ENTREGA 1

integrantes: Nahuel Panigo, Ramiro Lopes Canadell

1. Resolver los ejercicios 1 y 2 de la práctica

ejercicio1:

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.

El problema es que de la forma que estan ordenadas la matrices no pueden entrar en la cache los elementos que van a ser usados recientementes, por lo que hay una gran cantidad de fallos, por lo que se pierde mucho tiempo en ir a buscar los datos a memoria principal.
Para esto debemos cambiar la forma que se guarde la matriz A cuando la vamos guardando va a tardar un poco mas porque va a ir “saltando posiciones”, pero luego en la multiplicacion va a acceder a elemntos consecutivos, ya que una va a estar ordenado de a filas y otra de a columnas

1. Evaluar para N=512, 1024 y 2048.

procesador : Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

numero de N	tiempo en matrices.c	tiempo en matrices modificado
512	2.451059	0.548845
1024	38.536601	4.473785
2048	530.371826	35.501895

procesador : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz

numero de N	tiempo en matrices.c	tiempo en matrices modificado
512	2.080563	0.484709
1024	42.854186	4.041709
2048	461.996183	32.866403

Los tiempos son en segundos.

ejercicio2:

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?

El algoritmo recibe 3 parametros: El primer parametro es un numero que indica la cantidad de bloque por dimension, el segundo parametro es un numero que indica la dimension de cada bloque (es decir el lado del bloque, por lo tanto elevando al cuadrada este parametro me da la cantidad de elementos por bloque) y el tercer parametro tiene que ser 0 o 1 y es para imprimir o no las matrices formadas.

Multiplicando la cantidad de bloques por dimension por la dimension de cada bloque me da como resultado la dimension de la matriz. Si elevo este resultado al cuadrado me da como resultado la cantidad de elementos de la matriz. Para saber la cantidad de bloques puedo dividir la cantidad de elementos de la matriz por la cantidad de elementos de los bloques.

EJ: le paso como parametro un 4 (cant bloques por dimension) y un 10 (tamaño del lado del bloque) entonces voy a tener bloques de 4x4= 16 elementos , una matriz con lado 4 x 10 =40 y con 40 x 40 = 1600 elementos. Por lo tanto tendría 100 bloques de 16 elementos cada uno.

Para realizar la multiplicacion lo que se hace es multiplicar de a bloques. Primero se multiplica el primer bloque de A por el primer bloque de B como si fuesen matrices (es decir filas por columnas) y se va guardando en los respectivos elementos de c , que estos van a ir llenandose a medida que se complete la multiplicacion de todos los bloques que tengan que ver con la respectiva fila de c. Luego se pasa al siguiente bloque; A agarra los bloques por filas y B agarra los bloques por columnas (es decir que luego de multiplicar el primer bloque A va a seguir por el bloque de su derecha y B por el bloque de abajo).

Tamaño matriz	Numero N	Numero r	tiempo x bloque	matrices modifi
512	16	32	0.489159	0.361596
1024	16	64	4.001931	3.033393
2048	32	64	31.253305	24.088439

2. Utilizando las optimizaciones realizadas en el ejercicio 1 de la práctica 1,

resolver la multiplicación de matrices: D=ABC

Adjuntamos un archivo llamado **matrices.c** que es el que multiplica 2 matrices de forma optimizada y otro archivo llamado **matricesx3.c** que hace la multiplicacion de 3 matrices.

¿Cómo debe ordenar en memoria las matrices A, B, C y D para alcanzar un mejor rendimiento? Evaluar para N=512, 1024 y 2048.

