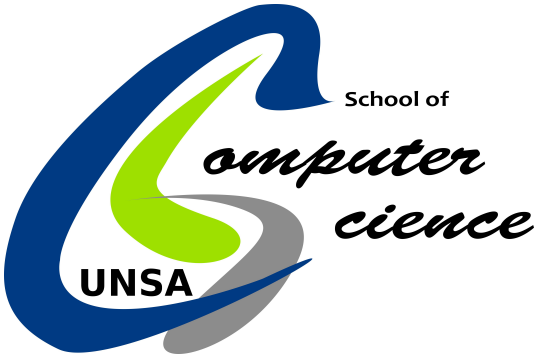
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN



TEMA:

ANÁLISIS Y RESULTADOS

Curso:

ALGORITMOS PARALELOS

Presentado por:

Christofer Chávez Carazas

Arequipa - Perú
2017

1. Descripción del Problema

Comparar y analizar dos algoritmos para multiplicar matrices (Simple Matrix Multiplication / Tiled Matrix Multiplication) en función del tiempo, los caché misses y la distancia de acceso a la memoria.

2. Especificaciones del Experimento

Ambos algoritmos están escritos en el lenguaje C. Para el experimento se generan dos matrices de n x n con números aleatorios comprendidos entre 0 y 100. Dentro del tiempo calculado sólo se toma en cuenta la multiplicación de las matrices (Los bucles anidados), no se toma en cuenta la creación de las matrices ni ninguna otra operación fuera de la multiplicación de las matrices. Para cada tamaño cada algoritmo se ejecutó cinco veces y se sacó el promedio de los tiempos.
Se utilizó valgrind y kcachegrind para ver el manejo de la memoria de ambos algoritmos, para esto se utilizaron matrices de 700x700.

3. Resultados y Análisis

En la Figura 1 se muestra un gráfico con el tiempo de demora de los algoritmos respecto al tamaño de las matrices. Claramente se observa que el algoritmo Tiled es más rápido que el algoritmo Simple cuando el tamaño aumenta. Esto se debe a la gran diferencia que hay entre los caché misses que cada algoritmo produce; en la Figura 2(a) (Algoritmo Simple) y en la Figura 2(b) (Algoritmo Tiled) esta remarcado el número de caché misses.
El Algoritmo Tiled divide la matriz en bloques más pequeños, de tal manera que una fila de esos bloques quepa en una línea de caché y poder utilizar esa fila lo más que se pueda, de modo que los caché misses se reducan considerablemente. En este experimento se supuso un tamaño de línea de caché de 64 bytes. Ya que los enteros ocupan 4 bytes, las divisiones hechas en el experimento son de 16 números por bloque.
El Algoritmo Simple no divide la matriz. Si el tamaño de una fila es más grande que el tamaño de una línea de caché, al querer utilizar esa fila para la multiplicación, se generan muchos más caché misses.

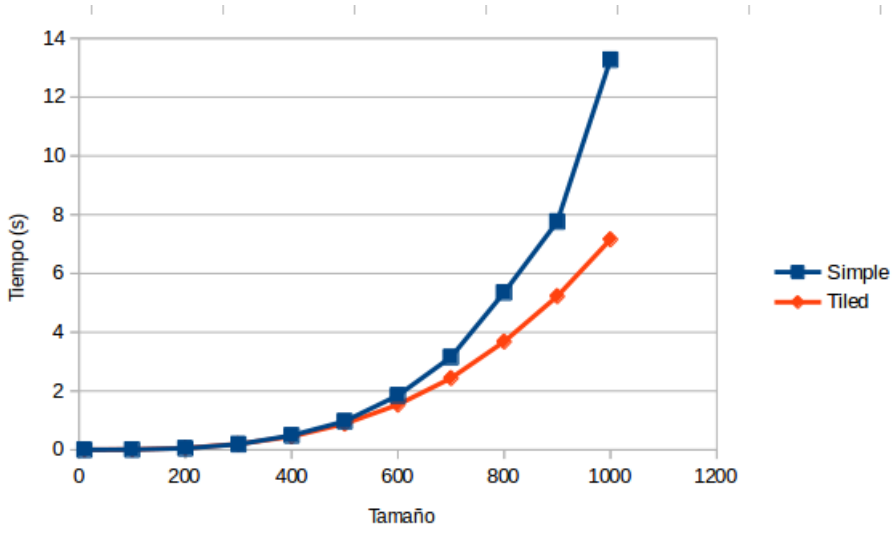


Figura 1: Resultados de las ejecuciones

Instruction Fetch	17 161 783 168	17 161 783 168	Ir
L1 Instr. Fetch Miss	9	9	I1mr
LL Instr. Fetch Miss	9	9	ILmr
Data Read Access	7 551 891 222	7 551 891 222	Dr
L1 Data Read Miss	359 950 879	359 950 879	D1mr
LL Data Read Miss	88 129	88 129	DLmr
Data Write Access	343 982 822	343 982 822	Dw
L1 Data Write Miss	30 891	30 891	D1mw
LL Data Write Miss	30 188	30 188	DLmw
L1 Miss Sum	359 981 779	359 981 779	L1m = I1mr + D1mr + D1mw
Last-level Miss Sum	118 326	118 326	LLm = ILmr + DLmr + DLmw
Cycle Estimation	20 773 433 558	20 773 433 558	CEst = Ir + 10 L1m + 100 LLm

(a) Caché misses del algoritmo Simple

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	11 818 407 143	11 818 407 143	Ir	
L1 Instr. Fetch Miss	11	11	I1mr	
LL Instr. Fetch Miss	11	11	ILmr	
Data Read Access	5 345 265 188	5 345 265 188	Dr	
L1 Data Read Miss	1 861 490	1 861 490	D1mr	
LL Data Read Miss	88 090	88 090	DLmr	
Data Write Access	398 430 325	398 430 325	Dw	
L1 Data Write Miss	30 890	30 890	D1mw	
LL Data Write Miss	30 187	30 187	DLmw	
L1 Miss Sum	1 892 391	1 892 391	L1m	= I1mr + D1mr + D1mw
Last-level Miss Sum	118 288	118 288	LLm	= ILmr + DLmr + DLmw
Cycle Estimation	11 849 159 853	11 849 159 853	CEst	= Ir + 10 L1m + 100 LLm

(b) Caché misses del algoritmo Tiled

Figura 2: Resultados del Kcachegrind

4. Conclusiónes

- Es importante tener nocion de cómo se maneja la memoria para poder optimizar los algortimos o programas.
- Los caché misses influyen mucho en el tiempo de ejecución de un algoritmo o programa.
- En el problema de la multiplicación de matrices óptima, el algoritmo Tiled es una buena forma de aumentar la rapidez, aunque teóricamente sigue teniendo un costo computacional de $O(n^3)$.

Referencias

[1] Peter S. Pacheco *An introduction to parallel programming*

[2] Lam, Monica S.; Rothberg, Edward E.; Wolf, Michael E. (1991) *The Cache Performance and Optimizations of Blocked Algorithms.*