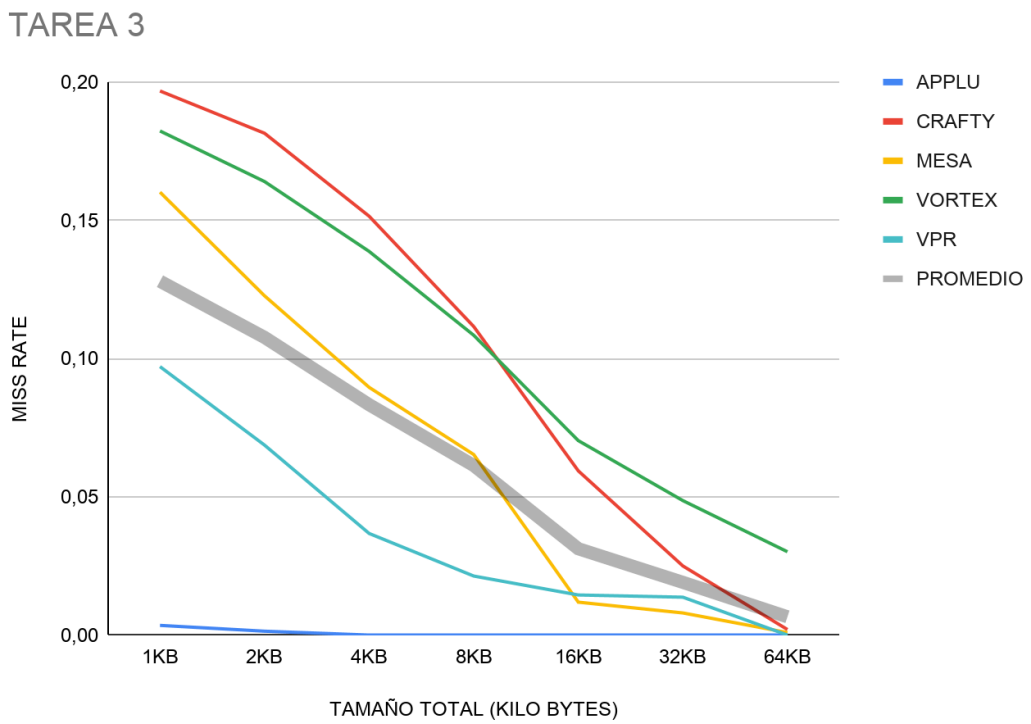
Para un tamaño de 32KB:

numSets = 32*1024 / 16 = 2048

Para un tamaño de 64KB:

numSets = 64*1024 / 16 = 4096

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Los resultados son los esperados ya que al aumentar el tamaño de la caché baja el miss ratio porque caben más datos en la caché de primer nivel IL1. Con un tamaño más grande nos estaríamos aprovechando de la localidad temporal, es decir, más datos se podrán almacenar en esta caché. La localidad temporal significa guardar los datos recientemente accedidos cerca del procesador.

TAREA 4: Tamaño UL2

En esta tarea hemos comprobado la tasa de fallos de la caché de datos/instrucciones de segundo nivel, UL2, a medida que su tamaño total aumenta que tiene la configuración de asociatividad 1 (directa), con un tamaño de línea de 16 bytes y una política de reemplazo LRU.

Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para un tamaño de 32KB:

numSets = 32*1024 / (64 * 1) = 512

Para un tamaño de 64KB:

numSets = 64*1024 / 64 = 1024

Para un tamaño de 128 KB:

numSets = 128*1024 / 64 = 2048

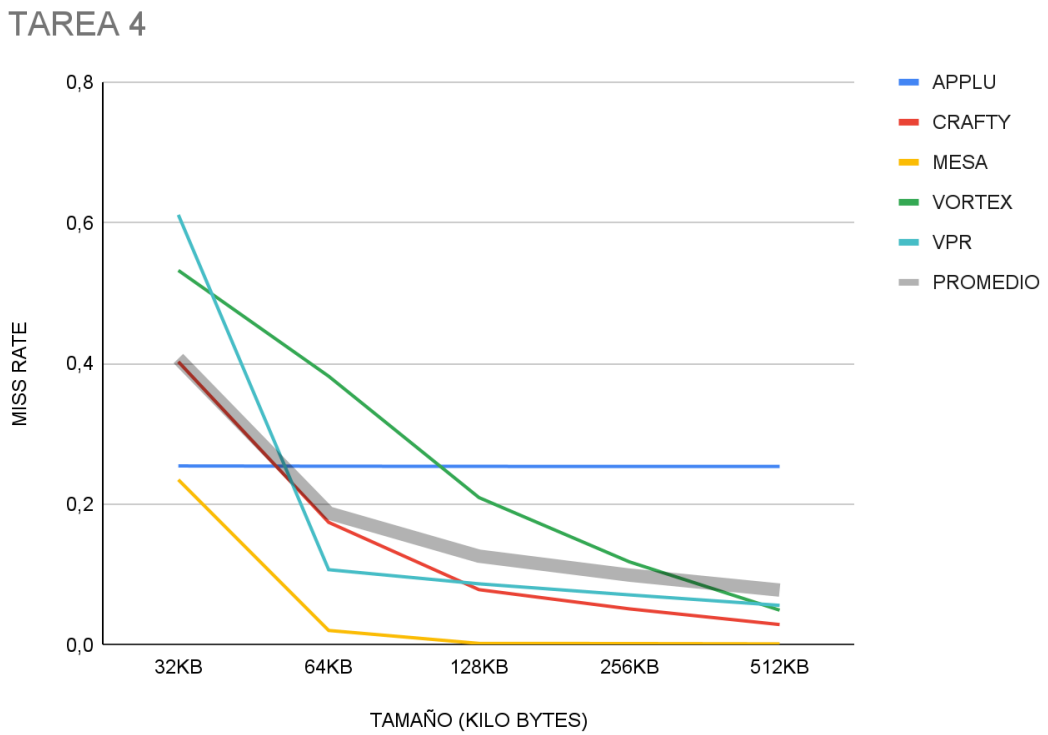
Para un tamaño de 256KB:

numSets = 256*1024 / 64 = 4096

Para un tamaño de 512KB:

numSets = 512*1024 / 64 = 8192

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Los resultados son los esperados ya que al aumentar el tamaño de la caché baja el miss ratio porque caben más datos en la caché de primer nivel UL2. Con un tamaño más grande nos estaríamos aprovechando de la localidad temporal, es decir, más datos se podrán almacenar en esta caché. La localidad temporal significa guardar los datos recientemente accedidos cerca del procesador.

TAREA 5: Asociatividad DL1

En esta tarea hemos comprobado la tasa de fallos de la caché de datos de primer nivel, DL1, a medida que su asociatividad aumenta y que tiene la configuración de un tamaño total de 16 Kbytes, un tamaño de línea de 16 bytes y una política de reemplazo LRU.
Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como el tamaño total, la asociatividad y tamaño de bloque lo sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para asociatividad de 1:

numSets = 16*1024 / (16 * 1) = 1024

Para asociatividad de 2:

numSets = 16*1024 / (16 * 2) = 512

Para asociatividad de 4 :

numSets = 16*1024 / (16 * 4) = 256

Para asociatividad de 8:

numSets = 16*1024 / (16 * 8) = 128

Para asociatividad de 16:

