

# Paralelización de la Factorización $QR$ por Tiles

Mihaita Alexandru Lupoiu

Computación Paralela y Distribuida, Universidad Politécnica de Valencia

18 de enero de 2016

## Resumen

Los sistemas multinúcleo están continuamente ganando terreno en el mundo de la informática general y más aún cuando se trata de High Performance Computing. Por esta razón los algoritmos de álgebra lineal deben ser reformulados o se deben desarrollar nuevos algoritmos con el fin de tomar ventaja de las características de estos nuevos procesadores. Este trabajo está basado en las Working Notes 190[2] y 191[3] de LAPACK que presenta un algoritmo para la factorización QR, donde las operaciones se pueden representar como una secuencia de tareas pequeñas que operan por bloques.

Estas tareas se pueden organizar de forma dinámica y la ejecución se realiza en base a las dependencias entre ellas y en la disponibilidad de recursos computacionales. Esto puede dar lugar a una ejecución fuera de orden de las tareas que ocultarán completamente la presencia de tareas secuenciales intrínsecas en la factorización.

*Keywords:*  $QR$ , matriz, factorización, LAPACK, paralelización, BLAS.

## 1. Introduction

La factorización QR es una herramienta computacional muy útil para resolver sistemas de ecuaciones lineales, problemas de mínimos cuadrados lineales o cálculo de los valores propios.

Una forma de obtener las matrices ortogonales necesarias para realizar la factorización  $QR$  de una matriz  $A$   $n \times n$  dada, es a través del uso de las matrices de Householder.

En este trabajo se analizarán las técnicas y propuestas de algoritmos para realizar la factorización QR por "tiles" utilizando Householder. En concreto, se centra en los artículos "Working Notes 190" y "Working Notes 191" de LAPACK.

El trabajo explicará en que consiste la Factorización QR por Bloques para posteriormente explicar en la segunda parte como se ha planteado la realización de la Factorización QR por Tiles. Para acabar se explicará la implementación que se ha intentado realizar del mismo.

## 2. La Factorización $QR$ por Bloques

La factorización QR es una transformación que factoriza una matriz inicial  $A$  de tamaño  $m \times n$  en dos matrices  $Q$  y  $R$  donde  $Q$  es una matriz unitaria de tamaño  $n \times n$  y  $R$  es una matriz triangular de tamaño  $m \times m$ .

Esta factorización se realiza aplicando  $\min(m, n)$  reflexiones de Householder a la matriz  $A$ . Como las reflexiones de Householder son transformaciones ortogonales, esta factorización es más estable comparado con la LU pero a cambio de un mayor coste computacional. La factorización QR tiene un coste de  $2n^2(m - n/3)$  Flops, mientras que la LU tiene un coste de  $n^2(m - n/3)$  FLops.

En la librería de LAPACK se utiliza una versión peculiar de la factorización QR que consigue mejores prestaciones en arquitecturas con varios niveles de memoria gracias a la división por bloques.

Este algoritmo se basa en acumular una cantidad de transformaciones de Householder y es denominado *panel factorization*. Luego se aplica todas a la vez aprovechando las rutinas de BLAS de nivel 3. Esta técnica utilizada para acumular las transformaciones de Householder se denomina técnica compacta WY.

La rutina de LAPACK que realiza la factorización QR se denomina DGEQRF y se explicará a continuación.

Considerando una matriz  $A$  de tamaño  $m \times n$  que se puede representar como:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

Donde:

- $A_{11}$  es de tamaño  $b \times b$
- $A_{12}$  es de tamaño  $b \times (n - b)$
- $A_{21}$  es de tamaño  $(m - b) \times b$
- $A_{22}$  es de tamaño  $(m - b) \times (n - b)$

El algoritmo de LAPACK para realizar la factorización QR se puede describir como una secuencia de pasos, y cada paso consiste en realizar la transformación de la Ecuación (1):

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \Rightarrow \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix}, \begin{pmatrix} R_{11} & R_{12} \\ 0 & \tilde{A}_{22} \end{pmatrix} \quad (1)$$

La transformación de la Ecuación (1) se obtiene en dos pasos:

1. **Factorización del Panel.** En este paso se realiza una transformación QR del panel  $(A_{*1})$  como se puede observar en la Ecuación (2).

$$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} \Rightarrow \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix}, (T_{11}), (R_{11}) \quad (2)$$

