Step-by-Step Guide to How to Use the Distributed-Parallel Processing Scheme and Histogram Routines in Cosmos (updated version)

August 6, 2007

Contents

This guide applies to Cosmos V7.28 or later[1]. The guide is mainly intended to those who are able to use a kind of PC cluster and want to use distributed-parallel processing of an event (except for the histogram routines)[2].

1 Distributed-parallel processing of one event

We assume only one event generation by the distributed-parallel processing scheme since the event we want to generate is of very high energy (for the plural event generation, see section6). As of March 2006, with 50 sets of 2 GHz cpu's, a full M.C of $\sim 10^{19}$ eV proton primary with a minimum kinetic energy of 500 keV is the practical limit. It needs about 10∼14 days. However, an extended scheme enables us to do essentially the full M.C simulation at 10^{20} eV within the similar order of days. We note the present scheme dose not use the so called thin sampling method, though the user can employ it simultaneously (deep use of thin sampling would spoil the merit of the current method).

The distributed-parallel processing scheme applies the skeleton/flesh method which has been implemented in Cosmos long time.

First we make a skeleton. Instead of fleshing out the skeleton directly (as it is the case in usual skeleton/flesh jobs), we smash the skeleton into a number of sub-skeletons and flesh them by a number of cpu's and finally assemble fleshed results to obtain one event as if it were generated by one cpu. Therefore we need 4 steps which are symbolically denoted as: skeleton-smash-flesh-assemble (SSFA). During fleshing at many hosts, we don't need complex communications among the hosts nor special distributed-parallel processing software.

We prepared two fleshing/assembling procedures as prototypes.

basic Outputting individual particle information and/or that of hybrid AS information.

histo In addition to "basic", outputting histograms.

The user can extend these in various ways. In fact, for the Telescope Array project, the second one is extended to output more information. The routines for the first case is in UserHook/DisPara/FleshBasic and those for the latter is in UserHook/DisPara/FleshHist.

The current scheme may typically be applied to primaries over $\sim 10^{17}$eV, but we show a step-by-step guide using a 10^{15}eV proton primary case so that the user can finish the example within a few min or so.

[1]Some routines related to .hyb data are updated and the data structure is now different from the description in the old manual (before 16 Jul. 2007) . The manual for V7.20 contained some description which did not correspond to the actual program. This version is upgraded largely and many processes are now automated.

[2]If the user installs a new Cosmos overriding an old Cosmos, it is strongly recommended to remove old UserHook/Hist, UserHook/DisPara, UserHook/DisParaRescue beforehand.

In what follows, sentences with blue letters indicate that you can use another value (name etc), but it would be better to use as it is because it is used as default in later processes, or using another value or name is almost no meaning.

A line like
$ make
indicates a command prompt and a command you may enter.

1.1 Making a skeleton

We first assume a **basic** type application. We observe individual particle as well as hybrid AS.

1. Go to UserHook/SkelFlesh
2. Edit paramBasicDemo. Some essential parameters for this job are

   ```
   ASDepthList = 1000 2000 3000 4000 5000 6000 7000 8000 9000  10000
   CosZenith = (1.0, 1.0)
   DepthList=  1000 2000 3000 4000 5000 6000 7000 8000 9000  10000
   DestEventNo = 1 1
   Freec = F,
   Generate = 'em'
   HeightOfInj = 20000.0,
   IntModel ='"dpmjet3"'
   Job = 'newskel'
   KEminObs = 100.0,

   SkeletonFile = '../DisPara/FleshBasic/Sparam'
   Generate2 = 'em/as'
   KEminObs2 = 500.e-6
   UserHookc = '../DisPara/FleshBasic/Skeleton',
      '/tmp/skelwork_#','noappend',
   UserHooki =  15,  16,   0       0,          0,
   UserHookr =             00,    00,
   ```

   We observe hybrid AS at 100 to 1000 g/cm^2 with a step of 100 g/cm^2 (ASDepthList). Individual particles may be observed at the same depth (DepthList; Actual output depends on the user).

   The number of events to be generated is 1 (DestEventNo). The first collision point is fixed (Freec=F) to be at the injection height (HeightOfInj: 20km $\sim$ 55 g/cm^2) [3].

   For the skeleton, we generate electromagnetic particles besides hadrons and muons, but don't generate hybrid AS. (Generate='em'). The "Job" must be 'newskel'. The observation minimum energy for skeleton is 100 GeV. (The appropriate value of KEminObs depends on the number of cpu's, the primary energy and so on. To make each sub-skeleton almost identical,

[3]This is intended to generate a typical shower. Of course, Freec=T may be used.

a smaller value is desirable. 1/1000∼1/50000 of the primary energy would be a choice. You may have to adjust Zprivate2.h in UserHook/DisPara/Smash at smashing.

).

For the parallel job, we need a copy of this input parameter file. It is automaticall put in '../DisPara/FleshBasic/Sparam' (=SkeletonFile). In the fleshing job, we generate hybrid AS (Generate2='em/as'). individual particles are followed down to 500 keV (KEminObs2). The additional parameters are UserHookc, UserHooki and UserHookr. The first value of UserHookc is the file name of the skeleton itself. The second value is the working file (# is replaced by a process number).

3. Edit 'primary' to set proton primary of energy 10^{15}eV.

4. $ make clean; make -f chookSkel.mk

5. $ ./skelXXX < paramBasicDemo

   where XXX should be your $ARCH (say, PCLinuxIFC).

   Then the skeleton making job starts and ends with the following messages in a short time.

```
...
...
  first Z=   56.2305271392052       g/cm2   20000.0000000009       m
++++++++++++
       1 events are memorized as skeleton
their  seeds are also memorized
-----------
    No of cummulative events =           1 No of events in this run=           1
    comp.    sampled    accepted
      1           1           1
###end of run###
```

   For some compilers, these may be followed by

```
****** Important *******
*  Namelist 'Sparam' put in
*  ../DisPara/FleshBasic/Sparam
*  cannot be read by your future job
*  Therefore, copy present namelist file as 'Sparam'  and
*  change Job='newskel' to Job='newflesh' there
```

   In this case, your Fortran compiler is unable to write namelist data which in turn is readable by a program again; therefore, you must do some manual business as indicated above.

1.2 Preparation for a distributed-parallel job

Before going into the smashing process, we have to prepare the following. We show two cases for submitting jobs: jobs are submitted via the SGE (Sun Grid Engine) job control system or SSH (secure shell). The latter ssh would be available on any modern unix OS but cannot judge which cpu is at leisure automatically. Therefore, it is better to use SGE, if it is available. For other job control systems, we need some modification of the job submitting system (see section8). For SSH, it is almost mandatory that the user can login to another host without using pass word[4]. The method of doing so would be found somewhere on the web. The data files we should prepare here are used both by SGE and SSH. However, for SGE, the files look like a help we could live without.

1. Go to UserHook/DisPara
2. Make a file allHosts in which you must give a number of lines each of which has a number and host name like

   ```
   1 tasim503       2       1
   2 tasim504       2       1
   3 tasim505       2       1
   4 tasim506       2       1
   5 tasim507       2       1
   ...
   ```

   The number may be a maximum of 4 digit integers[5]. They must be unique but could be random (not recommended though). The same host name may appear only once. It is followed by the number of cpu's (or cores) it contains. The last one is the relative cpu power; a larger value indicates faster cpu. The list must cover all the possible hosts to which SGE may submit a job. The number and host name correspondence is used only SSH jobs; SGE jobs uses only number. The value is used at smashing time. Higher power hosts will be allotted a larger number of sub-skeletons (Hereafter we may omit 'sub'). Since the SGE job cannot specify the execution host, the value should not be given for the SGE job (or give the same values). The lines with # at first item or blank lines are neglected.
3. Make a file Hosts in which a subset of allHosts is gvien, like,

   ```
   1 tasim503 1
   2 tasim503 1
   3 tasim504 1
   ...
   8 tasim506 1
   9 tasim507 1
   10 tasim507 1
   ```

   The number of hosts there is the number of parallel jobs. For SSH job, you must select alive hosts (and idle hosts as much as possible). For SGE, only the number of hosts and the number attached to each host are important to identify the files created by the job; the hostname and number correspondence will be lost in the SGE job).