numSets = 16*1024 / (16 * 16) = 64

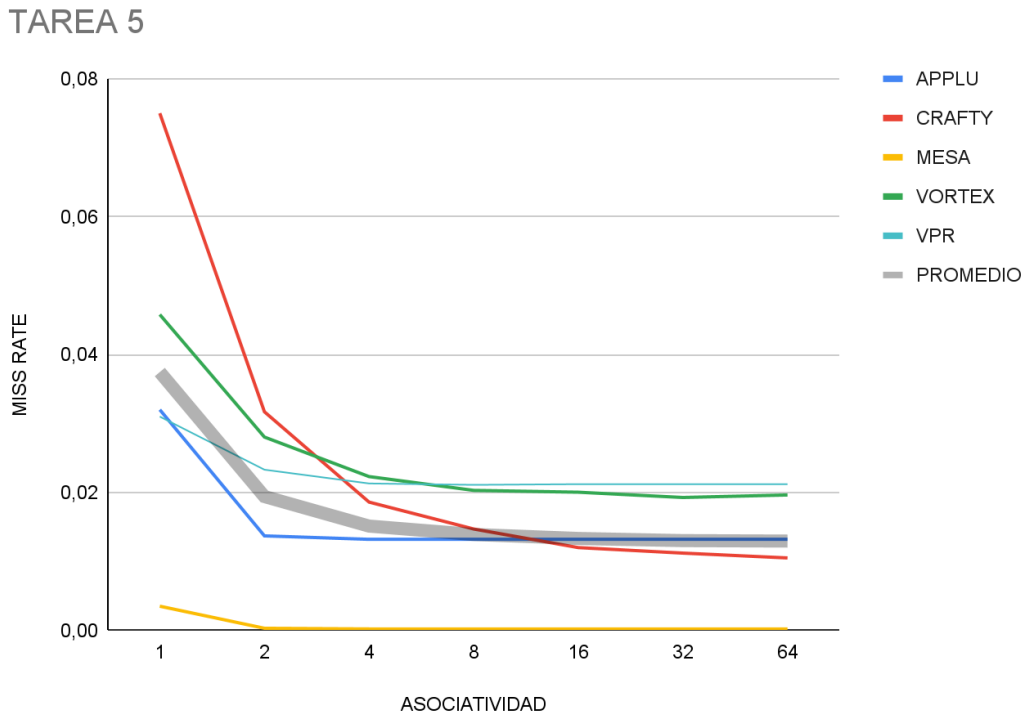
Para asociatividad de 32:

numSets = 16*1024 / (16 * 32) = 32

Para asociatividad de 64:

numSets = 16*1024 / (16 * 64) = 16

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Cuando se aumenta la asociatividad el hit ratio sube, aunque sube más en tamaños totales de caché más pequeños, en este caso al ser 16 KB como no es un tamaño ni muy grande ni muy pequeño hay el suficiente espacio para que quepan todos los datos. Y podemos ir viendo como si disminuye el miss ratio a medida que aumenta la asociatividad, a pesar de que a partir de asociatividad de 8 vías el miss ratio se estabiliza y no baja. Esto se debe a que se disminuye el numero de conjuntos y hay menos bloques en los que buscar.

TAREA 6: Asociatividad IL1

En esta tarea hemos comprobado la tasa de fallos de la caché de datos de primer nivel, IL1, a medida que su asociatividad aumenta y que tiene la configuración de un tamaño total de 16 Kbytes, un tamaño de línea de 16 bytes y una política de reemplazo LRU.
Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para asociatividad de 1:

numSets = 16*1024 / (16 * 1) = 1024

Para asociatividad de 2:

numSets = 16*1024 / (16 * 2) = 512

Para asociatividad de 4 :

numSets = 16384 / (16 * 4) = 256

Para asociatividad de 8:

numSets = 16*1024 / (16 * 8) = 128

Para asociatividad de 16:

numSets = 16*1024 / (16 * 16) = 64

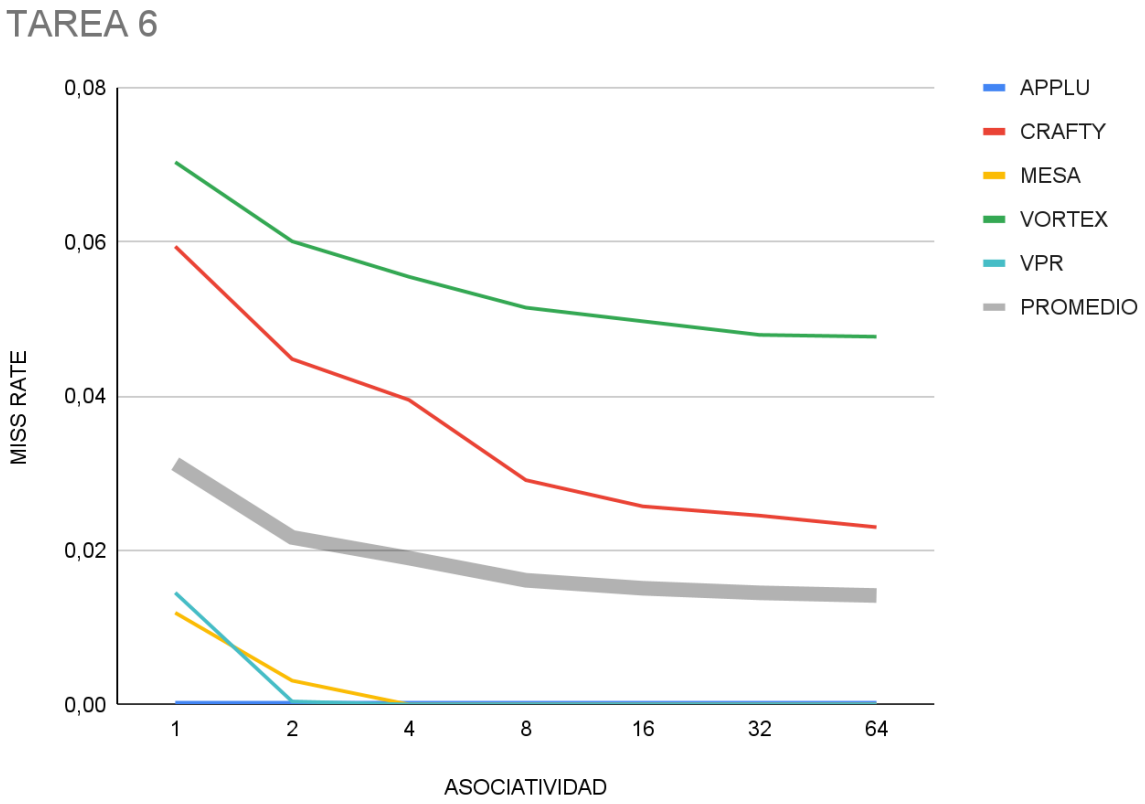
Para asociatividad de 32:

numSets = 16*1024 / (16 * 32) = 32

Para asociatividad de 64:

numSets = 16*1024 / (16 * 64) = 16

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Cuando se aumenta la asociatividad el hit ratio sube, aunque sube más en tamaños totales de caché más pequeños, en este caso al ser 16 KB como no es un tamaño ni muy grande ni muy pequeño hay el suficiente espacio para que quepan todos los datos. Y podemos ir viendo como si disminuye el miss ratio a medida que aumenta la asociatividad, a pesar de que a partir de asociatividad de 8 vías el miss ratio se estabiliza y no baja mucho.

TAREA 7: Tamaño Bloque DL1

En esta tarea hemos comprobado la tasa de fallos de la caché de datos de primer nivel, DL1, a medida que su tamaño de bloque aumenta y que tiene la configuración de un tamaño total de 16 Kbytes, una asociatividad de 1 (directa) y una política de reemplazo LRU.

Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para tamaño de bloque 8 bytes:

numSets = 16*1024 / 8 = 2048

Para tamaño de bloque 16 bytes:

numSets = 16*1024 / 16 = 1024

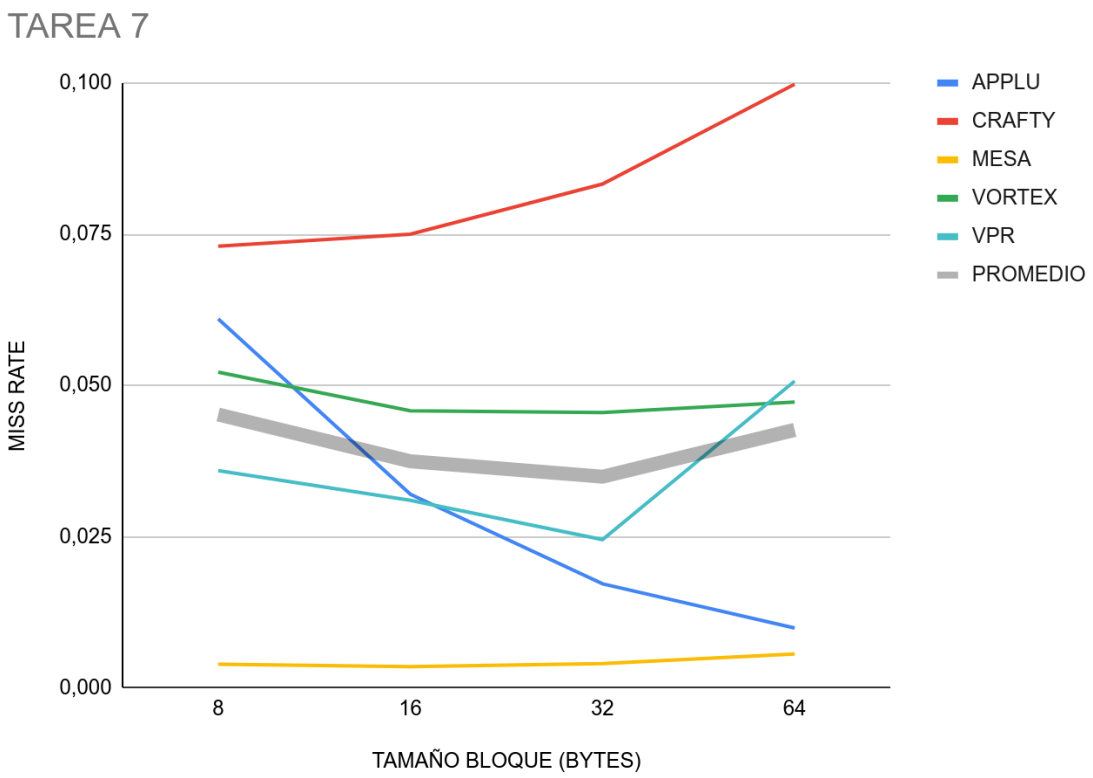
Para tamaño de bloque 32 bytes:

numSets = 16*1024 / 32 = 512

Para tamaño de bloque 64 bytes:

numSets = 16*1024 / 64 = 256

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Al aumentar el tamaño de bloque se debería reducir el miss ratio por localidad espacial pero como tenemos un tamaño fijo de caché al aumentar el tamaño de bloque estamos reduciendo el número de bloques por lo que los datos tendrían que competir por ellos (polución) y aumentando así el miss ratio.

TAREA 8: Tamaño Bloque IL1

En esta tarea hemos comprobado la tasa de fallos de la caché de instrucciones de primer nivel, DL1, a medida que su tamaño de bloque aumenta y que tiene la configuración de un tamaño total de 16 Kbytes, una asociatividad de 1 (directa) y una política de reemplazo LRU.

Para ejecutar la configuración de la caché hemos tenido que calcular el número de sets ya que varía para cada tamaño, hemos utilizado las siguientes fórmulas:

numLineas = numSets * asociatividad

tamañoTotal = numLineas * blockSize

Como los tamaños totales los sabemos, lo que nos falta por saber es el número de sets para poder configurar bien la cache para el simulador. Los cálculos son los siguientes:

numSets = tamañoTotal / (blockSize * asociatividad)

Para tamaño de bloque 8 bytes:

numSets = 16*1024 / 8 = 2048

Para tamaño de bloque 16 bytes:

numSets = 16*1024 / 16 = 1024

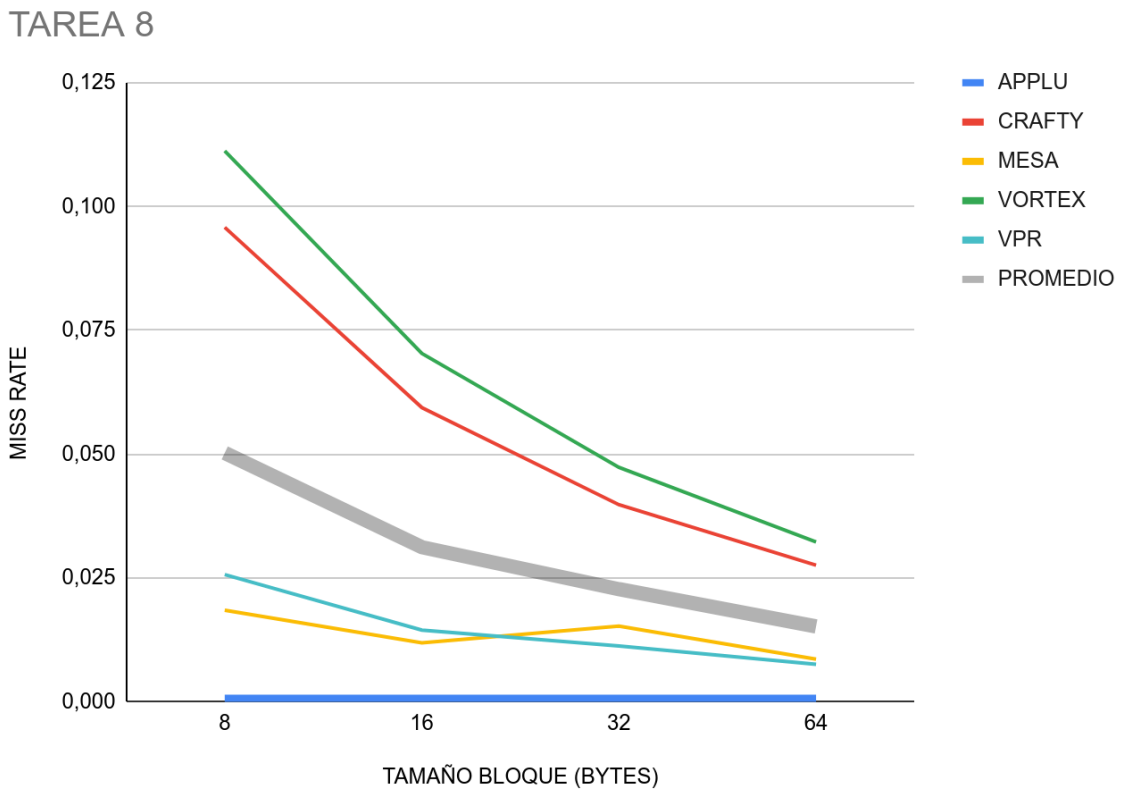
Para tamaño de bloque 32 bytes:

numSets = 16*1024 / 32 = 512

Para tamaño de bloque 64 bytes:

numSets = 16*1024 / 64 = 256

En la siguiente gráfica se ven los resultados de cada benchmark para cada tamaño.



RESULTADOS

Al aumentar el tamaño del bloque en la caché IL1 hacemos que haya mucho más espacio en un mismo bloque para que quepan las instrucciones enteras en uno mismo y no tengan que ocupar más de un bloque. Como se ve en el gráfico, a medida que se hace grande el tamaño de bloque hay más instrucciones en esta memoria, por lo tanto hay menos fallos en ella, se aumenta el hit rate.

