

APPENDIX A

ARTIFACT DESCRIPTION APPENDIX: PRELIMINARY PERFORMANCE EVALUATION OF THE SHIFTED CHOLESKY QR ALGORITHM FOR ILL-CONDITIONED MATRICES

A. Abstract

Here, we give the artifact associated with our poster.

B. Description

1) *Check-list (artifact meta information): Fill in whatever is applicable with some informal keywords and remove the rest*

- **Algorithm:** Cholesky QR algorithm, Householder QR algorithm, Gram-Schmidt algorithm
- **Program:** Fortran90, using double precision
- **Compilation:** Intel Fortran Compiler (ver. 17.0.6), options: -O3 -mcmodel=medium -shared-intel -qopenmp -ipo -xHost, Intel MKL (ver. 2017.0.6) option: -mkl=parallel
- **Transformations:**
- **Binary:** Fortran executable
- **Data set:** Randomly generated matrix that has designated condition number
- **Run-time environment:** Intel MKL,
- **Hardware:** Two 18-core Intel Xeon E5-2695v4 (2.1GHz) processors, 128GB memory
- **Run-time state:** Number of threads: 36 (one thread per core)
- **Execution:**
- **Output:** Execution time, loss of orthogonality, and residual
- **Experiment workflow:** Generate a test matrix, compute its QR factorization by each routine and measure the execution time by `omp_get_wtime()`, and then check the accuracy (loss of orthogonality and residual)
- **Experiment customization:** Using the optimal size of working array required in LAPACK routines, which is obtained by the query for each routine.
- **Publicly available?:** No

2) *Software dependencies:*

- LAPACK: `dgeqrf`, `dorgqr`, `dgeqr`, `dgemqr`, `dpotrf`, `dgesvd` (for checking the actual condition number of generated matrices)
- BLAS: `dgemm`, `dsyrk`, `dtrsm`, `dtrmm`, `dgemv`, `dger`
- Other: `random_number`

C. Notes

We give our program code: Cholesky QR, shifted Cholesky QR, shifted CholeskyQR3, test matrix generation routine, and accuracy check routines.

Listing 1. Cholesky QR

```

1  subroutine chol_qr(m, n, A, lda, R, ldr, use_gemm,
   info)
2  implicit none
3  integer, intent(in) :: m, n, lda, ldr, use_gemm
4  integer, intent(out) :: info
5  real(8), intent(inout) :: A(lda,*)
6  real(8), intent(out) :: R(ldr,*)
7
8  !-- Gram matrix
9  if(use_gemm == 1) then
10   call dgemm('T', 'N', n, n, m, 1.0d0, A, lda, A, lda
   , 0.0d0, R, ldr)
11 else
12   call dsyrk('U', 'T', n, m, 1.0d0, A, lda, 0.0d0, R,
   ldr)
13 end if
14
15 !-- Cholesky factorization of the Gram matrix
16 call dpotrf('U', n, R, ldr, info)
17 if(info /= 0) then

```

```

18   return
19 end if
20
21 !-- Q factor
22 call dtrsm('R', 'U', 'N', 'N', m, n, 1.0d0, R, ldr,
   A, lda)
23
24 return
25 end subroutine

```

Listing 2. Shifted Cholesky QR

```

1  subroutine shifted_chol_qr(m, n, A, lda, R, ldr,
   use_gemm, info)
2  implicit none
3  integer, intent(in) :: m, n, lda, ldr, use_gemm
4  integer, intent(out) :: info
5  real(8), intent(inout) :: A(lda,*)
6  real(8), intent(out) :: R(ldr,*)
7  real(8) :: sigma
8  integer :: i
9
10 !-- Gram matrix
11 if(use_gemm == 1) then
12   call dgemm('T', 'N', n, n, m, 1.0d0, A, lda, A, lda
   , 0.0d0, R, ldr)
13 else
14   call dsyrk('U', 'T', n, m, 1.0d0, A, lda, 0.0d0, R,
   ldr)
15 end if
16
17 !-- shift
18 sigma = 0.0d0
19 do i = 1, n
20   sigma = sigma + R(i,i)
21 end do
22 sigma = sqrt(dble(m))*1.0d-16*sigma
23 do i = 1, n
24   R(i,i) = R(i,i) + sigma
25 end do
26
27 !-- Cholesky factorization of the gram matrix
28 call dpotrf('U', n, R, ldr, info)
29 if(info /= 0) then
30   return
31 end if
32
33 !-- Q factor
34 call dtrsm('R', 'U', 'N', 'N', m, n, 1.0d0, R, ldr,
   A, lda)
35
36 return
37 end subroutine

```

Listing 3. Shifted CholeskyQR3

```

1  subroutine shifted_chol_qr3(m, n, A, lda, R, ldr,
   use_gemm, W, lwork, info)
2  implicit none
3  integer, intent(in) :: m, n, lda, ldr, use_gemm,
   lwork
4  integer, intent(out) :: info
5  real(8), intent(inout) :: A(lda,*)
6  real(8), intent(out) :: R(ldr,*), W(*)
7  integer :: i, j
8
9  !-- query for the size of W
10 if(lwork == -1) then
11   W(1) = dble(n*n)
12   info = 0
13   return
14 end if
15 !-- check the size of W
16 if(lwork < n*n) then
17   print*, "lwork is too small"
18   info = -1
19   return
20 end if
21
22 !-- Cholesky QR with shift
23 call chol_qr_with_shift(m, n, A, lda, R, ldr,
   use_gemm, info)
24 if(info /= 0) then
25   return
26 end if
27