Esta operación produce  $b$  reflexiones de Householder  $(V_{*,1})$  y una matriz triangular superior  $R_{11}$  de tamaño  $b \times b$  que representa una parte de la matriz final  $R$ , mediante la rutina DGEQR2 disponible en LAPACK. En este paso se genera la matriz triangular  $T_{11}$  de tamaño  $b \times b$  mediante la rutina DLARFT disponible en LAPACK. La matriz  $V_{11}$  es una matriz triangular inferior unitaria de tamaño  $b \times b$ . Los arrays de  $V_{*,1}$  y  $R_{11}$  no necesitan espacio extra dado que sobrescribe  $A_{*,1}$ , pero se necesita un espacio temporal de trabajo para  $T_{11}$ .

2. **Actualización de la submatriz.** En este paso la transformación realizada en el apartado anterior es aplicado al resto de matriz como se puede observar en la ecuación (3).

$$\begin{pmatrix} R_{12} \\ \tilde{A}_{22} \end{pmatrix} = \begin{pmatrix} I - \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix} \cdot (T_{11}) \cdot \begin{pmatrix} V_{11}^T & V_{21}^T \end{pmatrix} \end{pmatrix} \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \quad (3)$$

Esta operación se realiza mediante la rutina de LAPACK DLARFB y genera la matriz  $R_{12}$  de tamaño  $b \times (n - b)$ , que es una parte de la matriz final de  $R$  y una matriz  $\tilde{A}_{22}$ .

Esta transformación (1) se vuelve a repetir, pero esta vez a la submatriz  $\tilde{A}_{22}$  y así sucesivamente hasta llegar al final de la matriz  $A$ .

EL valor de  $b \ll m, n$  por defecto tiene el valor de 32 en LAPACK-3.1.1, pero este valor se puede modificar para intentar buscar uno más óptimo dependiendo de las características de la arquitectura.

### 3. Tiled QR Factorization

La idea de distribución dinámica y la ejecución fuera de orden de las tareas se puede aplicar a toda clase de algoritmos que permiten la paralelización de las operaciones de Álgebra Lineal comunes. Un ejemplo es la factorización de Cholesky donde no es necesario realizar ningún cambio algorítmico ya que se puede paralelizar de forma natural por "tiles", mientras que el algoritmo por bloque de la factorización QR tiene un cuello de botella computacional por el tipo de matriz que se emplean. Con el fin de tener una mayor granularidad en la QR, las operaciones se tienen que paralelizar por tareas y por lo tanto se va a necesitar un cambio algorítmico importante en la QR.

El cambio algorítmico que se propone en el artículo "Working Notes 191" es el siguiente:

Suponiendo que la matriz  $A$  es de tamaño  $pb \times qb$

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1q} \\ A_{21} & A_{22} & \dots & A_{2q} \\ \vdots & & \ddots & \vdots \\ A_{p1} & A_{p2} & \dots & A_{pq} \end{pmatrix}$$

donde  $b$  es el tamaño de bloque para cada  $A_{ij}$  de tamaño  $b \times b$ , la factorización QR se puede realizar mediante el Algoritmo(1).

---

**Algorithm 1:** ALGORITMO DE LA FACTORIZACIÓN QR POR BLOQUES.

---

```
1 for  $k = 1, 2, \dots, \min(p, q)$  do
2   DGEQRT( $A_{kk}, V_{kk}, R_{kk}, T_{kk}$ );
3   for  $j = k + 1, k + 2, \dots, q$  do
4     DLARFB( $A_{kj}, V_{kk}, T_{kk}, R_{kj}$ );
5   end
6   for  $i = k + 1, k + 1, \dots, p$  do
7     DTSQRT( $R_{kk}, A_{ik}, V_{ik}, T_{ik}$ );
8     for  $i = k + 1, k + 1, \dots, p$  do
9       DSSRFB( $R_{kj}, A_{ij}, V_{ik}, T_{ik}$ );
10    end
11  end
12 end
```

---

Como se puede observar en el Algoritmo(1), la factorización QR por bloques se basa en las siguientes cuatro operaciones:

**DGEQRT.** Esta subrutina se utiliza para realizar la factorización QR por bloques del bloque  $A_{kk}$  de tamaño  $b \times b$ . El tamaño de bloque interno es  $s$ . El resultado de esta operación es una matriz triangular superior  $R_{kk}$ , una matriz triangular inferior  $V_{kk}$  que contiene las  $b$  reflexiones de Householder y una matriz triangular superior  $T_{kk}$  como fue definida por la técnica de WY para acumular las transformaciones. Esta subrutina se basa en la DGEQRF de LAPACK, por lo que aprovecha las rutinas de BLAS 3.

