UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

LABORATORIO 3 ORGANIZACIÓN DE COMPUTADORES

CACHE

DANY RUBIANO JIMENEZ

Profesores: Felipe Garay
Erika Rosas
Nicolás Hidalgo
Ayudante: Ian Mejias

Santiago - Chile
6 de marzo de 2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

CAPÍTULO 1. INTRODUCCIÓN

1.1 MOTIVACIÓN Y ANTECEDENTES

En la actualidad se busca mejorar los procesos que realizan los programas de tal forma que se optimice la velocidad de ejecución. Uno de estos procesos yace en la forma como el programa accede a los datos en memoria, entonces es importante saber como sacar provecho de los componentes de hardware que se tienen, de forma que dada una implementación se optimice este proceso. En este laboratorio, se busca analizar como diferentes implementaciones de un programa interactúan con la memoria de forma que se encuentren las tasas de miss y de hits asociadas a la búsqueda de los datos en caché. Como lo es habitual en la asignatura, se utiliza el simulador Mars de MIPS para el desarrollo de las distintas implementaciones para el programa requerido.

1.2 OBJETIVOS

El objetivo del presente documento es exponer al lector el desarrollo y resultado de las implementaciones del programa ”Fold” a través de arreglos y de listas enlazadas, tomando en cuenta la forma en la que estas implementaciones acceden a los datos en memoria principal.

1.3 ORGANIZACIÓN DEL DOCUMENTO

Para cumplir con los objetivos respectivos el presente documento distribuye su información de la manera siguiente: Primero se encuentra un marco teórico en el cual se expone al lector los conceptos teóricos que son utilizados a lo largo del informe con el fin de facilitar la comprensión del mismo.

Posteriormente se presenta el proceso del desarrollo del programa requerido, seguido del análisis de los resultados obtenidos sobre el manejo del cache. Por último se presentan las conclusiones con las reflexiones que conllevo el trabajo realizado lo cual también incluye menciones a los procesos utilizados.

CAPÍTULO 2. MARCO TEÓRICO

2.1 LISTA ENLAZADA

Una lista enlazada es un tipo de dato autorreferenciado que contiene un puntero o enlace a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

2.1.1 Listas enlazadas lineales

2.1.1.1 Listas simples enlazadas

Es una lista enlazada de nodos, donde cada nodo tiene un único campo de enlace. Una variable de referencia contiene una referencia al primer nodo, cada nodo (excepto el último) enlaza con el nodo siguiente, y el enlace del último nodo contiene para indicar el final de la lista. Aunque normalmente a la variable de referencia se la suele llamar top, se le podría llamar como se desee

2.1.1.2 Listas doblemente enlazadas

Artículo principal: Lista doblemente enlazada Un tipo de lista enlazada más sofisticado es la lista doblemente enlazada o lista enlazadas de dos vías. Cada nodo tiene dos enlaces: uno apunta al nodo anterior, o apunta al valor NULL si es el primer nodo; y otro que apunta al nodo siguiente, o apunta al valor NULL si es el último nodo. En algún lenguaje de muy bajo nivel, XOR-Linking ofrece una vía para implementar listas doblemente enlazadas, usando una sola palabra para ambos enlaces, aunque esta técnica no se suele utilizar.

2.1.2 Listas enlazadas circulares

En una lista enlazada circular, el primer y el último nodo están unidos juntos. Esto se puede hacer tanto para listas enlazadas simples como para las doblemente enlazadas. Para recorrer una lista enlazada circular podemos empezar por cualquier nodo y seguir la lista en cualquier dirección hasta que se regrese hasta el nodo original. Desde otro punto de vista, las listas enlazadas circulares pueden ser vistas como listas sin comienzo ni fin. Este tipo de listas es el más usado para dirigir buffers para ”ingerir” datos, y para visitar todos los nodos de una lista a partir de uno dado.

