COMPARACIÓN DE LOS RESULTADOS ENTRE ALGORITMOS SECUENCIALES Y PARALELOS CON DIFERENTES TAMAÑOS DE MATRICES

Yair Antonio Castillo Castillo

1 Introducción

Este tabajo consiste en diseñar y codificar una paralelización a los algoritmos de suma/resta, multiplicación e inversa (por medio de reducción Gauss-Jordan) de matrices. Así mismo, en el presente hay comparativa de los resultados de los algoritmos secuenciales y paralelos probados con matrices de diferentes tamaños. Y se explica los funcionamientos de los algoritmos secuenciales y paralelos.
Para los algoritmos secuenciales se tomó como base el proyecto anterior donde se hacían sobrecarga de operaciones y memoria dinámica, para los algoritmos paralelos se modificó estos mismo algoritmos. Dentro de este reporte, solo pondré las partes importante del código, lo demás está en el de c++.

2 Desarrollo

Primero se explicará el código para ver que diferencia que hay entre el secuencial y el paralelo. Se dividirá en suma/resta, que basicamente son el mismo código, multiplicación e inversa.
Para las matrices se definió una clase como en el proyecto anterior, también usando la sobrecarga de operaciones:

```
class matriz{      // Aqui se define la clase matriz
private:
public:
    double **Tab;  // El apuntador para crear la matriz
    int nfilas;    // El numero de filas de la matriz
    int ncolumnas; // El numero de columnas de la matriz
    matriz(int filas, int columnas);   // Aqui se crea la matriz (Constructor)
    ~matriz();                         // Aqui se destruye la matriz (Destructor)
    void print();                      // Aqui se imprime la matriz
    void generacion_aleatoria();       // Aqui se llena la matriz con valores aleatorios
    matriz inversa();                 // Funcion para obtener la inversa
    matriz operator+(const matriz &A); // Funcion sumar 2 matrices (sobrecarga de operador)
    matriz operator-(const matriz &A); // Funcion restar 2 matrices (sobrecarga de operador)
    matriz operator*(const matriz &A); // Funcion mult. 2 matrices (sobrecarga de operador)
    void operator=(const matriz A); // Funcion copiar 2 matrices (sobrecarga de operador)
};
```

Para el código en paralelo se hizo de la siguiente manera, en la suma/resta de 2 matrices a cada hilo/thread se le asignó una cantidad de filas, es decir, si la matriz tiene 16 filas, y tengo solamente 4 threads, entonces cada hilo va a sumar/restar exactamente 4 filas. El primer hilo las primeras cuatro, el segundo hilo las siguientes 4 y así sucesivamente.

Para la la multiplicación es igual, solo que en la multiplicación se agarra los mismo índices de las filas pero en la columnas de las siguiente matriz.

Para la inversa, se utiliza eliminación Guass-Jordan, por lo tanto a los hilos se le va a asingar a cuales la filas el pivote fila (donde se encuentra el pivote) va a eliminar. Si se reparte igual que en ejemplo de la suma/resta y el pivote es 4, el primer hilo va a eliminar las primeras 4, pero como el 4 es pivote, ahí no hace nada y así sucesivamente con los demás hilos.

Además de esto, cada función solo toma una struct como entrada, por lo que antes de definir todas las funciones, tenemos que definir como se inician los hilos que se utilizarán para la paralelización, también definir las matrices globalmente ya que funciones toman estas matrices, por lo que tenemos:

```
#include <pthread.h>. // Para hacer paralelizacion
#include <iostream>
#include <chrono>   // Para tomar el tiempo

#define HAVE_STRUCT_TIMESPEC
#define NTHREADS 4
struct Argumentos{  // Aqui definimos los argumentos que tomara las funciones
    int ini;        // Limites
    int fin;
    int pivote;     // Para la inversa
};

int n1 = 0;  // Defimos los tamanos de las matrices, con 0 ya que despues podemos pedir
int m1 = 0;  // a la persona que inserte las dimensiones que desee
int n2 = 0;
int m2 = 0;

matriz A(n1, m1); // Defimos las matrices principales
matriz B(n2, m2);

matriz CS(n1, m1); // Defimos las matrices de los resultados
matriz CR(n1, m1);
matriz CM(n1, m2);
matriz CI(n1, m1);
}

// Esto va en cada una de la funciones
pthread_t *thr = new pthread_t[NTHREADS]; //inicializo un array con los threads
Argumentos *args = new Argumentos[NTHREADS];   // inicializo un array con los argumentos
    para cada thread
int subint = floor(n1/NTHREADS);
delete[] thr;
delete[] args;
```

En argumentos definimos el límites de las filas.
En este caso se utilizó una computadora con 4 hilos, también definimos los tamaños igual a 0, esto debido a que en el main se pedirá el tamaño, se crearan nuevas matrices y luego se copiará con la sobrecarga del operador "=" a estas matrices definidas globalmente. También al final del código anterior están definidos la inicialización de los hilos, esto estará al inicio y al final de cada función.

Es todo lo que se necesita para hacer la paralelización, y ya en cada operación se verá más a detalle como usar lo anterior, para el tiempo se utilizará lo sigueinte:

```
#include <chrono>
std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now(); //INICIO
std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now(); //FIN
std::cout<<"Tiempo(seg)= "<<(std::chrono::duration_cast<std::chrono::microseconds>(end-begin
    ).count())/1000000.0<<std::endl; //TIEMPO EN SEGUNDOS
```

donde se pondrá antes y después de cada función para poner medir el tiempo.