[4]Your pc cluster system may prohibit login to an individual host. In such a case, the SSH job system cannot be used.

[5]This means current system supports up to 10000 hosts (rather cpu's). However, most applications would use cpu's less than 1000, therefore we put such a limit as "maxCPU" in UserHook/DisPara/Smash/Zprivate2.h. You must change the value if needed.

You may use mkHosts.csh to create Hosts:

```
./mkHosts.csh  10 > Hosts
```

This is to create 10 lines in Hosts. mkHosts.csh assumes allHosts is ready.

1.3 Smashing the skeleton

1. Go to UserHook/DisPara/Smash
2. Edit setupenv.sh. All shell scripts relating to the distributed-parallel job *submission* are written in sh (bash).
3. FLESHDIR=FleshBasic must be given as the future flesh directory. This must be compatible with the ones given in the parameter file at Skeleton making time; (i.e. SkeletonFile and UserHookc).
4. NCPU=10 is specified as the number of cpu's to be used. This must be the same as the number of hosts in the Hosts file.
5. MCPU=10, MARGIN=0,ENHANCE=1 must be given. For these, see later (section 5).
6. Others are those with blue letter categories.
7. $ bash (or sh)

   Do this, if your shell is not bash (sh). We must change the shell here; (required only at Smash).
8. $ source ./setupenv.sh

   This is to set environmental variables used in the smash process. If there are some files inside the FleshBasic/SkelDir, you will be asked they should be kept or not (normal answer is “remove all”). Besides, this will check the degeneracy of the numbers in Hosts file. If it complains, you have to remake the Hosts.
9. $ make clean ; make
10. $ ./smashSkelPCLinuxIFC

    (This is PC Linux with Intel fortran case).

    This will generate output something like

```
 # of cpu's=          10
output directory is ../FleshBasic/SkelDir/
          10 files will be created there as Skeleton0001 etc
------------
        5096 ptcls are observed ones in skeleton
 # of total ptcls at flesh=      119042
 cpu#    cpuPW    Sum E        # of ptcls
   1     1.0    94805.70        11904
   2     1.0    94805.70        11904
   3     1.0    94805.70        11904
```

```
  4     1.0    94805.70         11904
  5     1.0    94805.70         11904
  6     1.0    94805.70         11904
  7     1.0    94805.70         11905
  8     1.0    94805.70         11905
  9     1.0    94805.70         11904
 10     1.0    94805.70         11904
all events have been smashed
```

Here all cpu power is equal (=1, cpuPW). The "sum E" means the sum of particle energy and the "# of ptcls" the number of particles in the skeleton given alloted to each cpu. We see they are almost the same so that the fleshing jobs will need almost the same cpu time.

1.4 Fleshing the skeletons

1. Go to DisPara/FleshBasic
2. Edit setupenv.sh
3. Fix EXECID to be something to symbolize the job. We put EXECID=p15cos1.0, implying a proton primary of 10^{15} eV with vertical incidence. The first letter must be an alphabet (restriction by SGE). It must be such one that can be a part of a file name and within 32 characters. (Absolute path name of a file must be within 128 characters in the distribute-parallel job scheme).
4. `OUTDIR=/tmp/$USER`. This is the directory in which the main output from the job is stored. Since we suppose a lot of output for individual particle information (more than 50 kbyte/s from each cpu), writing the output to `$TOPDIR/Assemble/OutDir` (best place for later handling) might be overburden to the NFS so that we choose a local disk (/tmp). (Although this depends on the environment, if the NFS usage is very heavy, the job could spend a real time 100 times more than properly needed !)[6].

   If you have something important not to be deleted in `/tmp/$USER` you may choose deeper directory such as `/tmp/$USER/Basic` or `/tmp/${USER}2` etc. The directory will be created in the next step, if they are not present.

5. $ ./setupenv.sh

   This is to test your setting and delete files stored in a kind of working directories. Since we specified `/tmp/$USER` we are notified to delete files there[7]. If you are sure the directory exists in all hosts and nothing remains there, you may bypass the delete process. If the directory is non existent, you must also try the delete process; it will create the directory. All others are

[6]As mentioned earlier, your system may not permit using local disk or may not permit to login to a local host. If login by ssh is not permitted, you will have difficulty to gather data created in the local disk. Such a system generally supports high performance NFS, so you may use $TOPDIR/Assemble/OutDir.

[7]If you are going to use SGE in the fleshing step, the delete process must scan all the host listed in allHosts, since you cannot specify the host at execution. For SSH job submission, you can limit the hosts as you listed in Hosts.

If there is no target files to be deleted in a host, you will see warning-like messages.

You have to wait a few second for a dead host. Don't worry about that. However, if a host can respond to ping but cannot respond to ssh–this may happen in some case–you have to cancel the script and delete the host in "allHosts".

blue category. The script also copies 'primary' file used in the skeleton making time to this directory.

6. Edit chookFlesh.f

   In this example, we want output individual particle information at 600 g/cm^2 which is the 6th depth; we modify the program to write data for aTrack.where = 6.

7. $ make clean; make

8. $ ../execflesh.sh sge

   Since we want to use SGE, the argument should be sge. If it is ssh, the SSH job submitting scheme will be used. You will see the next interactive mesages.

```
/home/Users/kasahara/Cosmos/UserHook/DisPara/FleshBasic
parameter files have been created in
 /home/Users/kasahara/Cosmos/UserHook/DisPara/FleshBasic/ParamDir
 ENHANCE = 1 is forced
1) Do you flesh all skeletons by 10 cpus listed in ../Hosts
2) Or specify some numbers  among them  for flesh job ?
3) Or stop here
Enter 1, 2 or 3
1
You selected 1;  Enter y, if it is correct
y
command used for cpu 0001  is

COSMOSTOP=~/Cosmos
export COSMOSTOP
source $COSMOSTOP/UserHook/DisPara/FleshBasic/setupenv.sh $0

source  /home/Users/sgeadmin/default/common/settings.sh
source  /Loft1/Intel/ifc/bin/ifortvars.sh
...
```

   The red letter is your input. This will submit 10 sge jobs. The NUMB (= number listed in Hosts) is important to identify job result. If you find some host becomes down after the submission and re-submit the job, we should find this number (not host name). In that case, select, 2 instead of 1 in the previous number selection. Then, you will be asked to enter a list of such numbers. If such an accident is found after a long run (say, 10 days), re-submission will need another 10 days. This is a tragedy. To ease such a crisis, DisParaRescue is prepared. This will be mentioned later (section 2).

9. Status check. For SGE jobs, you may use "qstat" to see the status of the submitted jobs. For SSH and SGE, a handy way to see the job status is to go to ErrDir.

   - ls -l

     will tell you when each of .err file is updated. A dormant job may be a dead job.
   - `tail -n 4 *.err | more`

     will show such as

```
 stack tops=   100.187637329102
 stack tops=   100.100700378418
 stack tops=   100.094741821289
 stack tops=   100.087478637695
```

This shows the highest energy particles in the stack. Some change there indicates that the job is close to the end.
If the job normally ends, you will see

```
 No of cummulative events =           1 No of events in this run=           1
 comp.    sampled    accepted
   1            1           1
###end of run###
```

The last line "`###end of run###`" is important since it is used to judge that the job is ended normally or not by further steps.

