Algoritmo Para Múltiplas Cadeias Paralelas

```
program prog
use ManagerMod
use RandomGenaratorMod
implicit none
include 'mpif.h'

!Parallel variables.
integer, parameter :: master = 0
integer :: np, myproc, ierr, stat(MPI_STATUS_SIZE), thin
real :: piMeanGlobal, meanVarGlobal, meanVareGlobal, finishGlobal
real, dimension(:), allocatable :: meanbGlobal
integer :: timeArray1Global(3), timeArray2Global(3)

!Local variables.
real :: rhs, invLhs, mean, xpx, v0, v1, logPi, logPiComp, logDelta0, logDelta1
real :: probDelta1, u, lhs, add, aa, bb, start, finish, df, vara, scalec
real :: mean2pq, piMean, meanVar, varEffects, meanVare, vare, nua, pi
real, dimension(:), allocatable :: mean_b
integer :: j, loop, countLoci, numiter(2)
integer :: timeArray1(3), timeArray2(3) !Holds the hour, minute, and second.

!Variables to reset the random number generator.
integer :: clock, k
integer, dimension(:), allocatable :: seed

!>>>>>>>>>>>>>>>>> Parallel initiations <<<<<<<<<<<<<<<
call mpi_init(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myproc, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)
numiter(1) = 10000
numiter(2) = 129000
thin = 150

do j=0,np-1
        if (myproc==j) then
                call random_seed(size=k) !How big is the intrinsic seed?
                allocate(seed(k))
                call system_clock(count=clock)
                seed = clock + 37 * (/ (i - 1, i = 1, k) /) + myproc
                call random_seed(put=seed) !Assign the seed
                deallocate(seed)
        end if
end do

!>>>>>>>>>>>>>>>>> Input data <<<<<<<<<<<<<<<
!Reading datafile.
call readTrainData(nrecords)
```

```
allocate(x(nrecords,nmarkers+1))
allocate(ycorr(nrecords))
allocate(b(nmarkers + 1))
allocate(meanb(nmarkers + 1))
allocate(mean_b(nmarkers + 1))
allocate(var(nmarkers))
allocate(ppa(nmarkers))
allocate(newMeanb(nmarkers + 1))

!Manipulating data.
call dataTrainManip(x)

!Initial values.
call dataTrainManip(x)
call initialVal(b, meanb, var, ppa, ycorr)
piMean = 0.0
meanVar = 0.0
meanVare = 0.0
mean_b = 0.0
vara = 200
pi = 0.5 !Probability that SNP k has zero effect.
mean2pq = 0.5
nua = 4.2
logPiComp = log(1-pi)
varEffects = vara/(nmarkers*(1-pi)*mean2pq)
scalec = varEffects*(nua-2)/nua

!>>>>>>>>>>>>>>>>>>>> MCMC Sampling <<<<<<<<<<<<<<<<<<<<
df = nrecords + 3 !df is degrees of freedom.
open(1,file = 'output.out')

do loop = 1,2 !First loop is the burning

        call itime(timeArray1)
        call cpu_time(start)

        do i = 1,numiter(loop)
                !Sample vare.
                vare = dot_product(ycorr,ycorr)/random_chisq(df)

                !Sample intercept.
                ycorr = ycorr + x(:,1)*b(1)
                rhs = sum(ycorr)/vare
                invLhs = 1.0/(nrecords/vare)
                mean = rhs*invLhs
                b(1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,1)*b(1)
                meanb(1) = meanb(1) + b(1)

                !Sample delta and effect for each locus
```

```
do j = 1,nmarkers
        ycorr = ycorr + x(:,j + 1)*b(j + 1)
        rhs = dot_product(x(:,j + 1),ycorr)
        xpx = dot_product(x(:,j + 1),x(:,j + 1))
        v0 = xpx*vare
        v1 = (xpx**2*varEffects + xpx*vare)
        logDelta0 = -0.5*(log(v0) + rhs**2/v0) + logPi
        logDelta1 = -0.5*(log(v1) + rhs**2/v1) + logPiComp
        probDelta1 = 1.0/(1.0 + exp(logDelta0-logDelta1))

        call random_number(u)
        if (u < probDelta1) then
                lhs = xpx/vare + 1.0/varEffects
                invLhs = 1.0/lhs
                mean = invLhs*rhs/vare
                b(j + 1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,j + 1)*b(j + 1)
                meanb(j + 1) = meanb(j + 1) + b(j + 1)
                ppa(j) = ppa(j) + 1
                var(j) = varEffects
        else
                b(j + 1) = 0.0
                var(j) = 0.0
        end if
end do

!Sample common variance.