2.1 Suma/Resta

Para la suma se utiliza la función del proyecto anterior, que es recorrer cada entrada de la matriz de resultado y sumar/restar las entradas de las 2 matrices.

2.1.1 Suma/Resta secuencial

Primero empezaremos con la suma secuencial que su código es:

```
matriz matriz::operator+(const matriz &M){   // Funcion para sumar matrices (Se usa sobrecarga de operador)
    matriz M1(nfilas,ncolumnas);             // Aqui se crea matriz extra donde se guardara el resultado de la suma
    if (nfilas!=M.nfilas || ncolumnas!=M.ncolumnas ){  // Aqui se checa si cumplen que las dimensiones son iguales
        cout << "Las dimensiones no son iguales por lo tanto no se pueden sumar" << endl; // Aqui se regresa un mensaje de error
        M1.Tab=NULL;                         // en caso de que las dimensiones no coincidan
        return M1;                           // Aqui se regresa un apuntador doble igua a NULL
    }
    for (int i=0; i<M.nfilas;i++){           // En cada entrada de la matriz creada se suma las entradas de las matrices
        for(int j=0;j<M.ncolumnas;j++){
            M1.Tab[i][j] = (this->Tab)[i][j]+M.Tab[i][j];
        }
    }
    cout << "La suma sequencial es:" << endl; // Aqui se regresa un mensaje
    return M1;                               // Aqui se regresa la matriz que es la suma de las matrices
}
```

Lo cual es el mismo que el anterior proyecto, solo quitando el hecho de definiar las matrices resultantes con memoria dinánima. Y para ser llamado es la de la siguiente manera:

```
A.generacion_aleatoria();  // Llenamos de numeros aleatorios
B.generacion_aleatoria();
matriz CSP(n1,m1);    // Definimos la matriz resultante
CSP = A + B;              // Se usa el operador "+"
```

Para el caso de la resta es igual, solo hay un signo “-” en vez de “+”.

2.1.2 Suma/Resta paralela

Ahora nuestra función de suma/resta paralela, aquí en el primer “ for” podemos ver que le asignamos las filas que va a sumar, limitadas por lo se le da de argumentos. Solo recorre esas filas y todo lo demás es igual.

```
void* suma_paral(void *args){                          // Funcion de suma en paralelo
    Argumentos *_args = (Argumentos*) args;            // Definimos las variables
    for (int i=((_args->ini)-1);i<=((_args->fin)-1);i++){//Aqui sumamos el fina inicial y la final
        for (int j = 0; j < B.ncolumnas; j++){      // que viene en argumentos para cada hilo
            CS.Tab[i][j] = A.Tab[i][j] + B.Tab[i][j];// Sumamos cada elemento de las filas
        }
    }
    return NULL;
}
```

En este se utilizó una struct Argumentos definidos anteriormente que nos servirá de argumentos para nuestra funciones, y para llamar a nuestra función se usa lo siguiente