10. Output.

    After several min or so (if the sge job hosts are not crowded and have ∼2 GHz cpu), the sge jobs will end. In `/usr/$USER` of an sge job host, you will find files such as

    ```
    p15cos1.0-xxxx-tasim503.0001.dat
    p15cos1.0-tasim503.0001.hyb
    ```

    p15cos1.0 is the EXECID, xxxx (say 2015) the sge job number (SSH job lacks this), tasim530 the host name. .dat file contains individual particle info. .hyb file contains the transition of photons, electrons, muons and hadrons together with other information such as depth, "age", "cog detph" and electron number by hybrid AS calculation (of a given skeleton). The file name must conform the rule that last part is
    `-hostname.NUMB.extention`. (extension is such as hyb, hist, dat).

1.5 Assembling the results

It is a simple task to assemble all .dat data; simply concatenate all the files. To assemble the .hyb AS data, we need some work.

1. Go to DisPara/Assemble
2. Edit setupenvHyb.sh
3. HYBFILE0=./$EXECID.hyb. This is to specify the file to become the final assembled hybrid data. Another one is also a blue category.
4. $ ./assemHyb.sh

   This command does "make" and issues the following messages[8].

   ($USER is kasahara in this case).

[8]If you used OUTDIR=$TOPDIR/Assemble/OutDir, this message will not come out.

```
Output seems  in /tmp/kasahara  of each  host
You have to gather .hyb data into /tmp/kasahara of this host
Now we are going to gather .hyb files in many hosts to /tmp/kasahara of this host
You have some files in /tmp/kasahara of the current host
1--Delete all files in /tmp/kasahara before gathering files (normal)
2--Delete only some files specifying file extesion(s).
3--Keep all files in /tmp/kasahara and gather the files
4--Files have been already gathered so keep them and proceed
5--Keep all files in /tmp/kasahara and quit
Select number
1
5, tasim503 is being inspected
...
```

Then, all of the .hyb data is gathered in the /tmp/kasahara/ of the current host. The script will execute a program to combine the .hyb data and generate final result in ./$EXECID.hyb.

5. Assembled result.
   The final .hyb data will look like

```
h    1  6 -1  1  1.000E+06  0.0000000  0.0000000  1.0000000  56.23   537.   537.
t   1  100.0 2072.4 0.219 0.186  1.361E+04  4.427E+03  1.281E+03  9.030E+02  2.424E+03  1.585E+01
t   2  200.0  433.4 0.655 0.372  3.467E+05  8.620E+04  5.193E+03  4.036E+03  6.939E+04  2.471E+02
t   3  300.0  241.5 0.840 0.558  1.532E+06  3.301E+05  9.912E+03  8.799E+03  2.921E+05  9.417E+02
t   4  400.0  180.1 0.978 0.744  3.118E+06  5.970E+05  1.350E+04  1.306E+04  5.504E+05  1.724E+03
t   5  500.0  144.7 1.099 0.930  3.907E+06  6.774E+05  1.571E+04  1.674E+04  6.239E+05  2.005E+03
t   6  600.0  122.2 1.201 1.116  3.483E+06  5.588E+05  1.642E+04  1.873E+04  5.027E+05  1.695E+03
t   7  700.0  106.0 1.276 1.302  2.496E+06  3.785E+05  1.612E+04  1.926E+04  3.280E+05  1.180E+03
t   8  800.0   94.3 1.332 1.488  1.549E+06  2.251E+05  1.502E+04  1.879E+04  1.874E+05  7.196E+02
t   9  900.0   84.8 1.368 1.674  8.747E+05  1.249E+05  1.368E+04  1.764E+04  9.797E+04  4.172E+02
t  10 1000.0   77.3 1.391 1.860  4.662E+05  6.360E+04  1.244E+04  1.618E+04  4.850E+04  2.032E+02
```

   The first line with “h” shows the event number (=1), primary code(=6), subcode(=-1), charge(=1), total energy(1.000E+06 GeV), direction cosines (0 0 1), first interaction depth(=56.23 g/cm^2), center of gravity (cog) depth of the electron number transition (537, 537 g/cm^2; the second one is better). The lines with “t”, the depth index, depth in g/cm^2, the Moliere unit (in m; 2 r.l above the depth along the primary), “age” of the hybrid Ne, depth/cog, Ng the number of photons, Ne that of electrons, Nmu that of muons, Nh that of hadrons, Ne computed by the hybrid method, dE/dx (GeV/(g/cm^2)).

6. Gathering .dat files and concatenate them.
   For this, you may use

```
 ./assemDat.sh  dir  file_name
```

   All .dat files will be collected in `$OUTDIR (=/tmp/$USER/` in this case), and concatenated to make a single file with name “`file_name`” in “dir”. Note that this process takes much longer time than for .hyb data due to large file sizes.

2 Rescuing a failed distributed-parallel job

Suppose a following situation. You have 50 hosts to flesh smashed skeletons; each host need 10 days to complete fleshing. Among them, one or few hosts failed to flesh the skeletons due to, say, some accidents (power failure, malfunction of network card etc). This happened when the job was reaching the end. So if you repeat the job on another host, it will take another 10 days while other hosts will finish the jobs soon. You need 20 days for final Assembling. If the smashed skeleton is smashed once more, and if you distribute the job to 10 hosts, you need only 1 day, so 11 days are enough for final assembling.

The outline of the rescue process is as follows:

- Failed jobs skeletons are collected and merged.
- The merged skeleton is smashed into a number of sub-skeletons and fleshed at a number of hosts.
- The fleshed result is assembled to form a one .hyb file and .dat file in the DisParaRescue/Assemble/.
- The first failed job is replaced by this rescue result:

  To the first failed job's .err file in `DisPara/FleshBasic/ErrDir/` is added "`###end of run###`". The .err files of other failed jobs are move to `DisPara/FleshBasic/ErrDir/Failed/`.

  Assembled .hyb and .dat file are copied to `$OUTDIR` of the original job. The name of the files are rescued-HostName.Number.hyb etc. where HostName and Number is the same one used in the first failed job.
- Rescued data is assembled together with the successful original data.

The details of the steps are as follows:

1. To rescue the job, first you have to gather data files from cpu's which finished fleshing successfully, if your output directory is /tmp/... of each execution host. (That is, if `$OUTDIR` is DisPara/Assemble/OutDir, you may skip this process). To do this, you may

   go to DisPara/Assemble

   and use

   assemHyb.sh

   and

   assemDat.sh.

   Note that you should keep .hyb and .dat files in the output directory.
2. Then, go to UserHook/DisParaRescue/Smash.
3. Edit setupenv.sh. Probably you may only specify the number of cpu's to be used in the rescue job. In the example, we give NCPU=5

4. $ ./getfailedNum.sh

   This is to collect failed job numbers and compile a program to merge skeletons of failed jobs. The output from this process will look like

```
SKELDIR is ../../DisPara/FleshBasic/SkelDir
Job numbers of the failed jobs are:
0007
0010
cppFCPCLinuxIFC -w -DPCLinuxIFC -I/home/Users/kasahara/Cosmos/cosmos            -c -c
********* ifc -Vaxlib
...
 ../../DisPara/FleshBasic/SkelDir/Skeleton0007 is copied to  ./Skeleton
making  tempskel from ./Skeleton +  ../../DisPara/FleshBasic/SkelDir/Skeleton0010
mv tempskel to ./Skeleton
All files have been  merged  and saved as ../FleshBasic/Skeleton
```

   or much simpler if failed job is only 1:

```
SKELDIR is ../../DisPara/FleshBasic/SkelDir
Job numbers of the failed jobs are:
0008
```

5. $ ./mkHosts.sh (This is in Smash).

   This will create ../Hosts. The numbers used there are those not used in original Hosts. In out example, ../Hosts will look like

```
999 tasim503 1
998 tasim503 1
997 tasim504 1
996 tasim504 1
995 tasim505 1
```

6. $ bash (if your shell is not sh/bash).

7. source ./setupenv.sh

   You may asked if files in SkelDir may be removed. Normally, you may remove them.