countLoci = 0
add = 0.0
do j = 1,nmarkers !k represents the locus.
        if (var(j) > 0.0) then
                countLoci = countLoci + 1
                add = add + b(j + 1)**2
        end if
end do
varEffects = (scalec*nua + add)/random_chisq(nua + countLoci)

!Sample pi.
aa = nmarkers - countLoci + 1
bb = countLoci + 1
pi = random_beta(aa,bb)
scalec = ((nua - 2)/nua)*(vara/((1-pi)*nmarkers*mean2pq))
logPi = log(pi)
logPiComp = log(1-pi)

!Thinning.
if (mod(i,thin) == 0.0) then
        meanVar = meanVar + varEffects
        meanVare = meanVare + vare
        mean_b = mean_b + meanb
```

```
                piMean = piMean + pi
            end if
        end do

        !Means.
        ppa = ppa/numiter(loop)
        piMean = piMean/(numiter(loop)/thin)
        mean_b = mean_b/(numiter(loop)/thin)
        meanVar = meanVar/(numiter(loop)/thin)
        meanVare = meanVare/(numiter(loop)/thin)

        call MPI_Reduce (mean_b, meanbGlobal, 10001, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (piMean, piMeanGlobal, 1, MPI_REAL, MPI_SUM, &
        master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (meanVar, meanVarGlobal, 1, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (meanVare, meanVareGlobal, 1, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)

        !Getting the current time for each processor.
        call itime(timeArray2)
        call cpu_time(finish)
        call MPI_Reduce(timeArray1, timeArray1Global, 3, MPI_INTEGER, &
        MPI_SUM, master, MPI_COMM_WORLD)
        call MPI_Reduce(timeArray2, timeArray2Global, 3, MPI_INTEGER, &
        MPI_SUM, master, MPI_COMM_WORLD)
        call MPI_Reduce(finish, finishGlobal, 1, MPI_REAL, MPI_SUM, &
        master, MPI_COMM_WORLD)

        if (myproc == master) then
            write (1,'("piMean: ",f20.10)')piMeanGlobal/np
            write (1,'("meanVar: ",f20.10)')meanVarGlobal/np
            write (1,'("meanVare: ",f20.10)')meanVareGlobal/np

            !>>>>>>>>>>>>>>> Cross Validation <<<<<<<<<<<<<
            !Reading datafile.
            call readTestData(n)

            !Determining GEBV and accuracy using test data.
            allocate(z(n,nmarkers+1))
            call dataTestManip(z)
            allocate (ahat_test(n))
            ahat_test = matmul(z,meanbGlobal/np)
            call pearsonCorr(r_test, ahat_test)
            write (1,'("r_test: ",f20.10)')r_test
            deallocate(ahat_test, z)

            !Final global time.
```

```
            write (1,'("Start: ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
            (i2,1x),"seconds.")') timeArray1Global/np
            write (1,'("End : ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
            (i2,1x),"seconds.")') timeArray2Global/np
            write (1,'("Time to run the MCMC: ",f10.3," seconds.")') &
            finishGlobal/np - start
            write (1,'("")')
            write (1,'("")')
        end if

        !inital values to be used after burning.
        meanb = 0.0
        piMean = 0.0
        meanVar = 0.0
        meanVare = 0.0
        mean_b = 0.0
        ppa = 0.0
        var = 0.0

end do
close(1)
deallocate(x, meanb, ycorr, b, var, ppa, meanbGlobal, mean_b)

call mpi_finalize (ierr)
stop
end program prog
```

Algoritmo Para Cadeias Paralelizadas

```
program prog
use ManagerMod
use RandomGenaratorMod
implicit none
include 'mpif.h'

!Parallel variables.