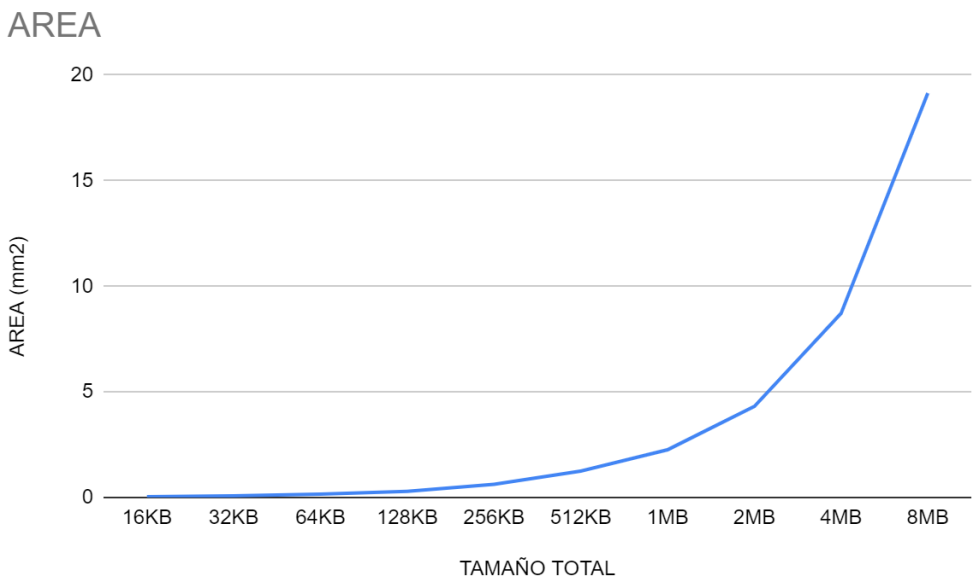
FASE 2: Tiempo de acceso, área y consumo

En esta fase estudiaremos el tiempo de acceso, el área y consumo de una caché, para realizar este estudio utilizaremos el simulador Cacti. El tiempo de acceso es lo que tardaría en obtener un resultado de la caché. El área es el tamaño necesario de silicio para implementar la caché. El consumo está dividido en 2 tipos, el estático y el dinámico. El estático es el que se produce en un procesador por el mero hecho de tener transistores. Y el dinámico es el que se produce por el mero hecho de que un transistor conmuta, es decir pasa de 0 a 1 o de 1 a 0.

TAREA 9: Tamaño Total Caché

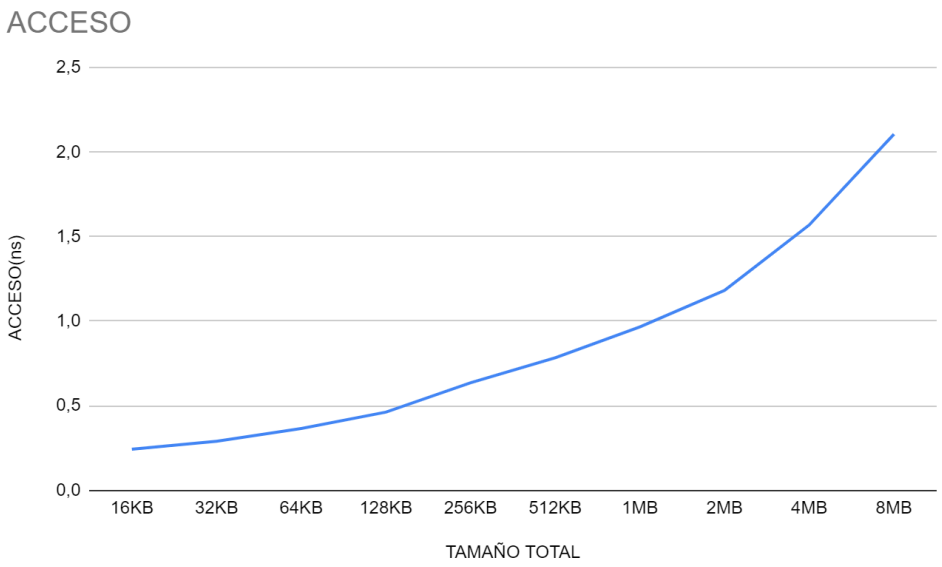
En esta tarea hemos analizado cómo varía el área, el tiempo de acceso y el consumo de la caché a medida que su tamaño total aumenta. Utilizaremos un tamaño de bloque de 16 bytes, asociatividad de 1 vía y escala de integración de 32 nm como valores base. Para analizar estos valores hemos utilizado el simulador Cacti al que le hemos pasado como entrada un fichero de configuración que contiene la configuración mencionada anteriormente pero cambiando el tamaño total.
En las siguientes gráficas se pueden ver los resultados para el área, el tiempo de acceso y consumo, respectivamente.

ÁREA



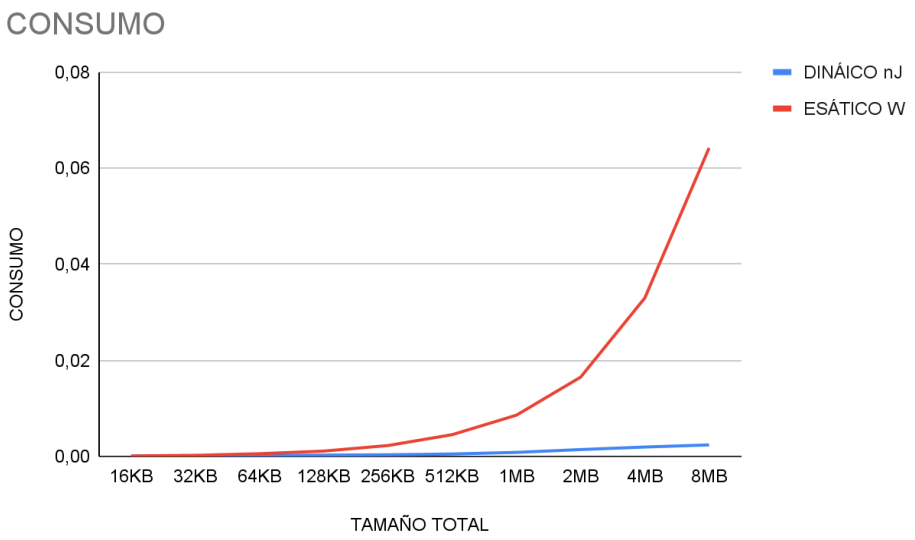
La gráfica muestra cómo a medida que aumenta el tamaño total también lo hace el área porque al aumentar el tamaño total se pueden guardar más datos.

ACCESO



La gráfica muestra cómo a medida que aumenta el tamaño total también lo hace el tiempo de acceso ya que ahora hay más entradas en las que buscar.

CONSUMO



En la gráfica se ve el comportamiento del consumo estático y el consumo dinámico.
Como el consumo estático depende de los transistores, a medida que aumenta el tamaño total de la caché necesitaremos más transistores, en consecuencia el consumo estático también lo hará.
El consumo dinámico, en cambio, no varía tanto ya que el uso de transistores no varía mucho, aumentar el tamaño de caché no implica que computen más.

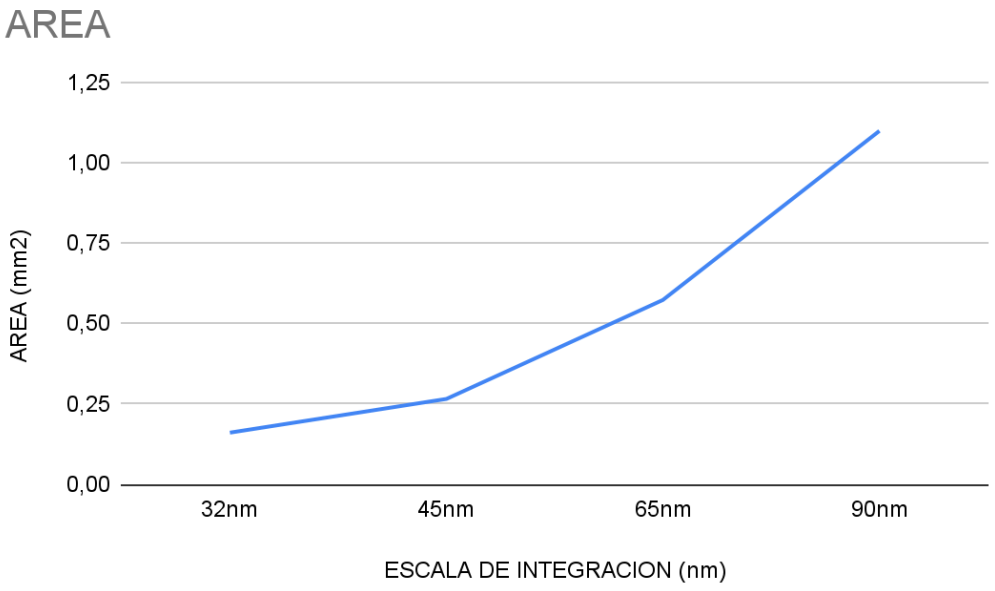
TAREA 10: Escala de integración

En esta tarea hemos analizado cómo varía el área, el tiempo de acceso y el consumo de la caché a medida que su escala de integración aumenta. La escala de integración es el tamaño de los transistores que tiene la cpu. Normalmente se usa una escala de integración de 90nm.