8. make clean; make -f smashSkel.mk

9. $ ./smashSkelPCLinuxIFC

   (This is for Linux with Intel Fortran). This smashes failed jobs skeletons (if there are more than 2 failed jobs, the skeletons are merged and smashed). The output will look like

```
 # of cpu's=           5
output directory is ../FleshBasic/SkelDir/
           5 files will be created there as Skeleton0001 etc
------------
           0 ptcls are observed ones in skeleton
```

```
  # of total ptcls at flesh=       20414
  cpu#   cpuPW    Sum E        # of ptcls
    1    1.0    38343.31         4083
    2    1.0    38343.31         4082
    3    1.0    38343.31         4083
    4    1.0    38343.31         4083
    5    1.0    38343.31         4083
 all events have been smashed
```

10. Then, go to DisParaRescue/FleshBasic
11. Edit setupenv.sh

    You may give EXECID and OUTDIR. They may be the same as the original job.
12. $ make clean; make
13. ./setupenv.sh

    Probably, you may proceed the default way.
14. ../execflesh.sh `[ssh|sge]`
15. Go to DisParaRescue/Assemble when fleshing of the rescue job is ended.
16. Do assembling by using assemHyb.sh and assemDat.sh as in the normal assembling. The assembled files will be put in the same directory with file names, `$EXECID.hyb` and `$EXECID.dat`.
17. ./patching.sh

    This selects one failed job (top of "failedHostNum" file in DisParaRescue/Smash), add `###end of run###` to its .err file, copies assembled files into the original output directory with names "rescued-Host.Num.hyb" etc (Host.Num is the top item in the failedHostNum file). Then, the script changes the directory to the DisPara/Assemble, and uses RassemHyb.sh and RassemDat.sh to get final assembled results. If the process fails during this last assembling phase, you may go to DisPara/Assemble and use RassemHyb.sh or RassemDat.sh directly.

3 Distributed-parallel processing with histogram output

Note: Histogram routines can be used only with the Intel Fortran at present. It is needed to allocate a storage dynamically within a structure construct. This feature is not supported by, say, Absoft Fortran 90 yet.

How the histogram routines are organized will be explained later (see section 10). We shall go without knowing it for a while.

At very high energies, if we output individual particle information, we will need 100 GB or even several TB disk size. This situation may be overcome by taking histograms during the program run without outputting individual particle information. The other solution may be outputting only a fraction of individual particle information. Of course, mixture of the two is possible. In this section, we describe how to take histograms on fly, although we also output individual particle information.

The latter solution will be discussed later (5).

3.1 Preparation

1. Go to UserHook/Hist
2. $ `make clean; make`

   This may be done only once. This is to create `k90whist*.o` in the library.
3. We need "allHosts" and "Hosts" in UserHook/DisPara as in **basic**. We use the same one.

3.2 Making a skeleton

This is almost the same as for **basic** applications. For the demo, we use the same condition.

1. Go to UserHook/SkelFlesh
2. Edit paramHistDemo

   This is nothing but a copy of paramBasicDemo except for "FleshBasic" is changed to "Flesh-Hist" (at two places).
3. $ `make clean; make -f chookSkel.mk`
4. $ `./skelPCLinuxIFC < paramHistDemo`

   "skelPcLinuxIFC" here again means the binary by the Intel Fortran case.

3.3 Smashing the skeleton

1. The only difference from the **basic** case is to put `FLESHDIR=FleshHist` in setupenv.sh.
2. $ bash   (If you shell is not sh/bash).

   Don't forget we must use sh for smashing.
3. $ source ./setupenv.sh
4. $ make clean;make
5. $ ./smashSkelPCLinuxIFC

   Then, you will see the following result:

```
  # of cpu's=          10
output directory is ../FleshHist/SkelDir/
          10 files will be created there as Skeleton0001 etc
------------
        4808 ptcls are observed ones in skeleton
 # of total ptcls at flesh=      113333
 cpu#   cpuPW    Sum E        # of ptcls
   1    1.0    94470.89        11333
   2    1.0    94470.89        11333
   3    1.0    94470.89        11333
   4    1.0    94470.89        11333
   5    1.0    94470.89        11333
   6    1.0    94470.89        11333
   7    1.0    94470.89        11333
   8    1.0    94470.89        11334
   9    1.0    94470.89        11334
  10    1.0    94470.89        11334
all events have been smashed
```

These are almost the completely the same as the **basic** case.

3.4 Fleshing the smashed skeletons

The difference from the FleshBasic is that the first line of interface.f is `#define  USEHISTO` while it is `#undef  USEHISTO` in FleshBasic/interface.f. Other difference is that interfaceH???.f and Zprivate?.f are prepared which are included in interface.f.

1. Go to `UserHook/DisPara/FleshHist`.
2. All codes that the user may want to modify are gathered in interface.f and interfaceH???.f. However, most output would be controlled by setupenv.sh.
3. Edit setupenv.sh
4. Many will be set similarly as for setupenv.sh in FleshBasic.
5. Fix `EXECID`. Must start with an alphabet and is such that it can be a file name.
6. `HISTDEP='3 4 5 6 7 8 9 10/'`

   We specify at which depth we tack histograms by this. In this example, we specified 3 to 10-th depths. (I.e, 300,400,,..1000 g/cm^2).
7. `INDIVDEP='0/'`

   This is not used in our example. For individual output, we put “write statements” in interface.f so that info. at the 6-th depth can be obtained.

8. OUTPUT='t t t t t t t t t t t f f/'

   There is a number of histograms (=12) predefined in the program. If `t` is given at n-th position, the corresponding histogram is taken and output. If `f`, no output. The n-th one is for:

   1 In fact, the first one is not for histogram but to control individual particle output. (Since INDIVDEP='0/', `t/f` is not referred.)

   2 Lateral distribution (of γ. e, μ. Unless otherwise stated, the same is true below) (lat)

   3 Lateral distribution of dE/dx (in GeV/(gm/cm^2)) (of e, μ and e+μ). (dEdxlat)

   4 Energy spectrum parameterized by radial distance r. (re).

   5 Zenith angle distribution (in $\cos = z$) parameterized by r. (rz).

   6 f distribution parameterized by the zenith angle (in cos). f is defined as $f = \vec{r}\cdot\vec{d}/r$, where $\vec{r}$ is the 2D position vector and $\vec{d} = (\cos\varphi, \sin\varphi)$ with φ being the azimuthal angle. (zf)

   7 f distribution parameterized by r. (rf)

   8 f distribution parameterized by e.(ef)

   9 t distribution parameterized by r and e. ti is the arrival time (in ns). (ret)

   10 t distribution parameterized by r. (rt)

   11 z distribution parameterized by r and e. (rez)

   12 f distribution parameterized by r and z. (rzf)

   13 f distribution parameterized by r and e. (ref)

   The last symbol in parentheses is used as a quick reference in graphic display interface (Is is used as "category").

9. OUTDIR. `We use /tmp/$USER`.

10. Others are blue letter category.

11. $ ./setupenv.sh

    If some files remain in a kind of working directory, you will be asked to delete them.

12. $ ../execflesh.sh sge

    This process is the same in FleshBasic.

```
/home/Users/kasahara/Cosmos/UserHook/DisPara/FleshHist
parameter files have been created in /home/Users/kasahara/Cosmos/UserHook/DisPara/Flesh
ENHANCE = 1 is forced
1) Do you flesh all skeletons by 10 cpus listed in ../Hosts
2) Or specify some numbers  among them  for flesh job ?
3) Or stop here
Enter 1, 2 or 3
1
You selected 1;  Enter y, if it is correct
y
```

    Then, SGE jobs will be successively submitted.

13. At ErrDir, you will be able to check the job status as in the FleshBasic case.

3.5 Assembling

The only difference from the FleshBasic case is we need to use assemHist.sh besides assemHyb.sh and assemDat.sh.

