Práctica Nro. 1

Optimización de algoritmos secuenciales

Esta práctica cuenta con una entrega obligatoria que los alumnos deberán entregar en grupos de dos personas.

Pautas:

En todos los ejercicios de matrices probar con tamaños de matriz potencias de 2 (32, 64, 128, 256, 512, 1024, 2048) etc.

Compilar en Linux gcc:

gcc –o salidaEjecutable archivoFuente.c

1. *Algebra de Matrices*

a. Analizar el algoritmo matrices.c que resuelve la multiplicación de matrices cuadradas de N*N, ¿dónde cree se producen demoras? ¿cómo se podría optimizar el código? Optimizar el código y comparar los tiempos probando con diferentes tamaños de matrices.

Ejecución: *./matrices N*

b. Analizar los algoritmos que resuelven distintas operaciones sobre matrices de N*N:

expMatrices1.c: dos versiones que resuelven la operación AB + AC + AD
expMatrices2.c: dos versiones que resuelven la operación AB + CD
expMatrices3.c: dos versiones que resuelven la operación BA + CAD

Ejecutar con distintos valores de N. Comparar los tiempos de ejecución de las dos versiones en cada caso. ¿Qué versión es más rápida? ¿Por qué?

Ejecución: *./expMatrices1 N*
./expMatrices2 N
./expMatrices3 N

c. Describir brevemente cómo funciona el algoritmo multBloques.c que resuelve la multiplicación de matrices cuadradas de N*N utilizando la técnica por bloques. Ejecutar el algoritmo utilizando distintos tamaños de matrices y distintos tamaño de bloques, 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?

Ejecución: *./multBloques r B [0|1]*

r: cantidad de bloques por lado de la matriz

B: tamaño de bloque

[0|1]: = 1 o 0 para imprimir o no las matrices en pantalla

Ejemplo:

*2 bloques de 512 elementos da una matriz de N=2*512=1024, sin mostrar en pantalla:*

./multBloques 2 512 0

d. Analizar el algoritmo triangular.c que resuelve la multiplicación de una matriz cuadrada por una matriz triangular inferior ambas de N*N, ¿cómo se podría optimizar el código? Optimizar el código y comparar los tiempos probando con diferentes tamaños de matrices.

Ejecución: *./triangular N*

2. El algoritmo fib.c resuelve la serie de Fibonacci para un numero N dado utilizando dos métodos, el método recursivo y el método iterativo. Analizar los tiempos de ambos métodos ¿Cuál es más rápido? ¿Por qué?

Ejecución: *./fib N*

Probar con N entre 0 y 50.

3. 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 formas de resolverla. ¿Cuál de las dos formas es más rápida? ¿Por qué?

Ejecución: *./funcion*

4. *Optimización de instrucciones*

a. 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?

   Ejecución: *./instrucciones*

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

   Ejecución: *./instrucciones2*

c. El algoritmo modulo.c compara el tiempo de ejecución de dos versiones para obtener el resto de un cociente m (m potencia de 2) de los elementos enteros de un vector de tamaño N. ¿Qué análisis se puede hacer de las dos versiones?

   Ejecución: *./modulo N m*

   N: tamaño del vector

   m: potencia de 2

5. *Iteraciones*

a. Analizar el algoritmo iterstruc.c que resuelve una multiplicación de matrices utilizando dos estructuras de control distintas. ¿Cuál de las dos estructuras de control tiende a acelerar el cómputo?

Compilar con la opción –O3

Ejecución: *./iterstruct N R*
N: tamaño de la matriz
R: cantidad de repeticiones

b. Analizar el algoritmo optForArray.c que inicializa un vector con sus valores en 1 de dos formas. ¿Cuál es más rápida?

Ejecución: *./optForArray N R*
N: tamaño de la matriz
R: cantidad de repeticiones

6. 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.

7. El algoritmo porcentaje.c calcula el porcentaje de un número dado. Ejecutar el programa y verificar que los resultados son correctos:
   El 50% de 90 debería dar 45.
   El 10% de 2530 debería dar 253.
   El 90% de 20000000 debería dar 18000000.
   El 10% de 50000000 debería dar *5000000.*
   El 40% de 50000000 debería dar *20000000.*
   Analizar ¿Por qué ocurre un error al ejecutar el 50% de *50000000? ¿Cómo lo solucionaría?*

Ejecución: *./porcentaje P N*
P es el tanto por ciento de N

(calcular el 50% de 90: ./porcentaje 50 90)