

Análisis de Caché

Introducción

En esta Práctica De Algoritmos Paralelos se hace un análisis de errores de caché. Se comparó dos algoritmos que hallan la multiplicación de matrices, donde el primer algoritmo es el clásico conocido formado por 3 bucles, mientras el segundo algoritmo es una mejora del primero usando bloques. La comparación se realizó mediante el uso de herramientas tales como valgring y kcacheingring, quienes nos dan una evaluación más precisa en términos de L1, L2 entre otros para conocer los errores de caché.

Comparación

Se midió mediante el tamaño de matriz en nuestro caso matrices cuadradas. Donde el tiempo es dado en milisegundos:

Cant	100	200	500
Clasica	8.053	41.214	280.062
Bloques	18.006	13.753	13.753

Viendo con más detalle la memoria caché los mejores resultados

Clásica 500 datos:

```
==24626==
==24626== Events      : Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLMw
==24626== Collected : 37623713 13690547 2268635 1757 86716 4576 1663 7659 3691
==24626==
==24626== I   refs:      37,623,713
==24626== I1  misses:      1,757
==24626== L1i misses:      1,663
==24626== I1  miss rate:      0.00%
==24626== L1i miss rate:      0.00%
==24626==
==24626== D   refs:      15,959,182 (13,690,547 rd + 2,268,635 wr)
==24626== D1  misses:      91,292 ( 86,716 rd + 4,576 wr)
==24626== L1d misses:      11,350 ( 7,659 rd + 3,691 wr)
==24626== D1  miss rate:      0.6% ( 0.6% + 0.2% )
==24626== L1d miss rate:      0.1% ( 0.1% + 0.2% )
==24626==
==24626== LL refs:      93,049 ( 88,473 rd + 4,576 wr)
==24626== LL  misses:      13,013 ( 9,322 rd + 3,691 wr)
==24626== LL  miss rate:      0.0% ( 0.0% + 0.2% )
```

main				
Types	Callers	All Callers	Callee Map	Source Code
Event Type	Incl.	Self	Short	Formula
Instruction Fetch	94.28	0.00	Ir	
Data Read Access	96.17	0.00	Dr	
Data Write Access	91.38	0.00	Dw	
L1 Instr. Fetch Miss	15.08	0.34	I1mr	
L1 Data Read Miss	84.52	0.00	D1mr	
L1 Data Write Miss	48.86	0.00	D1mw	
LL Instr. Fetch Miss	14.01	0.30	ILmr	
LL Data Read Miss	0.34	0.00	DLMr	
LL Data Write Miss	58.98	0.00	DLMw	
L1 Miss Sum	81.45	0.01	L1m = I1mr + D1mr + D1mw	
Last-level Miss Sum	18.72	0.04	LLm = ILmr + DLMr + DLMw	
Cycle Estimation	91.51	0.00	CEst = Ir + 10 L1m + 100 LLm	

Bloques 500 datos:

```

==24474==
--24474-- warning: L3 cache found, using its data for the LL simulation.
==24474== For interactive control, run 'callgrind_control -h'.
1.079418
==24474==
==24474== Events      : Ir Dr Dw I1mr D1mr D1mw ILmr DLMr DLMw
==24474== Collected : 69762268 29230255 6608416 1022 7381 2868 1009 1942 2505
==24474==
==24474== I   refs:      69,762,268
==24474== I1 misses:      1,022
==24474== L1i misses:      1,009
==24474== I1 miss rate:      0.00%
==24474== L1i miss rate:      0.00%
==24474==
==24474== D   refs:      35,838,671 (29,230,255 rd + 6,608,416 wr)
==24474== D1 misses:      10,249 ( 7,381 rd + 2,868 wr)
==24474== L1d misses:      4,447 ( 1,942 rd + 2,505 wr)
==24474== D1 miss rate:      0.0% ( 0.0% + 0.0% )
==24474== L1d miss rate:      0.0% ( 0.0% + 0.0% )
==24474==
==24474== LL refs:      11,271 ( 8,403 rd + 2,868 wr)
==24474== LL misses:      5,456 ( 2,951 rd + 2,505 wr)
==24474== LL miss rate:      0.0% ( 0.0% + 0.0% )

```

main				
Types	Callers	All Callers	Callee Map	Source Code
Event Type	Incl.	Self	Short	Formula
Instruction Fetch	99.87	82.59	Ir	
Data Read Access	99.92	80.93	Dr	
Data Write Access	99.84	49.51	Dw	
L1 Instr. Fetch Miss	24.76	1.57	I1mr	
L1 Data Read Miss	83.31	66.63	D1mr	
L1 Data Write Miss	82.25	68.13	D1mw	
LL Instr. Fetch Miss	24.98	1.59	ILmr	
LL Data Read Miss	45.98	0.05	DLmr	
LL Data Write Miss	80.80	67.27	DLmw	
L1 Miss Sum	77.73	61.11	$L1m = I1mr + D1mr + D1mw$	
Last-level Miss Sum	58.08	31.20	$LLm = ILmr + DLmr + DLmw$	
Cycle Estimation	99.51	82.16	$CEst = Ir + 10 L1m + 100 LLm$	

Conclusion

En el caso del primer algoritmo el bucle más interno de su multiplicador de matriz lee filas o columnas enteras en secuencia, el cache se llena gradualmente de datos, pero el tamaño del cache es limitado, por lo que si las filas son realmente largas, el cache debe tirar lo que cargó inicialmente, para dar cabida a cosas nuevas, es decir que cuando llegue al final e inicie el siguiente, necesitará algunos datos que estuvieron recientemente en la caché, y de esta manera esperar a que regresen de la memoria otra vez.

En conclusión el segundo algoritmo podrá tener más bucles anidados, pero su manera de acceso a memoria es mas rapida que el primero, para hacer esto posible debemos ver la manera de cargar los datos próximos a usar en la memoria caché, y esto debemos aplicarlo desde nuestro programa, buscando la mejor manera de acomodar nuestros datos temporales y espaciales.