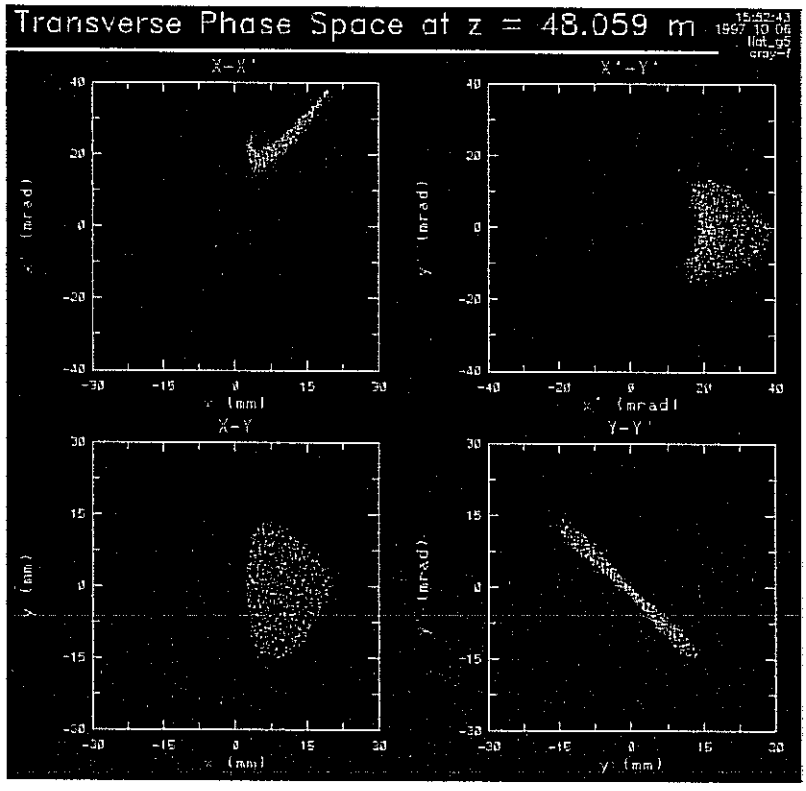
HIFAR 496
LBNL-42173

The HIBEAM Manual



Version 1.1 - February 2000

William M. Fawley

CHAPTER 1

Introduction, History, and Physics Model

Sec. 1-1 *Introduction*

HIBEAM is a 2 1/2D particle-in-cell (PIC) simulation code developed in the late 1990's in the Heavy-Ion Fusion research program at Lawrence Berkeley National Laboratory. The major purpose of HIBEAM is to simulate the transverse (*i.e.*, X-Y) dynamics of a space-charge-dominated, non-relativistic heavy-ion beam being transported in a static accelerator focusing lattice. HIBEAM has been used to study beam combining systems, effective dynamic apertures in electrostatic quadrupole lattices, and emittance growth due to transverse misalignments. At present, HIBEAM runs on the CRAY vector machines (C90 and J90's) at NERSC, although it would be relatively simple to port the code to UNIX workstations so long as IMSL math routines were available.

Sec. 1-2 *History, Porting to Fortran90, and Improvements*

The original author of HIBEAM was Kyoung Hahn. The code, as he left it in early 1996, was an updated Fortran77 version of the venerable SHIFT-XY code written by I. Haber (NRL), targeted toward the Cray C90 machine at NERSC. However, the original HIBEAM had a number of both minor and more serious flaws. Many simulation pararameters (*e.g.* macroparticle number, grid cell number, beam combiner dimensions) were "hard-wired". Hence, to do something as simple as increase the macroparticle number from 4096 to 8192 required one to recompile and relink the entire code. The Hahn version also had the non-optimal feature of only allowing one set of input

file names and, in similar fashion, always assigning the same names to the output diagnostic text and graphics files. Therefore, one always risked inadvertently having a new simulation run write over and destroy some important previous run. Graphics were generated both by the main code and by a post-processor code, the latter doing rather simple particle scatter and history plots. The Fast Fourier Transform (FFT) routines originally used in HIBEAM were apparently adapted from a NIST modification of quite general (*i.e.* not targeted toward a particular computing platform) NCAR routines written in the early 1980's and thus were not optimised for vector platforms such as the Cray-J90's. Finally, the code was not particularly robust and could suffer unpredictable "hard" crashes (*i.e.* core dumps), for example if small changes were made to grid sizes.