La matriz A se encuntra ordenada por filas, la matriz B por columnas y la matriz C tambien por columnas. Primero se multiplica A x B y se guarda en una matriz llamada AB que está ordenada por filas. Se multiplica AB x C y el resultado lo guardo en una matriz D ordenada por filas.

numero de N	**Intel(R) Core(TM) i5-8250U CPU @ 1.60G**	**Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz**
512	1.131716	1.002574
1024	9.239611	8.306224
2048	70.635169	66.167312

3. Resolver el ejercicio 5 de la práctica 1.

ejercicio 5
Dada la ecuación cuadrática: x^2 − 4.0000000 x + 3.9999999 = 0, sus raíces son r1 = 2.000316228 y r2 = 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?

en el caso de la solucion float los aproximo a 0 a ambos, y la solucion double en cambio se aproximo mas, ambas soluciones tienen mas digitos en su parte decimal.Por lo tanto, la decicion float al truncar, se fija en el siguiente valor en caso de ser mayor a 5 aumenta en 1 el valor y si es menor deja el que esta.
quedo con 5 numeros despues de la coma, ya que al imprimirlo se indica %.5f de esta manera se indica la cantidad de unidades que se quieren despues de la coma

resultados:
Soluciones Float: 2.00000 2.00000
en el caso de las funciones hechas con float dieron un resultado mas distante, debido que las funciones de math.h trabajan con valores double, por lo que pudo haber un error de bits en la parte decimal.

Soluciones Double: 2.00032 1.99968
en el caso de la solucion double al truncarlo le sumo en el ultimo digito ya que el siguiente era 6.

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?

TIMES	Tiempo solucion double	Tiempo solucion float
10	1.812953	2.071735
50	8.555850	10.334417
100	16.490923	20.148656

notamos que a medida que se va incrementando la variable TIMES el tiempo del algoritmo incrementa de igual forma.Es decir, que hay una correlacion entre la variable TIMES y el tiempo del mismo.

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?

TIMES	Tiempo solucion double	Tiempo solucion float
10	1.809019	2.434114
50	9.017651	11.538612
100	17.734010	22.195793

En este caso al cambiar la variable TIMES con valores muy altos ,el tiempo no se correlaciona exactamente con el tiempo que tarda sino que va a tardar un poco menos

A diferencia de quadratic2.c en este caso cuando declara los valores de las constantes lo que hace es declarar algunas en float y otras como double:
quadractic2:

```
#define A 1.0
#define B -4.0000000
#define C 3.9999999
```

```
for (j=0; j<TIMES ; j++)
        for (i=0; i<N ; i++) {
//              flt_solve(fa[i], fb[i], fc[i]);
                float d = pow(fb[i],2) - 4.0*fa[i]*fc[i];
                float sd = sqrt(d);
                float r1 = (-fb[i] + sd) / (2.0*fa[i]);
                float r2 = (-fb[i] - sd) / (2.0*fa[i]);
        }
```

quadratic3:

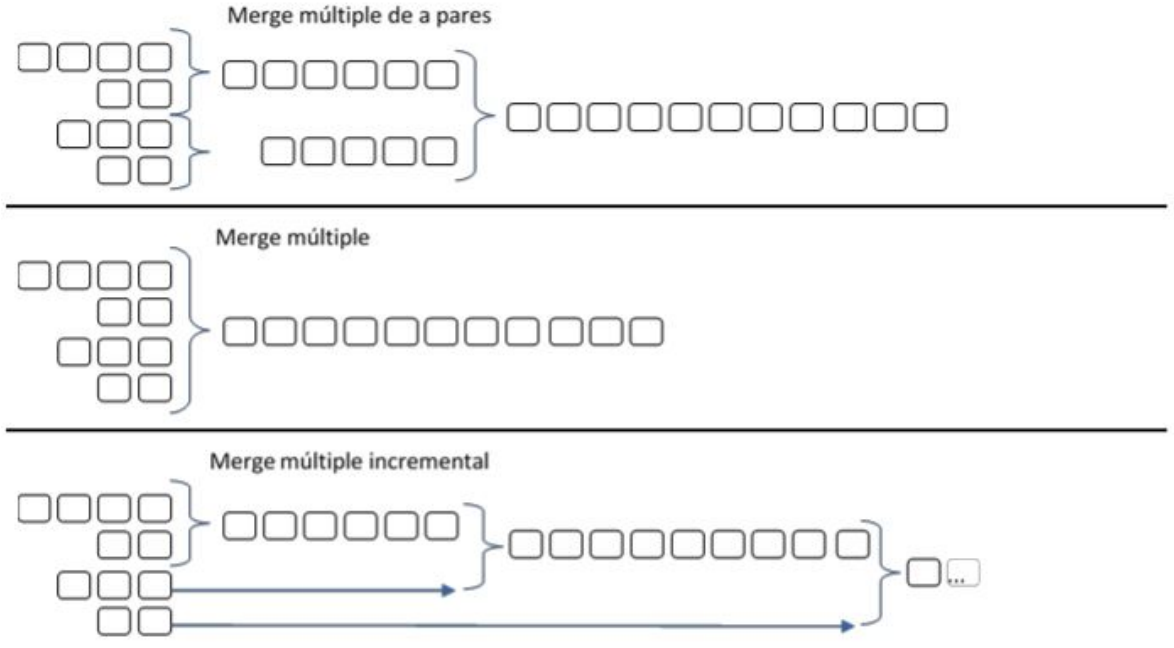
```
#define FA 1.0f
#define FB -4.0000000f
#define FC 3.9999999f

#define DA 1.0
#define DB -4.0000000
#define DC 3.9999999
```

```
or (j=0; j<TIMES ; j++)
      for (i=0; i<N ; i++) {
/             flt_solve(fa[i], fb[i], fc[i]);
              float d = powf(fb[i],2.0f) - 4.0f*fa[i]*fc[i];
              float sd = sqrtf(d);
              float r1 = (-fb[i] + sd) / (2.0f*fa[i]);
              float r2 = (-fb[i] - sd) / (2.0f*fa[i]);
      }
```

La f se usa para indicar que el numero es un float. Ademas en este caso en la parte que multiplica los float tambien le agrega una f a los numeros, porque sino cuando hace la multiplicacion entre un float y un double lo que ocurre es que el float se convierte en double, luego se realiza la multiplicacion y por ultimo los convierte en float para guardarlo en la variable float.
Ademas en el caso de cuadratic c para la interaccion con float utiliza las funciones powf y sqrtf respectivamente para no cambiar el valor del mismo, lo que tambien hace un mayor tiempo computacional.

4. Resolver el ejercicio 12 de la práctica 1.

ejercicio 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.

Cant Listas	Modo	Min	Max	Tiempo de	Tiempo	Tiempo de a

				a multiple	**incremental**	**pares**
128	Pares	2000	2000	0.925557	0.583808	0.058708
128	Uniforme_ Creciente	2000	2000	1.199346	0.603572	0.065171
128	Secuencia	2000	2000	0.928679	0.598548	0.059126
32	Secuencia	200 mil	200 mil	0.664950	0.374426	0.105689
32*4=128	Secuencia	200mil/4=50mil	50 mil	2.802618	1.459483	0.142517

tiempo multiple= (cant Listas)*cantidad de elementos de todas las listas.

tiempo incremental= [cant=(L1+L2)] sumatoria de (cant listas-2) cant+Ll(i)
la cantidad de for que se hacen va a depender de la cantidad de listas -1.
el for de adentro va a ir cambiando las iteraciones en el primer caso va a empezar con la suma de los tamanos de L1 y L2 el siguiente L1,L2 y L3 hasta llegar el ultimo que va a hacer cantidad de elementos de todas las listas

tiempo de a Pares= el tiempo de a pares es el que lo realiza mas rapido, porque este tambien realiza cant listas -1 iteraciones, pero en este caso el for anidado no va a ir incrementando con la suma de listas sino que se van a ir sumando de a pares las listas, por lo tanto esto lleva a menos iteraciones y a retardar las listas grandes que ocupan gran lugar en memoria hasta los ultimos pasos.