Práctica Nro. 1

Optimización de algoritmos secuenciales

Pautas generales

- Las pruebas se deben realizar en un equipo con sistema operativo Linux nativo (no virtualizado). La virtualización puede alterar el comportamiento y la toma de los tiempos de ejecución, imposibilitando la ocurrencia de los resultados y análisis esperados.
- Al momento de realizar las pruebas, deberá cerrar todo programa que tenga abierto (editores, navegadores etc). Mantenga abierta únicamente una consola/terminal para ejecutar las pruebas.
- En todos los ejercicios con matrices, pruebe con tamaños que sean potencias de 2 (512, 1024, 2048, etc).
- Tener en cuenta que, para poder notar cambios en la ejecución, los algoritmos deben ejecutarse al menos varios segundos (más de 15 segundos si es posible).

Información útil para compilar y ejecutar

- Para compilar en Linux con gcc*:* *gcc –o salidaEjecutable archivoFuente.c*
- Para ejecutar*:* *./salidaEjecutable arg1 arg2 … argN*

Ejercicios

1. El algoritmo *fib.c* resuelve la serie de Fibonacci, para un número N dado, utilizando dos métodos: recursivo e iterativo. Analice los tiempos de ejecución de ambos métodos ¿Cuál es más rápido? ¿Por qué?
   Nota: ejecute con N=1..50.

2. El algoritmo *funcion.c* resuelve, para un x dado, la siguiente sumatoria:

$$\sum_{i=0}^{100\,000\,000} 2 * \frac{x^3 + 3x^2 + 3x + 2}{x^2 + 1} - i$$

   El algoritmo compara dos alternativas de solución. ¿Cuál de las dos formas es más rápida? ¿Por qué?

3. Investigue en la documentación del compilador o a través de Internet qué opciones de optimización ofrece el compilador *gcc* (*flag O*). Compile y ejecute el algoritmo *matrices.c*, el cual resuelve una multiplicación de matrices de NxN. Explore los diferentes niveles de optimización para distintos tamaños de matrices. ¿Qué optimizaciones aplica el compilador? ¿Cuál es la ganancia respecto a la versión sin optimización del compilador? ¿Cuál es la ganancia entre los distintos niveles?

4. Dada la ecuación cuadrática: $x^2 - 4.0000000\,x + 3.9999999 = 0$, sus raíces son r_1 = 2.000316228 y r_2 = 1.999683772 (empleando 10 dígitos para la parte decimal).

    a. El algoritmo *quadratic1.c* computa las raíces de esta ecuación empleando los tipos de datos *float* y *double*. Compile y ejecute el código. ¿Qué diferencia nota en el resultado?
    b. El algoritmo *quadratic2.c* computa las raíces de esta ecuación, pero en forma repetida. Compile y ejecute el código variando la constante TIMES. ¿Qué diferencia nota en la ejecución?
    c. El algoritmo *quadratic3.c* computa las raíces de esta ecuación, pero en forma repetida. Compile y ejecute el código variando la constante TIMES. ¿Qué diferencia nota en la ejecución? ¿Qué diferencias puede observar en el código con respecto a *quadratic2.c?*

    Nota: agregue el flag *-lm* al momento de compilar. Pruebe con el nivel de optimización que mejor resultado le haya dado en el ejercicio anterior.

5. Analice el algoritmo *matrices.c*. ¿Dónde cree que se producen demoras? ¿Cómo podría optimizarse el código? Al menos, considere los siguientes aspectos:
    - Explotación de localidad de datos a través de reorganización interna de matrices A, B o C (según corresponda).
    - El uso de *Setters* y *getters* es una buena práctica en la programación orientada a objetos. ¿Tiene sentido usarlos en este caso? ¿cuál es su impacto en el rendimiento?
    - ¿Hay expresiones en el cómputo que pueden refactorizarse para no ser computadas en forma repetida?
    - En lugar de ir acumulando directamente sobre la posición C[i,j] de la matriz resultado (línea 72), pruebe usar una variable local individual y al finalizar el bucle más interno, asigne su valor a C[i,j]. ¿Esta modificación impacta en el rendimiento? ¿Por qué?

    Combine las mejoras que haya encontrado para obtener una solución optimizada y compare los tiempos con la solución original para diferentes tamaños de matrices.

6. Analice y describa brevemente cómo funciona el algoritmo *mmblk.c* que resuelve la multiplicación de matrices cuadradas de NxN utilizando una técnica de multiplicación por bloques. Luego, ejecute el algoritmo utilizando distintos tamaños de matrices y distintos tamaños de bloque (pruebe con valores que sean potencia de 2; p.e. N={512,1024,2048} y TB={16,32,64,128}). Finalmente, compare los tiempos con respecto a la multiplicación de matrices optimizada del ejercicio anterior. Según el tamaño de las matrices y de bloque elegido, responda: ¿Cuál es más rápido? ¿Por qué? ¿Cuál sería el tamaño

de bloque óptimo para un determinado tamaño de matriz? ¿De qué depende el tamaño de bloque óptimo para un sistema?

7. Analice el algoritmo *triangular.c* que resuelve la multiplicación de una matriz cuadrada por una matriz triangular inferior, ambas de NxN. ¿Cómo se podría optimizar el código? ¿Se pueden evitar operaciones? ¿Se puede reducir la memoria reservada? Implemente una solución optimizada y compare los tiempos probando con diferentes tamaños de matrices.