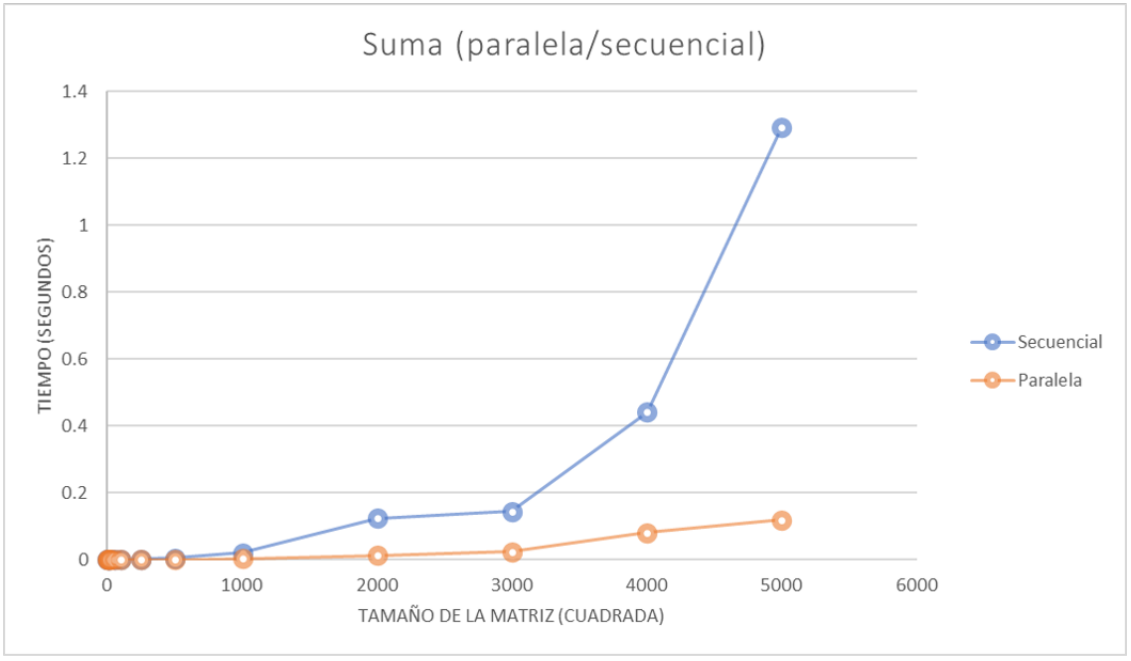

Figure 1: Podemos ver que el secuencial dura más para matrices grande

```
A = matriz(n1, m1); // Defimos las matrices principales
B = matriz(n2, m2)
CS = matriz(n1, m1); // Defimos las matrices de los resultados
A.generacion_aleatoria();  // Llevamos de numero
B.generacion_aleatoria();
if (A.nfilas!=B.nfilas || A.ncolumnas!=B.ncolumnas ){  // Aqui se checa si cumplen que las dimensiones son iguales
    cout << "Las dimensiones no son iguales por lo tanto no se pueden sumar" << endl; // Aqui se regresa un mensaje de error
    break;
}else{
    for(int i=0; i<NTHREADS; i++){ // Solo esta definiendo los rangos de donde a donde van las filas
        if(i==NTHREADS-1){
            args[i].ini = subint*i+1;
            args[i].fin = n1;
        }else{
            args[i].ini = subint*i+1;
            args[i].fin = subint*(i+1);
        }
        pthread_attr_t attr; // Definimos la asignacion de tareas a los hilos
        pthread_attr_init(&attr); // Incializamos
        pthread_create(&thr[i],&attr, suma_paral, &args[i]); // Aqui creamos los hilos
    }
    for(int i=0;i<NTHREADS;i++){ // A que terminen todos los hilos
        pthread_join(thr[i], NULL);
    }
}
```

Se puede ver que hay bastantes diferencias, en el paralelo, aparte de definir los límites para los hilos, hay un “if” para ver si son iguales a las dimensiones y está afuera de la función suma. Para el caso de la resta es igual solo cambiando "+" por un "-".

2.1.3 Resultados

La tabla de resultados (en segundos para la suma son

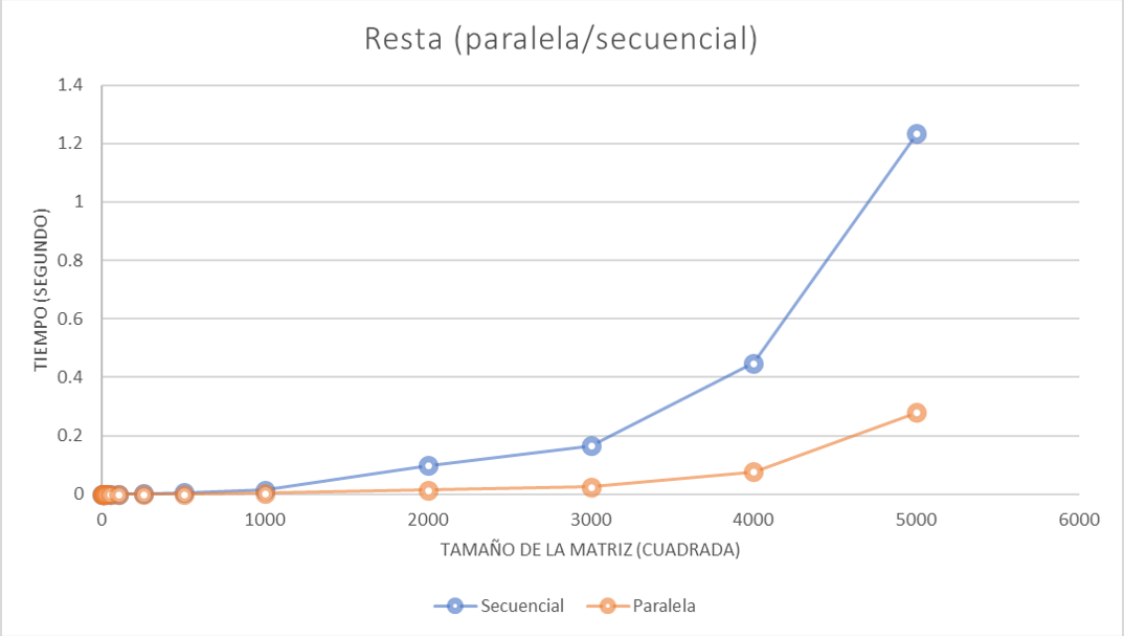

Figure 2: Podemos ver que el secuencial dura más para matrices grande

Tamaño (cuad)	1	2	5	10	25	50	100
Secuencial:	0.000017	0.000017	0.00003	0.000061	0.000088	0.000289	0.00046
Paralela:	0.00058	0.000286	0.000241	0.000353	0.000224	0.000393	0.000675

Tamaño (cuad)	250	500	1000	2000	3000	4000	5000
Secuencial:	0.002061	0.004799	0.021966	0.125086	0.144492	0.441604	1.29442
Paralela:	0.000811	0.001249	0.003894	0.013638	0.024291	0.081098	0.12

La tabla de resultados (segundos) para la resta son

Tamaño (cuad)	1	2	5	10	25	50	100
Secuencial:	0.000029	0.000014	0.000012	0.000014	0.000056	0.000096	0.000293
Paralela:	0.00067	0.000286	0.000276	0.000283	0.00016	0.000295	0.000293

Tamaño (cuad)	250	500	1000	2000	3000	4000	5000
Secuencial:	0.001403	0.005003	0.014902	0.097312	0.166323	0.44918	1.23313
Paralela:	0.000409	0.001082	0.003379	0.014356	0.024853	0.076824	0.28

Podemos ver las gráficas para la suma en Figure 1 y para la resta en Figure 2. Se puede observar que para matrices pequeñas la secuencial dura menos, pero esto se va volteando cuando las matrices son grandes, donde la paralela dura menos.