DGEQRT( $A_{kk}, V_{kk}, R_{kk}, T_{kk}$ ):

$$A_{kk} \longrightarrow (V_{kk}, R_{kk}, T_{kk}) = QR(A_{kk})$$

**DLARFB.** Esta subrutina de LAPACK basado en BLAS 3 se utiliza para aplicar las transformación de  $(V_{kk}, T_{kk})$  realizadas por la subrutina DGEQRT al bloque  $A_{kj}$  generando  $R_{kj}$ .

DLARFB( $A_{kj}, V_{kk}, T_{kk}, R_{kj}$ ):

$$A_{kj}, V_{kk}, T_{kk} \longrightarrow R_{kj} = (I - V_{kk}T_{kk}V_{kk}^T)A_{kj}$$

**DTSQRT.** Esta subrutina realiza la factorización QR por bloques que se aplica al bloque  $R_{kk}$  (submatriz triangular superior) y al bloque  $A_{ik}$  de forma conjunta. Esta subrutina devolverá una matriz triangular superior  $\tilde{R}_{kk}$  que contiene las  $b$  reflexiones de Householder, donde  $b$  es el tamaño del bloque. También se crea una matriz triangular superior  $T_{ik}$  definida por la técnica de WY comprimido para cumular las transformaciones de householder.

**DTSQRT**( $R_{kk}, A_{ik}, V_{ik}, T_{ik}$ ):

$$\begin{pmatrix} R_{kk} \\ A_{ik} \end{pmatrix} \longrightarrow (V_{ik}, T_{ik}, R_{kk}) = QR \begin{pmatrix} R_{kk} \\ A_{ik} \end{pmatrix}$$

**DSSRFB.** Esta subrutina actualiza la matriz formada por los dos bloques  $A_{kj}$  y  $A_{ij}$ , después de aplicar transformaciones realizadas por **DTSQRT**

**DSSRFB**( $R_{kj}, A_{ij}, V_{ik}, T_{ik}$ ):

$$\begin{pmatrix} R_{kj} \\ A_{ij} \end{pmatrix}, V_{ik}, T_{ik} \longrightarrow \begin{pmatrix} R_{kj} \\ A_{ij} \end{pmatrix} = (I - V_{ik}T_{ik}V_{ik}^T) \begin{pmatrix} R_{kj} \\ A_{ij} \end{pmatrix}$$