```

```

28  !-- set 0 into the lower triangular part of R
29  do j = 1, n
30    do i = j+1, n
31      R(i,j) = 0.0d0
32    end do
33  end do
34
35  !-- Cholesky QR
36  call chol_qr(m, n, A, lda, W, ldr, use_gemm, info)
37  if(info /= 0) then
38    return
39  end if
40
41  !-- R := W * R
42  call dtrmm('L', 'U', 'N', 'N', n, n, 1.0d0, W, n, R,
43    ldr)
44
45  !-- Cholesky QR
46  call chol_qr(m, n, A, lda, W, ldr, use_gemm, info)
47  if(info /= 0) then
48    return
49  end if
50
51  !-- R := W * R
52  call dtrmm('L', 'U', 'N', 'N', n, n, 1.0d0, W, n, R,
53    ldr)
54
55  info = 0
56  return
57 end subroutine

```

Listing 4. Generation of a test matrix

```

1  subroutine init_rand_mat_with_required_cond(m, n, A,
2    lda, sigma, seed)
3    implicit none
4    integer, intent(in) :: m, n, lda, seed
5    real(8), intent(in) :: sigma
6    real(8), intent(out) :: A(lda,*)
7    integer :: i, j
8    real(8) :: tmp, span, r
9    real(8) :: d(n)
10   real(8) :: X(m,n), Y(n,n)
11
12   call init_random_orth_mat(m, n, X(1,1), m, seed+11)
13   call init_random_orth_mat(n, n, Y(1,1), n, seed+11)
14   do i = 1, n
15     d(i) = sigma*(dble(i-1)/dble(n-1))
16   end do
17   do j = 1, n
18     do i = 1, m
19       X(i,j) = X(i,j) * d(j)
20     end do
21   end do
22   call dgemm('N', 'N', m, n, n, 1.0d0, X(1,1), m, Y
23     (1,1), n, 0.0d0, A(1,1), lda)
24
25   return
26 end subroutine
27
28 subroutine init_random_orth_mat(m, n, A, lda, seed)
29   implicit none
30   integer, intent(in) :: m, n, lda, seed
31   real(8), intent(out) :: A(lda,*)
32   real(8), allocatable :: W(:)
33   real(8) :: tau(n), r(n), tmp
34   integer :: lwork, info
35   integer :: i, j
36   !-- generate a random matrix
37   call init_random_matrix(m, n, A(1,1), lda, seed)
38
39   !--- compute QR factorization of A
40   !----- query the required size of the working array
41   call dgeqrf(m, n, A(1,1), lda, tau(1), tmp, -1, info
42     )
43   lwork = int(tmp)
44   allocate(W(lwork))
45   !----- QR
46   call dgeqrf(m, n, A(1,1), lda, tau(1), W(1), lwork,
47     info)
48   !----- copy Rii
49   do i = 1, n
50     r(i) = A(i,i)
51   end do
52   !----- generate Q

```

```

49   call dorgqr(m, n, n, A(1,1), lda, tau(1), W(1),
50     lwork, info)
51   !--- Q = Q D, D:=diag(sign(r_ii),...)
52   do j = 1, n
53     if(r(j) < 0.0d0) then
54       do i = 1, m
55         A(i,j) = -A(i,j)
56       end do
57     end if
58   end do
59   deallocate(W)
60   return
61 end subroutine
62
63 subroutine init_random_matrix(m, n, A, lda, seed)
64   implicit none
65   integer, intent(in) :: m, n, lda, seed
66   real(8), intent(out) :: A(lda,*)
67   integer :: i, j
68   real(8) :: val
69
70   call set_random_seed(seed)
71   do j = 1, n
72     do i = 1, m
73       call random_number(val)
74       A(i,j) = val - 0.5d0
75     end do
76   end do
77   return
78 end subroutine

```

Listing 5. Check of the accuracy of computed results

```

1  subroutine check_ortho(m, n, Q, ldq, ortho)
2    implicit none
3    integer, intent(in) :: m, n, ldq
4    real(8), intent(in) :: Q(ldq,*)
5    real(8), intent(out) :: ortho
6    real(8) :: W(n,n)
7    integer :: i, j
8
9    call dgemm('T', 'N', n, n, m, 1.0d0, Q(1,1), ldq, Q
10      (1,1), ldq, 0.0d0, W(1,1), n)
11
12    do i = 1, n
13      W(i,i) = W(i,i) - 1.0d0
14    end do
15
16    ortho = 0.0d0
17    do j = 1, n
18      do i = 1, n
19        ortho = ortho + (W(i,j)**2)
20      end do
21    end do
22
23    ortho = sqrt(ortho) / sqrt(dble(n))
24    return
25 end subroutine
26
27 subroutine check_resid(m, n, Q, ldq, R, ldr, A, lda,
28   resid)
29   implicit none
30   integer, intent(in) :: m, n, ldq, ldr, lda
31   real(8), intent(in) :: A(lda,*)
32   real(8), intent(inout) :: Q(ldq,*)
33   real(8), intent(out) :: resid
34   real(8) :: a_norm
35   integer :: i, j
36
37   call dtrmm('R', 'U', 'N', 'N', m, n, 1.0d0, R(1,1),
38     ldr, Q(1,1), ldq)
39
40   resid = 0.0d0
41   a_norm = 0.0d0
42
43   do j = 1, n
44     do i = 1, m
45       resid = resid + ((A(i,j) - Q(i,j))**2)
46     end do
47   end do
48
49   resid = sqrt(resid) / sqrt(a_norm)
50   return
51 end subroutine

```