1. Go to DisPara/Assemble.
2. First we assemble .hyb data. Confirm setupenv.sh. Probably you need not modify it.
3. $ ./assemHyb.sh

   This will create $EXECID.hyb in the current directory. This is the final assembled hybrid data.
4. Confirm `setupenvHist.sh`. Probably you need not do anything.
5. $ ./assemHist.sh

   This will assemble all .hist data and generate $EXECID.hist in the current directory. This is a binary file so you cannot recognize the contents directly.
6. $ ./assemDat.sh ./
7. The binary .hist file is not yet really the final one. It has some hybrid AS information which is not based on the finally assembled .hyb data. So we must replace it by the true hybrid information as in the next step.

3.6 Modifying the .hist data with final .hyb data and getting files for plotting

No modification is needed if the .hist data dose not utilize hybrid AS information (You could organize so in interfaceH???.f). In our example, we utilize hybrid AS size, age, etc for the "ID (or key)". Note that, even if we don't modify the .hist, the graph itself is correct. Only keys (in terms of gnuplot) become inappropriate.

1. Probably you don't need to Edit setupManipHistEnv.sh, but have a look at it FYI.

   We shall express an Assembled histogram file as .hist file. This is in default not complete. So we modify it with .hyb data and obtain complete file which we call .chist file. The binary file .hist and .chist can be converted into ascii format. We call the .ahist and .achist, respectively. .achist can be obtained from .hist + .hyb or .chist. From .achist, we can produce a number of files for plotting. Also from .ahist we can do the same, though the "key" is in default, not correct. In summary, we have following type of jobs.

```
#   Following type of jobs are supposed.
#  1)  .hist + .hyb ---> .chist
#  2)  .hist + .hyb ---> .achist
#  3)  .chist       ---> .achist
#  4)  .achist      --->  plotting files
#
#  5)  .hist        ---> .ahist
#  6)  .ahist       --->  plotting files: graph itself is corrcect but
#                                key comment becoems wrong.
#  7)  .chist + .hyb ---> .achist (possilble but redundant)
```

Although, 2+4 in the above processes is the shortest way to get graphs, it is recommended to keep the complete binary histogram file (.chist); it containes everything and even it affords some possibility to get a different ascii output. So we prefer to using 1 + 3 + 4.

2. Therefore, we set

```
#   Job type
JOBTYPE=" 1  3 4"
```

3. Input files and output files are blue letter category.

4. As directory to store files for plotting. we set

```
PLOTDIR=./$EXECID-Plot
export PLOTDIR
```

5. $ ./manipHisto.sh

   Then, everything is managed by the script. Making plotting files will take some time. After the script is finished, you will see `$EXECID-Plot` directory in which many files for plotting are stored. Also several `$EXECID.*hist` files are created.

6. Without using script, the user can make plotting files from .ahist (.achist) files.

   $ `awk -f $COSMOSTOP/Scrpt/splitHisto.awk maindir=dir .ahist-file`

   whee dir is the diectroy to store the generated files, and .ahist-file the soure ascii histogram file.

7. In an actual job, the final .chist, .hyb file, .dat file and files for plotting are to be kept somewhere.

8. To get files for plotting
   splitHisto.sh dir xxx.achist
   where dir the directory to store the files for plotting.

How to plot the data inside the plotting directory will be described later.

3.7 Rescuing a distribute-parallel job with histogram output

The method of rescuing a failed job in this case is quit similar to the previous case. You may simply remember about .hist data; If you do some for .hyb data, you have to do the similar one for .hist data too. After you get assembled data for .hist, you have to do manipHisto.sh for which .hyb has no counterpart.

4 Multiple distributed-parallel processing

In some case you may want to run plural sets of distributed-parallel jobs. If you do two distributed-parallel jobs within the DisPara directory simultaneously, there will be a confusion about environmental variable setting. The simplest way to do multiple distributed-parallel jobs will be copying DisPara and its descendent to a different directory under UserHook/ (Let's name it DisPara2).

1. Go to UserHook/
2. cp -r DisPara DisPara2
3. Go to DisPara2.
4. $ ./chgDisPara.csh
5. Edit setupenv.sh

   Change “ARENA” value to DisPara2
6. For rescue job, you may use DisParaRescue, provided that you have to change setupenv.sh under DisParaRescue; give DisPara2 to ORIGIN.
7. If you are afraid of that such change may cause confusion, you may also make a copy of DisParaRescue (say, as DisParaRescue2), and use chgDisPara.ch there. In this case, you have to also change the values of ARENA and ORIGIN in setupenv.sh there; give DisParaRescue2 to ARENA and DisPara2 to ORIGIN.

Note: You must remember that in the parameter file for skeleton making (in UserHook/SkelFlesh), DisPara2 must be specified instead of DisPara (at two places). The “primary” file there must not be changed until fleshing process starts. When fleshing job starts, a copy of the “primary” file should have been put in the fleshing directory so that the copy will be used hereafter by other scripts. If you use /tmp/... as an output directory, you have to specify a different subdirectory in /tmp/ from other distributed-parallel jobs.

5 Managing a huge output

At 10^{18} eV or higher energies, outputting individual particle information leads to a disk size crisis. Individual particle information would be needed, for example, when we want to get detector response. For such a purpose, normally we don’t need to record all particles. One may randomly select particles to be recorded. There will be various methods for doing so.

The method used for the TA project is to record particles at every 25 g/cm^2. The observation area at a given depth is divided like a spider-web (see Fig.1). In each area we count the total number of particles for each particle type, but as to individual particle information such as energy, position, angle, arrival time etc, a maximum of a fixed certain number of particles is recorded in each sector.

To be able to randomly select particles at each sector, we must have an expected total number there, which is rather difficult to know before calculation. As a result, for safety, we record particles larger than the fixed number, and sometimes it becomes 10 times more than that number. Therefore, we have to do random selection again after completion of the job.

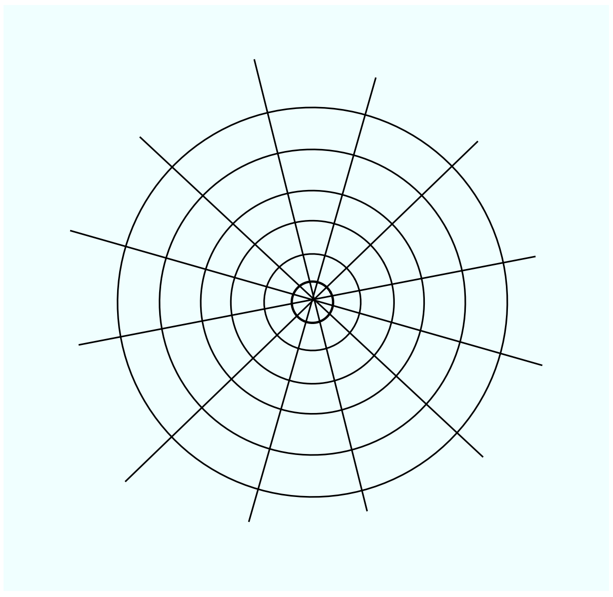

Figure 1: Web sectors

The TA project needs air showers in the GZK energy region. With 100 cpu's, full M.C of a 10^{20} eV shower needs 2.5 months. This is still too long. However, in conjunction with the fact mentioned above, we are led to the following method.

We already know that every smashed sub-skeleton looks almost identical. Therefore, if we make 1000 sub-skeletons, and flesh only 100 among them, we can get an shower scaled to 1/10 of the full version. Fluctuation among the sub-skeletons are very small so that this scaling is very accurate. This is a kind of thin sampling of ensemble and there is no weight on each particle at all. The number of particles from 100 sub-skeletons is still too large to record so that we need not worry about one particle with a very large weight.

Some variables are prepared in setupenv.sh in Smash. for such a purpose.

