support files

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++script.txt++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# sed script
#
# 2000/7/10     H.Kobayashi
#
# A scipt making formatted input file from a user written
# free formed parameter file.
#

{
# remove comment & null lines
        /^$/d
        /^¥#/d
        s/[[:space:]]//g
        s/¥#.*$//
# if the line ends with "," it will continure to the next line
        /,$/H
        /,$/d

# if the line includes "}" then it is the end line of
# the concatination (in really, the line composed of only with numbers)
        /^[^a-zA-Z]*}/{
        x
        G
        s/¥n//g
        }

# clean up lines
        s/[[:space:]]//g
        s/{//g
        s/}//g
        s/,/ /g
        s/=/ /g
        p
}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++g3++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#!/bin/sh
# 2000/7/10     H.Kobayashi
#
```

```
#

FILTER=./script         # lint script for free formed user parameters
SOURCE=./param          # user parameters file
KEYWORDS=./keywords     # pre-defined keywords list

#
# function to test the keyword is defined in the table
#
testtoken()
{
arg=$(grep ^$1 $KEYWORDS)
if  [ -z $(echo $arg) ] || [ $arg != $1 ]
#if [ ! $arg = $1 ]
then
        echo "error: keyword ¥"$1¥" does not exist " >&2
fi
}

#
# function to insert the number of parameters after the keyword
#
maketoken()
{
echo -n $2' '$(($1-1))' '
shift 2
echo $*
}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++keywords++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

HITRAN_DIRECTORY
OUTPUT_FILE
INTERIM_FILE
TEMPERATURE
PRESSURE
MOLECULE
MIN_WAVENUMBER
MAX_WAVENUMBER
TARGET_MIN_WAVENUMBER
TARGET_MAX_WAVENUMBER
LINE_INTENCITY_CRITERIA
BAND_WIDTH
FOURIER_SIZE_INDEX
APODIZATION
BULK_H2O
BULK_CO2
BULK_O3
```

```
BULK_N2O
BULK_CO1
BULK_CH4

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++param.text++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# Wednesday, September 13, 2000
HITRAN_DIRECTORY=/lbyl/hitran/hitran96/by_molec/
OUTPUT_FILE=./spectrum
INTERIM_FILE=./alist
TEMPERATURE=270.7
PRESSURE=0.798
MAX_WAVENUMBER=1300.0
MIN_WAVENUMBER=650
TARGET_MAX_WAVENUMBER=1180.5
TARGET_MIN_WAVENUMBER=980.0
LINE_INTENCITY_CRITERIA=1.0e-22
BAND_WIDTH=2
FOURIER_SIZE_INDEX=18
MOLECULE={3}
BULK_O3={0.0010993}
APODIZATION= 2

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++parameter+++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

HITRAN_DIRECTORY 1 /lbyl/hitran/hitran96/by_molec/
OUTPUT_FILE 1 ./spectrum
INTERIM_FILE 1 ./alist
TEMPERATURE 1 270.7
PRESSURE 1 0.798
MAX_WAVENUMBER 1 1300.0
MIN_WAVENUMBER 1 650
TARGET_MAX_WAVENUMBER 1 1180.5
TARGET_MIN_WAVENUMBER 1 980.0
LINE_INTENCITY_CRITERIA 1 1.0e-22
BAND_WIDTH 1 2
FOURIER_SIZE_INDEX 1 18
MOLECULE 1 3
BULK_O3 1 0.0010993
APODIZATION 1 2


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

mail c program

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++l2ftvMain.c++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


/*****************************************************************

        2000/7/13
        2000/8/16 ver.1.0
        2000/12/14 ver. for paralell for lines dimension

        An object linked with -o option didn't work well on the
        tested linux server, however, it may relate to the
        configulation of the server file system that adopting
        a hardware disk array.