Due to all these problems, I decided to do a major rewrite to the code with the following goals:

- Use modern FORTRAN90 constructs, including the "module" structure for global variables and "type" definitions for structures such as multiple beamlets, lattice arrays, *etc.*
- Use FORTRAN90 memory management/allocation constructs such as allocatable pointer arrays and stack-based "automatic" arrays
- Improve the robustness of particle and field subroutines to eliminate array bound-caused crashes when particle coordinates exceeded the nominal physical boundaries
- Replace the NIST/NCAR FFT routines with CRAY-supplied 2D FFT library routines that are optimized for multiprocessor, vector architectures such as the C90 and J90
- Replace most hard-wired simulation parameters by default values that can be easily over-ridden by user-specified input
- Eliminate the need for a graphics post-processor code but give the user the ability to write particle and history dump files for later post-processing if wanted
- Allow the user to specify names for input and output files to make different runs easily identifiable and unique
- Put together a "MAD"-style lattice input parser (see Sec. 3-2 "Lattice Input File Specifications" on page 18) to reduce the pain and error-proneness of having to specify individual lattice elelents in multiple, huge input arrays
- Collect all graphics calls into the main program (*i.e.*, eliminate post-processor program) and use a "standard" graphics library whose core routines are built upon GKS and NCAR

Furthermore, modern code management practices were implemented for HIBEAM. All of the approximately 15 HIBEAM Fortran source files were put under the *Source Code Control System* (SCCS) which is a standard UNIX utility. Among other features, SCCS allows one to a) "check" in and out individual source files for user editing and annotate changes made to each version of a given file b) easily "back up" the entire source code to a version corresponding to a given past date (useful, for example, if necessary to benchmark a recent version against a much older version). Moreover, all HIBEAM source and "make" files were put in a single directory under the *Andrew*

File System (AFS), now supported at NERSC on each of the CRAY J90 PVP platforms. Thus, it no longer is necessary to FTP the most modern version of the source to each individual CRAY following each modification.

With these changes, HIBEAM now is quite robust and core dumps essentially only happen when new bugs are introduced due to source modifications. The code runs significantly faster due to better vectorization and also due to increased parallelism, although this latter area was not specifically addressed and could probably be much improved. The source management under SCCS and AFS makes version control and updating straightforward and is recommended to anyone with source codes exceeding a couple of thousand lines. There still remains some work to be done concerning elimination of "hard-wired" parameters, especially with regards to beam combiner specification. Likewise, as new transport problems are studied, undoubtedly one will want other diagnostic plots and improvements to the physics models such as the addition of magnetic focusing, including both quadrupoles and solenoids.

Sec. 1-3 *Physics Model*

The overall sequence of events in HIBEAM is rather standard for a PIC code. First, all beam, grid, and lattice quantities are read in through a series of FORTRAN namelists contained in several input files (see CHAPTER 3 "HIBEAM Input/Output File Specifications" beginning on page 13 for an explanation of the individual files). Then the grids, graphics, and beam macroparticle distributions are initialized. Then an outer loop over lattice element is entered. The properties of the given element are brought in, including calculating the appropriate capacity matrix if necessary and the z-step size is adjusted so that the element spans an integer number of z-steps. Then an inner loop over the given lattice element is begun. Each loop step involves a particle move and field and history diagnostic calculation. If wanted, diagnostic "snapshots" such as particle scatter plots or field contour maps are written to the output graphics metafile. Particle dump files may also be written at user-chosen z locations. At the end of the final lattice element, diagnostic history plots are produced and the program exits.

Staggered Leap Frog Scheme

The particle move and acceleration sequence is written as a staggered leap-frog to maximize diagnostic symmetry between x and v. In short, at the beginning of z-step N, both the particle position and velocity are known. Then using v_N, the position x_N is advanced to $x_{N+1/2}$. At this point, the charge density, potentials, and electric fields are calculated. Then the velocity is advanced a full z step from N to N+1 using $E_{N+1/2}$. Finally, the position is advanced from $x_{N+1/2}$ to x_{N+1} using v_{N+1}, thus leaving the particles and velocities known at the same point in z again. Since the z step can

change from one optics element to the next (this is done in order that elements are entered/exited at the exact beginning/end of a z step), this procedure is not equivalent to a pure leap-frog, even for a constant v_z. It also is "anti-symmetric" to the modified leap-frog scheme used in WARPXY where the velocity is advanced a half-step, and then the position. For appropriately small z-steps, the net differences between the two move algorithms are likely to be quite small. If v_z varies from particle to particle, the equivalent time step, $dt = dz / v_z$, is computed and used for each individual particle. In this case, the resulting algorithm is no longer exactly symplectic.