3 Multiplicación

Aquí se define un variable auxiliar, que solo guardará los elementos de la multiplicación y se le asignará a la entrada de la matriz de resultados.

3.1 Multiplicación secuencial

El código es:

```
matriz matriz::operator*(const matriz &M){   // Funcion para multiplicar matrices (Se usa sobrecarga de operador)
    matriz M1(nfilas,M.ncolumnas);           // Aqui se crea matriz extra donde se guardara el resultado de la multiplicacion
    if (this->ncolumnas!=M.nfilas){          // Aqui se checa si el # de columnas de la 1 matriz coincide con el # de renglones de la 2   matriz
        cout << "El numero de columnas de la primer matriz no coicididen con el numero de renglonoes de la segunda matriz, por lo tanto no se pueden multiplicar" << endl;
        M1.Tab=NULL;                         //  Aqui se regresa un mensaje de error en caso de que las dimensiones no coincidan
        return M1;                           // Aqui se regresa un apuntador doble igua a NULL
    }
    for (int i=0; i<nfilas;i++){             // En cada entrada de la matriz creada se multiplica el renglon de la primer matriz
        for(int j=0;j<M.ncolumnas;j++){      // por la columna de la segunda matriz
            int y=0;                         // Aqui guardamos la suma de la multiplicacion
            for (int k=0;k<(this->ncolumnas);k++){
                y=y+Tab[i][k]*M.Tab[k][j];
            }
            M1.Tab[i][j] = y;                // Aqui se asigna a cada entrada la suma
        }
    }
    cout << "La multiplicacion sequencial es:" << endl;
    return M1;                               // Aqui se regresa la matriz que es la multiplicacion de las matrices
}
```

Lo cual es el mismo que el anterior proyecto, solo quitando el hecho de definiar las matrices resultantes con memoria dinánima. Y para llamarlo es lo siguiente:

```
A.generacion_aleatoria(); //Llenamos  de  numeros  aleatorios
B.generacion_aleatoria();
matriz CMP(n1,m2);   //  Definimos  la  matriz  resultante
CMP = A * B;              // Se usa el operador "*"
```

3.1.1 Multiplicación paralela

Ahora nuestra función de suma paralela, aquí en el primer “for” podemos ver que le asignamos las filas que va a multiplicar, limitadas por lo se le da de argumentos. Solo recorre mutiplicar filas y todo lo demás es igual

```
void* mult_paral(void *args){                        // Funcion de multiplicacion en paralelo
    Argumentos *_args = (Argumentos*) args;          // Argumentos
    for (int i=((_args->ini)-1);i<=((_args->fin)-1);i++){// Aqui sumamos el fina inicial y la final
        for (int j = 0; j < m2; j++){        // En cada entrada se multiplica el renglon de la primer matriz por la columna de la segunda matriz
            double y = 0;                            // Aqui guardamos la suma de la multiplicacion
            for (int k = 0; k < m1; k++){
                y=y+A.Tab[i][k]*B.Tab[k][j];
            }
            CM.Tab[i][j] += y;                // Aqui se asigna a cada entrada la suma
        }
    }
    return NULL;
}
```

En este se utilizó una struct Argumentos definidos anteriormente que nos servirá de argumentos para nuestra funciones, para llamar a nuestra función se usa lo siguiente

