Lab5 Arquitectura de Computadoras

José Leonidas García Gonzales

June 2019

1 Metodología seguida

Comencé implementando una versión directa para la multiplicación de matrices en C++. Sin embargo, al ejecutarlo en las computadoras del laboratorio para multiplicar dos matrices cuadradas obtuve:

Tabla 1. Tiempos de ejecución para una implementación sin optimizaciones

dimensión	tiempo
3000	1m32s
4000	4m32s
5000	<10m

Luego, comencé a realizar optimizaciones. La primera optimización fue utilizando el principio de localidad.

$$AxB = C$$

$$c_{i,j} = A_{row_i} \cdot B_{col_j}$$

De las fórmula anterior nos damos cuenta que para calcular $c_{i,j}$ necesitamos podemos recorrer A en row-order y B en column-order. Por ello, declaré A como un array de memoria continua (así las filas estarían continuas) y lo mismo para B, con la diferencia que los elementos de B los guarde como si estuviera leyendo su matriz transpuesta (así, tenemos las columnas en espacios de memoria continua). Con ello pude obtener los siguientes resultados en mi laptop:

Tabla 2. Tiempos de ejecución promedio para una implementación usando el principio de localidad

dimensión	tiempo (s)
1000	0.684557
2000	5.413270
3000	18.194445
4000	43.108473
5000	85.151837

Finalmente, adapté la implementación para utilizar hilos de manera que si vemos la matriz resultante como una fila, cada hilo calcule $\lfloor cantidaddeelementos/cantidaddehilos \rfloor$ términos, con el último hilo calculando más elementos que los demás si la cantidad de hilos no es divisor de la cantidad de elementos.

Además, creé un script en bash para comparar que la última implementación multiplique correctamente las matrices comparando resultados con la primera implementación para matrices de 1000x1000 y para ejecutar repetidamente la multiplicación de matrices para distintas entradas e ir guardando los tiempos de ejecución en archivos para que estos puedan ser leídos por un script que creé en python para graficar la cantidad de hilos vs el perfomance y obtener una curva suave a partir de los resultados obtenidos. Estos resultados lo podemos observar en la Tabla 3 y el Gráfico 1.

Tabla 3. Tiempos de ejecución promedio para una implementación usando el principio de localidad y threads para multiplicar 2 matrices cuadradas de orden 5000

número de hilos	tiempo (s)
1	84.022631
2	43.461688
4	42.234993
8	42.354470
16	42.271656
32	42.360322

2 Conclusiones

- El principio de localidad puede optimizar incluso por un factor de 10 una implementación.
- El uso de hilos permite un uso más optimo de los recursos de un computador, sin embargo más hilos no suponen necesariamente mejor perfomance.

3 Opservaciones

- Para la ejecución de los programas el computador solo estuvo ejecutando la multiplicación de matrices y los procesos del sistema
- Durante la multiplicación de las matrices se mantuvo la laptop cargando pues se noto que los programas requerían más tiempo cuando la laptop no se estaba cargando
- Las escepcificaciones de la laptop donde se corrieron los tests son las siguientes

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
```

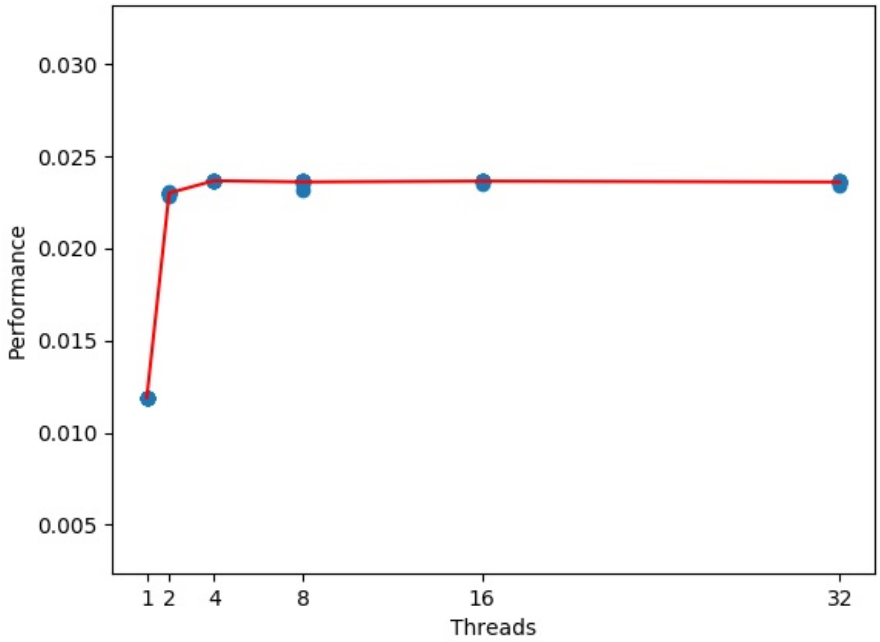

Figure 1: Gráfico 1. Number of threads vs Performance

```
Byte Order:          Little Endian
Address sizes:       39 bits physical, 48 bits virtual
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               142
Model name:          Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Stepping:            9
CPU MHz:             500.121
CPU max MHz:         3100.0000
CPU min MHz:         400.0000
BogoMIPS:            5426.00
Virtualization:      VT-x
L1d cache:           32K
```

```
L1i cache:           32K
L2 cache:            256K
L3 cache:            3072K
```