Field Solver

At present, HIBEAM models only forces due to electrostatic fields. Thus, all fields are due to a gradient of a potential and are curl-free. If one desires to add magnetic fields such as solenoids or quads, something equivalent to a "Boris" mover should be coded. Likewise, additional coding will be necessary if acceleration gaps are desired. HIBEAM, like its predecessor SHIFT-XY, uses a capacity matrix method together with a periodic Fast Fourier Transform (FFT) to solve for the electrostatic potential. Consequently, the field is presumed periodic in both x and y and if the beam and focusing electrodes are not completely surrounded by a conducting surface (*i.e.* open boundary conditions), the user must be careful to deal with (or prevent) non-real forces from adjoining periodic regions due to any net charge in the central simulation region.

Determining the x and y components of the total electric field requires the following steps:

1. Determine the capacity matrix C for the given conductor/electrode configuration. Since a given geometrical configuration with N capacity "nodes" requires N FFT's followed by inversion of a NxN matrix with N normally ranging from 128 to 256, it is computationally expensive to determine C and the CPU charges would be enormous if the matrix had to be computed anew for every single z-step in the simulation. Fortunately, the node configuration does not change within the z-span of a given lattice element (which normally contains many z steps) and, moreover, most focusing lattices contain multiple elements which are identical save for a change of voltage. Thus, C only needs to be computed once for a given conguration. To take advantage of this, HIBEAM can now internally store (via allocated pointer arrays) up to 8 different capacity matrices, with N being a free parameter for each. The original Hahn version could write a single matrix to disc to allow re-use by later runs; this feature was disabled (mainly because it requires the grid geometry remain the same from the old to the new run) but could easily be revived.
2. Assign the beam space charge to the grid using area weighting. Particles whose trajectories have exceeded the grid boundaries and/or have struck electrode surfaces are marked as "lost" and their charge is not weighted to the grid.
3. Determine the electrostatic potential on the grid using an FFT with periodic boundary conditions in both x and y.

4. Interpolate the potential to the locations of the capacity matrix nodes. Determine the image charges at the nodes from multiplying the capacity matrix C times the difference between the node external voltage and the computed potential from the beam charge. Weight the determined image charge to the grid, adding it to the beam charge density.
5. Compute the electrostatic potential a second time with the periodic FFT, now using the "total" charge density including both beam and image charges.
6. Compute E_x and E_y from a centered, two-point finite differencing of the potential.

There are a number of input variables and switches that allow one to modify the field solution. One may change the number of grid cells through **nx** and **ny**, and the cell size through the simulation transverse boundary limits of **xdsize** and **ydsize**. If **ibcf=0** rather than the default of 1, voltages on the conductors and electrodes are set to zero and hard-wired external focusing is applied (but image charges are still calculated via the capacity matrix). If **l_use_capmatrix** is input as false, **ibcf** is set to 0, and neither the capacity matrix nor the image charges are calculated. The number of capacity nodes for a given focusing element is set via **n_cap_nodes** in the lattice input file; if not set, the default value is the parameter **nc_default**=144 in the module HIB_CAP in *hib_cap_mod.f90*. The index of the capacity matrix for a particular element is set by **i_cap_pointer** in the lattice input file.

In the original Hahn version of HIBEAM, a set of neutralizing "ghost" charges were put at the four corners of the simulation grid to balance the non-neutral beam charge. Due to unease with the artifical solution, this feature was eliminated (it should have no effect whether present or not for a round pipe drift section but will have an effect in the case of an electrostatic quadrupole element). Due to requests from a particular HIFAR individual, the option to including neutralizing "ghost" charges was re-implemented into HIBEAM via the **l_neut_charge** input switch. When true, neutralizing charges are put uniformly along all four border edges (as opposed to the four corners only).

In the past it was apparent that unwanted effects could be produced by the combination of periodic boundary conditions and a "squirrel cage" element whose average voltage was biased away from zero (as is true for the MBE-4 combiner experiment). In order to eliminate erroneous low order multipoles, the average voltage of the cage wires was subtracted from the individual wires. Once this was done, the interior fields of the squirrel cages in the absence of beam became extremely close to the wanted, analytical values. All this refers to code behavior before the option of neutralizing charges was re-implemented. It is not clear if such neutralizing charges would have eliminated the problem of a non-zero average cage voltage.