2.1.2.3 Listas enlazadas simples circulares

Cada nodo tiene un enlace, similar al de las listas enlazadas simples, excepto que el siguiente nodo del último apunta al primero. Como en una lista enlazada simple, los nuevos nodos pueden ser solo eficientemente insertados después de uno que ya tengamos referenciado. Por esta razón, es usual quedarse con una referencia solamente al último elemento en una lista enlazada circular simple, esto nos permite rápidas inserciones al principio, y también permite accesos al primer nodo desde el puntero del último nodo. 1

2.1.2.4 Listas enlazadas doblemente circulares

En una lista enlazada doblemente circular, cada nodo tiene dos enlaces, similares a los de la lista doblemente enlazada, excepto que el enlace anterior del primer nodo apunta al último y el enlace siguiente del último nodo, apunta al primero. Como en una lista doblemente enlazada, las inserciones y eliminaciones pueden ser hechas desde cualquier punto con acceso a algún nodo cercano. Aunque estructuralmente una lista circular doblemente enlazada no tiene ni principio ni fin, un puntero de acceso externo puede establecer el nodo apuntado que está en la cabeza o al nodo cola, y así mantener el orden tan bien como en una lista doblemente enlazada. (Wikipedia, s.f.-a).

2.2 PRINCIPIO DE LOCALIDAD

La localidad de las referencias, también conocida como el principio de localidad, es un fenómeno según el cual, basándonos en el pasado reciente de un programa podemos predecir con una precisión razonable qué instrucciones y datos utilizará en un futuro próximo. Los casos más importantes de localidad son la localidad espacial, la localidad secuencial y la localidad temporal.

2.2.1 Localidad Temporal

Si en un momento una posición de memoria particular es referenciada, entonces es muy probable que la misma ubicación vuelva a ser referenciada en un futuro cercano. Existe proximidad temporal entre las referencias adyacentes a la misma posición de memoria. En este caso es común almacenar una copia de los datos referenciados en caché para lograr un acceso más rápido a ellos.

2.2.2 Localidad Espacial

Si una localización de memoria es referenciada en un momento concreto, es probable que las localizaciones cercanas a ella sean también referenciadas pronto. Existe localidad espacial entre las posiciones de memoria que son referenciadas en momentos cercanos. En este caso es común estimar las posiciones cercanas para que estas tengan un acceso más rápido.

2.2.3 Localidad Secuencial

Las direcciones de memoria que se están utilizando suelen ser contiguas. Esto ocurre porque las instrucciones se ejecutan secuencialmente.

Para obtener beneficios de la gran frecuencia con la que ocurren casos de localidad espacial o temporal, muchos sistemas de memoria utilizan una jerarquía de niveles de memoria. (Wikipedia, s.f.-b).

2.3 CACHE

En informática, la caché es la memoria de acceso rápido de una computadora, que guarda temporalmente los datos recientemente procesados (información).1 La memoria caché es un búfer especial de memoria que poseen las computadoras, que funciona de manera similar a la memoria principal, pero es de menor tamaño y de acceso más rápido. Es usada por el microprocesador para reducir el tiempo de acceso a datos ubicados en la memoria principal que se utilizan con más frecuencia.

2.3.1 Política de ubicación

Decide dónde debe colocarse un bloque de memoria principal que entra en la memoria caché. Las más

utilizadas son:

2.3.1.5 Mapeo Directo

Al bloque i-ésimo de memoria principal le corresponde la posición i módulo n, donde n es el número de bloques de la memoria caché. Cada bloque de la memoria principal tiene su posición en la caché y siempre en el mismo sitio. Su inconveniente es que cada bloque tiene asignada una posición fija en la memoria caché y ante continuas referencias a palabras de dos bloques con la misma localización en caché, hay continuos fallos habiendo sitio libre en la caché.

2.3.1.6 Full Asociativo

Los bloques de la memoria principal se alojan en cualquier bloque de la memoria caché, comprobando solamente la etiqueta de todos y cada uno de los bloques para verificar acierto. Su principal inconveniente es la cantidad de comparaciones que realiza.

2.3.1.7 Asociativo por Conjuntos

Cada bloque de la memoria principal tiene asignado un conjunto de la caché, pero se puede ubicar en cualquiera de los bloques que pertenecen a dicho conjunto. Ello permite mayor flexibilidad que la correspondencia directa y menor cantidad de comparaciones que la totalmente asociativa.

2.3.2 Política de reemplazo

Determina qué bloque de memoria caché debe abandonarla cuando no existe espacio disponible para un bloque entrante. Básicamente hay cuatro políticas:

2.3.2.8 Aleatoria (Random)

El bloque es reemplazado de forma aleatoria.