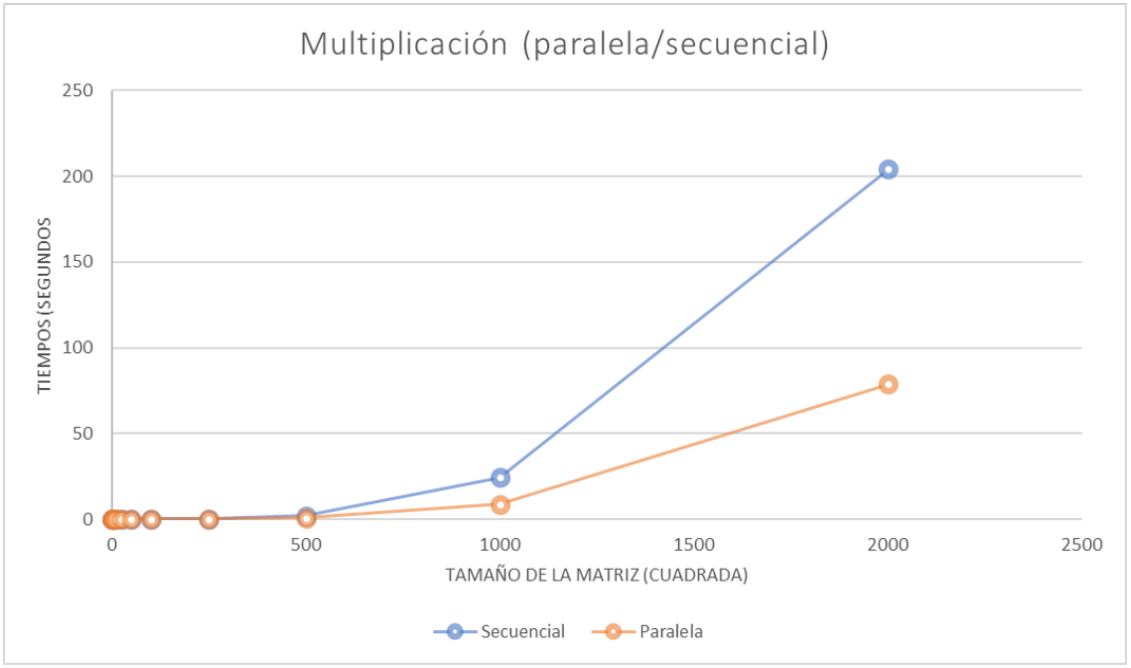

Figure 3: Podemos ver que el secuencial dura más para matrices grande

```
A = matriz(n1, m1); // Defimos las matrices principales
B = matriz(n2, m2);
CM = matriz(n1, m2); // Defimos las matrices de los resultados
A.generacion_aleatoria();
B.generacion_aleatoria();
if (A.ncolumnas!=B.nfilas){      // Aqui se checa si el # de columnas de la 1   matriz
    coincide con el # de renglones de la 2   matriz
    cout << "El numero de columnas de la 1   matriz no coicididen con el numero de
    renglonoes de la 2   matriz, por lo tanto no se pueden multiplicar" << endl;
    break;
}else{
    for(int i=0; i<NTHREADS; i++){ // Solo esta deinfiendo los rangos de donde a a donde van
     las filas
        if(i==NTHREADS-1){
            args[i].ini = subint*i+1;
            args[i].fin = n1;
        }else{
            args[i].ini = subint*i+1;
            args[i].fin = subint*(i+1);
        }
        pthread_attr_t attr; // Definimos la asignacion de tareas a los hilos
        pthread_attr_init(&attr); // Inicializamos
        pthread_create(&thr[i],&attr, mult_paral, &args[i]); //Aqui creamos los hilos
    }
    for(int i=0;i<NTHREADS;i++){
        pthread_join(thr[i], NULL); // A que se terminen los hilos
    }
}
```

Podemos ver que se parece mucho a la de suma/resta paralela, definiendo de la misma manera las filas que les van a asignar a los hilos, y también que el anterior, el "if" está afuera de la función.

3.1.2 Resultados

La tabla de resultados (segundos) para la multiplicación son

Tamaño (cuad)	1	2	5	10	25	50
Secuencial:	0.000005	0.00001	0.000013	0.000015	0.000121	0.001304
Paralela:	0.00018	0.000303	0.000236	0.000177	0.000227	0.000727

Tamaño (cuad)	100	250	500	1000	2000
Secuencial:	0.007188	0.124876	1.91805	24.4182	204.279
Paralela:	0.003172	0.053482	0.761775	8.81111	78.8364

La gráfica de esta función está en Figura 3, igual que la suma y resta, para matrices pequeñas la secuencial es más rápida, pero si para matrices grandes hay un gran diferencia, casi 3 veces dura más la secuencial que la paralela para el tamaño de muestra igual a 2000 (matriz cuadrada).

4 Inversa

La inversa se utilizar la eliminación de Gauss-Jordan, donde para cada elemento del pivote se va a eliminar a las filas arriba de este y abajo a la vez.

4.1 Inversa secuencial

El código es:

```
matriz matriz::inversa(){                    // Funcion para obtener la inversa usando Gauss-Jordan
    matriz Inver(nfilas,ncolumnas);    // Aqui se crea una matriz extra donde se guardara la inversa con las dimensiones cambiadas
    if (ncolumnas!=nfilas){                  // Aqui se checa si la matriz es cuadrada para ver si podemos obtener la inversa
        cout << "No se puede obtener la inversa, las matriz tiene que ser cuadrada" << endl;
     // Aqui se regresa un mensaje de error
        cout << endl;                        // en caso de que la matriz no es cuadrada
        Inver.Tab=NULL;                     // Aqui se regresa un apuntador doble igua a NULL
        return Inver;
    }
    for (int i=0; i<nfilas;i++){             // Para obterner la inversa se tiene que crear una matriz identidad cuadrada y poder aplicar
        for(int j=0;j<ncolumnas;j++){        // Gauss-Jordan
            Inver.Tab[i][j] = (double) 0;   // 0 en todas las entradas
            if (i==j){
                Inver.Tab[i][j] =  (double) 1; // 1 en la diagonal
            }
        }
    }
    for (int i=0;i<(nfilas);i++){            // Aqui se empieza haciendo la matriz escalonado por renglones, donde se ira eliminando todos
        double a = Tab[i][i];                // los elementos que est n debajo de la diagonal
        int k=0;
        while (a==0){                       // Si el elemento de la diagonal es 0, ver que exista un elemento abajo de esa diagonal que no sea 0
            for (k=i;k<nfilas;k++){
                a = Tab[k][i];      // Ver cada una
                if (a!=0){
                    break;
                }else if (k==(nfilas) && a==0){// En caso no es existir, entonces regresar error de que no existe inversa para esa matriz
                    cout << "La matriz no es invertible " << endl;
                    cout << endl;
```

```
                    Inver.Tab = NULL;       // Aqui se regresa un apuntador doble igua a NULL
                    return Inver;
                }
            }
        }
        if (k>i){                       // Si la diagonal fue 0 y existe otro valor abajo que no es 0, entonces se intercambian los renglones
            matriz aux(1,nfilas);
            matriz aux1(1,nfilas);
            aux.Tab[1] = Tab[k];      // Se intercambian para la matriz original
            Tab[k]=Tab[i];
            Tab[i] = aux.Tab[1];
            aux1.Tab[1] = Inver.Tab[k];    // Y tambien se cambia en la matriz identidad (Ya modificada si no es la primera interacion)
            Inver.Tab[k]=Inver.Tab[i];
            Inver.Tab[i] = aux1.Tab[1];
        }
        for (int k=0;k<nfilas;k++){  // Aqu dividimos la fila del pivote entre el pivote
            Inver.Tab[i][k]=Inver.Tab[i][k]/a;
            Tab[i][k]=Tab[i][k]/a;
        }
        a = Tab[i][i]=1;            // Aqui vamos eliminando de arriba para abajo para cada pivote
        for (int p=0;p<nfilas;p++){
            if (p!=i){
                double b = Tab[p][i];
                for (int j=0;j<ncolumnas;j++){
                    Tab[p][j] = (double) Tab[p][j] - (double)Tab[i][j]*(double)b;
                    Inver.Tab[p][j] = (double)Inver.Tab[p][j]-(double)Inver.Tab[i][j]*(double)b;
                }
            }
        }
    }
    cout << "La matriz inversa sequencial es:" << endl; // Aqui se imprime un mensaje
    return Inver;                           // Regresamos la matriz transpuesta
}
```

