Lab 2 | Ejercicio 2 (Informe)

Integrantes:

- Emanuel Nicolás Herrador (DNI 44.898.601)
- Juan Bratti (DNI 44.274.011)

Materia: Arquitectura de Computadoras 2023

Introducción

En este presente informe se presentarán los resultados obtenidos para los distintos tipos de procesadores a considerar (*in order* y *out of order*), respecto a varios tipos de caché de dato principal (*dcache*, cambiando cantidad de vías), y respecto a dos tipos principales de predictores de salto (*local* y *por torneos*), aplicándose la benchmark **simFisica** explicitada en el enunciado del ejercicio.

Además de todos estos datos, se realizará su respectivo análisis y justificación para entender por qué funciona de ese modo y cuál es la mejor combinación que podemos considerar para optimizar la *performance* para esta *benchmark*.

(b, c) Datos obtenidos para *predictor local* y 32kB para la caché de dato principal pero cambiando cantidad de vías

Comportamiento esperado

Antes de pensar cómo debería ser el comportamiento esperado que se debe cumplir en la simulación *teóricamente*, tengamos en cuenta los siguientes datos:

- Como $N = 64$ y consideramos datos de 8 bytes, entonces el tamaño de X y X_temp es de $64^2 * 8B = 32kB$ cada uno.

- En el *loop* principal (de las iteraciones), debemos tener en cuenta que en la primer parte se lee sobre los valores de X y se escribe sobre los de X_temp, mientras que en la segunda parte, se lee sobre los de X_temp y se escribe en los de X.

Ya teniendo esto en cuenta, notemos que si tuviéramos una caché de datos de 32kB y 1 sola vía, entonces los valores **siempre se estarían pisando** dado que en cada momento vamos a ver y buscar valores de los dos arreglos (uno para escribir y el otro para leer). Luego, esto genera una gran cantidad de *replacements* y *misses* en la caché.

En contraposición, si tenemos una caché pero de 2 o más vías, entonces la cantidad total de datos que va a cubrir va a ser mayor a 64kB, es decir, al espacio ocupado por los dos arreglos que necesitamos y consideramos. Motivo de ello, implica que la caché va a poder cubrir los valores de ambos arreglos sin pisar las posiciones ni los bloques de esta.

Por ello mismo, entonces, lo esperado es que:

- El *peor caso* sea el de 32kB y 1 vía, con una **gran diferencia** por sobre los demás casos.
- Los *mejores casos* serían los de 32kB y 2 o más vías, siendo el de 2 el más “chico” que es óptimo para nuestra solución ⇒ Porque cubre los dos arreglos.

Luego, se espera que la mejor configuración para la caché para **simFisica** sea de **32kB y 2 vías.**

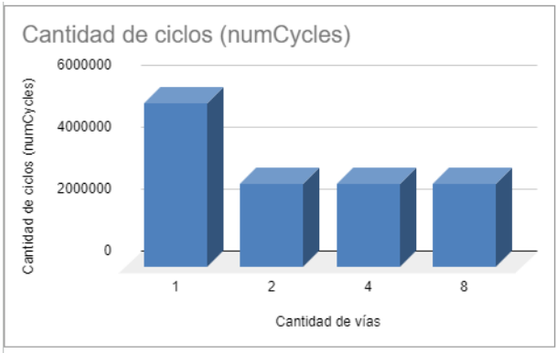
Datos obtenidos (resultados + gráficos)

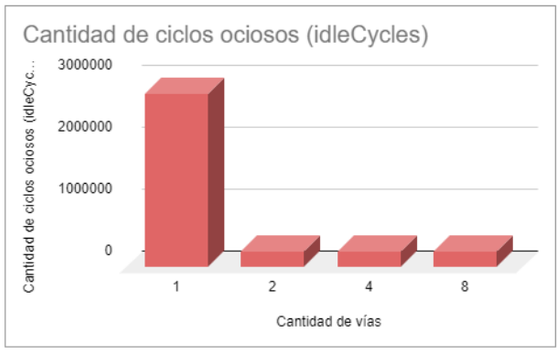
Respecto a los datos que se obtuvieron en la *simulación* de este punto, son los siguientes:

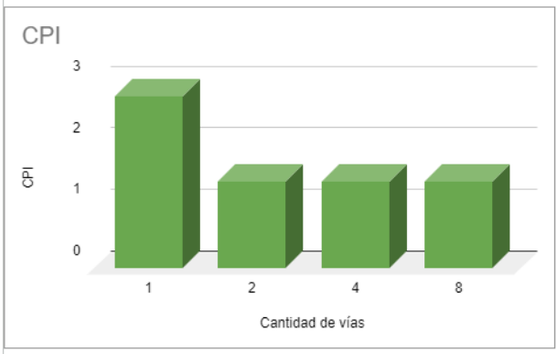
Tipo de caché	**Mapeo directo (1 vía)**			
Tipo de predictor	**Local**			
Tamaño de la caché	**32KB**			
Cantidad de vías	1	2	4	8
Cantidad de ciclos (numCycles)	5304518	2697610	2698818	2699398
Cantidad de ciclos ociosos (idleCycles)	2791021	246111	247063	247515
CPI	2,79	1,42	1,42	1,42
Conditional branchs predicted (condPredicted)	341098	341096	341096	341096
Conditional branchs not predicted (condIncorrect)	2047	2047	2047	2047
MissRate de la predicción	0,60%	0,60%	0,60%	0,60%
Hits de la caché principal de datos (overallHits)	136216	276554	276507	276475
Misses de la caché principal de datos (overallMisses)	151989	11663	11710	11742
MissRate de la caché principal de datos	52,74%	4,05%	4,06%	4,07%

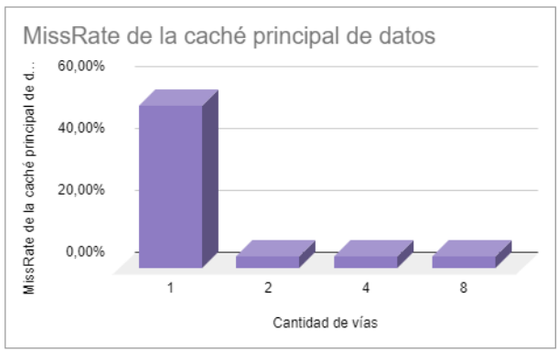
Si bien colocamos varios datos, para nuestro análisis nos concentraremos principalmente en **numCycles, idleCycles, CPI** y **MissRate de la caché.**

Motivo de ello, entonces, adjuntamos los gráficos que se utilizarán para referencia en el análisis:









Análisis de resultados

Podemos ver en las métricas obtenidas gracias a la simulación, que como mencionamos anteriormente, el caso de 2 vías en un tamaño de 32kB presenta un **gran aumento de la performance** respecto al de 1 vía, notando que:

- La cantidad de ciclos (**numCycles**) disminuye en un 49,15%.
- La cantidad de ciclos ociosos (**idleCycles**) disminuye en un 91,18%.
- El valor **CPI** (*clocks per instruction*) también disminuye en un 49,10%.
- La cantidad de **missRates** disminuye en un 92,33%.

Lo cual significa una mejora significativa en la optimización de la ejecución de la *benchmark.*

Como podemos ver, gracias a que las métricas mejoran, podemos concluir que toda *configuración con 2 o más vías en 32kB son las mejores soluciones*, siendo óptima la de menor tamaño (**2 vías + 32kB de dato**).

(d) Datos obtenidos para *predictor local* vs. *predictor por torneos* para caché de 32kB y 2 vías

Comportamiento esperado

En este punto, como queremos considerar la mejor configuración de caché (32kB de dato + 2 vías) con distintos tipos de *predictores de salto*, tenemos que tener en especial atención el tipo de código que consideramos en la benchmark de *daxpy*.

Como podemos ver en el código que dejamos adjuntado, la *benchmark* presenta varios bloques *for* y bloques *if/else*, generando que durante la ejecución se realicen muchísimos saltos de cada tipo (los relacionados con *for* y los relacionados con *if/else*).

En base a las características de los tipos de predictores, una primera idea a tener en cuenta es que los tipos de predictores locales comunes nos *ayudan* con los bloques *for* para predecir *taken* de las iteraciones, mientras que no ayudan con los bloques *if/else* dado que necesitan del *shift register* (i.e., saber los patrones de saltos anteriores para poder realizar una buena predicción usando la máquina de estados). En contraparte, los predictores *globales*, nos ayudan con los bloques *if/else* gracias a que se tienen en cuenta **todos** los saltos y sus patrones, pero empeora la predicción de *taken* para los *for*.

Motivo de ello, notamos que la mejor opción para realizar la ejecución de nuestro programa es el *predictor de saltos por torneos* dado que “trae lo mejor de los dos mundos”, dado que tiene un predictor local, uno global y uno que predice cuál va a ser el mejor resultado de los resultados predecidos por cada uno (para saber cuál de las dos opciones elegir),

Luego, esta configuración nos permite tener en cuenta tanto los *for* como los *if/else*.