Due to speed considerations in the periodic FFT employed for the potential solution, HIBEAM requires that the number of grid cells in the *x* and *y* direction, **nx** and **ny** respectively, be powers of two (e.g. 16, 32, 64 ...) or three times a power of two (e.g. 24, 48, 96, 192, ...). Normally, **nx=ny** and the grid sizes are the same in the *x* and *y* directions (**xdsize** and **ydsize**). In "theory", there

should be no problem if **nx** differs from **ny** and/or **xdsize** differs from **ydsize**, but no exhaustive check has been done in this regard.

Numerical Emittance Growth Study

A small series of tests have been done to study numerical emittance growth as a function of particle number and grid cell size. For a given set of grid parameters, macroparticle number, and time step, a non-neutral beam propagating through a FODO lattice will have a emittance growth term that is approximately linear with z. As the macroparticle number is increased, the numerical emittance growth rate decreases inversely as $(NP)^{-1}$. For example, with 8192 particles in a 256 by 256 grid with a 72-degree undepressed and six-degree depressed phase advance per lattice period, the emittance has a normalized growth rate of 7.6×10^{-4} per lattice period. As the grid cell size is decreased by increasing **nx** and **ny**, the growth rate will increase as $(\mathbf{nx} * \mathbf{ny})^{+1/2}$. Most runs employed 4096-8192 macropartices and a 256 by 256 grid. Most of the field quantiities are stored in pointer or allocated arrays and it is concievable that if one uses too many grids cells that one might overflow the stack size. The macroparticle number is limited to 16***ndim** with **ndim** currently 1024; this value can be increased by editing the module HIB_PAR in the file *hib_par_mod.f90*. There are unlikely to be any problems until the product 16***ndim** approaches 100,000 and/or the product **nx***ny exceeds 1,000,000.

CHAPTER 2

Obtaining and Running the HIBEAM code

As previously mentioned, HIBEAM runs on the NERSC C90 and J90 vector machines. The executable file is normally named **xhibe**, although this can be easily changed either directly through the UNIX **mv** command or in the Makefile. A user can either run the code in "interactive" mode from a terminal window or, alternatively, submit a job to the NERSC batch system. We first discuss how to obtain HIBEAM and then how to make a run.

Sec. 2-1 *Obtaining the HIBEAM executable* **xhibe**

For the immediate future, a copy of **xhibe** will be kept in W. Fawley's HFS public space under NERSC user number **u1532** in the directory **pub**. As of early 2000, **u1532** HPSS files belong to in the **/nserc/agp** HPSS directory (even though **u1532** is in the **ajb** repository). Presuming you are on the killeen interactive J90 machine, you can obtain HIBEAM by typing:

```
killeen[2] hsi
*------------------------------------------------------*
*          NERSC HPSS ARCHIVE SYSTEM                   *
*------------------------------------------------------*
V1.5  Username: u1111  UID: 1111
? cd /nersc/agp/u1532/pub
? ls
```

/nersc/agp/u1532/pub:
S5new.wire inmbe4_S5g mbe4_S5mad.lat xhibe.c90.dec97
xhibe.j90.aug98 xhibe.j90.dec97 xhibe.j90.may99
? **get xhibe.j90.dec97**
Transfer started for [xhibe.j90.dec97]
get xhibe.j90.dec97:/nersc/agp/u1532/pub/xhibe.j90.dec97 (97/12/12 00:00:00 5142960 bytes, 3848.3 KBS)
? **quit**
killeen[3] **ls -l xh***
*-rwx------ 1 u1111 ajb 5142960 May 20 10:23 xhibe.j90.dec97**
killeen[4] **mv xhibe.j90.dec97 xhibe**
killeen[5]

The **/nersc/agp/u1532/pub/** HPSS space should have executables for both the C90 and J90 machines with the date of each version appended to the name. There should also be archives (written and readable with the UNIX utility "ar") of input decks and a README file.

Sec. 2-2 *Running HIBEAM directly from a terminal window*

For short runs (*e.g.* less than a couple of CPU minutes), it is generally most efficient to run HIBEAM directly from a "shell" window. Examples include Xterm windows (as would be produced by logging in from a UNIX workstation running X-Windows or a Macintosh running MacX),Telnet windows (normally VT100 or VT220 emulations) from a remote login and/or any other window that supports a "Command Line Interface" (CLI).