2.3.2.9 FIFO

Se usa el algoritmo First In First Out (FIFO) (primero en entrar primero en salir) para determinar qué bloque debe abandonar la caché. Este algoritmo generalmente es poco eficiente.

2.3.2.10 Menos recientemente usado (LRU)

Sustituye el bloque que hace más tiempo que no se ha usado en la caché, traeremos a caché el bloque en cuestión y lo modificaremos ahí.

2.3.2.11 Menos frecuencias usadas (LFU)

Sustituye el bloque que ha experimentado menos referencias. (Wikipedia, s.f.-a).

CAPÍTULO 3. DESARROLLO

3.1 DESARROLLO DE LOS PROGRAMAS

3.1.1 Fold mediante arreglo

3.1.2 Fold mediante listas enlazadas

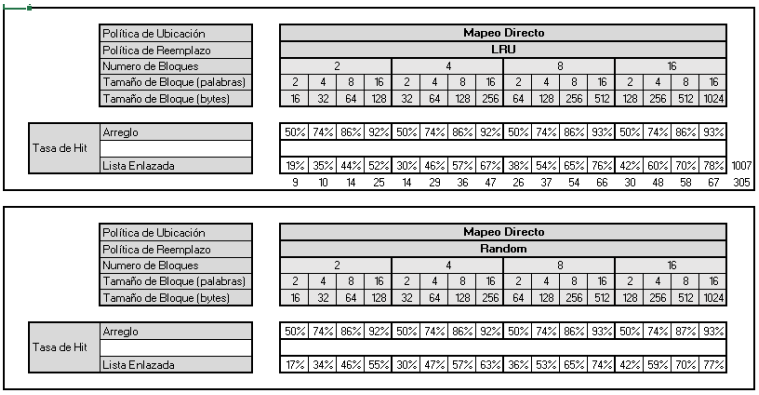
3.2 RESULTADOS

Debido a los problemas que se tienen para insertar elementos en la lista enlazada, tal como se detalló en la sección anterior, se toman siete elementos a considerar para realizar las pruebas requeridas. Estos elementos se presentan en la siguiente tabla.

Tabla 3.1: Elementos de prueba

168	695	415	988	90	496	84	1017	534	607
433	660	646	940	64	916	954	671	663	308
955	352	822	153	709	429	1011	475	394	956
400	718	370	961	742	457	1023	969	965	419
948	913	460	659	532	206	251	17	51	644
877	214	234	677	539	558	402	941	548	776
568	799	697	824	158	239	860	981	876	679
497	763	616	891	401	533	74	652	932	499
693	851	332	516	249	16	1020	549	618	43
873	847	746	282	637	906	794	238	487	276

Para estudiar la memoria caché del simulador MIPS, se fueron modificando las variables que permite observar el comportamiento del mismo respecto de las tasas de hit y miss en caché. Las variables que se modificar en el simulador MARS MIPS, son los tres tipos de caché antes explicados (Mapeo directo, full-asociativo y N-asociativo), el número de bloques, el tamaño de cada bloque (palabras), el tamaño del caché en bytes. Para el estudio de este documento, la información central radica en la tasa de hits en caché, de tal forma que una mayor tasa de hit, implicaría un mayor rendimiento del sistema debido a la baja tasa de miss.

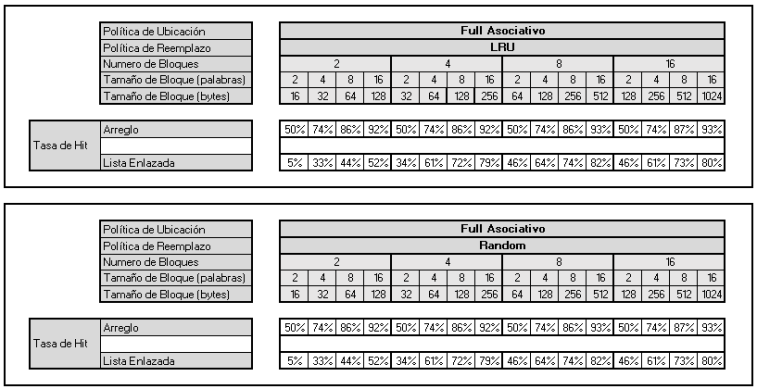


Política de Ubicación	Mapeo Directo															
Política de Reemplazo	LRU															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	86%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	19%	35%	44%	52%	30%	46%	57%	67%	38%	54%	65%	76%	42%	60%	70%	78%
	9	10	14	25	14	29	36	47	26	37	54	66	30	48	58	67