Por ello, lo *esperado* es que la *performance* de la ejecución de nuestro programa aumente significativamente y el *missRate* de la predicción tenga una gran disminución.

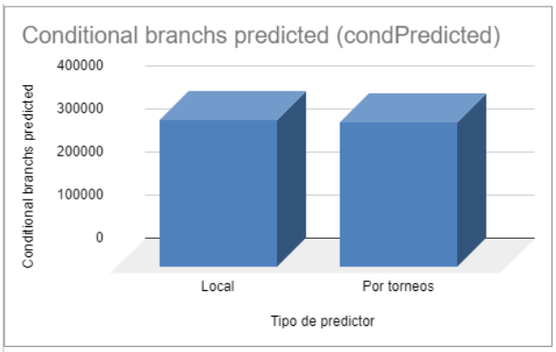
Datos obtenidos (resultados + gráficos)

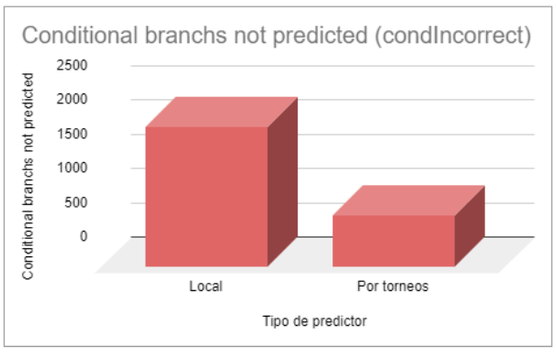
Respecto a los datos que se obtuvieron en la *simulación* de este punto, son los siguientes:

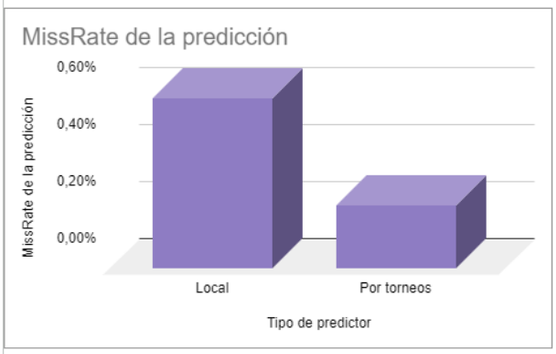
Tipo de caché	**Mapeo directo (1 vía)**	
Cantidad de vías	**2**	
Tamaño de la caché	**32KB**	
Tipo de predictor	Local	Por torneos
Cantidad de ciclos (numCycles)	2697610	2682897
Cantidad de ciclos ociosos (idleCycles)	246111	245755
CPI	1,42	1,41
Conditional branchs predicted (condPredicted)	341096	335975
Conditional branchs not predicted (condIncorrect)	2047	752
MissRate de la predicción	0,60%	0,22%
Hits de la caché principal de datos (overallHits)	276554	276554
Misses de la caché principal de datos (overallMisses)	11663	11663

Si bien colocamos varios datos, para nuestro análisis nos concentraremos principalmente en **condPredicted, condIncorrect** y **MissRate de la predicción**

Motivo de ello, entonces, adjuntamos los gráficos que se utilizarán para referencia en el análisis:







Análisis de resultados y *comparativa*

Teniendo en cuenta las métricas + lo analizado en el comportamiento que sería esperado, podemos observar que el predictor por torneo mejora la *performance* de la ejecución respecto al local. Esto se puede visualizar en los resultados obtenidos, específicamente en:

- El **missRate** disminuye en un 62,56%.
- **condIncorrect** disminuye en un 63,26%.

Entonces, podemos concluir que el predictor por torneo es el mejor, e incluso disminuye el CPI.

(e) Datos obtenidos para *predictor por torneos*, caché de 32kB y 2 vías, y procesador *out of order* vs. *in order*

Comportamiento esperado

En este caso, si consideramos la mejor configuración que tenemos hasta el momento (32kB de dato + 2 vías para la dcaché + predictor por torneos), entonces podemos concentrarnos en ver cuál es la diferencia que generaría usar un procesador *in order* (los anteriores puntos) vs. uno *out of order*.

En general, el *out of order* va a tener una mejor *performance* dado que siempre que no haya dependencia de dato, no importa el orden en que se ejecuten las instrucciones y, mientras haya una que se esté calculando, puede proceder con otra que sea posterior.

Motivo de ello, nos permite aprovechar muchísimo los ciclos de nuestro procesador y disminuir **significativamente** los *ciclos ociosos* dado que, mientras estamos esperando que se traiga un dato de la memoria principal, otra operación posterior puede cargarse, ejecutarse y escribir sus resultados (siempre que no haya dependencia de dato real)

sin problema y sin tener que esperar. Por ello, estos ciclos ahora son utilizados y permite que nuestro *CPI* también disminuya significativamente.

