Título: Laboratorio 3 - 20213812

Respuestas:

1. **Resultados de Ejecución:**
    - ⓾ Los resultados de cada ejecución se encuentran en los archivos `reporte_v1.txt` y `reporte_v2.txt`.
2. **Código Agregado:**
    - ⓾ Se ha implementado código adicional para mejorar el rendimiento.
3. **Reportes Extendidos:**
    - ⓾ Se presentan reportes extendidos con detalles adicionales.
4. **Resultados en Captura:**
    - ⓾ Se adjunta una captura de pantalla `captura_20213812.png` con los resultados.
5. **Ejecuciones Individuales:**
    - ⓾ Las ejecuciones individuales se almacenan en las carpetas `reportes_v1` y `reportes_v2`. Los resultados de cada ejecución se guardan en archivos con el formato `reporte_v1_xx.txt` o `reporte_v2_xx.txt`, donde "xx" indica el número de ejecución. Cada carpeta contiene una tabla adjunta con todos los tiempos de ejecución de cada run (1 - 15) en formato odt y csv.
6. **Análisis de Rendimiento:**
    - ⓾ Localidad temporal: Los elementos a los que se accedió recientemente tienden a ser accedidos nuevamente en el futuro cercano.
    - ⓾ Localidad espacial: Si se accede a una posición de memoria una vez, es probable que el programa acceda a una ubicación de memoria cercana.
    - ⓾ Entonces al acceder al total de pago del primer alumno, los datos no están en la cache por lo que se da un miss y se mueve un bloque para los siguientes elementos cercanos.
7. **Versión V1 del Código:**

```
//Imprimir reporte
/*C@digo*/
struct timespec ti, tf;
double elapsed;
printf("Alumno\tCosto/Credito\tTotal_Creditos\tTotal_a_Pagar\tTiempo_de_ejecucion_(ns)\n");
for(int i=0; i<nAlumnos; i++){
    printf("%d\ts/.%5.2f\t\t%5.2f\t\t\ts/.", arrCodAlumno[i], arrEscAlumno[i], arrTotalCreditos[i]);

    clock_gettime(CLOCK_REALTIME, &ti);
    printf("%5.2f", arrEscAlumno[i]*arrTotalCreditos[i]);
    clock_gettime(CLOCK_REALTIME, &tf);
    elapsed = (tf.tv_sec - ti.tv_sec) * 1e9 + (tf.tv_nsec - ti.tv_nsec);
    printf("\t\t\t%5.2f\n", elapsed);
}
```

    - ⓾ Previamente se accede a los valores de arrEscAlumno[i] y arrTotalCreditos[i] y luego son usados para calcular el valor del total gastado. Por localidad temporal creo que si se intercambian los sumandos se podria aprovechar, puesto que se accede 2 veces a arrTotalCreditos[i]. En el caso de la localidad espacial se aprovecha que se accede a valores cercanos, sin embargo el hecho de acceder a 2 arreglos double más realizar el calculo agrega tiempo de ejecucion.
8. **Versión V2 del Código:**

```
//Imprimir reporte
/*C@digo*/
struct timespec ti, tf;
double elapsed;
printf("Alumno\tCosto/Credito\tTotal_Creditos\tTotal_a_Pagar\tTiempo_de_ejecucion_(ns)\n");
for(int i=0; i<nAlumnos; i++){
    printf("%d\ts/.%5.2f\t\t%5.2f\t\t\ts/.", arrCodAlumno[i], arrEscAlumno[i], arrTotalCreditos[i]);

    clock_gettime(CLOCK_REALTIME, &ti);
    printf("%5.2f", arrTotalApagar[i]);
    clock_gettime(CLOCK_REALTIME, &tf);
    elapsed = (tf.tv_sec - ti.tv_sec) * 1e9 + (tf.tv_nsec - ti.tv_nsec);
    printf("\t\t\t%5.2f\n", elapsed);
}
```

- ➓ En esta situacion se aprovecha unicamente la localidad espacial, puesto que nunca se vuelve acceder a una misma ubicacion 2+ veces. Además, los datos ya estan previamente calculados por lo que es un simple acceso adicional a un arreglo de double.

9. **Comparación de Versiones:**

- ➓ Según la tabla de ejecución, `MatriculaV2.c` muestra mejores resultados en términos de rendimiento y en dos ejecuciones se observa una mejora considerable.

10. **Consideraciones sobre el Tamaño de los Datos:**

- ➓ En el caso de datos tipo char que solo ocupan 1 byte es posible arrastrar más datos en un solo bloque lo que mejora la oportunidad de hit a diferencia de usar un dato tipo long long que usa 16 bytes.