1007
305

Política de Ubicación	Mapeo Directo															
Política de Reemplazo	Random															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	87%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	17%	34%	46%	55%	30%	47%	57%	63%	36%	53%	65%	74%	42%	59%	70%	77%

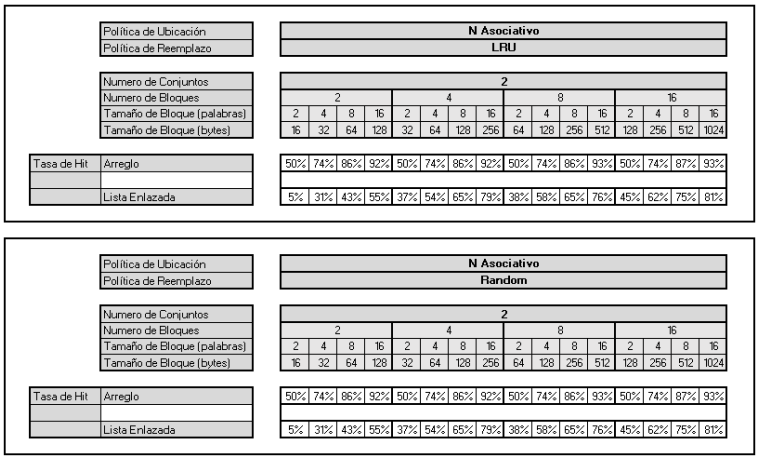
Figura 3-1: Pruebas con Caché Mapeo Directo



Política de Ubicación	Full Asociativo															
Política de Reemplazo	LRU															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	87%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	5%	33%	44%	52%	34%	61%	72%	79%	46%	64%	74%	82%	46%	61%	73%	80%

Política de Ubicación	Full Asociativo															
Política de Reemplazo	Random															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	87%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	5%	33%	44%	52%	34%	61%	72%	79%	46%	64%	74%	82%	46%	61%	73%	80%

Figura 3-2: Pruebas con Caché Full Asociativo



Política de Ubicación	N Asociativo															
Política de Reemplazo	LRU															
Numero de Conjuntos	2															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	87%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	5%	31%	43%	55%	37%	54%	65%	79%	38%	58%	65%	76%	45%	62%	75%	81%

Política de Ubicación	N Asociativo															
Política de Reemplazo	Random															
Numero de Conjuntos	2															
Numero de Bloques	2				4				8				16			
Tamaño de Bloque (palabras)	2	4	8	16	2	4	8	16	2	4	8	16	2	4	8	16
Tamaño de Bloque (bytes)	16	32	64	128	32	64	128	256	64	128	256	512	128	256	512	1024
Tasa de Hit Arreglo	50%	74%	86%	92%	50%	74%	86%	92%	50%	74%	86%	93%	50%	74%	87%	93%
Tasa de Hit																
Tasa de Hit Lista Enlazada	5%	31%	43%	55%	37%	54%	65%	79%	38%	58%	65%	76%	45%	62%	75%	81%

