Práctica 1 - Optimización de Algoritmos Secuenciales

Consideraciones para la práctica

- *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*
- *Obtención de tiempos de ejecución:*
  - *Para todos los algoritmos se debe utilizar la función provista por la cátedra (dwalltime).*
  - *Por convención sólo deberá tomarse el tiempo de ejecución del procesamiento de datos.*
  - *El tiempo de ejecución NO debe incluir:*
    - *Alocación y liberación de memoria.*
    - *Impresión en pantalla (printf)*
    - *Inicialización de estructuras de datos.*
    - *Impresión y verificación de resultados.*

1. *Álgebra de Matrices*

a. Analizar el algoritmo matrices.c que resuelve la multiplicación de matrices C=AB donde A, B y C son matrices cuadradas de N*N:
   i. Comparar el tiempo de ejecución original con el tiempo de eliminar las funciones getValor y setValor. ¿Son necesarias estas funciones?¿Qué puede decirse del overhead introducido por los llamados a estas funciones?
   ii. Partiendo del código sin las funciones getValor y setValor, comparar la solución que almacena A, B y C por filas con las solución que almacena B por columnas. ¿Qué conclusión puede obtenerse de la diferencia en los tiempos de ejecución? ¿Cómo se relaciona con el principio de localidad?

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. Implementar una solución a la multiplicación de matrices AA donde la matriz A es de N*N. Plantear dos estrategias:

1. Ambas matrices A están ordenadas en memoria por el mismo criterio (filas o columnas) .
2. La matriz A de la izquierda sigue un criterio de ordenación en memoria (por ej: filas). La matriz A de la derecha se construye (por ej: ordenada por columnas) a partir de modificar el criterio de ordenación en memoria de la matriz A izquierda. Se debe tener en cuenta el tiempo de construcción de la nueva matriz.

¿Cuál de las dos estrategias alcanza mejor rendimiento?

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

Ejecución: *./multBloques N BS*
BS: tamaño de bloque

e. Implementar las soluciones para la multiplicación de matrices MU, ML, UM y LM. Donde M es una matriz de N*N. L y U son matrices triangulares inferior y superior, respectivamente. Analizar los tiempos de ejecución para la solución que almacena los ceros de las triangulares respecto a la que no los almacena.

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. Analizar los algoritmos productoVectorialRegistro.c y productoVectorialSinRegistro.c. Ambos programas parten de dos conjuntos de N vectores matemáticos y realizan el producto vectorial uno a uno de acuerdo al orden de cada vector en el conjunto. ¿Cuál de las dos soluciones es más rápida? ¿Por qué?:

   Ejecución: *./* productoVectorialRegistro N
   ./ productoVectorialSinRegistro N
   N: tamaño de los conjuntos de vectores

4. Optimización de instrucciones

a. El algoritmo instrucciones1.c compara el tiempo de ejecución de las operaciones básicas suma (+), resta (-), multiplicación (*) y división (/), aplicadas sobre los elementos que se encuentran en la misma posición de dos vectores x e y. ¿Qué análisis se puede hacer de cada operación? ¿Qué ocurre si los valores de los vectores x e y son potencias de 2?

   Ejecución: *./instrucciones1 N r*
   N: tamaño de los vectores
   r: número de repeticiones

b. En función del ejercicio anterior analizar el algoritmo instrucciones2.c que aplica dos operaciones distintas a cada elemento de un vector x.

   Ejecución: *./instrucciones2 N r*
   N: tamaño de los vectores
   r: número de repeticiones

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 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

b. Analizar el algoritmo GaussFor.c que calcula la suma de N números naturales consecutivos y lo compara con la suma de Gauss.

¿Por qué la suma para N=2147483647 da resultado correcto y para N=2147483648 el resultado es erróneo? ¿Cómo lo solucionaría?

Ejecución: *./GaussFor N*
N: número de elementos a sumar

6. El algoritmo overheadIF.c da tres soluciones al siguiente problema: dado un vector V y una posición P, el algoritmo cuenta la cantidad de números del vector V que son menores al elemento en la posición P.
   Analizar los tiempos obtenidos de las tres soluciones y evaluar las fuentes de overhead en cada caso.

   Ejecución: *./overheadIF N*
   N: tamaño del vector

7. Compilar y ejecutar el archivo precision.c que calcula el número de fibonacci para los elementos de un vector de tamaño N. El algoritmo compara el resultado de aplicarlo a elementos de tipo de datos entero respecto a aplicarlo a elementos de coma flotante en simple y doble precisión.
   Analizar los tiempos obtenidos para cada tipo de datos. ¿Qué conclusiones se pueden obtener del uso de uno u otro tipo de dato?

   Compilar en simple precisión: gcc -O2 -lm -o singlep precision.c
   Compilar en doble precision: gcc -O2 -DDOUBLE -lm -o doblep precision.c

   Ejecución simple precisión: *./singlep N*
   Ejecución doble precisión: *./doblep N*
   N: tamaño del vector

8. El algoritmo nreinas.c resuelve el problemas de las N Reinas. Entender el problema que resuelve el algoritmo y analizar el comportamiento del tiempo de ejecución a medida que crece N. Probar para N, de uno en uno, desde 4 a 20 ¿Qué orden de ejecución tiene?

   Ejecución doble precisión: *./nreinas N*
   N: número de reinas