        The linking must do as "... -L /usr/local/lib -lfftw -lm"
        when fftw libralies are in /usr/local/lib.
*****************************************************************/
/*---------------------------------------------------------------*/
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "math.h"
#include "mpi.h"
#include "fftw.h"
#define HITRAN_FILE_FORM "%s%02d_hit96.par"
#define NUMBER_OF_LAYERS 40
#define MAX_NODES 16

/*===============begin of definitions==============================*/

/*----------definition of physical principal parameters---------*/
double
centimeter= 0.01,       /* default is MKS system (m)  */
gram= 0.001,            /* default is MKS system (kg) */
c_light_speed=  2.99792458e8,   /*  (m/s)   */
bolzmann_c=     1.380658e-23,   /*  (J/K)   */
avogadro_c=     6.0221367e23,   /*  (/mole) */
planck_c=       6.6260755e-34,  /*  (J*s)   */
Pi=             3.14159265359,
temperature_std= 296.0,
pressure_std=   1013.0;
/*                             dmy H2O,CO2, O3,N2O, CO,CH4    */
double molecular_weight[]=   {  0, 18, 44, 48, 44, 28, 16};
double s_coefficient_beta[]= {0.0,1.5,1.0,1.5,1.0,1.0,1.5};

/*----------definition of world environment---------------------*/
```

```
double temperature, pressure;
int mol_id[32], mol_id_index;
double
bulk_H2O[NUMBER_OF_LAYERS],
bulk_CO2[NUMBER_OF_LAYERS],
bulk_O3[NUMBER_OF_LAYERS],
bulk_N2O[NUMBER_OF_LAYERS],
bulk_CO1[NUMBER_OF_LAYERS],
bulk_CH4[NUMBER_OF_LAYERS];

/*----------definition of given parameter environment------------*/
int fourier_size_index, fourier_size;
double
nyu_max, nyu_min,               /* define calculation scope */
target_nyu_max, target_nyu_min, /* define lines screening scope */
band_width,
intencity_criterion= 3.5e-24; /* FASCODE default value for O3 */

/*-------definition of parameter environment induced from given---*/
double
compensation,
alpha_H, alpha_L, alpha_D, resolution_scale,
kScalingCoefficient,
kernel_step;
char data_file_name[256], out_file_name[256], interim_file_name[256];

/*-----definition of calculation environment---------------------*/
double node_performance[MAX_NODES];
double performance, performance_sum= 0.0;
MPI_Status *Mpi_status;
int Mpi_my_id, Mpi_n_procs, Mpi_err;
int v_level, apodization_type= 0;
typedef struct Line{
        double nyu;             /* wavenumber of line center */
        double gamma1;
        double temp_d_a_broad; /* temp. dependency for air broad */
        double intencity;
        double r_of_L_and_D;    /* ratio of Lorentz and Doppler */
        } Line;

/*------definition of strages to be allocated-------------------*/
int max_number_of_lines= 0;     /* number included in the cal. scope */
int number_of_lines=0;          /* screened lines number */
Line *line;
fftw_complex *kernel, *kernel_buf, *out;
double *tau, *xaxs, *apodize;

/*===============end of definitions==============================*/
```

```
/*-----------------------------------------------------------------
                main program
-----------------------------------------------------------------*/
int main(int argc, char *argv[]){

FILE *fp, *gp;
fpos_t *line_position, *lp;

int i, j, k, nparam;
char *ptr, parambuf[1024];
char line_buf[256], buf[20];
double wave_number= 0.;
double line_intencity;
double tr, trr, pr, d;
double log2, sqrt_log2;

double performance_meter();
int sp,ep;

if (argc <=2){
        fprintf(stderr, "Usage: l2ftvMain parameter_file_name
[verbose_level]¥n");
        exit(1);
        }
v_level= 1;     /* 0: dumm, 1: system, 2: parameters, 3: interim_result */
if (argc ==  3){
        v_level= atoi(argv[2]);
        if (v_level > 0) printf("v_level=%d¥n", v_level);
        }

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &Mpi_n_procs);
MPI_Comm_rank(MPI_COMM_WORLD, &Mpi_my_id);