integer, parameter :: master = 0
integer :: np, myproc, ierr, stat(MPI_STATUS_SIZE), w(nmarkers)
real, dimension (nmarkers) :: XPX, V0, V1

!Local variables.
real :: rhs, invLhs, mean, logPi, logPiComp, logDelta0, logDelta1, probDelta1
real :: lhs, add, aa, bb, start, finish, df, meanVare, vara, scalec, vare, nua
real :: mean2pq, piMean, meanVar, varEffects, pi
real, allocatable :: newMeanb(:), mean_b(:)
integer :: nloci, j, loop, countLoci,thin,timeArray1(3), timeArray2(3),numiter(2)

!Variables to reset the random number generator.
integer, dimension (12) :: old, seeds
integer :: k
seeds(1) = 12345

call mpi_init(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myproc, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)
numiter(1) = 10000
numiter(2) = 1290000
thin = 150

!Reading datafile.
call readTrainData(nrecords)
allocate(x(nrecords,nmarkers+1))
allocate(ycorr(nrecords))
allocate(b(nmarkers + 1))
allocate(meanb(nmarkers + 1))
allocate(var(nmarkers))
allocate(ppa(nmarkers))
allocate(newMeanb(nmarkers + 1))

!Initial values.
call dataTrainManip(x)
call initialVal(b, meanb, var, ppa, ycorr)
piMean = 0.0
meanVar = 0.0
meanVare = 0.0
mean_b = 0.0
```

```
vara = 200
pi = 0.5 !Probability that SNP k has zero effect.
mean2pq = 0.5
nua = 4.2
logPiComp = log(1-pi)
varEffects = vara/(nmarkers*(1-pi)*mean2pq)
scalec = varEffects*(nua-2)/nua

!************************ Master task only ************************
if (myproc == master) then
        call RANDOM_SEED()
        call RANDOM_SEED(SIZE=K)
        call RANDOM_SEED(PUT=SEEDS(1:K))
        do j = 1, nmarkers
                XPX(j) = dot_product(x(:,j + 1),x(:,j +1))
        end do
        do loop = 1,2 !First loop is the burning
                call itime(timeArray1)
                call cpu_time(start)
                do i = 1,numiter(loop)
                        call MPI_Recv(vare, 1, MPI_REAL, 2, 200, &
                        MPI_COMM_WORLD, stat, ierr)
                        call MPI_Recv(ycorr, nrecords, MPI_REAL, 2, 201, &
                        MPI_COMM_WORLD,stat, ierr)
                        call MPI_Recv(V0, nmarkers, MPI_REAL, 2, 202, &
                        MPI_COMM_WORLD, stat, ierr)
                        call MPI_Recv(V1, nmarkers, MPI_REAL, 2, 203, &
                        MPI_COMM_WORLD, stat, ierr)

                        !Thinning to vare.
                        if (mod(i,thin) == 0.0) then
                                meanVare = meanVare + vare
                        end if

                        !Sample delta and effect for each locus
                        call MPI_Recv(w, nmarkers, MPI_REAL, 1, 100, &
                        MPI_COMM_WORLD, stat, ierr)
                        do j = 1,nmarkers !j represents the locus.
                                ycorr = ycorr + x(:,j + 1)*b(j + 1)
                                rhs = dot_product(x(:,j + 1),ycorr)
                                logDelta0 = -0.5*(log(V0(j)) + rhs**2/V0(j)) + logPi
                                logDelta1 = -0.5*(log(V1(j)) + rhs**2/V1(j)) + &
                                logPiComp
                                probDelta1 = 1.0/(1.0 + exp(logDelta0-logDelta1))
                                if (w(j) < probDelta1) then
                                        lhs = XPX(j)/vare + 1.0/varEffects
                                        invLhs = 1.0/lhs
                                        mean = invLhs*rhs/vare
                                        b(j + 1) = random_normal (mean,sqrt(invLhs))
                                        ycorr = ycorr - x(:,j + 1)*b(j + 1)
```

```
                    var(j) = varEffects
                else
                    b(j + 1) = 0.0
                    var(j) = 0.0
                end if
            end do
            call MPI_SEND(ycorr, nrecords, MPI_REAL, 2, 204, &
            MPI_COMM_WORLD, ierr)
            call MPI_SEND(b(2:nmarkers + 1), nmarkers, &
            MPI_REAL, 3, 300, MPI_COMM_WORLD, ierr)

            !Sample common variance.