Utilizaremos un tamaño de bloque de 16 bytes, asociatividad de 1 vía y un tamaño total de 64 KB como valores base. Para analizar estos valores hemos utilizado el simulador Cacti al que le hemos pasado como entrada un fichero de configuración que contiene la configuración mencionada anteriormente pero cambiando la escala de integración.

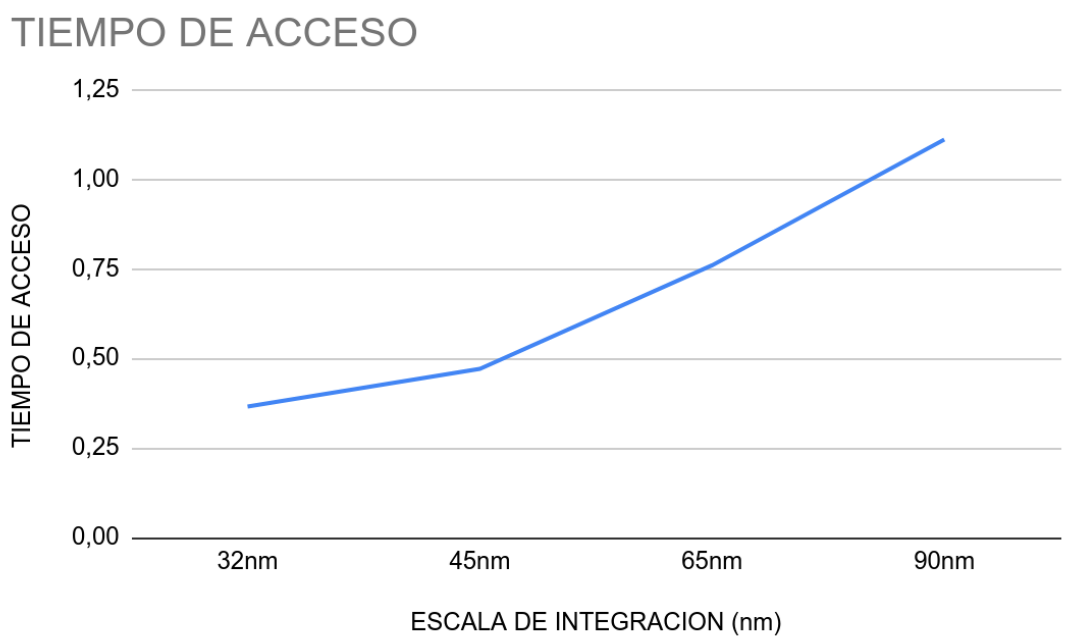
En las siguientes gráficas se pueden ver los resultados para el área, el tiempo de acceso y consumo, respectivamente.

ÁREA



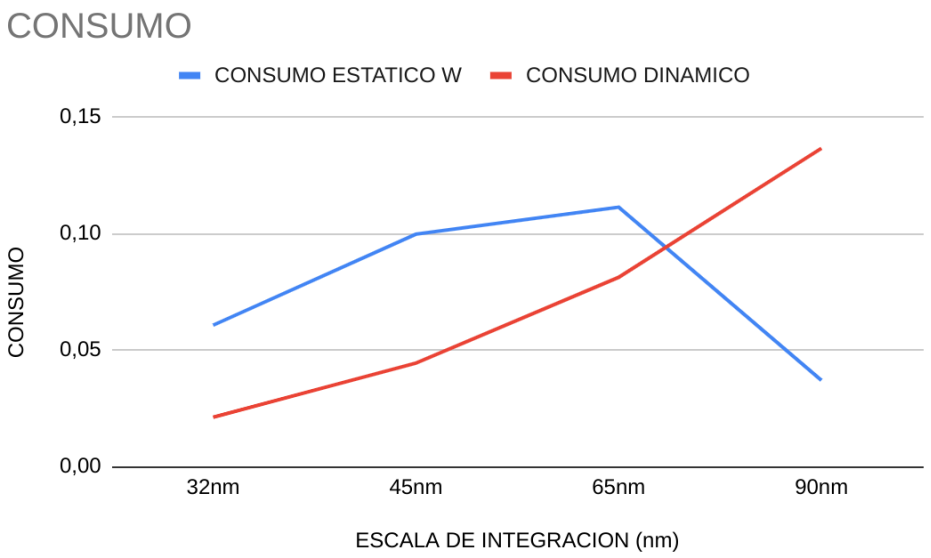
Como indica el gráfico, a mayor tamaño tenga la escala de integración se va a aumentar el área ya que aumenta el tamaño de transistores.

ACCESO



La gráfica muestra cómo a medida que aumenta el tamaño de la escala de integración también lo hace el tiempo de acceso ya que ahora hay más entradas en las que buscar.

CONSUMO



En la gráfica se ve el comportamiento del consumo estático y el consumo dinámico.

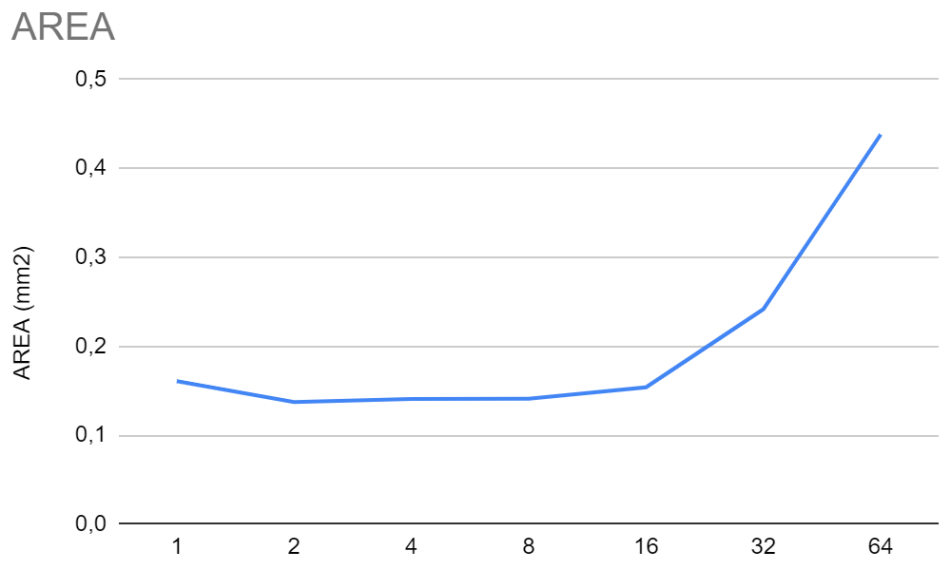
Como el consumo estático depende de los transistores, a medida que aumenta el tamaño de ellos, hay menos transistores para el mismo espacio, por lo tanto el consumo estático decae.

El consumo dinámico, en cambio, aumenta ya que a medida de que el tamaño del transistor aumenta, caben menos y habría que usarlos más.