/*-------------------measurement of node performance------------------*/
performance=  performance_meter();
MPI_Gather(&performance, 1, MPI_DOUBLE, node_performance,
                1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
if (Mpi_my_id == 0){
        for(i= 0; i<Mpi_n_procs; i++) node_performance[i]= 1/
node_performance[i];
        node_performance[0] *= 1.2;
        for(i= 0; i<Mpi_n_procs; i++) performance_sum +=
node_performance[i];
        for(i= 0; i<Mpi_n_procs; i++) node_performance[i] /=
performance_sum;
        if (v_level > 0)
                for(i=0; i<Mpi_n_procs; i++) printf("P(%d)
performance=%e¥n",
                                                i, node_performance[i]);
```

```
        }
MPI_Bcast(node_performance, Mpi_n_procs, MPI_DOUBLE, 0, MPI_COMM_WORLD);


/*-------------------get parameters------------------------------*/
nparam= get_param_str(argv, "MOLECULE", parambuf);
ptr= parambuf;
for (i=1; i <= nparam; i++){
        mol_id[i]= atof(ptr);
        ptr= strchr(ptr, ' ')+1;
        }
if (v_level > 1){
        printf("molecule_id=");
        for (i=1; i <= nparam; i++) printf("%d,", mol_id[i]);
        printf("¥n");
        }
mol_id_index= mol_id[1];  /* we can handle now only 1 species */

nparam= get_param_str(argv, "BULK_O3", parambuf);
ptr= parambuf;
for (i=1; i <= nparam; i++){
        bulk_O3[i]= atof(ptr);
        ptr= strchr(ptr, ' ')+1;
        }
if (v_level > 1){
        printf("bulk_O3=");
        for (i=1; i <= nparam; i++) printf("%lf,", bulk_O3[i]);
        printf("¥n");      /* we can handle now only 1 layer */
        }

nparam= get_param_str(argv, "MIN_WAVENUMBER", parambuf);
nyu_min= atof(parambuf)/ centimeter;
if (v_level > 1) printf("min_wave_number=%lf¥n", nyu_min);

nparam= get_param_str(argv, "MAX_WAVENUMBER", parambuf);
nyu_max= atof(parambuf)/ centimeter;
if (v_level >1) printf("max_wave_number=%lf¥n", nyu_max);

nparam= get_param_str(argv, "TARGET_MAX_WAVENUMBER", parambuf);
target_nyu_max= atof(parambuf)/ centimeter;
if (v_level > 1) printf("tartget_max_wave_number=%lf¥n", target_nyu_max);

nparam= get_param_str(argv, "TARGET_MIN_WAVENUMBER", parambuf);
target_nyu_min= atof(parambuf)/ centimeter;
if (v_level > 1) printf("target_min_wave_number=%lf¥n", target_nyu_min);

nparam= get_param_str(argv, "TEMPERATURE", parambuf);
temperature= atof(parambuf);
if (v_level > 1) printf("temperature=%lf¥n", temperature);
```

```
nparam= get_param_str(argv, "PRESSURE", parambuf);
pressure= atof(parambuf);
if (v_level > 1) printf("pressure=%lf¥n", pressure);

nparam= get_param_str(argv, "HITRAN_DIRECTORY", parambuf);
sprintf(data_file_name, HITRAN_FILE_FORM, parambuf, mol_id_index);

nparam= get_param_str(argv, "OUTPUT_FILE", parambuf);
sprintf(out_file_name, "%s", parambuf);
if (v_level > 1) printf("output file name=%s¥n", out_file_name);

nparam= get_param_str(argv, "INTERIM_FILE", parambuf);
sprintf(interim_file_name, "%s", parambuf);
if (v_level > 1) printf("interim file name=%s¥n", interim_file_name);

nparam= get_param_str(argv, "FOURIER_SIZE_INDEX", parambuf);
fourier_size_index= atoi(parambuf);
if (v_level > 1) printf("Fourier_size_index=%d¥n", fourier_size_index);

nparam= get_param_str(argv, "BAND_WIDTH", parambuf);
band_width= atof(parambuf);
if (v_level > 1) printf("band_width=%lf¥n", band_width);

nparam= get_param_str(argv, "APODIZATION", parambuf);
apodization_type= atoi(parambuf);
if (v_level > 1) printf("apodization_type=%d¥n", apodization_type);

nparam= get_param_str(argv, "LINE_INTENCITY_CRITERIA", parambuf);
intencity_criterion= atof(parambuf);
if (v_level > 1) printf("intencity_criterion=%e¥n", intencity_criterion);

fourier_size= (int) pow(2.0, (double)fourier_size_index);
if (v_level > 1) printf("fourier size= %d¥n", fourier_size);


