Laboratorio 1

Victor Manuel Toledo Gamarra Escuela de Ciencia de la Computación
Universidad Católica San Pablo
Arequipa, Arequipa
Email: victor.toledo@ucsp.edu.pe

1. Introducción

En el siguiente trabajo realizaremos un análisis de dos códigos con misma complejidad , con el fin de poder ver cual es la diferencia entre ellos si es que existe.

2. Implementación

La implementación fue sacada del libro de Parallel Programming del autor Peter Pacheco.

2.1. Primero observaremos el primer código

```
/* First pair of loops */
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
      y[i] += A[i][j]*x[j];
```

Fig. 1. primer algoritmo

2.2. observemos el segundo código

```
/* Second pair of loops */
for (j = 0; j < MAX; j++)
   for (i = 0; i < MAX; i++)
      y[i] += A[i][j]*x[j];
```

Fig. 2. segundo algoritmo

3. Análisis

Procederemos hacer un conjunto de pruebas en la que testearemos ambos códigos con diferentes tamaños de A, ingresaremos con cantidades de 1000, 5000, 7500, 10000 y 20000 con la matriz de A relleno de números random.

Como podemos observar en la figura 3 nos muestra los tamaños que le asignamos en cada caso que vendría a ser la columna de color plomo y la columna naranja como la azul son los tiempos de cada algoritmo. Visualmente no podemos lograr ver la diferencia gráfica de los valores de cuanto fue y si es significativa por lo que lo pasaremos a una gráfica de estadística que vendría a ser la siguiente figura 4, donde podemos observar los que el segundo

	Algoritmo 1	Algoritmo 2
1000	0.019	0.03
5000	0.682	0.987
7500	1.493	2.15
10000	2.71	3.911
20000	9.713	14.345

Fig. 3. Tiempo de los intervalos dados por cada código

Fig. 4. Tiempo de los intervalos dados por cada código

resultado nos da muchísimo más tiempo significativo que el primero. Si hacemos una comparación significativa , podemos ver que los primeros 3 tamaños nos da resultado que es un x3 de tiempo más.

Si observamos el primer código recorre los bucles de una manera que está accediendo a los datos de la matriz en bloques contiguos, aunque la complejidad del código sea igual los bucles acceden a los datos de diferente forma donde la cache solo podrá almacenar 8 elementos de A donde se podrá acceder A[0][1], A[0][2], A[0][3] ya que están en la cache, mientras que el otro bucle tendrá problemas al acceder en A[1][0] , ya que la cache almacenará dos filas u ocho elementos de A,y cuando se quiera leer el primer elemento de la fila 2 o 3 una de las líneas que ya está en la caché tendrá que ser desalojada de la caché , una vez que línea es desalojada, el primer par de bucles no necesitará acceder a los elementos de esa línea de nuevo.

Mientras que el segundo código va a acceder a A[1][0], A[2][0], A[3][0], el cuál ninguno está en la cache , como la caché es pequeña, las A[2][0] y A[3][0] requerirán que las líneas que ya están en la caché sean desalojadas. Como A[2][0] está almacenado en la línea 2 de la caché, la lectura de su línea desalojará la línea 0, y la lectura de A[3][0] desalojará la línea 1. Después de terminar la primera pasada por el bucle externo, necesitaremos acceder a A[0][1], que fue desalojado con el resto de la primera fila. Así que vemos que cada vez que leamos un elemento de A, tendremos un fallo.

4. Conclusión

Por lo que el primer código es mucho más eficiente por los resultados dados en las pruebas dadas de los testeos , como también la forma de utilizar el cache diferente a segundo algoritmo ya que nos reduce el tiempo de consulta.