Challenge 1: Performance of Multiple Multicore Algorithms

Diego Madariaga Román

1. Descripción

Este desafío consiste en medir el impacto de ejecutar múltiples algoritmos paralelos en una misma máquina *multicore*.

A continuación, se presenta una explicación de los experimentos a realizar, los cuales consisten en la ejecución de 3 programas con ejecución en paralelo, en donde se varía el número de threads a utilizar y la afinidad de procesadores para cada programa. Luego, se presentan los resultados experimentales obtenidos y finalmente, se realiza un análisis sobre estos resultados.

2. Experimentación

El código utilizado y las instrucciones para reproducir los experimentos realizados se encuentran disponibles en `https://git.io/dmadariaga-performance`

Para la experimentación realizada se utilizaron 3 programas con ejecución en paralelo:

- `string_matching`: Búsqueda de un patrón de largo 1 dentro de un texto de largo 1,410,065,408.
- `parallel_quicksort`: Ordenamiento de un arreglo de enteros aleatorios de tamaño 50,000,000.
- `prefix_sum`: Cálculo de la suma de prefijos sobre un arreglo de enteros de tamaño 100,000,000.

Se crearon 5 casos de pruebas utilizando los 3 programas, los cuales se muestran a continuación:

1. Ejecución de los 3 algoritmos en paralelo, cada uno utilizando 4 threads, sin especificar la afinidad de procesadores.

Código Fuente 1: Primer programa de prueba

```
#!/bin/bash
CILK_NWORKERS=4 ./string_matching out D &
CILK_NWORKERS=4 ./parallel_quick_sort 50000000 &
CILK_NWORKERS=4 ./prefix_sum 100000000
wait
```

2. Ejecución de los 3 algoritmos en paralelo, cada uno utilizando 4 threads, y afinidad con 4 procesadores consecutivos.

Código Fuente 2: Segundo programa de prueba

```
#!/bin/bash
CILK_NWORKERS=4 taskset -c 0-3 ./string_matching out D &
CILK_NWORKERS=4 taskset -c 4-7 ./parallel_quick_sort 50000000 &
CILK_NWORKERS=4 taskset -c 8-11 ./prefix_sum 100000000
wait
```

3. Ejecución de los 3 algoritmos en paralelo, cada uno utilizando 4 threads, y afinidad con 4 procesadores no consecutivos.

Código Fuente 3: Tercer programa de prueba

```
#!/bin/bash
CILK_NWORKERS=4 taskset -c 0,3,6,9 ./string_matching out D &
CILK_NWORKERS=4 taskset -c 1,4,7,10 ./parallel_quick_sort 50000000 &
CILK_NWORKERS=4 taskset -c 2,5,8,11 ./prefix_sum 100000000
wait
```

4. Ejecución de los 3 algoritmos en paralelo, cada uno utilizando 3 threads, y afinidad con 4 procesadores consecutivos.

Código Fuente 4: Cuarto programa de prueba

```
#!/bin/bash
CILK_NWORKERS=3 taskset -c 0-3 ./string_matching out D &
CILK_NWORKERS=3 taskset -c 4-7 ./parallel_quick_sort 50000000 &
CILK_NWORKERS=3 taskset -c 8-11 ./prefix_sum 100000000
wait
```

5. Ejecución de los 3 algoritmos en paralelo, cada uno utilizando 2 threads, y afinidad con 4 procesadores consecutivos.

Código Fuente 5: Quinto programa de prueba

```
#!/bin/bash
CILK_NWORKERS=2 taskset -c 0-3 ./string_matching out D &
CILK_NWORKERS=2 taskset -c 4-7 ./parallel_quick_sort 50000000 &
CILK_NWORKERS=2 taskset -c 8-11 ./prefix_sum 100000000
wait
```

Cada programa de prueba fue ejecutado midiendo el tiempo total de ejecución y el número de *cache-misses*.

Cada experimento fue realizado 5 veces y se han reportado los valores correspondientes a las medianas obtenidas en cada caso.