/*----- count the line number of the molecule in the HITRAN file
        and obtain line data ---------------------------------*/

if (v_level > 1) printf("datafile=%s¥n", data_file_name);

if (NULL == (fp= fopen(data_file_name, "r"))){
        fprintf(stderr, "HITRAN data file open error ¥n");
        exit(1);
        }
if (v_level > 0) printf("P(%d): file opened¥n", Mpi_my_id);

while (NULL != fgets(line_buf, sizeof(line_buf), fp)){

        strncpy( buf, &line_buf[3], 12); buf[12]= '¥0';
        wave_number= atof( buf )/ centimeter;
```

```
        if (wave_number > target_nyu_max) break;

        if (target_nyu_min <= wave_number){
                max_number_of_lines++;

                strncpy( buf, &line_buf[16], 9); buf[9]= '¥0';
                line_intencity= atof(buf)* centimeter;
                if(line_intencity >= intencity_criterion* centimeter)
                        number_of_lines++;
                }
        else{
                k= fgetpos(fp, lp);
                line_position= lp;
                }
        }  /*end of while */

if (v_level > 1) printf("max lines number=%d¥n", max_number_of_lines);
if (v_level > 1) printf("target_line number=%d¥n", number_of_lines);

if (NULL == (line= (Line*)malloc(sizeof(Line)* number_of_lines ))||
NULL == (kernel= (fftw_complex*)malloc(sizeof(fftw_complex)*
fourier_size))||
NULL == (kernel_buf= (fftw_complex*)malloc(sizeof(fftw_complex)*
fourier_size))||
NULL == (out= (fftw_complex*)malloc(sizeof(fftw_complex)* fourier_size))||
NULL == (tau= (double*)malloc(sizeof(double)* fourier_size))||
NULL == (apodize= (double*)malloc(sizeof(double)* fourier_size))||
NULL == (xaxs= (double*)malloc(sizeof(double)* fourier_size))){
        fprintf(stderr, "Error: memory allocation fault¥n");
        exit(1);
        }

for(i=0; i< fourier_size; i++){
        kernel[i].re= 0.0;
        kernel[i].im= 0.0;}

/*-----------------start data reading-----------------------*/

if (0 != fsetpos(fp, line_position)){
        fprintf(stderr, "file position set error ¥n");
        exit(1);
        }

if (v_level > 0) printf ("P(%d): file positioned¥n", Mpi_my_id);

j= 0;
for (i= 0; i < max_number_of_lines; i++){
        fgets(line_buf, sizeof(line_buf), fp);

        strncpy( buf, &line_buf[16], 9); buf[9]= '¥0';
```

```
        line_intencity= atof(buf)* centimeter;
        strncpy( buf, &line_buf[3], 12);  buf[12]= '¥0';
        wave_number= atof( buf )/ centimeter;

        if (wave_number >= target_nyu_min &&
                wave_number <= target_nyu_max &&
         line_intencity >= intencity_criterion* centimeter){
                (line+ j)->intencity= line_intencity;
                (line+ j)->nyu= wave_number;
                strncpy( buf, &line_buf[35], 5); buf[5]= '¥0';
                (line+ j)->gamma1= atof(buf)/ centimeter;
                strncpy( buf, &line_buf[56], 3); buf[3]= '¥0';
                (line+ j)->temp_d_a_broad= atof(buf);/* not normalized to
MKS */
                j++;
                }
        }

fclose( fp );
if (v_level > 0) printf("P(%d): file closed¥n", Mpi_my_id);