Figura 3-3: Pruebas con Caché N Asociativo

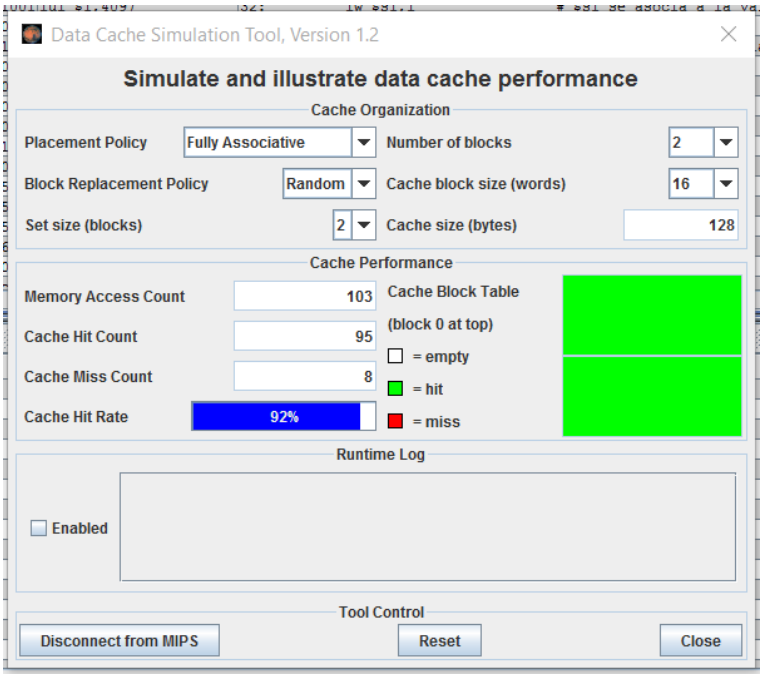


Figura 3-4: Ejemplo de prueba de caché en el fold para el arreglo

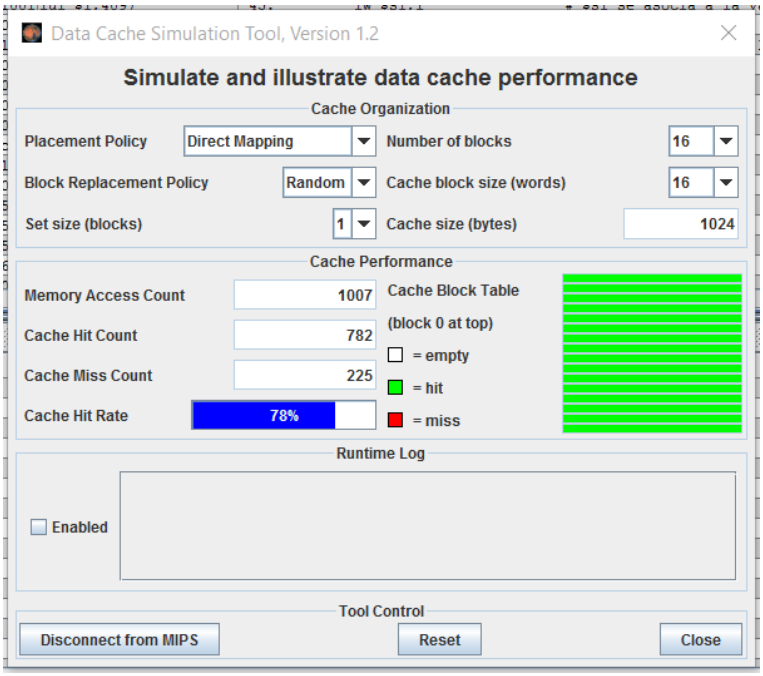


Figura 3-5: Ejemplo de prueba de caché en el fold para la lista enlazada incluyendo el llenado de elementos

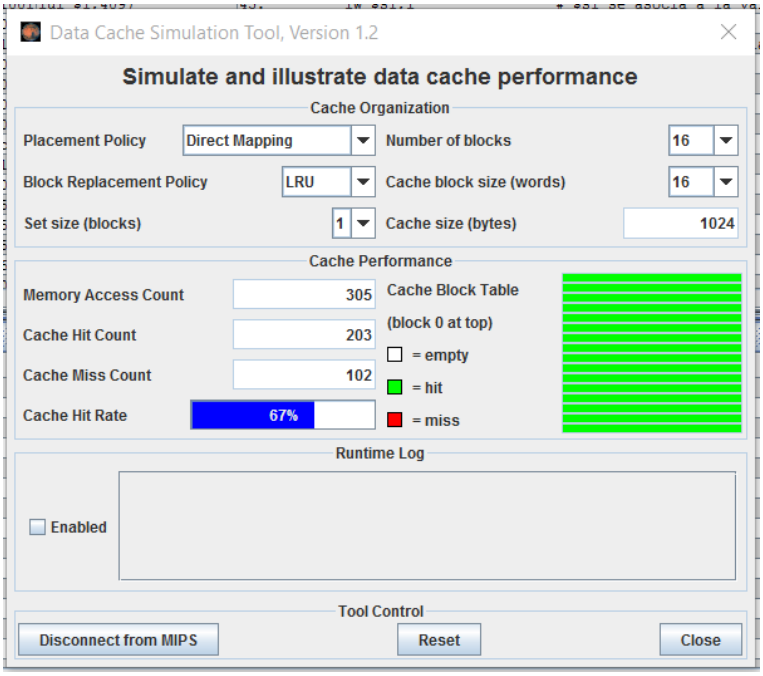


Figura 3-6: Ejemplo de prueba de caché para el llenado de elementos a través del arreglo

