```
/*************************************************************************/
/*    parallel 3D convolution                                            */
/* out(i,j,k)=\sum_{p=0,q=0,r=0}^{Nx,Ny,Nz}DemagTensor(i-p,j-q,k-r)aM(p,q,r) */
/*************************************************************************/
PetscScalar *a,*K; //arrays presenting DemagTensor and M in real space,with zero-padding
Vec         x, w;  //vectors presenting DemagTensor and M in real space,with zero-padding
Vec         y, z;   //vectors presenting DemagTensor and M in fourier space
Vec         hfft;         //resultant vector
Vec         inpx, inpw, out;  //real data
Mat         A;
PetscInt    DIM=3, N, *dim, index, idx1, idx2, idx3, idx1_wrap, idx2_wrap, idx3_wrap;
PetscInt    Nx=26, Ny=26, Nz=3;  //dimensions of real data

PetscMalloc(DIM*sizeof(PetscInt), &dim);
dim[0]=64;  //expanded to be the power of 2 for circular convolution
dim[1]=64;
dim[2]=8;
N=dim[0]*dim[1]*dim[2];  //total number of variables in 3D, with zero-padding

//preallocate
PetscMalloc(N*sizeof(PetscScalar), &a);
PetscMalloc(N*sizeof(PetscScalar), &K);

/* define the vector */
ierr = VecCreate(PETSC_COMM_WORLD,&inpx);      CHKERRQ(ierr);
ierr = VecSetSizes(inpx, PETSC_DECIDE, N);     CHKERRQ(ierr);
ierr = VecSetFromOptions(inpx);                CHKERRQ(ierr);
ierr = VecDuplicate(inpx,&inpw);              CHKERRQ(ierr);
ierr = VecDuplicate(inpx,&out);               CHKERRQ(ierr);

/* create FFTW object */
ierr = MatCreateFFT(PETSC_COMM_WORLD,DIM,dim,MATFFTW,&A);     CHKERRQ(ierr);

/* create FFTW vectors that are compatible with parallel layout of A */
ierr = MatGetVecsFFTW(A, &x, &y, &hfft);       CHKERRQ(ierr);
ierr = MatGetVecsFFTW(A, &w, &z, &hfft);       CHKERRQ(ierr);

// initilization
for(int i=0; i<N; i++)
{
       a[i]=0.0;
       K[i]=0.0;
}

//define a with zero padding
for(int p=0; p<Nx; p++)
{
       for(int q=0; q<Ny; q++)
       {
              for(int r=0; r<Nz; r++)
              {
                     node2=r*(Nx*Ny)+q*Nx+p;
                     index=r*(dim[0]*dim[1])+q*dim[0]+p;
                     a[index]=aM[node2];
              }
       }
}
for(int i=0; i<N; i++)
```

```
{
       ierr = VecSetValue(inpx, i, a[i], INSERT_VALUES);   CHKERRQ(ierr);
}

ierr = VecAssemblyBegin(inpx);   CHKERRQ(ierr);
ierr = VecAssemblyEnd(inpx);     CHKERRQ(ierr);

ierr = VecScatterPetscToFFTW(A, inpx, x);      CHKERRQ(ierr);

//define K with zero padding
for(int i=0; i<Nx; i++)
{
    for(int j=0; j<Ny; j++)
    {
        for(int k=0; k<Nz; k++)
        {
            node1=k*(Nx*Ny)+j*Nx+i;
            for(int p=0; p<Nx; p++)
            {
                for(int q=0; q<Ny; q++)
                {
                    for(int r=0; r<Nz; r++)
                     {
                        node2=r*(Nx*Ny)+q*Nx+p;
                        idx1=i-p+int(dim[0]/2)-1;
                        idx2=j-q+int(dim[1]/2)-1;
                        idx3=k-r+int(dim[2]/2)-1;

                        //wrap around order
                        if(idx1>=0&&idx1<=int(dim[0]/2)-2)
                        {
                           idx1_wrap=idx1+int(dim[0]/2);
                        }
                        else if(idx1>=int(dim[0]/2)-1&&idx1<=dim[0]-2)
                        {
                            idx1_wrap=idx1-int(dim[0]/2)+1;
                        }
                        else if(idx1==dim[0]-1)
                        {
                            idx1_wrap=idx1;
                        }
                        if(idx2>=0&&idx2<=int(dim[1]/2)-2)
                        {
                            idx2_wrap=idx2+int(dim[1]/2);
                        }
                        else if(idx2>=int(dim[1]/2)-1&&idx2<=dim[1]-2)
                        {
                           idx2_wrap=idx2-int(dim[1]/2)+1;
                         }
                        else if(idx2==dim[1]-1)
                       {
                           idx2_wrap=idx2;
                        }
                        if(idx3>=0&&idx3<=int(dim[2]/2)-2)
                        {
                            idx3_wrap=idx3+int(dim[2]/2);
                        }
                        else if(idx3>=int(dim[2]/2)-1&&idx3<=dim[2]-2)
```

```
                    {
                        idx3_wrap=idx3-int(dim[2]/2)+1;
                    }
                    else if(idx3==dim[2]-1)
                    {
                        idx3_wrap=idx3;
                    }
                   index=idx3_wrap*(dim[0]*dim[1])+idx2_wrap*dim[0]+idx1_wrap;
                   K[index]=DemagTensor[node1-node2];
                }
            }
        }
    }
}
for(int i=0; i<N; i++)
{
    ierr = VecSetValue(inpw, i, K[i], INSERT_VALUES);   CHKERRQ(ierr);
}
ierr = VecAssemblyBegin(inpw);   CHKERRQ(ierr);
ierr = VecAssemblyEnd(inpw);     CHKERRQ(ierr);
ierr = VecScatterPetscToFFTW(A, inpw, w);      CHKERRQ(ierr);

/* Apply FFTW forward and FFTW backward  */
ierr = MatMult(A,x,y);CHKERRQ(ierr);
ierr = MatMult(A,w,z);CHKERRQ(ierr);
ierr = VecPointwiseMult(y, y, z);     CHKERRQ(ierr);
ierr = MatMultTranspose(A,y,hfft);    CHKERRQ(ierr);

/* Output from Backward DFT */
ierr = VecScatterFFTWToPetsc(A, hfft, out);    CHKERRQ(ierr);

PetscFree(a);
PetscFree(K);
PetscFree(col2);
PetscFree(dim);

ierr = VecDestroy(&x);CHKERRQ(ierr);
ierr = VecDestroy(&w);CHKERRQ(ierr);
ierr = VecDestroy(&y);CHKERRQ(ierr);
ierr = VecDestroy(&z);CHKERRQ(ierr);
ierr = VecDestroy(&hfft);CHKERRQ(ierr);
ierr = VecDestroy(&inpx);CHKERRQ(ierr);
ierr = VecDestroy(&inpw);CHKERRQ(ierr);
ierr = VecDestroy(&out);CHKERRQ(ierr);
ierr = MatDestroy(&A);  CHKERRQ(ierr);
```