Práctica Nro. 1

Optimización de algoritmos secuenciales

Pautas generales:

- En todos los ejercicios de matrices probar 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 en Linux gcc: gcc –o salidaEjecutable archivoFuente.c

1. Analizar el algoritmo *matrices.c* que resuelve la multiplicación de 2 matrices cuadradas de NxN. ¿Dónde cree que se producen demoras? ¿Cómo podría optimizarse el código? Implementar una solución optimizada y comparar los tiempos probando con diferentes tamaños de matrices.

2. Describir brevemente cómo funciona el algoritmo *multBloques.c* que resuelve la multiplicación de matrices cuadradas de NxN utilizando una técnica de multiplicación por bloques. Ejecutar el algoritmo utilizando distintos tamaños de matrices y distintos tamaños de bloque. Comparar los tiempos con respecto a la multiplicación de matrices optimizada del ejercicio 1. Según el tamaño de las matrices y de bloque elegido ¿Cuál es más rápido? ¿Por qué? ¿Cuál sería el tamaño de bloque óptimo para un determinado tamaño de matriz?

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 nuevamente alguno de los algoritmos de multiplicación de matrices de los ejercicios 1 y 2 pero usando 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. Analizar 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? Implementar una solución optimizada y comparar los tiempos probando con diferentes tamaños de matrices.

5. 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 y sin opciones de optimización.

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

7. 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?

8. El algoritmo *instrucciones.c* compara el tiempo de ejecución de las operaciones básicas: suma (+), resta (-), multiplicación (*) y división (/), para dos operandos dados x e y. ¿Qué análisis se puede hacer de cada operación? ¿Qué ocurre si x e y son potencias de 2?

9. En función del ejercicio 7 analizar el algoritmo *instrucciones2.c* que resuelve una operación binaria (dos operandos) con dos operaciones distintas.

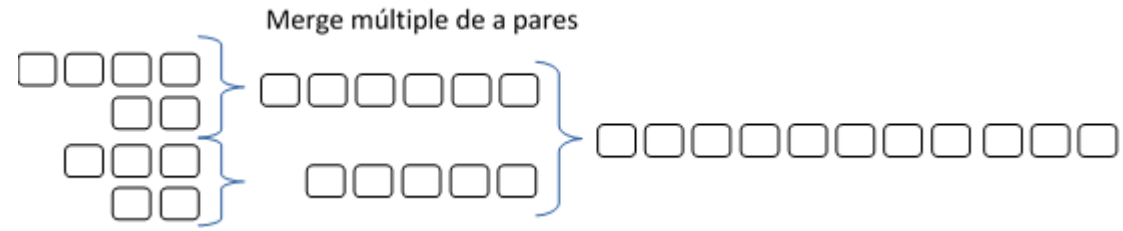
10. Analizar los algoritmos *iterstruc1.c* y *iterstruc1.c* que resuelven una multiplicación de matrices utilizando dos estructuras de control distintas. ¿Cuál de las dos estructuras de control tiende a acelerar el cómputo? Compile con y sin opciones de optimización.

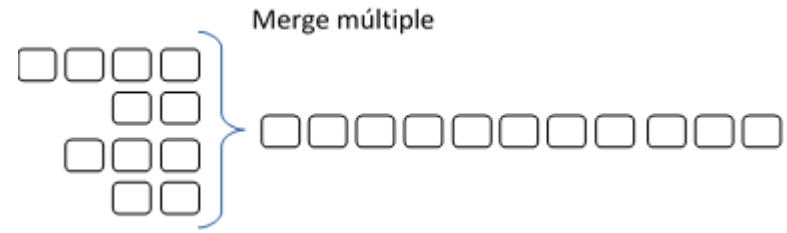
11. Dado un vector de N elementos de números reales distintos de 0, realizar la reducción por cociente consecutivo. Ejemplo:

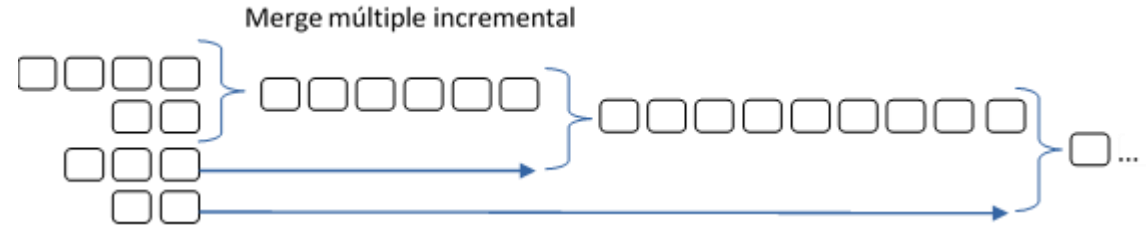
500	**10**	**6**	**3**	**60**	**2**	**18**	**3**
500/10 = 50		6/3 = 2		60/2 = 30		18/3 = 6	
50		**2**		**30**		**6**	
50/2 = 25				30/6 = 5			
25				**5**			
25/5 = 5							
5							

Utilizar vectores con N potencias de 2 y se debe minimizar el espacio de almacenamiento.

12. Analizar el algoritmo *merge_multiple.c* que compara tres métodos diferentes para combinar las listas de un arreglo de listas. A continuación, se describen las versiones en forma gráfica:







¿Qué estrategia alcanza mejor rendimiento? ¿Por qué?
Nota: Al momento de ejecutar, siga el comentario al inicio del código fuente.