Lo cual es el mismo que el anterior proyecto, solo quitando el hecho de definiar las matrices-resultantes con memoria dinánima. Y para llamarlo es lo siguiente:

```
A = matriz(n1, m1); // Definimos la matriz principal
matriz CIP(n1,m1);  // Definimos la matriz de resultado
matriz A2(n1,m1);   // Definimos una alternativa
A.generacion_aleatoria(); // La llenamos de numero aleatorios
for (int i=0;i<n1;i++){   // Duplicamos A a A2
    for (int j=0;j<m1;j++){
        A2.Tab[i][j]=A.Tab[i][j];
    }
}
CIP = A2.inversa();  //Sacamos la inversa
```

La función de obtener inversas paralela modifica la matriz que se le obtendrá la inversa, por lo que si se aplica paralela y luego secuencial (es lo que sucede), da que la inversa de A es la identidad, por lo cual hay un error, así que se dicidió duplicar A con el nombre de A2. En el proyecto anterior lo que se hacia era eliminar hacía abajo, primero dejando una tringular superior y luego eliminar de abajo hacía arriba. En este caso se modificó para que cuando vaya tomando los pivotes elimine hacia abajo y hacia arriba de una vez, esto servirá para la inversa paralela.

4.1.1 Inversa paralela

Ahora nuestra función de inversa paralela, primero tenemos una función que no paralelizada “inter_filas” que es checar si es invertible revisando los pivotes y evitar que no haya 0 en la diagonal, de haber eso, intercambiarlos (Esto está dentro la inversa secuencial, se puso afuera aquí por simplicidad). Luego ya tenemos la función del inversa “inversa_paral”, aquí en el segundo “for” podemos ver que le asignamos las filas que va a eliminar usando la fila pivote que es la fila donde está el pivote, estos límites están dado por el argumentos.

```
1  void inter_filas(int pivot){
2      int i=pivot;                        // Aqui se empieza haciendo la matriz escalonado por
       renglones, donde se ira eliminando todos
3      double a = A.Tab[i][i];             // los elementos que esten debajo de la diagonal
4      int k=0;
5      while (a==0){                       // Si el elemento de la diagonal es 0, ver que exista
       un elemento abajo de esa diagonal que no sea 0
6          for (k=i;k<A.nfilas;k++){
7              a = A.Tab[k][i];            // Ver cada elemento de la columna
8              if (a!=0){
9                  break;
10             }else if (k==(A.nfilas) && a==0){// En caso no es existir, entonces regresar
       error de que no existe inversa para esa matriz
11                 cout << "La matriz no es invertible " << endl;
12                 cout << endl;
13                 A.Tab = NULL;           // Aqui se regresa un apuntador doble igua a NULL
14                 return;
15             }
16         }
17     }
18     if (k>i){                           // Si la diagonal fue 0 y existe otro valor abajo que no
        es 0, entonces se intercambian los renglones
19         matriz aux(1,A.nfilas);
20         matriz aux1(1,A.nfilas);
21         aux.Tab[1] = A.Tab[k];          // Se intercambian para la matriz original
22         A.Tab[k]=A.Tab[i];
23         A.Tab[i] = aux.Tab[1];
24         aux1.Tab[1] = CI.Tab[k];        // Y tambien se cambia en la matriz identidad (Ya
       modificada si no es la primera interacion)
25         CI.Tab[k]=CI.Tab[i];
26         CI.Tab[i] = aux1.Tab[1];
27     }
28 }
29 void* inversa_paral(void *args){            // Funcion para obtener la inversa usando Gauss-
       Jordan
30     Argumentos *_args = (Argumentos*) args; // Los argumentos
31     int filacom = (_args->ini)-1;           // Definimos variables
32     int filafin = (_args->fin)-1 ;
33     int i = (_args->pivote);
34     double a = A.Tab[i][i];
35     for (int k=0;k<A.nfilas;k++){           // Empezamos diviendo la fila entre el pivote
36         CI.Tab[i][k]=CI.Tab[i][k]/a;
37         A.Tab[i][k]=A.Tab[i][k]/a;
38     }
39     a = A.Tab[i][i]=1;                      // Hacemos el pivote
40     for (int p=filacom;p<=filafin;p++){     // Empezamos a eliminar las filas que le dimos a
        cada hilo
41         if (p!=i){                          // Aqu  por si a un hilo le toca la fila que
       toma como pivote que no haga nada
42             double b = A.Tab[p][i];         // Empiezan a eliminar la filas
43             for (int j=0;j<A.ncolumnas;j++){
44                 A.Tab[p][j] = (double) A.Tab[p][j] - (double)A.Tab[i][j]*(double)b;
45                 CI.Tab[p][j] = (double)CI.Tab[p][j]-(double)CI.Tab[i][j]*(double)b;
46             }
47         }
48     }
49     return NULL;
50 }
```