- **NCPU. The number of virtual cpu's (i.e, sub-skeletons), say, 1000.**

  After fixing Smash/setupenv.sh, you must make Hosts file containing NCPU host list by using mkHosts.csh at DisPara (Say, ./mkHosts.csh 1000 ¿ Hosts). Max number of NCPU is currently 9999.

- **MCPU. The number of cpu's actually used. say, 50.**

  You must make ThinHosts file containing MCPU host list using mkThinHosts.sh at DisPara, after Smash/setupenv.sh is fixed. (./mkThinHosts.sh)

- **MARGIN. The number of cpu's for margin. The results from these cpu's may be used when some hosts among MCPU failed to complete the job (MARGIN+MCPU $\leq$ NCPU).**

- ENHANCE. The scaling factor, NCPU/MCPU.

The ENHANCE factor is already considered in every quantities from interface*.f (i.e, .hyb and .hist data). Each particle has an equal weight of ENHANCE.

You may utilize these variables for your own work.

6 Plural events generation at a time

So far we have been showing only 1 event generation. It is possible to generate plural events in one parallel job. Simply give a number greater than 1 at Skeleton making. Smashing, Fleshing and Assembling can be done without paying attention to the fact we are dealing with plural events. After making .achist file, we have to take a little bit different way from the one event case; we have to split the events into individual event for plotting.

1. $ `splitEvent.sh .achist-file event1 event2`

   where *.achist-file* is the path to a .achist file which contains a number of events. *event1 event2* are the first and last event number to be extracted from the file. If *event2* is not given, *event1* to the last event are extracted. If *event1* is not given, all events are the target. The extracted events are stored in the same directory as the source file. The name of each extracted file will be composed using the source file. Fo example, if a source file name is a.b.c.achist, output will be a.b.c-1.achist a.b.c-2.achist,...etc.

2. Apply splitHisto.sh to these individual files.

3. Output of individual particle information in .dat file.

   The concatenated final file has events not in a sequential way. You have to collect data of desired event number: For example, if you want gather event number 3,

   ```
   awk  'BEGIN{put="no"}   \
   $1=="i" && $2==ev {put="yes"; print;next} \
   $1=="i" && $2!=ev {put="no";next} \
   put=="yes" {print} '   ev=3 concatenated-file.dat > event3.dat
   ```

   will give you the comprehensive data.

7 .hist file from other place

We may use histogram routines in other places than parallel processing scheme. We could modify setupManipHistEnv.sh for such a case. But it may be also good to

remember the basic treatment. Such histograms would not use .hyb data, so we don't worry about incomplete binary .hist; the .hist file is complete from the first.

1. **To convert it to .achist, go to UserHook/Hist.**
2. **Fix the HISTFILE0 environmental variable to be the source .hist file.**
3. **$ make -f bin2ascii.mk**
4. **$ ./bin2ascii$ARCH > xxx.achist**[9] **($ARCH is such as PCLinuxIFC)**

8 Other job control systems than SGE

If the user uses non SGE job control system, probably the files to be modified are execSGEtemplate.sh in FleshBasic and FleshHist. In DisPara, there are 3 files: execflesh.sh execflesh_one.sh and execflesh_all.sh. They have a branch instruction depending on user input ssh/sge. This part must be also modified.

9 Plotting graphs

The histogram files are organized to be ready for plotting in the p15cos1.0-Plot directory in this example. The basic data file is a table of histograms showing x vs dN/dx. and some others. The details will be given in a later section (10). Such data will be plotted by many softwares; we use gnuplot here.

9.1 1D histogram

1. **Go to inside of the directory. You see a directory list**
   dEdxLat ef lat re ret ret2 rez rf rt rz zf
   which has been explained in 8 of p.17.
2. **Go to the "lat" directory where you see a directory list**
   Electrons Muons Photons
   which implies the lateral distribution of electrons, muons and photons are available.
3. **Go to Electrons. You will see 5 .dat files which contains table; its basic ingredient lists**

   ```
   x  dN/dx
   ...
   ...
   ```

[9]Unfortunately, it is not easy to make the input by redirection as ./bin2ascii$ARCH < somebin.hist > xxx.achist.

The second item may sometimes be $\frac{1}{N}\frac{dN}{dx}$.

4. Each .dat corresponds to a different depth which you specified by setupenv.sh in 6 of p.16.

5. To have a quick view of superposed graphs of lateral distribution at these 5 different depths,
   $ gnuplot plog.gp

6. You will see a graph and notice the vertical scale is in $r^2\frac{1}{N}\frac{dN}{dr}$ with r in Moliere unit. This means the graph is normalzied as $\int \frac{1}{N}\frac{dN}{dr}dr = 1$. Note that, if the particle density is expressed by $\rho(r)$, $\frac{dN}{dr} = \rho(r)2\pi r$. Also, the 2nd item in the data file is $\frac{1}{N}\frac{dN}{dr}$, and r^2 weight is used only at display.

7. The keys on the right top show the depth index, depth, age, depth/cog, m.u and cog. This should be correct one since we did a such work previously.

8. With this plot, we cannot do anything except for, say, printing it; no curve fitting, no different weighted graph, no change of key position.

9. To be able to control every detail of the plotting, you have to first invoke gnuplot and load 'plot.gp' as
   $ gnuplot
   ```
   gnuplot> load "plot.gp"
   ```

10. To do a curve fitting at this point, you may define a function, say,

    ```
    gnuplot > f(x) = p1 * x**(-p4) * (1.+ x/p2)**(-p3*log10(x)-p5)
    gnuplot > p1=4; p2=0.2; p3=0.6; p4=0.1; p5=2
    gnuplot > fit f(x) "3.dat" via p1,p2,p4,p5
    gnuplot > rep f(x)*x*x
    ```

    Our target is "3.dat" data which corresponds to 700 g/cm^2. The fitting is tried with fixed $p3 = 0.6$ (via p1,p2,p4,p5). Since the fit has been done to non weighted data while the display has r^2 weight, we have to write `rep f(x)*x*x` to see the fitted result. It is not bad at $r < 10$.

11. I some case, fitting in a wide range by a single function may be difficult. It is sometimes a good idea to divide the fitting region into two.

    ```
    gnuplot > g(x) = r1 * x**(-r4) * (1.+ x/r2)**(-r3*log10(x)-r5)
    gnuplot > r1=4; r2=20; r3=0.6; r4=0.1; r5=2
    gnuplot > fit [10:100] g(x) "3.dat" via r1,r4,r5
    gnuplot > rep g(x)*x*x
    ```

```
gnuplot > load "plot.gp"
gnuplot > rep x<20 ? f(x)*x*x: 1/0
gnuplot > rep x>5 ? g(x)*x*x: 1/0
```

12. Different weight representation.

    Edit plot.gp

13. change pw=2.00 to pw=1
14. change ylabel accordingly (”rdN/Ndr”)
15. change key position. (set key 1,0.01)
16. load ”plot.gp”

    In this case, rep is not enough.

17. Thicker lines or dot presentation.

    change the last line ”w his” to ”w his lw 2” or ”w p”. and load again.

18. Unnormalized plot.

    Change the last \$2 to \$3 and ylabel to ”rdN/dr”. The key position must also be changed.

19. If you want to use this presentation as default for other similar M.C results, you may rename the plot.gp file and keep it there. You may overwrite later files here. And use saved “plot.gp”. var.gp need not be saved; It holds only variable part from data to data.

9.2 2D histogram

It should be mentioned that our goal is not to produce so called Lego plot or a like for 2D/3D histograms. Such an output will be treated elsewhere. Our final output is a 1D distribution for a given parameter space. Say, energy spectrum at a given lateral distance.