CAPÍTULO 4. ANÁLISIS DE LOS RESULTADOS

En este capítulo se analizan los resultados obtenidos con ambos algoritmos de ordenamiento, y se explica el por qué de los mismos, empleando los términos de localidad temporal y espacial según corresponda. Continuando con las políticas de reemplazo de bloques en caché, la teoría dice que lo más eficiente corresponde al uso de la política LRU, debido a que hace uso del concepto de localidad temporal, beneficiando a estar en caché a aquellos bloques recientes (que pueden ser requeridos nuevamente) y prioriza el sacar los bloques con más tiempo en caché sin ser utilizados, por lo tanto poseen menos posibilidades de ser usados que los de menos tiempo en caché. Al contrario que un reemplazo de bloques por medio de la aleatoriedad, no se fundamenta en ninguno de las localidades descritas en el informe, sino que puede sacar cualquier bloque que esté en caché aunque sea el más reciente y usado. Por otra parte la localidad espacial, puede ser incluida junto con la política LRU, añadiendo otra variable a la selección del bloque, esto es por ejemplo, que el(los) bloque(s) más recientemente usado y los próximos al mismo poseen una mayor prioridad para permanecer en caché al contrario de aquellos bloques menos recientemente usados y que no se encuentran próximos a los bloques de uso más reciente. Por lo recién explicado, se puede esperar que la tasa de Hit en los caché con la política LRU son más eficientes que uno con un protocolo de reemplazo aleatorio. Con los resultados obtenidos es posible observar que mientras mayor es la cantidad de bloques en caché, mayor es la tasa de Hit y por ende una menor tasa de miss, debido a que hay más bloques en los cuales poder almacenar la información de los programas en ejecución. La reducción de la tasa de Miss se debe a la mayor capacidad de almacenar más datos en caché, y como resultado, es muy probable que cuando el procesador necesite un dato, este se encuentre en caché por lo que se genera un hit, es decir que se encontró el dato en el primer nivel de la jerarquía de memoria, ahorrándose ciclos del procesador, dado que no hay castigo por miss. En resumen una mayor cantidad de bloques en caché es beneficioso para el rendimiento del sistema, ahorrándose operaciones de transferencia para traer un bloque de un nivel a otro.

CAPÍTULO 5. CONCLUSIONES

Es importante considerar la cantidad de bloques y el tamaño de los mismos cuando se desea estudiar la memoria caché. Mediante los conceptos de localidad es posible inferir y concluir resultados esperados de la memoria caché bajo diferentes políticas de reemplazo, de la estructura. Es importante mencionar que aunque en los resultados expuestos, la tasa de Hit va en aumento junto con el tamaño y cantidad de bloques, pero respecto del tamaño de los bloques, es cierto que disminuye la tasa de Miss, pero el castigo por el mismo es mayor, debido a que se necesitan más ciclos de reloj para poder traer a caché y entre las jerarquías de memoria bloques más grandes, por lo tanto tamaños grandes de bloque pueden alcanzar un límite de eficiencia por los mismos, dado que esta decrece luego de un aumento considerable de caché. Esta eficiencia se puede observar en el tiempo de respuesta del programa, que tiende a ser mayor para bloques muy grandes de caché. Por otra parte, los resultados que se obtienen casi nunca son iguales, debido a que el procesador ejecuta diversos procesos en conjunto al programa implementado para este laboratorio, por tanto los resultados obtenidos pueden ser similares considerando un equipo de características parecidas y cantidad equivalente de programas en ejecución, junto con la cantidad de enteros a ordenar.

CAPÍTULO 6. BIBLIOGRAFÍA

Wikipedia. (s.f.-a). Caché. https://es.wikipedia.org/wiki/Cach_(informtica).

Wikipedia. (s.f.-b). Cercanía de referencias. https://es.wikipedia.org/wiki/Cercana_de_referencias.

Wikipedia. (s.f.-c). Lista enlazada. https://es.wikipedia.org/wiki/Lista_enlazada.