The standard execution line from a shell is:

xhibe { r=*run-name* l=*lattice-file* w=*wire-file* i=*input-file* }

where each of the options within the bracket is optional. Table 2-1 on page 9 gives a summary of these options. The string corresponding to *run-name* is used to identify the particular HIBEAM run and forms the prefix for the output graphics which (which is an NCAR CGM metafile). Note that if neither the run-name nor input file are specified, the default name for the main input file will be ***intest***. If the run-name is specified (which is the usual case) but not the main input file name, then HIBEAM will look for an input file named "in*[run-name]*" where *[run-name]* stands for a 14 character or fewer ASCII string. One can mix on the same execution line the "=' notation (*e.g.* **r=test7**) (which was the standard form for LRLTRAN utilities) with the "-" notation (e.g, **-i intest8**) (which is the standard UNIX notation for options on execute lines). Permitted characters for these strings include alphabetical letters (both upper and lower case), numbers, and

TABLE 2-1 Optional arguments in the xhibe execute line

Option	Explanation	Default value	Maximum # characters
`r=` or `-r`	Run name to be used as a suffix for the input file (if not specified) and as a prefix for graphics and dump files	`test`	16
`l=` or `-l`	Full lattice input file name	`inlattice`	24
`w=` or `-w`	Full "wire" input file name	`iwire`	24
`i=` or `-i`	Full "main" input file name	`in[`*`run-name`*`]`	24

the underscore symbol ("_"). As is true for nearly all UNIX file systems, please do not use strings containing abnormal, politically incorrect characters such as slashes ("/" or "\"), periods or commas (":" or ","), brackets or parentheses of any sort, asterixes or similar symbols (e.g. "*", "=", "&", "@", "|", "~", "<", "%", "-", etc) or spaces (" ") . Note that any attempted inclusion of smileys ["(-:"] is officially viewed as subversive and may be grounds for immediate dismissal.

Example 1: `xhibe r=mbe4_A3  l=mbe4_A1.lat`

Here the run-name will be *mbe4_A3* and thus the output CGM graphics file will be named *mbe4_A3.cgm* and any dump files will also have the same prefix (e.g., *mbe4_A3.txt*). HIBEAM will also expect a main input file named *inmbe4_A3* and a lattice input file containing focusing lattice specifications named *mbe4_A1.lat*. If an input wire file is needed (normally only if one or more beams is run through a combiner), HIBEAM will attempt to read the default name of *iwire*.

Example 2: `xhibe r=mbe4_A5 i=inA5new -w std.wire l=mbe4_A1.lat`

As in Example 1, the graphics and dump files will use the run-name of *mbe4_A5* as their prefix (e.g. *mbe4_A5.cgm*). Since the user has specified a separate main input file named *inA5new*, HIBEAM will look for this file (rather than the default of *inmbe4_A5*). It will also look for a lattice input file named *mbe4_A1.lat* and a wire input file named *std.wire*.

Sec. 2-3 *Running HIBEAM under NERSC batch*

For multiple and/or long individual HIBEAM runs, it may be most convenient to run these via the "batch" procedure at NERSC. This gives the additional advantage that batch CPU time, at a given

priority, is charged at a much lower effective rate than is "interactive" time. Moreover, the much lower priority allowed in batch mode permits even greater savings. Disadvantages include the inability (without extensive script coding) to recover gracefully from execution line errors and/or input/lattice/wire file problems. Furthermore, if one sets the priority too low or requests an excessively large run time or memory allocation, it is possible that a batch request might not run until the next millennium. Fortunately, most HIBEAM runs require well under one hour of CPU time on a J90 and can run in 8 MW of memory or less.

Depending upon your personal masochism level and previous experiences, the necessity of dealing with NERSC UNICOS batch scripts may be yet another of life's little joys held in store for you. Apart from various "bookkeeping" requirements and niceties, there are three basic things that a batch script must do. First, by hook or crook, it must gather together the necessary files to run the wanted job(s) in the chosen directory (which in general, is probably not the actual directory where the executable resides but, more likely, some temporary directory that will disappear after the overall batch job is done). Usually, you will either read the files from a long-term storage medium (e.g. CFS) or copy them over from active user disk space (*e.g.*, your "superhome" directory area where you hopefully put a copy of HIBEAM). Second, the script must then have the UNICOS batch system run the job, being sure it previously specified sufficient CPU time and memory so that HIBEAM could finish. Fortunately, this part of the batch script is simple, being a one-liner of the type presented in the previous section (e.g. **`xhibe r=mbe4_A5 ...`**). Third, presuming your job(s) actually ran, the batch script must transfer the generated output files to a safe place such as archival storage (e.g. CFS) or your personal disk space (e.g. superhome space). Otherwise, you may find that you just spent 10 CRU's of your group's repository for nought.