            countLoci = 0
            add = 0.0
            do j = 1,nmarkers !j represents the locus.
                if (var(j) > 0.0) then
                    countLoci = countLoci + 1
                    add = add + b(j + 1)**2
                end if
            end do
            varEffects = (scalec*nua + add)/random_chisq(nua + &
            countLoci)
            call MPI_SEND(varEffects, 1, MPI_REAL, 2, 205, &
            MPI_COMM_WORLD, ierr)
            call MPI_SEND(varEffects, 1, MPI_REAL, 4, 400, &
            MPI_COMM_WORLD, ierr)

            !Sample Pi
            aa = nmarkers - countLoci + 1
            bb = countLoci + 1
            pi = random_beta(aa,bb)
            scalec=((nua - 2)/nua)*(vara/((1-pi)*nmarkers*mean2pq))
            logPi = log(pi)
            logPiComp = log(1-pi)
            call MPI_SEND(pi, 1, MPI_REAL, 4, 402, &
            MPI_COMM_WORLD, ierr)
        end do
        call MPI_Recv(mean_b(1), 1, MPI_REAL, 2, 206, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(mean_b(2:nmarkers + 1), nmarkers, & MPI_REAL,
        3, 301, MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(ppa, nmarkers, MPI_REAL, 3, 302, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(meanVar, 1, MPI_REAL, 4, 401, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(piMean, 1, MPI_REAL, 4, 403, &
        MPI_COMM_WORLD, stat, ierr)

        !Getting the current time.
        call itime(timeArray2)
```

```
                call cpu_time(finish)
                print '("Start: ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
                (i2,1x),"seconds.")',timeArray1
                print '("End : ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
                (i2,1x),"seconds.")',timeArray2
                print '("Time to run the MCMC: ",f10.3," seconds.")',finish - start
                write (1,'("piMean: ",f20.10)')piMean/numiter(loop)/thin
                write (1,'("meanVar: ",f20.10)')meanVar/numiter(loop)/thin
                write (1,'("meanVare: ",f20.10)')meanVare/numiter(loop)/thin

                !inital values to be used after burning
                newMeanb = mean_b !It will be used in cross validation
                mean_b = 0.0
                meanb = 0.0
                piMean = 0.0
                meanVar = 0.0
                ppa = 0.0
                var = 0.0
                call MPI_SEND(varEffects, 1, MPI_REAL, 2, 207, &
                MPI_COMM_WORLD, ierr)
        end do
        deallocate(x, ycorr, b, var, ppa)

        !>>>>>>>>>>>>>>> Cross Validation <<<<<<<<<<<<<
        !Reading datafile.
        call readTestData(n)

        !Determining GEBV using testData.
        allocate(z(n,nmarkers+1))
        call dataTestManip(z)
        allocate (ahat_test(n))
        ahat_test = matmul(z, newMeanb)
        call pearsonCorr(r_test, ahat_test)
        print*,'r = ',r_test
        deallocate(meanb, ahat_test, z, newMeanb)
end if

!*******************************Non-master tasks only*******************************
if (myproc == 1) then
        do loop = 1, 2
                do i = 1, numiter(loop)
                        call random_number(w)
                        call MPI_SEND(w, nmarkers, MPI_REAL, master, & 100,
                        MPI_COMM_WORLD, ierr)
                end do
        end do
end if

if (myproc == 2) then
        do j = 1, nmarkers
```

```
                XPX(j) = dot_product(x(:,j + 1),x(:,j +1))
        end do
        do loop = 1, 2
                do i = 1, numiter(loop)
                        !Sample vare.