También se tiene que hacer la matriz identidad afuera (esto está adentro de la función inversa secuencial), por lo que para llamarlo es lo siguiente:

```
1  if (A.ncolumnas!=A.nfilas){  // Aqui se checa si la matriz es cuadrada para ver si podemos
       obtener la inversa
2      cout << "No se puede obtener la inversa, la matriz tiene que ser cuadrada" << endl; //
       Aqui se regresa un mensaje de error
3      cout << endl;
4      break;
5  }else{
6      for (int i=0; i<CI.nfilas;i++){            // (Creeacion de inversa) Agregamos ceros a
       la afuera de la diagonal
7          for(int j=0;j<CI.ncolumnas;j++){       // Y 1 en la diagonal
8              CI.Tab[i][j] = (double) 0;         // 0 en todas las entradas
9              if (i==j){
10                 CI.Tab[i][j] =  (double) 1;    // 1 en la diagonal
11             }
12         }
13     }
14     for (int j=0;j<CI.ncolumnas;j++){
15         inter_filas(j);
16         if (A.Tab==NULL){
17             break;
18         }
19         for (int i=0; i<NTHREADS; i++){
20             if(i==NTHREADS-1){
21                 args[i].ini = subint*i+1;
22                 args[i].fin = n1;
23                 args[i].pivote=j;
24             }else{
25                 args[i].ini = subint*i+1;
26                 args[i].fin = subint*(i+1);
27                 args[i].pivote=j;
28             }
29             pthread_attr_t attr;
30             pthread_attr_init(&attr);
31             pthread_create(&thr[i],&attr, inversa_paral, &args[i]);
32         }
33         for (int i=0;i<NTHREADS;i++){
34             pthread_join(thr[i], NULL);
35         }
36     }
37 }
```

Aquí ya hay bastante diferencias, la matriz identidad se crea afuera de la función inversa paralela (en la inversa secuencial que está adentro), esto es porque si estaba adentro cada que se aplicara la función de inversa paralela se estaría creando la matriz identidad y por lo tanto no generaría la inversa. Aquí tenemos un "for" extra en el renglón 14 del código anterior, este sirve para ir recorriendo los pivotes, esto no se puede hacer de forma paralela, porque tenemos que esperar a que termine las eliminación de las filas del pivote anterior, así que por eso está afuera y de manera secuencial. Tambien el "pthread_join" está exactamente despues de que se termina el segundo "for", esto es que tiene que esperar a que se termine de eliminar las filas cuando se está usando un pivote, esto es para poder seguir continuando con el método de gauss-jordan.

4.1.2 Resultados

La tabla de resultados (segundos) para la multiplicación son

Tamaño (cuad)	1	2	5	10	25	50
Secuencial:	0.000005	0.00001	0.000015	0.000044	0.000184	0.001435
Paralela:	0.00023	0.000464	0.001143	0.001618	0.0003792	0.008163