Figure 2-1 on page 11 displays an example of batch script (named *HIBEAM.bat*) to run HIBEAM. The equivalent execute line interactively would have been: **`xhibe r=tbird l=llat_h.lat`**. The total batch job is allowed 3000 seconds of CPU time, asks for 8 MW of memory, and wants to run at a niceness of 14 which will put this job in the "medium" queue. This particular example was targeted (via the **#QSUB -q batchf command**) toward the "franklin" J90 machine at NERSC. The **setenv NCPUS 2** command requests parallel processing by 2 of the 32 CPUs on franklin; HIBEAM seems to run reasonably effectively with 1-4 CPU's. A log file named *logHIB* will contain the various accounting information generated by the "ja" account program. The necessary files to run the job (*xhibe, intbird, llat_h.lat*) are copied over to the user's temporary (only during the duration of the batch job) directory whose environment variable is $TMPDIR. After the job is complete, the NCAR graphics CGM file (named *tbird.cgm*) is copied over to the user's HIBEAM/WORK subdirectory in his/her "home" disk space on the J90 cluster. This is necessary since the copy created in $TMPDIR will disappear once the batch job is complete. **NOTE:** this batch script was written in 1997 and should be modified as per current NERSC practices (which evolved significantly in late 1999)

To submit this job to batch on the "franklin" machine, the user would type:

```
qsub -q batchf HIBEAM.bat
```

To check on the job status, one would type

```
qstat -h franklin  -a | grep uXXXX
```

where **uXXXX** is the user's login name. Depending upon the length of the run and the number of phase-space scatter plots requested, the graphics file sizes typically run in the 100kB to 1.5MB range In general, one obtains reasonably fast turn-around if one chooses a priority that lands the submitted batch job in either the fast or medium queue.

Until recently, there had been an anomaly that occured when running HIBEAM on the J90 "batch" machines. Namely, relative to an identical run on the "killeen" J90 interactive machine, a batch run would suffer an extremely high charge for system time (often comparable to the CPU charges!). The underlying reason for this was that the NCAR-based graphics routines in HIBEAM use more than one font. Apparently, each time a font change was made, a NFS-mount system call was made to read over the font information (typically only ten's of kilobytes!) from disk space on the killeen machine. In November 1997 this anomaly was "fixed" and system charges dropped to about 12% of the CPU charges.

```
#QSUB -s /bin/csh
#QSUB -q batchf
#QSUB -lM 8MW
#QSUB -lT 3000
#QSUB -ln 14
#QSUB -eo
#QSUB -o logHIB
set echo
setenv TERM vt100
alias cd cd
setenv NCPUS 2
set timestamp
ja
cd $HOME/HIBEAM/WORK
set name = tbird
set latf = llat_h.lat
cp xhibe in$name $latf $TMPDIR
cd $TMPDIR
xhibe r=$name l=$latf
cd $HOME/HIBEAM/WORK
cp $TMPDIR/*.cgm .
ja -cst
exit
```

FIGURE 2-1. Sample UNICOS batch script for HIBEAM

CHAPTER 3

HIBEAM Input/Output File Specifications

Sec. 3-1 *Main Input File Structure and Variables*

HIBEAM uses a number of separate ASCII input files and, within each input file, one or more namelists to specify the needed beam, lattice, and computational parameters. Output specifiers are also in the "main" input file but focusing lattice properties (in general) are given in a separate lattice input file whose requirements will be covered in the next section of this chapter. Yet another file, the wire input file, is required to specify the positions and voltages of a wire "squirrel-cage" focusing element if present (as is true in the MBE-4 combiner experiment). We now discuss each of these files in turn, beginning with the main input file.

Overall Structure of HIBEAM Main Input File

The main input file contains "header" information at its beginning, followed by two different input namelists: (1) ***hinit*** to specify various beam and grid properties and (2) ***ztime*** to specify z-locations for diagnostic output. The header is normally 1-10 lines of text in 80-column (*i.e.* A80) format. The last line of the header *must* be terminated by a $ (*i.e.* dollar) symbol which signals the input parser routine to then begin reading the namelists. If the header is not terminated by a $, normally HIBEAM will exit with an error message to the user that no $ sign was found (and no namelists either since the parser will advance past them to the end of the file in its mindless quest for a $). Apart from a $ sign, any normal ASCII character may be used anywhere in the header lines. Lines longer than 80 characters will be truncated. The input file header has two purposes. First, it provides

a simple means to identify various characteristics of the particular input file, *e.g.* what experiment configuration was being modeled, special beam characteristics, *etc.*. A well written header might remind the user what it was the made this file so incredibly special eight months ago. Second, since the header lines are echoed to the first page of the output graphics metafile, they also provide a means for identifying the graphics file beyond the simple run-name. Consequently, the user is advised to spend an extra 10-20 seconds revising and improving the header lines of a new or revised input file. This is a small price to pay relative to the clock and CPU time of a typical HIBEAM run.