TAREA 11: Asociatividad Caché

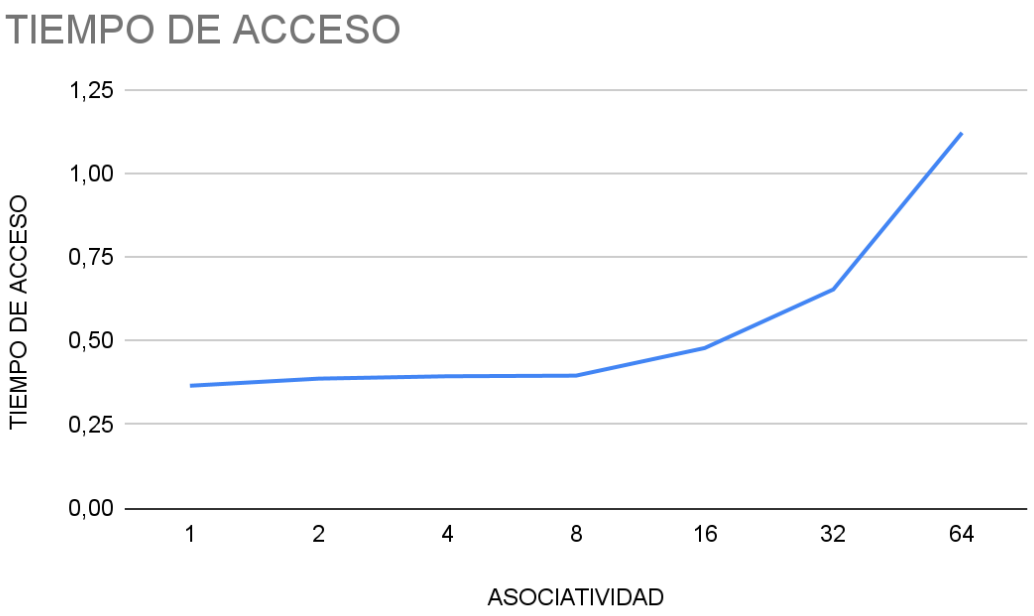
En esta tarea hemos analizado cómo varía el área, el tiempo de acceso y el consumo de la caché a medida que su asociatividad aumenta. Utilizaremos un tamaño de bloque de 16 bytes, un tamaño total de 64 KB y escala de integración de 32 nm como valores base. Para analizar estos valores hemos utilizado el simulador Cacti al que le hemos pasado como entrada un fichero de configuración que contiene la configuración mencionada anteriormente pero cambiando la asociatividad.
En las siguientes gráficas se pueden ver los resultados para el área, el tiempo de acceso y consumo, respectivamente.

ÁREA



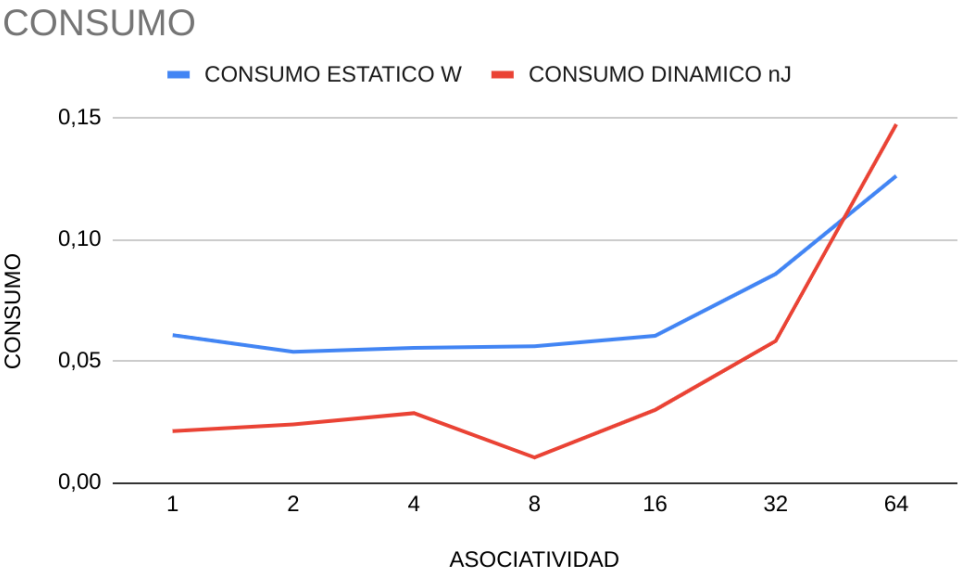
Como podemos ver en la gráfica el área aumenta a medida que el número de vías se hace grande. Esto se debe a que el área de la caché no depende solo del tamaño de esta sino que también hay que tener en cuenta el tamaño del tag, aumentando este a medida que hay más vías.

ACCESO



La gráfica muestra cómo a medida que aumenta la asociatividad también lo hace el tiempo de acceso ya que cada vez habrá más entradas por conjuntos en las que buscar.

CONSUMO



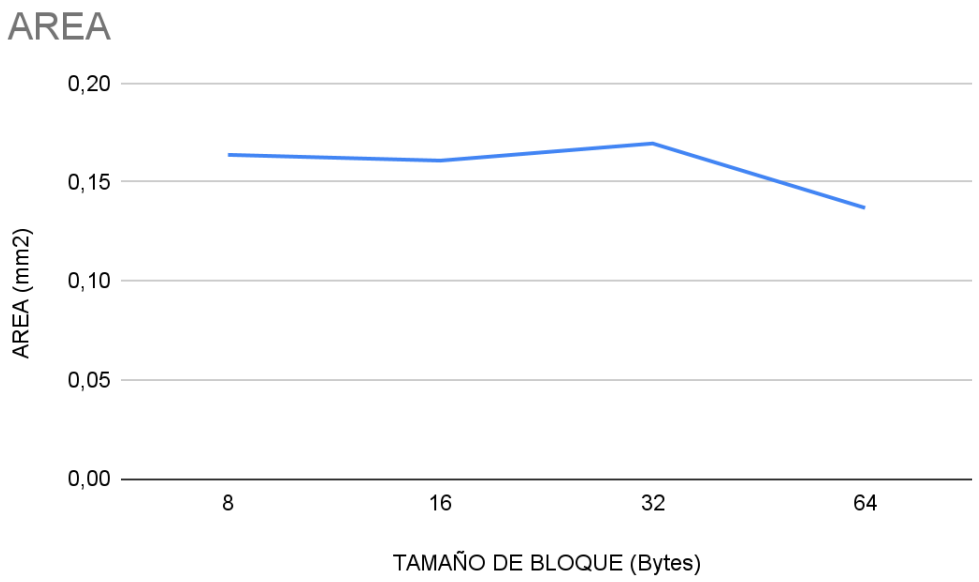
En la gráfica se ve el comportamiento del consumo estático y el consumo dinámico.
Como el consumo estático depende de los transistores, como el número de bits del tag aumenta, necesitaremos más transistores.

El consumo dinámico también aumenta ya que a medida que el número de asociatividad aumenta el número de conjuntos disminuye y hay más entradas en las que acceder, aumentando así el uso de los transistores.