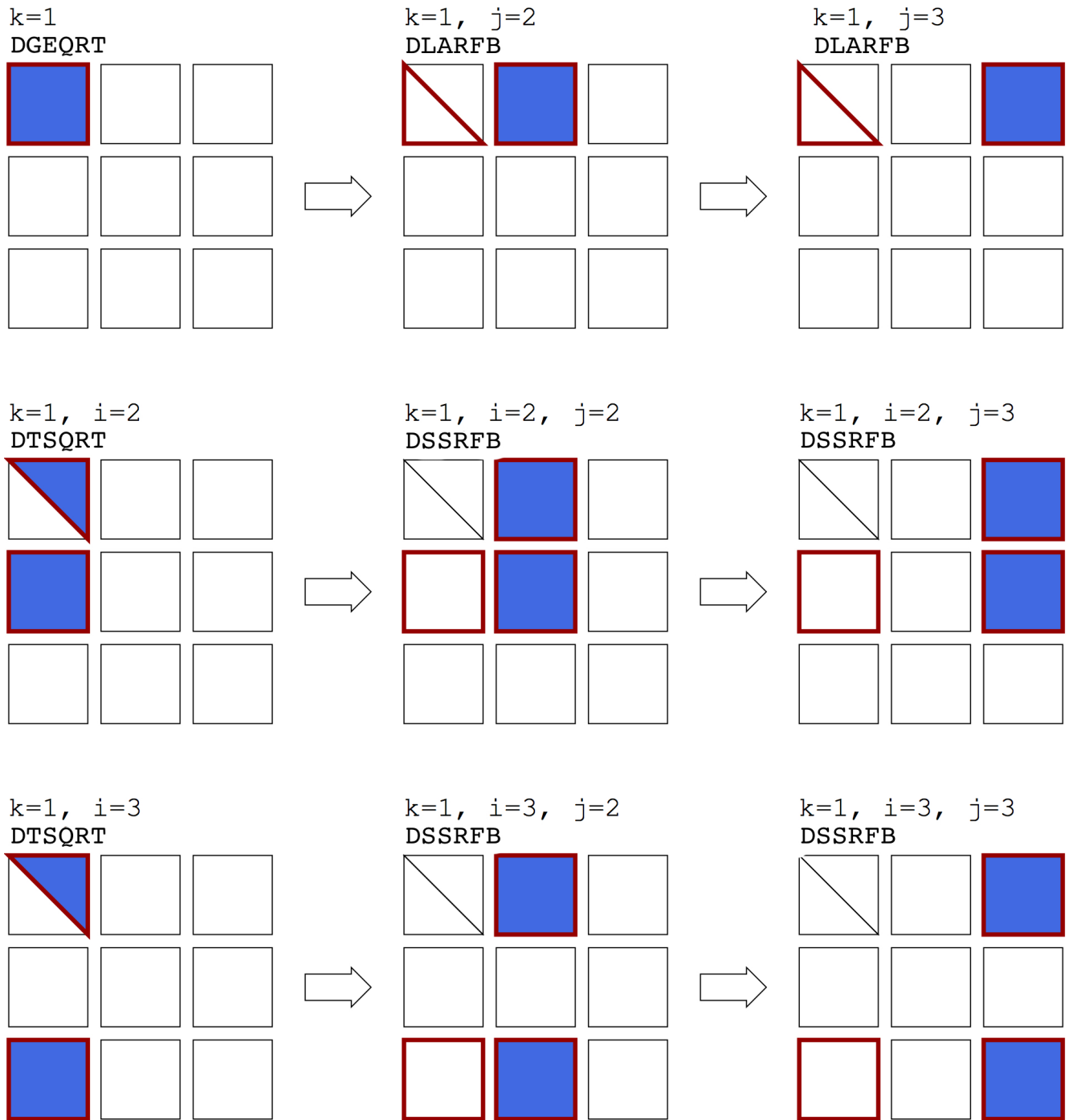


Figura 1: Graphical representation of one repetition of the outer loop in Algorithm(1) on a matrix with  $p = q = 3$ .

La figura 1 es una representación gráfica de una de las repeticiones ( $k = 1$ ) del algoritmo(1), donde  $p = q = 3$ . Los bordes rojos representan que bloques de la matriz se están leyendo y la área azul de los bloques representa los lugares donde se está escribiendo en la matriz. Las matrices  $T_{kk}$  no se muestran en esta figura para evitar confusiones.

El algoritmo(1) se puede representar como un Grafo Acíclico Dirigido (DAG) donde los nodos representan tareas que se aplican en los bloques de tamaño  $b \times b$  y las aristas representan las dependencias.