HINIT input namelist

The ***hinit*** namelist specifies the "physical" setup of the problem. All quantities are in MKS units - *e.g.*, all dimensions should be in meters. One exception is that the beam energy (**`emev0`**) is given in mega-electron volts. Most integer quantities start with the letters [i-n] and most logical (boolean) variables with the letter "l". String variables may be written using either single or double quotes. The namelist should begin with a line containing **`&hinit`** and end with a line containing **`/END`**. Table 3-1 gives the various quantities that can be set in the ***hinit*** namelist. The column labelled **Type** indicates Fortran variable type [*i.e.*, real (R), integer (I), logical (L)] together with their default values and a simple explanation.

TABLE 3-1 Variables in *hinit* Namelist

Variable Name	Type	Units	Default Value	Explanation
`a`	R	meters	5.e-3	initial radius of x-envelope
`amu`	R	amu	133.0	Mass of ion
`ap`	R		0.	initial dx/dz of envelope
`b`	R	meters	5.e-3	initial radius of y-envelope
`beam_name`	String		'Beam #1', 'Beam #2', *etc.*	label for individual beamlets
`bp`	R		0.	initial dy/dz of envelope
`charge`	R	electron charge	1.0	Beam ionization state (positive for ions)
`current0`	R	Amps	0.004	Current per beamlet
`current_fac`	R		1.0, 0.0, 0.0, ...	Individual beamlet current relative to current0 value
`dvzth`	R	m/sec	0.	longitudinal velocity spread
`emev0`	R	MeV	0.160	Initial beam energy

TABLE 3-1 Variables in *hinit* Namelist

Variable Name	Type	Units	Default Value	Explanation
emitn0	R	meter-rad	0.	Array containing normalized edge emittance for multiple beamlets
emit0	R	meter-rad	0.	Array containing un-normalized edge emittance for multiple beamlets
enorm0	R	meter-rad	2.0e-8	Normalized "edge" (4 x RMS) emittance of a single beam
fchrome	R		1.0	Chromaticity factor to multiply default energy (emev0)
ibcf	I		0	**0** => quad voltages set to 0 for capacity matrix inversion; hardwired external focusing applied 1 => quad voltages in capacity matrix inversion set to actual voltages
idist	I		2	transverse phase space distribution 1 => K-V 2 => semi-Gaussian (bit-reversed) 3 => semi-Gaussian (random)
irot	I		0	if 1, load beamlet major and minor axes at 45° rotation to X-Y axes
iseed_offset	I		0	random number seed for focusing element offsets
lambda	R	coulomb/m	-1.	Line charge density
l_env	L		.false.	.true. to calculate envelope equation (independent of macroparticles)
l_mad_lat_file	L		.false.	if .true.,read in MAD-style lattice file information
l_neut_charge	L		.true.	switch for neutralizing "ghost" charges along boundary edges
l_offset_one	L		.false.	if .true., load 1 (rather than 4) beams at multibeam creation point for combiner runs
l_use_capmatrix	L		.true.	switch for capacity matrix use
l_varydz	L		.false.	switch to allow vz variations
npart	I		4096	total number of macroparticles
nx	I		256	number of grid cells in x direction
ny	I		256	number of grid cells in y direction

TABLE 3-1 Variables in *hinit* Namelist

Variable Name	Type	Units	Default Value	Explanation
`par_dump_file`	String		'NOT SET'	file name containing particle phase space information from a previous HIBEAM run - needed for restart
`rapert`	R	meters	0.014	default quadrupole aperture
`rquad`	R	meters	0.016	default quad electrode radius
`rwall`	R	meters	0.020	default beam pipe radius (for drift zones)
`single_cage`	String		'NOT SET'	name of single beamlet to create and follow in combiner (other beamlets are neglected)
`vwire_fac`	R		1.0	overall scaling factor for wire cage voltages
`xdsize`	R	meters	0.05	full grid extent in x
`xoffset`	R	meters	0.	initial x offset of centroid
`xpoffset`	R		0.	initial value of centroid dx/dz
`ydsize`	R	meters	0.05	full grid size in y
`yoffset`	R	meters	0.	initial y offset of centroid
`ypoffset`	R		0.	initial value of centroid dy/dz