                        vare = dot_product(ycorr, ycorr)/random_chisq(df)

                        !sample intercept.
                        ycorr = ycorr + x(:,1)*b(1)
                        rhs = sum(ycorr)/vare
                        invLhs = 1.0/(nrecords/vare)
                        mean = rhs*invLhs
                        b(1) = random_normal(mean,sqrt(invLhs))
                        ycorr = ycorr - x(:,1)*b(1)
                        meanb(1) = meanb(1) + b(1)

                       !Thinning to mean_b.
                        if (mod(i,thin) == 0.0) then
                                mean_b(1) = mean_b(1) + meanb(1)
                        end if

                        do j = 1, nmarkers
                                V0(j) = XPX(j)*vare
                                V1(j) = (XPX(j)**2*varEffects + XPX(j)*vare)
                        end do

                        call MPI_SEND(vare, 1, MPI_REAL, master, 200, &
                        MPI_COMM_WORLD, ierr)
                        call MPI_SEND(ycorr, nrecords, MPI_REAL, master, &
                        201, MPI_COMM_WORLD, ierr)
                        call MPI_SEND(V0, nmarkers, MPI_REAL, master, & 202,
                        MPI_COMM_WORLD, ierr)
                        call MPI_SEND(V1, nmarkers, MPI_REAL, master, & 203,
                        MPI_COMM_WORLD, ierr)
                        call MPI_Recv(ycorr, nrecords, MPI_REAL, master, & 204,
                        MPI_COMM_WORLD, stat, ierr)
                        call MPI_Recv(varEffects, 1, MPI_REAL, master, 205, &
                        MPI_COMM_WORLD, stat, ierr)
                end do
                mean_b(1) = mean_b(1)/numiter(loop)
                call MPI_SEND(mean_b(1), 1, MPI_REAL, master, 206, &
                MPI_COMM_WORLD, ierr)
                call MPI_Recv(varEffects, 1, MPI_REAL, master, 207, &
                MPI_COMM_WORLD, stat, ierr)
        end do
end if

if (myproc == 3) then
do loop = 1,2
        do i = 1, numiter(loop)
```

```
                call MPI_Recv(b(2:nmarkers + 1), nmarkers, MPI_REAL, &
                master, 300, MPI_COMM_WORLD, stat, ierr)
                do j = 1, nmarkers !k represents the locus.
                        if (b(j + 1) /= 0.0) then
                                meanb(j + 1) = meanb(j + 1) + b(j + 1)
                                ppa(j) = ppa(j) + 1
                        end if
                end do

                !Thinning to mean_b.
                if (mod(i,thin) == 0.0) then
                        mean_b(2:nmarkers + 1) = mean_b(2:nmarkers + 1) + &
                        meanb(2:nmarkers + 1)
                end if
        end do
        meanb(2:nmarkers + 1) = meanb(2:nmarkers + 1)/numiter(loop)
        call MPI_SEND(mean_b(2:nmarkers + 1), nmarkers, MPI_REAL, &
        master, 301, MPI_COMM_WORLD, ierr)
        call MPI_SEND(ppa, nmarkers, MPI_REAL, master, 302, &
        MPI_COMM_WORLD, ierr)
end do
end if

if (myproc == 4) then
do loop = 1,2
        do i = 1, numiter(loop)
                call MPI_Recv(varEffects, 1, MPI_REAL, master, 400, &
                MPI_COMM_WORLD, stat, ierr)
                call MPI_Recv(pi, 1, MPI_REAL, master, 402, &
                MPI_COMM_WORLD, stat, ierr)

                !Thinning to meanVar and piMean.
                if (mod(i,thin) == 0.0) then
                        meanVar = meanVar + varEffects
                        piMean = piMean + pi
                end if
        end do
        call MPI_SEND(meanVar, 1, MPI_REAL, master, 401, &
        MPI_COMM_WORLD, ierr)
        call MPI_SEND(piMean, 1, MPI_REAL, master, 403, &
        MPI_COMM_WORLD, ierr)
        meanVar = 0.0
        piMean = 0.0
end do
end if
call mpi_finalize (ierr)
stop
end program prog
```