1. Go to “re/Photons” directory. “re” means energy spectrum at different lateral distances. This directory still has subdirectories:
   d400 d600 d700 d800 d1000
   implying data at depth 400 to 1000 g/cm^2.
2. Go to d600. Then, you will find r1.dat, r3.dat, r5.dat... plot.gp and var.gp. There is no r2.dat etc; this is because we specified lateral distances with some steps. The binary hist file contains data corresponding to r2.dat and it can be extracted if necessary.
3. The gnuplot graph can be shown same as in the 1D case. The rightmost part of the key will tell you the lateral distance.

4. Note that normalization for 2D histograms is such that $\int *\frac{dn}{dxdy}dy = 1$ for fixed x but not $\int *\frac{dn}{dxdy}dxdy = 1$.

9.3 3D histograms

We will need arrival time distribution at a given lateral distance as a function of energy. Therefore, we need a 3D histogram as "ret". The directory organization is rather awful for 3D. For example, you have to go to ret/Electrons/d600/r3 to be able to reach plot.gp. The normalization is $\int *\frac{dn}{dxdydz}dz = 1$ for fixed x and y.

10 Histogram routines

Note: Histogram routines can be used only with Intel Fortran at present. It is needed to allocate a storage dynamically within a structure construct. This feature is not supported by, say, Absoft Fortran 90 yet. Therefore, they are not included in the library. To include them into the library (for Intel Fortran),

1. Be sure your site.config is for Intel Fortran.
2. In UserHook/Hist
   $ make clean; make

   This should have been done already in the earlier section.

The routines can be used any applications. The routines support up to 3D histograms. The 1D histogram is obvious. For example, we can make an arrival time (T) distribution, disregarding all other constraints. We may wan to see the same distribution as a function of lateral distance (R). For this we may digitize R and T and accumulate the frequency of the digital bins. This is a 2D histogram. We can regards it as a distribution of R as a function of T, too. The variables may be expressed, in general, by (X), (X, Y) or (X, Y, Z) depending on 1D, to 3D histograms.

If we choose , some particular X and Y for 2D, the plotting routine for such a histogram understands our main purpose is to see the distribution of Y as a function of X. Thus, if we want too see T distributions at various R's, we should choose R as X and T as Y. That is, R is regarded as a parameter[10].

The same rule applies to the 3D histograms. If we want to get T distributions at different R's as a function of energy (E), we should organize as (R, E, T). R and E are regarded as parameters.

[10]The output for other purposes, say, lego plot, can be made from 2D histogram. But we don't touch it here.

Our output files for plotting are organized to be fitted directly to gnuplot, but the table data itself would be used any plotting applications (without any modification, or adding 1- or 2-line headers).

Test programs are supplied in UserHook/Hist as test1.f, test2.f and test3.f to see how 1D to 3D applications are written. The header files (Z90hist*.h) are here, not in Cosmos/cosmos so you must show the relative path to the files in your application.

1. Let's have a look at test1.f.

   We generate power spectrum of 3 different indexes and see the spectrum (3 histograms by k(3)). Besides, we also generate Gaussian distributions with two different averages, each of which has 5 different variances (10 histograms by h(2,5)).

2. Except for small applications, we shall use binary output.

   The output file must be opened by the user with form='unformatted' option.

   We first use
   `call kwhistso(2)`
   to declare that our output should be binary. (If the argument is 1, ascii output is obtained). This call is needed for any 1D, 2D, or 3D applications.

3. To initialize histograms,
   `call kwhisti(k(i), 1.5, 0.1, 30, b'01111')`
   (or generally `call kwhisti(area, min, bin, nbin, bitpattern)`
   is used. Here `i` runs from 1 to 3 so that we define 3 histograms. 1.5 is the minimum of the variable. 0.1 is the bin. 30 is the number of bins. The LSB of b'011111' specifies if we take $\log 10$ or not. The bit 1 indicates we take log. Even for the log case, the minimum value is not in log. The bin is for how we divide 1 log scale. (0.1 means 1 log decade is divided into 10).

Table 1: Bit pattern for the histogram initialization

Bit position		Meaning
1	11111	If 1, $\log 10$ is taken
2	11111	If 0, given min is the min value of the lowest bin If 1, given min is the middle value of the lowest bin
3	11111	If 0, underflow is neglected If 1, underflow is included in the lowest bin
4	11111	If 0, over flow is neglected If 1, over flow is included in the highest bin
5	11111	If 0, bin is the really bin If 1, bin is regared as the max of histogram. bin is determined automatically

4. The histogram area must be cleared.
   `call kwhistc(k(i))`

5. **Generate random variables and count them.**
   `call kwhist( k(i), sngl(x), 1.0 )`
   The variable must be in single precision. The last 1.0 means the weight. Suppose a thin sampling, then we have particle number not just 1, but 1.3, 2.0 etc. In such a case, we may use such a value.

6. **After counting, we perform some statistical calculations.**
   call kwhists(k(i), 0.)
   The 2nd argument 0. means you want to normalize the distribution to be 1. (area normalization). If you give 1.0, or some other positive value, $\frac{dN}{dx}$ is normalized by that value.

7. **Print the histogram. Actually we specified binary output, the file is created in this case.**
   `call kwhistp(k(i), fno)`

   If we have specified ascii output, and fno<0, the output would be to the "stdout ". If fno>0, the disk file is used. (The file must have been opened with formatted option).

8. **These are minimum of 1D histogram usage.**

9. **kwhists and kwhistp may be called as many times as you want for the same histogram. (Say, after calling kwhists with 0. you may call it with 1.0)**

10. **Other optional call's.**

    Next Gaussian case explains more subroutines of which call gives more comprehensive display of graphs.

11. **To give additional information to the histogram: some of them is used where the files for the plotting are to be stored, others when the graphs is displayed.**

```
              call kwhistai(h(i,j),
      *       "Test Gaussian dist.",
      *        "gauss", "event", .false., 0.,
      *          "x", "m")
```

    h(i,j) is the histogram area. "Test Gaussian dist" is the title of the graph. "gauss" is the "category" of the graphs. It could be used when we have lots of graphs where the files should be stored. It must be such one that can be a part of the file (directory) name. "event" is generally not so important argument; it shows the unit of "dN". .false. means the vertical scale should be ordinary scale when displayed (for gnuplot). 0. is the power index to be used for the graph: x^{power} is multiplied to the vertical quantity. "x" is the $x-$axis label. "m" is the unit for the $x-$axis.

12. `call kwhistid(h(i,j), key)`
    gives a key; it is displayed on the graph and serves for identifying histograms when multiple histograms are displayed.

13. `call kwhistdir(h(i,j), dirstr)`
    This may be a bit difficult to understand. This specifies the directory where the files for plotting should be stored.

    We used a loop

    ```
    do i = ..
         do j = ..
               call kwhist..(h(i,j)...)
         enddo
    enddo
    ```

    This means we have a lot of graphs. If we don't use this call, there will be lots of files in the same directory and we cannot classify which graph is which.

    The files for plotting will be organized in the following way

    ```
    maindir/category/dir(i)/fileJ1.dat
    maindir/category/dir(i)/fileJ2.dat
    maindir/category/dir(i)/fileJ3.dat
    ...
    ```

    where "maindir" is determined later when you create plotting files by the command line. "category" has been given by kwhistai. In our case "gauss". dir(i) should be a character string that the user must give by the `kwhistdir` call. As indicated by (i), it is a string to reflect index i. In our case, Gaussian average changes with i. File names fileJ1.dat etc will be automatically determined inside to reflect index j.

14. Finally we call kwhists for statistics and kwhistp for printing (in our case, to write binary file).

15. The optional routines may be called any place after initialization and before printing call. Optional calls are mandatory for a lot of graphs; without them no appropriate classification is possible.

16. If you had no loop correspoinding to "i" in the example, we need not call kwhistdir. The plotting fillies will be stored in the "category" directory.