TAREA 12: Tamaño Bloque

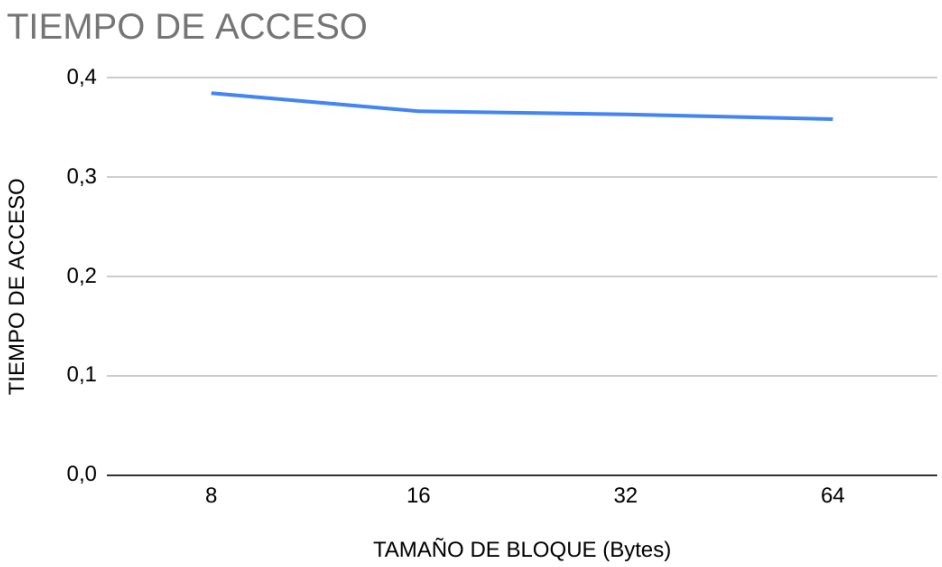
En esta tarea hemos analizado cómo varía el área, el tiempo de acceso y el consumo de la caché a medida que su tamaño de bloque aumenta. Utilizaremos un tamaño total de 64 KB, asociatividad de 1 vía y escala de integración de 32 nm como valores base. Para analizar estos valores hemos utilizado el simulador Cacti al que le hemos pasado como entrada un fichero de configuración que contiene la configuración mencionada anteriormente pero cambiando el tamaño bloque.
En las siguientes gráficas se pueden ver los resultados para el área, el tiempo de acceso y consumo, respectivamente.

ÁREA



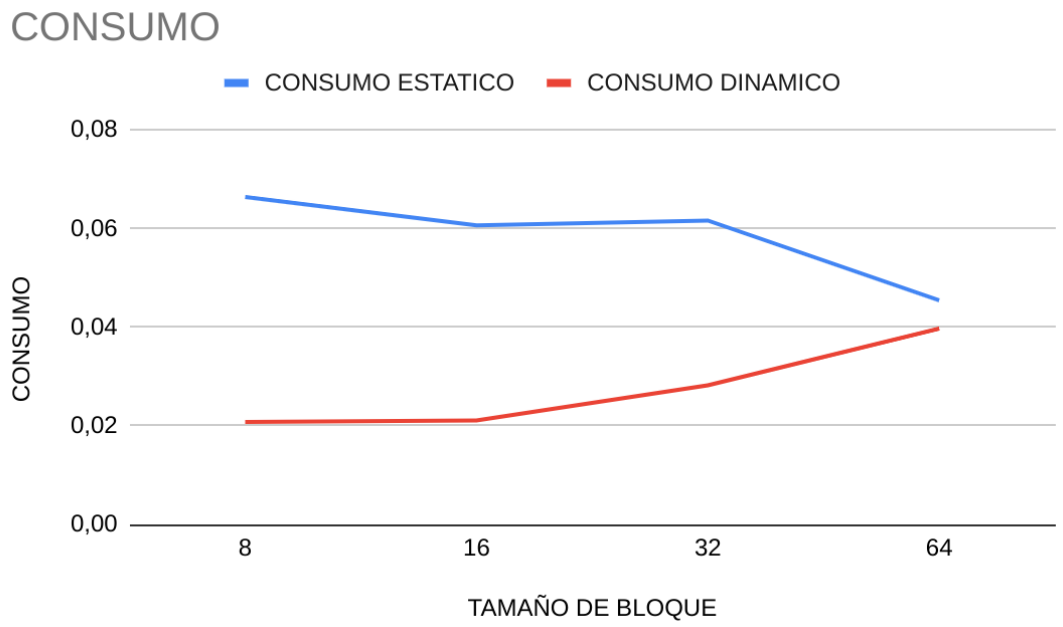
Como podemos ver en la gráfica el área se mantiene bastante estable ya que lo que estamos aumentado es el tamaño del bloque, por lo que la caché seguirá con un tamaño total fijo y el número de bits para el tag también disminuye.

ACCESO



Como podemos ver en la gráfica a medida que el tamaño del bloque va aumentando el tiempo de acceso disminuye un poco porque al estar aumentando el tamaño de bloque con una medida total fija de caché se reduce el número de bloques de la caché y por tanto el número de entradas de esta.

CONSUMO



En la gráfica se ve el comportamiento del consumo estático y el consumo dinámico.
Como el consumo estático depende de los transistores, como el tamaño total de la caché no varía pero si el tag, su consumo estático disminuye.
El consumo dinámico, en cambio, si aumenta ya que a medida que el tamaño del bloque aumenta el uso de transistores también.

FASE 3: Análisis de procesadores comerciales

TAREA 13: Análisis Cachés

El core del procesador ARM Cortex-A72 se compone de una caché de datos y otra de instrucciones de primer nivel y una de datos de segundo nivel.

Caché de datos de primer nivel (dl1)

- tamaño total: 32 Kb
- asociatividad: 2 vías
- tamaño de bloque: 64 bytes
- política de reemplazo: LRU

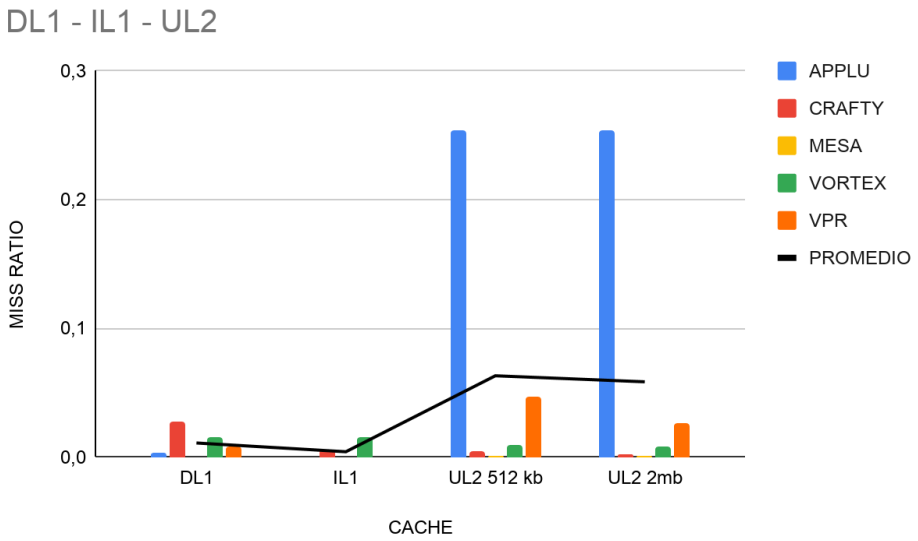
Caché de instrucciones de primer nivel (il1)

- tamaño total: 48 Kb
- asociatividad: 3 vías
- tamaño de bloque: 64 bytes
- política de reemplazo: LRU

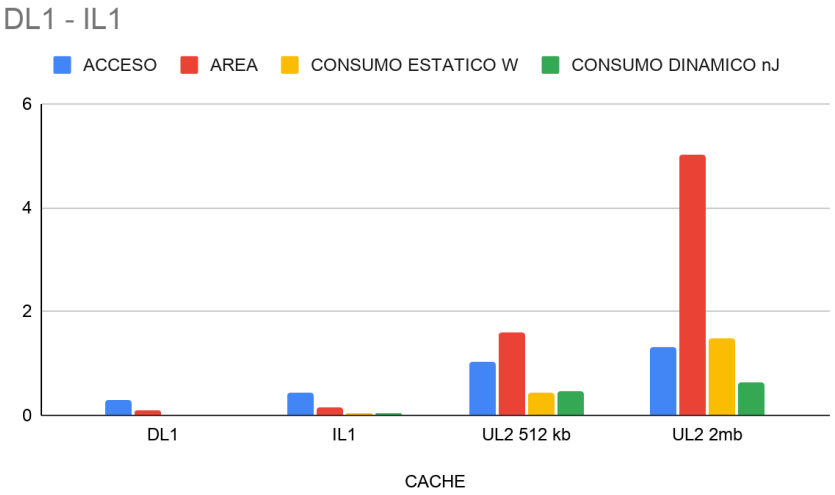
Caché de datos de segundo nivel (ul2)

- tamaño total: desde 512 Kb hasta 2Mb
- asociatividad: 16 vías
- tamaño de bloque: 64 bytes
- política de reemplazo: pseudo-LRU

Tamaño semiconductores → 16 nanómetros
Para hacer las simulaciones, como simplescalar y cacti no aceptan asociatividades que no sean potencias de 2 hemos optado por usar una asociatividad de 4 para la IL1.