/*-------------------preparing kernel calculation list--------------------
*/
pr= pressure/ pressure_std;
tr= temperature/ temperature_std;
trr= temperature_std/ temperature;
log2= log(2);
sqrt_log2=  sqrt(log(2));

compensation= pow(tr, s_coefficient_beta[mol_id_index]);
alpha_H= sqrt(molecular_weight[mol_id_index]* gram/ avogadro_c*
        c_light_speed* c_light_speed/ (2.0* bolzmann_c* temperature));
resolution_scale= (fourier_size+1)* band_width/ alpha_H;
kernel_step=  2.0* Pi* band_width/ alpha_H;
kScalingCoefficient=2* sqrt(Pi* resolution_scale* band_width/ alpha_H);

if (v_level > 2) printf("alpha_H= %e¥n",alpha_H);
if (v_level > 2) printf("resolution_scale= %e¥n",resolution_scale);
if (v_level > 2) printf("kernel_step= %e¥n",kernel_step);
if (v_level > 2) printf("kScalingCoefficient= %e¥n",kScalingCoefficient);
if (v_level > 2) printf("compensation= %e¥n",compensation);


for (i=0; i < number_of_lines; i++){
        alpha_L= (line+ i)->gamma1* pr*
                pow(trr, (line+ i)->temp_d_a_broad);
        alpha_D= (line+ i)->nyu/ alpha_H* sqrt_log2;
        (line+ i)->r_of_L_and_D= alpha_L/ alpha_D* sqrt_log2;
        (line+ i)->intencity *= compensation;
        }
```

```
/*-------------------kernel calculation--------------------------*/
if(-1 == fero_calc(argc, argv)){
        fprintf(stderr, "Interferogram calculation error ¥n");
        exit(1);
        }

free(kernel);
free(kernel_buf);
free(line);

if (v_level > 0) printf("P(%d): kernel function finished¥n", Mpi_my_id);

if (Mpi_my_id == 0){
        if (NULL == (fp= fopen(out_file_name, "w"))){
                fprintf(stderr, "data output file open error¥n");
                exit(1);
                }
        d= (nyu_max- nyu_min)/ fourier_size;
        for(i= 0; i< fourier_size; i++) xaxs[i]=  nyu_min+ i* d;
        for(i= 0; i< fourier_size; i++){
                if(xaxs[i] >= target_nyu_min &&
                xaxs[i] <= target_nyu_max)
                fprintf(fp, "%lf, %lf¥n", xaxs[i]* 0.01, tau[i]);
                }
        fclose(fp);
        }

MPI_Finalize();
exit(0);
}


/*-------------------------------------------------------------
        kernel calculation function
-------------------------------------------------------------*/

int fero_calc(int argc, char **argv){

int i, j;
double stime, ttime, etime;
double t, m, k_yy, k_nyu, k_intencity, k_bulk;
double nyur, t_nyur, dc, c0, c1, c2, u;
double e, e0, e1, e2;
double f, f0, f1, f1sqr, f2, f2sqr;
double gR, gI, gR0, gI0, gR1, gI1, gR2, gI2, gR_old, gI_old;
double h, h1, h2, h_old, h_old_old;
double m0, m1, m2;

fftw_plan p;
```

```
int sp, ep, my_size;
FILE *fp;

if (v_level > 1) printf("fero_calc size=%d¥n",fourier_size);

i= x_parameter_range(0, number_of_lines- 1, Mpi_n_procs, Mpi_my_id, &sp,
&ep);
if (v_level > 0) printf("P(%d): My range is (%d, %d)¥n",  Mpi_my_id, sp,
ep);
my_size= ep- sp+ 1;

k_bulk=  bulk_O3[1]* avogadro_c;

stime= MPI_Wtime();

for(j= sp; j< ep; j++){
   k_yy= (line+ j)-> r_of_L_and_D;
   k_nyu= (line+ j)->nyu;
   k_intencity= (line+ j)->intencity;
   nyur= k_nyu/ nyu_max;
   t= kernel_step;
   t_nyur= t* nyur;