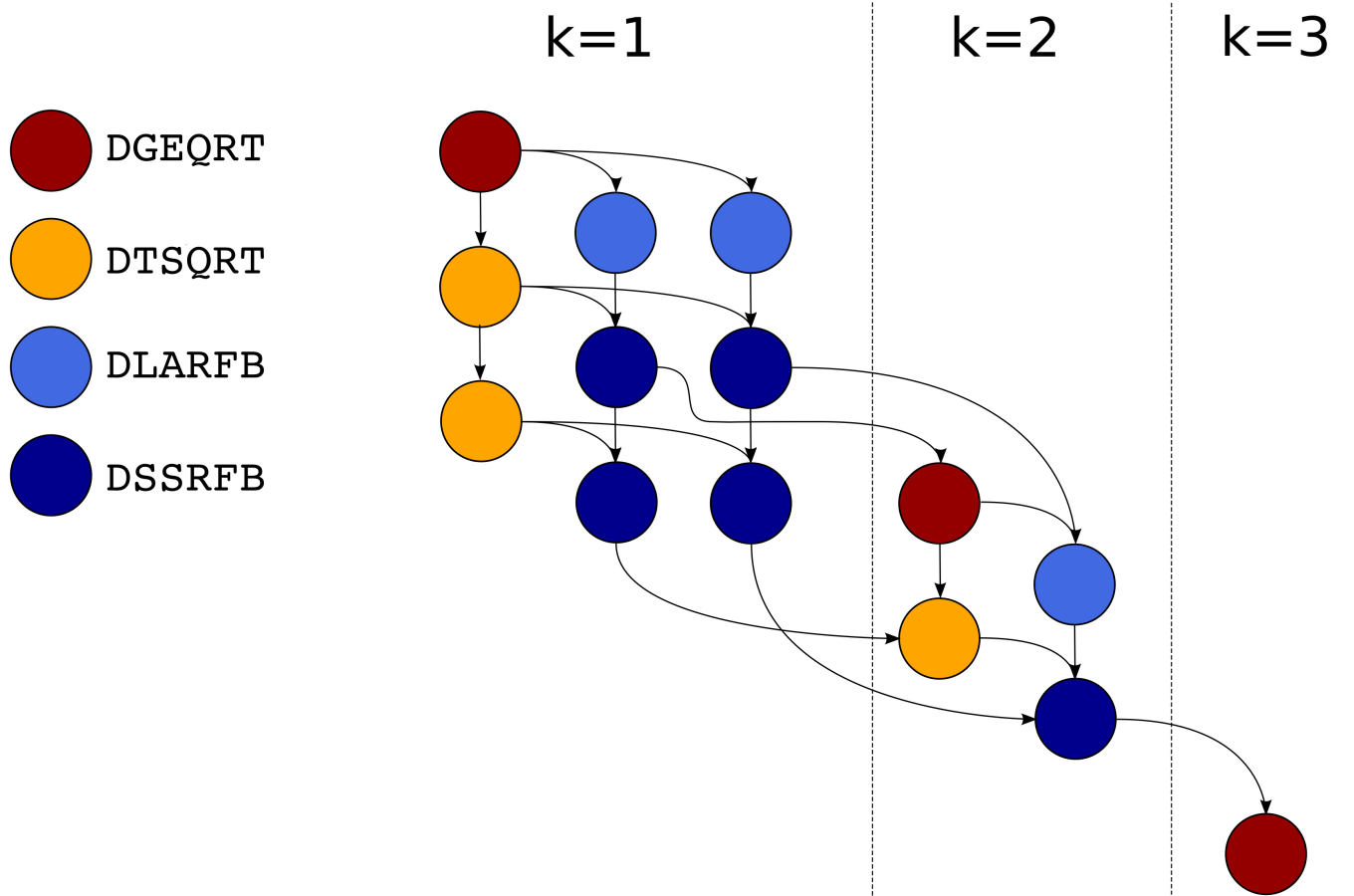


Figura 2: The dependency graph of Algorithm(1) on a matrix with  $p = q = 3$ .

La figura 2 muestra el DAG del algoritmo (1), que una vez conocido, las tareas se pueden distribuir de forma asíncrona mientras las dependencias entre ellas no son violadas. Una parte crítica es identificar en el DAG que nodos tienen mayor número de aristas, dado que en base a esa observación se le puede asignar mayor prioridad a los nodos que calculan bloques críticos.



En este caso los bloques que están calculando la factorización QR con la rutina DGEQRT tiene la prioridad más alta y el resto de tareas DLARFB, DTSQRT, DSSRFB se le puede asignar una prioridad descendente.

## 4. Implementación

A la hora de realizar la implementación el problema fue encontrar las rutinas del LAPACK para poder utilizarlas. El nombre de estas rutinas se habían cambiado y tras hablar con Alfredo Buttari<sup>1</sup>, descubrí que las rutinas a utilizar son las siguientes:

- DGEQRT
- DGEMQRT corresponde a DLARFB
- DTPQRT corresponde a DTSQRT
- DTPMQRT corresponde a DSSRFB

También cabe destacar que estas funciones requieren que la librería de LAPACK este en una versión más recientes(3.5.0). Para la instalación se puede utilizar el siguiente script:

Código:

---

```
1  %\begin{lstlisting}[basicstyle=\ttfamily]
2  #!/bin/bash
3
4  sudo apt-get update
5  sudo apt-get install make
6  sudo apt-get install gcc
7  sudo apt-get install liblapack-dev
8  sudo apt-get install liblapack3
9  sudo apt-get install libopenblas-base
10 sudo apt-get install libopenblas-dev
```

---

---

<sup>1</sup>Uno de los autores de los artículos

Una vez instalado y comprobado que se encuentran las rutinas, se tuvo que preparar la matriz de entrada para que se pueda procesar por bloques. El código correspondiente sería el siguiente:

---

```

1 void convtile(double *A, double *B, int n, int bs, int tofrom)
2 {
3     int i, j, ii, jj, i2, j2, nb=n/bs;
4
5     for (i=0; i<nb; i++) {
6         for (j=0; j<nb; j++) {
7             for (jj=0; jj<bs; jj++) {
8                 if (tofrom)
9                     memcpy(B+i*bs+(j*bs+jj)*n, A+(i+j*nb)*bs*bs+jj*bs, bs*sizeof(double));
10                else
11                    memcpy(B+(i+j*nb)*bs*bs+jj*bs, A+i*bs+(j*bs+jj)*n, bs*sizeof(double));
12            }
13        }
14    }
15 }

```

---

El código del algoritmo desafortunadamente aún no se ha acabado de programar, dados ciertos problemas con la ejecución de algunas de las rutinas.

---

```

1 void QR_LAPACK_Tile(double *A, int lA, int bs, int *info)
2 {
3     int i=0, j=0, k=0, nb=lA/bs, lda=bs, m=bs, n=bs, bs2=bs*bs;
4     int ldt=bs;
5     double * T = (double *) calloc(ldt * bs, sizeof( double ));
6     double *work = (double *) calloc(n*bs, sizeof( double ));
7     int K=m, ldc = lda;
8
9     for (k=0; k<nb; k++) { //min(p, q)
10        //DGEQRT(Akk, Vkk, Rkk, Tkk)
11        dgeqrt_(&m, &n, &bs, A+(k+k*nb)*bs2, &lda, T, &ldt, work, info);
12        if (*info != 0) return;
13        for (j=k+1; j<nb; j++){ //q
14            //DLARFB(Akj, Vkk, Tkk, Rkj)
15            dgemqrt_("L", "T", &m, &n, &K, &bs, A+(k+k*nb)*bs2, &lda, T, &ldt, A+(j+k*nb)*bs2, &ldc, work, &info);
16            if (*info != 0) return;
17        }
18        for (i=k+1; i<nb; i++){ //p
19            DTSQRT(Rkk, Aik, Vik, Tik)
20            dtpqrt_(&m, &n, &n, &bs, A, dla, B, ldb, T, ldt, work, info);
21
22            for (j=k+1; j<nb; j++){ //q
23                DSSRFB(Rkj, Aij, Vik, Tik)
24                dtpmqrt_("L", "N", &m, &n, &k, &l, &nb, v, &ldv, t, &ldt, A, &lda, b, &ldb, work, info);
25            }
26        }
27    }
28 }

```

---

Para la comprobación de los resultados obtenidos se pensaba utilizar la rutina del LAPACK y comparar los resultados.

---

```
1 void QR_LAPACK(double *A, int lA){
2     int n = lA;
3     int m = n;
4
5     int info = 0;
6     int k = n;          /* k = min(m,n);      */
7     int lda = m;        /* lda = max(m,1);   */
8     int lwork = n;      /* lwork = max(n,1); */
9     int max = lwork;    /* max = max(lwork,1); */
10
11     double *work;
12     double *tau;
13     double *vec;
14     work = (double *) calloc(max, sizeof( double ) );
15     tau  = (double *) calloc( k, sizeof( double ) );
16     vec  = (double *) calloc( m, sizeof( double ) );
17
18     dgeqrf_(&m, &n, A, &lda, tau, work, &lwork, &info);
19
20     free(work);
21     free(tau);
22     free(vec);
23 }
```

---

## 5. Conclusión y Trabajos futuros

El algoritmo de la factorización  $QR$  se puede implementar de manera concurrente por tiles y con unas mejoras considerables como se puede observar en los artículos Working Notes 190 y 191. El problema no es trivial y para obtener unos resultados se tiene que emplear mucho más tiempo del que se le ha podido dedicar.

Como trabajo futuro se debería de acabar la implementación del algoritmo secuencial para posteriormente utilizar OpenMP 4.4 o Quarc para paralelizar las tareas.

Una vez funcionando se deberían comparar las prestaciones que se han obtenido y compararlas con las de la factorización  $QR$  que ofrece LAPACK.

## Material Complementario

**GitHub:** Todo el código implementado está disponible en: <https://github.com/MihaiLupoiu/AMPI>.

## Referencias

Gene H. Golub, Charles F. Van Loan (1996). Matrix Computations Third Edition. *Chapter 8*.

Lapack Working Notes 191: <http://www.netlib.org/lapack/lawnspdf/lawn190.pdf>

Lapack Working Notes 191: <http://www.netlib.org/lapack/lawnspdf/lawn191.pdf>