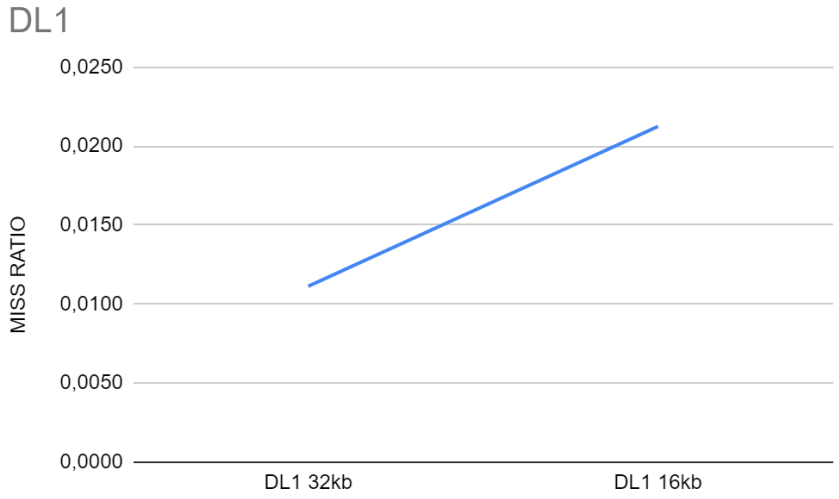
Los resultados obtenidos en la gráfica son los esperados ya que podemos ver que las cachés de primer nivel fallan menos que las de segundo nivel.



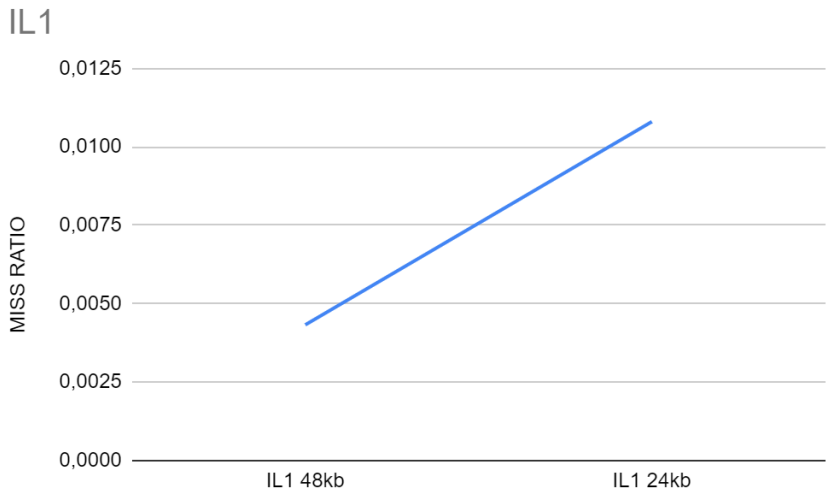
El análisis del área nos da los resultados esperados ya que las cachés que tienen menos tamaño total ocupan menos área que las de un tamaño mayor, por la parte del tiempo de acceso pasa lo mismo como todas tienen el mismo tamaño de bloque las que tienen mayor tamaño total tienen más entradas. El tiempo de acceso aumenta en la caché de segundo nivel ya que para poder acceder a esta se ha tenido que acceder primero a la de primer nivel.

TAREA 14: Dividir Capacidad DL1 y IL1

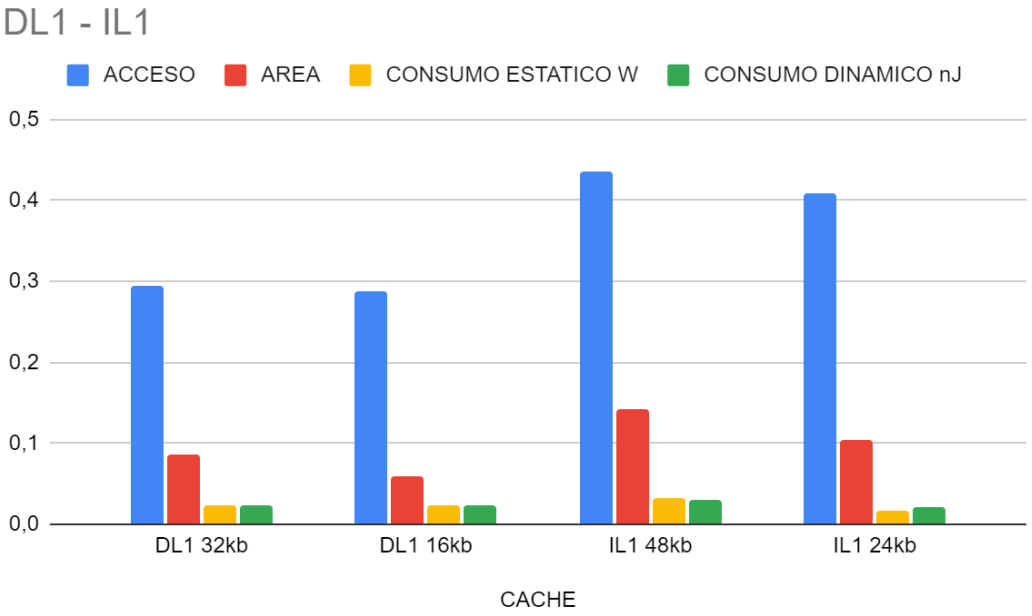
Para esta tarea hemos reducido el tamaño total de las caches de primer nivel pasando de 32 Kb de dl1 a 16 Kb y de 48 Kb a 24 Kb de la il1. Hemos repetido todas las simulaciones para las caches con los nuevos tamaños y hemos generado nuevas gráficas comparando los resultados anteriores con los nuevos.



Para la caché de datos de primer nivel podemos ver que con la mitad de tamaño falla el doble. Esto se debe a que hay menos espacio para poder almacenar los datos y obliga a tener que ir a buscarlos al siguiente nivel.



Para la caché de instrucciones de primer nivel podemos ver que con la mitad de tamaño falla el doble. Esto se debe a que hay menos espacio para poder almacenar los datos y obliga a tener que ir a buscarlos al siguiente nivel.



En el caso de tiempo de acceso, área y consumo, vemos que al reducir el tamaño el tiempo de acceso también se reduce muy poco. El área obviamente se reduce y el consumo se mantiene bastante estable.

Viendo los resultados, nos quedamos con las cachés más grandes ya que el miss ratio es la mitad de pequeño y el tiempo de acceso no hace una variación notable, el área tampoco y los consumos se mantienen.

TAREA 15: Comentario sobre el procesador

El Cortex A72 realiza instrucciones fuera de orden y no espera a que termine una para empezar una nueva, por lo que tenemos un procesador escalar multiciclo. Este procesador es mucho más rápido que el modelo anterior ya que sigue manteniendo el mismo tiempo de ciclo pero al ser multiciclo ya no se realiza una instrucción entera por ciclo sino que ahora se dividirán las instrucciones en tareas y se realizará una tarea por ciclo. La ventaja es que puede realizar múltiples tareas que requieran de recursos distintos de diferentes instrucciones a la vez.

Web-Grafía

ARM Cortex-A72

https://developer.arm.com/documentation/100095/0003/Level-1-Memory-System/About-the-L1-memory-system?lang=en