   /*****[0] step*****/
   gR0= 1; gI0= 0;
   m0= k_intencity/ k_nyu* k_bulk* nyur;        /* e0, f0= 1*/
   kernel[0].re += m0* gR0;
   kernel[0].im += 0;                           /* gI0= 0 */
   /*****[1] step*****/
   /* t= 1.0 * kernel_step */
   gR1= cos(-t_nyur* alpha_H); gI1= sin(-t_nyur* alpha_H);
   e1= exp(-k_yy* t_nyur);
   f1= exp(-t_nyur* t_nyur/ 4.0); f1sqr= f1* f1;
   h1= pow(f1, -5.0);
   m1= m0* e1* f1;
   kernel[1].re += m1* gR1;
   kernel[1].im += m1* gI1;
   /*****[2] step*****/
   t= 2.0* kernel_step;
   t_nyur= t* nyur;
   e2= e1* e1;
   f2= exp(-t_nyur* t_nyur/ 4.0); f2sqr= f2* f2;
   gR2= gR1* gR1- gI1* gI1; gI2= gR1* gI1+ gI1* gR1;
   h2= h1* f1sqr;
   m2= m0* e2* f2;
   kernel[2].re += m2* gR2;
   kernel[2].im += m2* gI2;
   /*****recursive step*****/
   e= e2;
   f= f2;
```

```
   gR_old= gR2; gI_old= gI2;
   h_old= h2; h_old_old= h1;
   for(i= 3; i< fourier_size; i++){
        e*= e1;
        f*= h_old_old* f2sqr;
        gR= gR_old* gR1- gI_old* gI1; gI= gR_old* gI1+ gI_old* gR1;
        h= h_old* f1sqr;
        m= m0* e* f;
        kernel[i].re += m* gR;
        kernel[i].im += m* gI;
        gR_old= gR; gI_old= gI;
        h_old_old= h_old; h_old= h;
        }
   }
ttime= MPI_Wtime();
if (v_level > 0) printf("P(%d): elapsed time for kernel calculation is
%lf¥n",
                        Mpi_my_id,  ttime- stime);

if (v_level > 2) printf("kernel_step,alpha_H=%e,%e¥n", kernel_step,
alpha_H);
if (v_level > 2) printf("m=%e, t_nyur=%e¥n", m, t_nyur);
if (v_level > 2) printf("bulk=%e, k_intencity=%e¥n", k_bulk, k_intencity);
/*
for(j= 0; j< number_of_lines; j++){
        k_yy= (line+ j)-> r_of_L_and_D;
        k_nyu= (line+ j)->nyu;
        k_intencity= (line+ j)->intencity;
        printf("nyu=%e, y=%e¥n", k_nyu, k_yy);
        }
*/

if (v_level > 0) printf("P(%d): kernel calculation finished¥n", Mpi_my_id);

/*----------data sending from clusters and receiving by the
server-------------*/
stime= MPI_Wtime();

MPI_Reduce(&kernel[0], &kernel_buf[0], fourier_size*2, MPI_DOUBLE, MPI_SUM,
0,
        MPI_COMM_WORLD);
/* however, my_size*2 is proper for sending & receiving for complex numbers
?? */

ttime= MPI_Wtime();
if (v_level > 0) printf("P(%d): elapsed time for data transfer is %lf¥n",
                        Mpi_my_id,  ttime- stime);
if(Mpi_my_id != 0) return 0;