17. Program run.
    ```
    $ make -f test1.mk
    $ ./test1PCLinuxIFC
    $ make -f bin2asci.mk
    $ setenv HISTFILE0 mytest1.chist (this is cshell case)
    $ ./bin2asciiPCLinuxIFC > mytest1.achist
    $ splitHisto.sh mytest1 mytest1.achist
    ```

18. Go to mytest1, and "gnuplot plot.gp" will show you the the power spectra.

19. We see the overflow and underflow values are included in the lowest and highest bins. There is no title, key, etc due to the fact that we used the minimum interface.

20. Go to gauss. There will be av1 and av2 directories.

21. Go to av1.
    `$ gnuplot plot.gp`
    This will show title, key, x, y labels.

22. However, we cannot control the graphwith this usage. To be able to control the display, we must load plot.gp within gnuplot.

    ```
    $ gnuplot
    gnuplot> load "plot.gp"
    gnuplot> set xr[-2:6]
    gnuplot> set log y
    gnuplot> rep
    gnuplot> a=1;s=1;m=2
    gnuplot> f(x) = a/sqrt(2*pi)/s *exp(-((x-m)/s)**2/2)
    gnuplot> ! ls
    1.dat   2.dat   3.dat   4.dat   5.dat   plot.gp var.gp
    gnuplot> fit f(x) "5.dat" via a,s
    gnuplot> set yr[1.e-5:10]
    gnuplot> rep f(x)  lw 3
    ```

23. More details will be controlled by editing plot.gp. For examle, we change the last part as
    `call "var.gp" "$1" "$2" "w p"`

    ```
    gnuplot> load "plot.gp"
    gnuplot> rep f(x)  lw 3
    ```

    When plot.gp is changed, it is not reflected by rep; we must load it again.

24. Direct ascii output. For small applications, we may make an ascii .achist file directly. For example, in test1.f, we may make the following changes

    ```
    call kwhistso(2) --> call kwhistso(1)
    fno= 3  -->  fno= -3
    open(fno ..)      --->  comment out
    ```

    and then, compile it.
    $ test1PCLinuxIFC > test1.achist
    will make an ascii file directory. splitHisto.sh will make plotting file as before.

10.1 2D and 3D histograms and summary

Test programs for 2D and 3D are test2.f and test3.f. The difference from the 1D case will be understood by Table 2

We summarize comments to Tables 2 and 3

kwhistso specify output method.
asciiorbin: If 1, ascii output. If 2, binary output

kwhisti make an instance of histogram.
ha: histogram area. include "Z90histi.h" and type(histogrami) ha (i=1,2,3).
min: value of lowest bin. (always not in log)
bin: bin width or highest value of bin. If log, it is for 1 $\log 10$ decade.
nbin: number of bins.
bitptn: 5bit pattern. bit pos 1-log. 2-min is middle of lowest bin. 3-inc. unf. 4-inc. ovf. 5-bin is the highest value of the bin. For 2D (3D), similar ones for Y (and Z).

kwhistc clear histogram area

kwhist digitize variables and count them.
x,(y,(z)): variables.
w: weight. Normally 1.0

kwhists statistical calculation and normalization.
norm: if 0.0, area normalization. if not 0.0, the value is used for normalization.

kwhistp Print a histogram (ascii) or write a histogram into binary file.
ascii or binary is determined by kwhistso.
fn: file number. if ascii output and < 0, "stdout" is used.

kwhistai give additional infromation.
title: string. used as title of the plot.
categ: category of the histogram. short string to symbolize it. say, 'ret' implying t (arrival time) distribution at different 'r' (distance) and 'e' (Energy). Used to make a directory.
dNunit: short string to show the unit of dN.
logv: if .true., vertical scale is displayed in log.
pw: if non zero, x^{pw} is multiplied to the vertical scale at display time.
axis-label: short string for x,(y,(z)) axis.
axis-unit: short string for unit of x,(y,(z)) axis.

kwhistdir specify directory where plotting files to be stored. dir: string. if white space is included, eliminated inside. Suppose the following loop structure

```
do i = 1,...
     do j = 1..
           ...
           call kwhisti(ha(i,j)...)
     enddo
enddo
```

(For example, i is for different depths, and j for particle type). We will have a lot of plotting files. So they must be organized in a different directory. When making plotting files, they will be stored as
For 1D, maindir/categ/dir(i)/file(j).dat

For 2D, maindir/categ/dir(i)/dir(j)/fileXn.dat
For 3D, maindir/categ/dir(i)/dir(j)/dirXn/fileYn.dat
where
maindir: is the directory specified when the command is executed to produce plotting files.
dir(i) is a short string which reflects index i,
dir(j) the same for index j.
fileXn means a file name composed of axis-label. n is for $n-$th such a file.
dirXn is a directory made by a similar fashion.
dir(i) and dir(j) must be supplied by the user. (for 3D case, string corresponding to dir(i)/dir(j) must be supplied).

If loop j dose not exist, we need not give "dir(i)/". More loops are not supported.

kwhistid Gives key (in terms of gnuplot).
id: short string to identify the plot. When multiple histograms are plotted on the same graph, this is used to identify the curve (or dot).

kwhistsetp2 Thinning the plot. (No 1D entry). Suppose a XY 2D histogram. If we plot Y distribution for all bin of X, we may have too many graphs.
step: if 2, X bin is employed with step 2. For 3D, step for Y can be given. If they are 1, equivalent to default.

kwhistev Mandatory. when you generate a number of events.
ev: event number.

kwhistd deallocate histogram area.

kwhista add two histograms of the same structure.

kwhistr read binary histogram file. Each histogram has #hist$N$ (N=1,2,or 3) depending on 1D, 2D or 3D. The user must read this 1 record and judge the histogram type, and then must call this.
fn: file number.
con: condition code. if 0, ok.

kwhistw binary write of histogram data. The user need not use this directory, but may use kwhistp.

Table 2: Summary of interface

function	1D	2D	3D
ascii/bin	kwhistso(asciiorbin)		
initialize	kwhisti(ha, min, bin, nbin, bitptn)	kwhisti2(ha, min, bin, nbin, bitptn, min, bin, nbin, bitptn)	kwhisti3(ha, min, bin, nbin, bitptn, min, bin, nbin, bitptn, min, bin, nbin, bitptn)
clear	kwhistc(ha)	kwhistc2(ha)	kwhistc3(ha)
count	kwhist(ha, x, w)	kwhist2(ha, x, y, w)	kwhist3(ha, x, y, z, w)
normalize	kwhists(ha, norm)	kwhists2(ha, norm)	kwhists3(ha, norm)
print/write	kwhistp(ha, fn)	kwhistp2(ha, fn)	kwhistp3(ha, fn)
Below: optional. needed complex output			
add info.	kwhistai(ha, title, categ, dNunit, logv, pw, axis-label, axis-unit)	kwhistai2(ha, title, categ, dNunit, logv, pw, axis-label, axis-unit, axis-label, axis-unit)	kwhistai3(ha, title, categ, dNunit, logv, pw, axis-label, axis-unit, axis-label, axis-unit, axis-label, axis-unit)
directory	kwhistdir(ha, dir)	kwhistdir2(ha, dir)	kwhistdir3(ha, dir)
id(key)	kwhistid(ha, id)	kwhistid2(ha, id)	kwhistid3(ha, id)
thinning		kwhiststep2(ha, step)	kwhiststep3(ha, step, step)
event #	kwhistev(ha, no)	whistev2(ha, no)	kwhistev3(ha, no)

Table 3: Other routines for special jobs

function	1D	2D	3D
free array	kwhistd(ha)	kwhistd2(ha)	kwhistd3(ha)
add 2 histo	kwhista(ha1,ha2,ha)	kwhista2(ha1,ha2,ha)	kwhista3(ha1,ha2,ha)
read from file	kwhistr(ha, fn, con)	kwhistr2(ha, fn, con)	kwhistr2(ha, fn, con)
write to file	kwhistw(ha, fn)	kwhistw2(ha, fn)	kwhistw3(ha, fn)