Todos los experimentos fueron llevados a cabo utilizando un computador de 12 cores físicos, distribuidos en una arquitectura NUMA de 2 nodos, cada uno con 6 cores y 3GB de memoria RAM. Los resultados se muestran a continuación:

Figura 2.1: Tiempo en [s] para la ejecución de los programas de prueba.

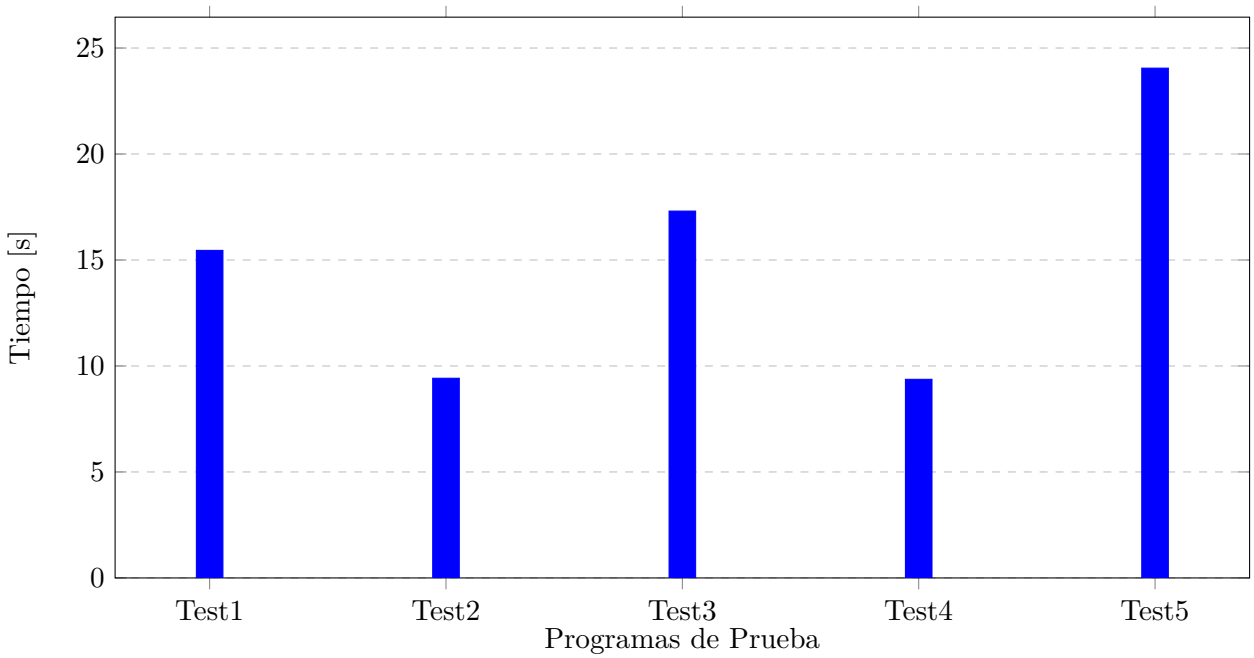
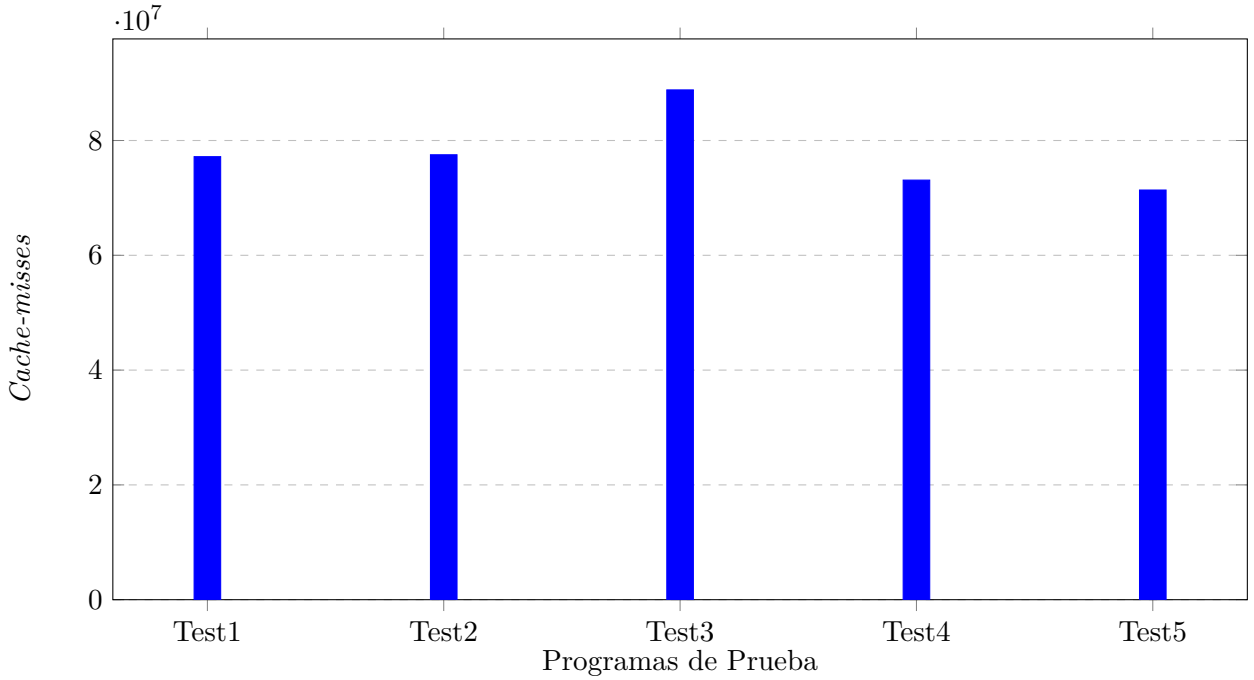