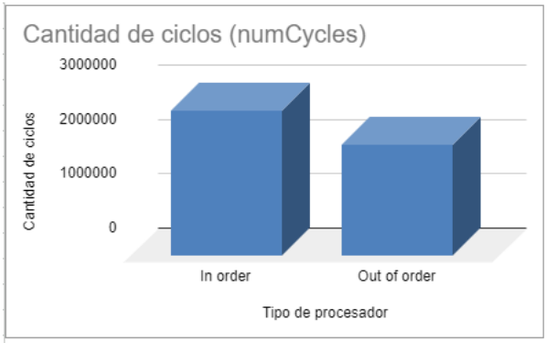
Datos obtenidos (resultados + gráficos)

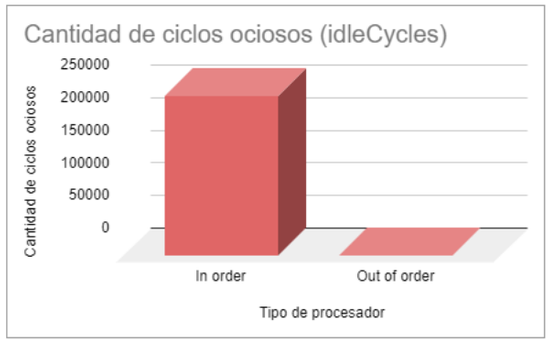
Respecto a los datos que se obtuvieron en la *simulación* de este punto, son los siguientes:

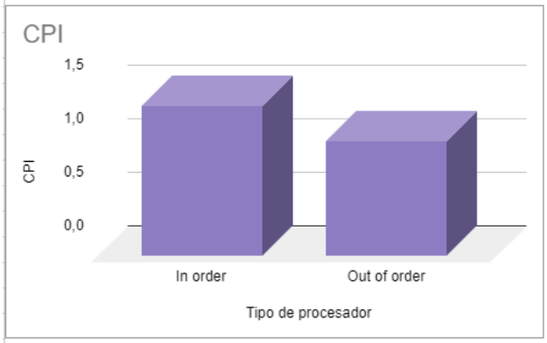
Tipo de caché	**Mapeo directo (1 vía)**	
Cantidad de vías	**2**	
Tamaño de la caché	**32KB**	
Tipo de predictor	**Por torneos**	
Tipo de procesador	In order	Out of order
Cantidad de ciclos (numCycles)	2682897	2043978
Cantidad de ciclos ociosos (idleCycles)	245755	896
CPI	1,41	1,07
Conditional branchs predicted (condPredicted)	335975	336829
Conditional branchs not predicted (condIncorrect)	752	747
MissRate de la predicción	0,22%	0,22%
Hits de la caché principal de datos (overallHits)	276554	227956
Misses de la caché principal de datos (overallMisses)	11663	60252
MissRate de la caché principal de datos	4,05%	20,91%

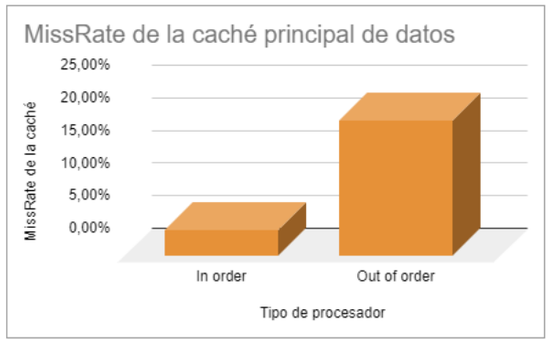
Si bien colocamos varios datos, para nuestro análisis nos concentraremos principalmente en **numCycles, idleCycles, CPI**, y haremos referencia al **MissRate de la caché** para explicar los beneficios de tener un procesador *out of order*.

Motivo de ello, entonces, adjuntamos los gráficos que se utilizarán para referencia en el análisis:









Análisis de resultados y *comparativa*

En este caso, tal y como se mencionó en la sección del *comportamiento esperado*, la **performance** de la ejecución de nuestro programa mejora significativamente, dado que, en términos de los datos obtenidos, tenemos:

- **numCycles** disminuye en un 24,8%.
- **idleCycles** disminuye en un 99,6%.
- **CPI** disminuye en un 24,11%

y es aquí donde se puede ver el *gran potencial* y *poder* que tienen los procesadores *out of order*, pasando de tener que una instrucción tarde (promedio) 1,41 ciclos de reloj, a 1,07 ciclos.