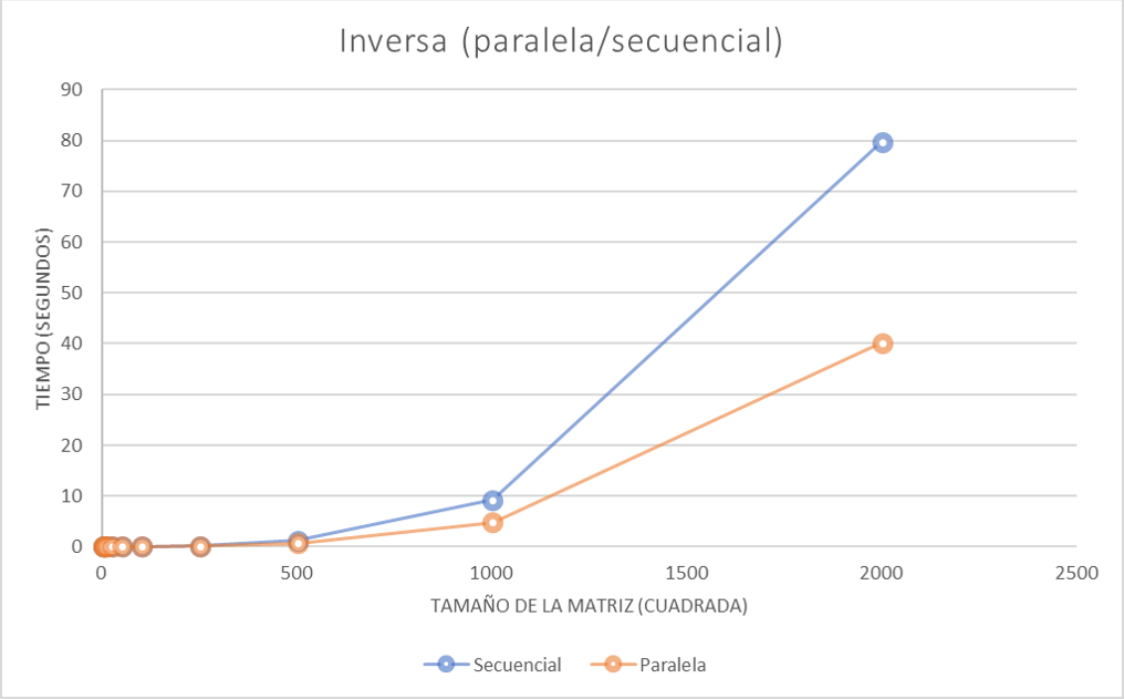

Figure 4: Podemos ver que el secuencial dura más para matrices grande

Tamaño (cuad)	100	250	500	1000	2000
Secuencial:	0.009026	0.148469	1.26046	9.19652	79.7634
Paralela:	0.022019	0.1107299	0.71616	6 4.78116	40.1454

La gráfica se puede ver en Figure 4. Igual que la anterior, la paralela es mejor para matrices grandes y la secuencial para matrices pequeñas. Aunque podemos ver que no es tan notable como la multiplicación, aquí es casi el doble.

5 Main y consola

Aquí se dirá como está construido el main.
El main es un función "switch" donde habrá varias opciones y en cada opción estará cada una de las funciones definidas anteriormente, cuando se corre el programa aparecerá en la consona lo siguiente:

```
Eliga la opcion de la operacion de algebra de matrices que se desea hacer:
1. Sumar 2 matrices
2. Restar 2 matrices
3. Multiplicar 2 matrices
4. Obtener la inversa
5. Todas las anteriores juntas
6. Salir
Opcion:
```

Cuando se elige la primera aparecerá que inserte las matrices y además le pedirá que si quiere imprimir las matrices:

```
Opcion:1
Inserte el numero de renglones de la 1 matriz:
Inserte el numero de columnas de la 1 matriz:
```

```
Inserte el numero de renglones de la 2 matriz:
Inserte el numero de columnas de la 2 matriz:

Desea imprimir las matrices? (y/n):
```

Una vez llenado lo anterior le aparecerá las matrices, la suma de matrices y el tiempo que duró el algoritmo secuencial y el algoritmo paralelo, y volverá al menú inicial.

Si inicia 2

```
Opcion:2
Se restara la 1 matriz menos la 2 matriz (en ese orden)
Inserte el numero de renglones de la 1 matriz:
Inserte el numero de columnas de la 1 matriz:
Inserte el numero de renglones de la 2 matriz:
Inserte el numero de columnas de la 2 matriz:

Desea imprimir las matrices? (y/n):
```

Al igual que el anterior, le aparecerán las matrices, la resta y el tiempo que duró.

Si inicia 3

```
Opcion:3
Se multiplicar  la 1   matriz por la 2   matriz (en ese orden)
Para esta operacion tiene que coincidir el no. de columnas de la 1   matriz con el no. de
    renglones de la 2   matriz

Inserte el numero de renglones de la 1 matriz:
Inserte el numero de columnas de la 1 matriz:
Inserte el numero de renglones de la 2 matriz:
Inserte el numero de columnas de la 2 matriz:

Desea imprimir las matrices? (y/n):
```

Al igual que el anterior, le aparecerán las matrices, la multplicacion y el tiempo que duró.

Si inicia 4

```

\nonindent Opcion:4

Inserte el numero de renglones de la matriz:
Inserte el numero de columnas de la matriz:

Desea imprimir las matrices? (y/n):
```

Al igual que el anterior, le aparecerá la matriz y el tiempo que duró.

Si inicia 5

```
Opcion:5

Inserte el numero de renglones de la 1 matriz:
Inserte el numero de columnas de la 1 matriz:
Inserte el numero de renglones de la 2 matriz:
Inserte el numero de columnas de la 2 matriz:

Desea imprimir las matrices? (y/n):
```

Aquí harán todas las operaciones anteriores, primero aparecerán las matrices, luego la suma y sus tiempo, resta y sus tiempos, multiplicación y sus tiempo y por ultima la inversa y sus tiempos. Y volverá al menú anterior.
Si inicia 6 se sale del programa.

6 Conclusiones

Podemos ver que claramente la paralelización es más eficiente cuando las matrices son grandes, y es menos eficiente cuando la matriz es pequeña. También hay poco diferencia de las funciones entre paralela y secuencial, lo que cambiar es lo que se le dará de entrada a las funciones y la creación de hilos que afuera de la matrices.

Faltaría comparar las varias formas de paralelizar estos códigos y ver cual es el mas eficiente.