/*------------post calculation on the server--------------------------*/
```

```
/*----------------------apodization-----------------------------------*/
if (apodization_type >= 3)
        fprintf(stderr, "not defined apodization type¥n");
/*--------Hanning window--------------------------------*/
else if (apodization_type == 2){
        c0= 0.26;
        c1= -0.154838;
        c2= 0.894838;
        for(i= 0; i< fourier_size; i++){
                t= (double)i* kernel_step;
                u= t/(2.0* Pi* resolution_scale- kernel_step);
                apodize[i]= c0+ c1*(1.0- pow(u, 2))+ c2* pow(1.0 -u*u, 2);
                }
        }
/*-------Norton Beer window-----------------------------*/
else if (apodization_type ==1 ){
        for(i= 0; i< fourier_size; i++){
                t= (double)i* kernel_step;
                u= t/(2.0* Pi* resolution_scale- kernel_step);
                apodize[i]= (1.0+ cos(Pi* u))/2.0;
                }
        }
/*-------operation of window-----------------------------*/
if (apodization_type != 0 ){
        for(i= 0; i< fourier_size; i++){
                kernel_buf[i].re*= apodize[i];
                kernel_buf[i].im*= apodize[i];
                }
        }

/*------------dc component compensation---------------------------------*/
dc= 0.0;
for(i=0; i< fourier_size; i++) dc += kernel_buf[i].re;
if (v_level > 2) printf("sum of kernel.re=%e¥n", dc);
dc *= kScalingCoefficient* alpha_H/ sqrt((double)fourier_size);
if (v_level > 2) printf("dc= %e¥n", dc);

/*-----------------output interim calcultion results--------------------*/
if (Mpi_my_id == 0){
        if (NULL == (fp= fopen(interim_file_name, "w"))){
                fprintf(stderr, "interim file open error¥n");
                exit(1);
                }
for(i= 0; i< fourier_size; i++) fprintf(fp, "%d, %e¥n", i,
kernel_buf[i].re);
        fclose(fp);
        }

/*------------------Fourier transform---------------------------*/
```

```
stime= MPI_Wtime();

p= fftw_create_plan(fourier_size, FFTW_BACKWARD, FFTW_ESTIMATE);
fftw_one(p, kernel_buf, out);
fftw_destroy_plan(p);

ttime= MPI_Wtime();
if (v_level > 0) printf("P(%d): elapsed time for fft is %lf¥n",
                        Mpi_my_id,  ttime- stime);

m= kScalingCoefficient* alpha_H/ sqrt((double)fourier_size);
for(i=0; i< fourier_size; i++) tau[i]= m* out[i].re- dc;

return 0;
}


/*-------------------------------------------------------------
  function to determine the calculaion range for clients
-------------------------------------------------------------*/

int parameter_range(int min_comp, int max_comp, int n_procs,
                                int rank, int *start, int *end){
int i, k;
k= (max_comp- min_comp)/ n_procs+ 1;
i= min(rank* k+ min_comp, max_comp+ 1);
*end= min(i+ k- 1, max_comp);
*start= i;
return 0;
}

int x_parameter_range(int min_comp, int max_comp, int n_procs,
                                int rank, int *start, int *end){
int i, j, k;

k= max_comp- min_comp;
*start= min_comp;
for (i= 0; i< n_procs; i++){
        j= k* node_performance[i];
        *end= min(max_comp, *start+ j);
        if(rank == i) return 0;
        *start= *end+ 1;
        }
return 0;
}

int min(int x, int y){
if(x >= y) return y; else return x;
}
```

```
/*-------------------------------------------------------------
        A function to scan a keyword from the file
        when keyword found then return the number of
        arguments.
-------------------------------------------------------------*/

int get_param_str(char *argv[], char *key, char *parambuf){
int argc;
char keyword[128], head[128];
FILE *param_file;

strcpy(keyword, key);
if((param_file= fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Can't open %s. ¥n", argv[1]);
        }
else
        while(fscanf(param_file, "%s %d %[^¥n]",
                                head, &argc, parambuf) != EOF){
                if(strcmp(keyword, head) == 0){
                        fclose(param_file);
                        return argc;
                        }
        }
parambuf=NULL;
fprintf(stderr, "not found keyword: %s.¥n", keyword);
return 0;
}


/*-------------------------------------------------------------
  function to measure the computation performance of node
-------------------------------------------------------------*/
double performance_meter(){
double stime, etime, junk;
int i;

stime= MPI_Wtime();
for (i= 0; i< 1000; i++) junk= exp(0.1);
etime= MPI_Wtime();
return etime- stime;
}

/*------------EOF-------------------------------------------*/
```