Figura 2.2: *Cache-misses* para la ejecución de los programas de prueba.



3. Discusión y Conclusiones

En las Figura 2.1 se puede observar por medio de los resultados de los Test 1, 2 y 3, que al permitir el uso de 4 threads por programa, se obtiene un menor tiempo de ejecución asignando afinidad con 4 procesadores consecutivos (Test 2) frente al no asignar afinidad alguna (Test 1) y al asignar afinidad con 4 procesadores no consecutivos (Test 3), en donde este último obtiene los tiempos de ejecución más alto. Asímismo, en la Figura 2.2 se puede ver que para los Test 1 y 2 se obtienen valores muy similares de *cache-misses*, por lo que se cree que el Test 1 en donde no se seteó afinidad alguna, ejecutó cada programa utilizando algunos procesadores consecutivos y otros no, a diferencia del Test 3 en donde todos los procesadores asignados para cada programa fueron no consecutivos, y en donde se obtuvo como consecuencia los valores más altos de *cache-misses*.

Con respecto a la variación de la cantidad de threads a utilizar por cada algoritmo, se puede ver en la Figura 2.1 por medio de los resultados de los Test 1 y 4, que al asignar a cada programa una afinidad con 4 procesadores consecutivos, no se encuentra diferencia en el tiempo de ejecución al utilizar 4 o 3 threads para cada programa (Test 1 y 4 respectivamente). Es más, al analizar la Figura 2.2 se observa que el utilizar 3 threads por programa (Test 4) genera menor cantidad de *cache-misses* que el utilizar 4 threads (Test 1). Lo anterior es importante, ya que muestra que no siempre se obtienen mejores resultados al asignar mayor cantidad de threads para la ejecución de distintos programas paralelos ejecutados en una misma máquina. En este caso, resulta conveniente asignar para cada programa una afinidad con 4 procesadores, pero limitar el número de threads a 3 (Test 4).

Finalmente, se observa en las Figuras 2.1 y 2.2 que el utilizar 2 threads y asignar afinidad con 4 procesadores consecutivos para cada programa (Test 5), genera una menor cantidad de *cache-misses* en relación a los otros experimentos, pero se obtienen mayores tiempos de ejecución, dado principalmente a la menor cantidad de trabajo que se puede realizar en paralelo dada la gran limitación de threads.