ZTIME input namelist

The *ztime* namelist is mainly used to control diagnostic locations and settings but also handles the z-limits of the simulation and some additional odds and ends concerning grid rescalings. The various namelist variables are given in Table 3-2; the notation M*(V) under "Default Value" indicates that the array is of length M with all values defaulted to value V. One can control the locations of phaseplots, "big" (single plot/frame) *X-Y* plots, electrostatic potential and charge density contour plots, and the locations at which particle dumps will be written. Many of the diagnostic locations can be set either as distinct locations in z (*e.g.* **`zphaseplot`** = 1.1, 1.5, 2.3, 2.95, ...) or as a periodic interval beginning at z=**`zstart`** (*e.g.*, **`dzphaseplot`** = 0.4 will produce plots at z=0.55, 0.95, 1.35 meters, and so on for **`zstart=0.15`**).

One can control the plotting range in x,y and dx/dz, dy/dz in the transverse phase space plots by defining components of **`phasep_info`** which is a "vector" of length 4 with the FORTRAN90 "type" of *phasep_range* whose codings is given by:

```
TYPE phasep_range
   real, DIMENSION(2) :: zrange, xrange, yrange, xprange, yprange
END TYPE phasep_range

TYPE(phasep_range) :: phasep_info(4)
```

Here **zrange** controls a range in z beginning with **zrange(1)** and ending with **zrange(2)** for which the user-defined limits of **xrange**, *etc.*, will be used in the phase space plots. If more than one element of **phasep_info** is defined, the **zrange** of the 1st element should be less than the of the second which should be less than that of the third, *etc.*; otherwise, peculiar behavior might ensue. Among the various reasons one might want to use this feature are if one wants to generate a movie with the axes labels and scales remaining constant from one frame to the next and/or needing a particular range in a given plot to compare with some other plot.

TABLE 3-2 Variables in *ztime* Namelist

Variable Name	Type	Units	Default Value	Explanation
zstart	R	meters	0.	start point of simulation; superceded by z location of dump restart file if used
zmax	Rl	meters	0.5	end point of simulation
zstep0	R	meters	0.005	nominal z-step size in push
zphaseplot	R	meters	24* (-1.0)	z-locations to plot phase space
dzphaseplot	R	meters	-1.0	periodic phase space plot interval
phasep_info	Struc-ture *see text*	meters	*see text*	structure to specify hard-wired plot ranges for x, y, dx/dz, dy/dz' and z-interval for each set of ranges
zparplot	R	meters	24* (-1.0)	z-locations for full graphics page X-Y particle plots
zfldplot **zplot** *obsolete*	R	meters	24* (-1.0)	z-locations for density and potential contour maps
dzfldplot	R	meters	-1.0	periodic density/potential map interval
dzhist	R	meters	-1.0	interval for storing history array values
l_print_hist	L		.false.	switch to write output text file containing derived beam envelope values
dump_type	String		'BINARY'	format for dump information; allowable = {'BINARY', 'ASCII', 'HDF' }
zpardump	R	meters	8* (-1.0)	z-locations to write particle dumps
l_hdf	L		.false.	switch to force **dump_type**='HDF'

TABLE 3-2 **Variables in *ztime* Namelist**

Variable Name	Type	Units	Default Value	Explanation
`imerge`	I		-1	combiner lattice element index for creation of multiple beams
`rzf_merge`	R		2.0	grid re-zoning factor at **imerge**
`irezone`	I		-1	lattice element index where grid should be re-zoned using **rzf**
`rzf`	R		1.0	grid re-zoning factor at **irezone**
`l_plot_cap_nodes`	L		.true.	if .true., make x-y plot of capacity nodes

Sec. 3-2 *Lattice Input File Specifications*

HIBEAM requires a completely separate input file from the "main" input file to specify the focusing lattice properties. This lattice input file can be written in two different "flavors": either in a stripped-down "MAD" format or in a somewhat painful series of arrays. In general, one will probably put together a lattice file that will be used in many separate runs with different input files (often one will do a series of runs where a quantity such as beam current varies but the lattice properties remain constant). The format of the "MAD"-style lattice input will explained first, followed by a explanation of the "array"-style format that is a holdover from the original (Hahn) version of HIBEAM.

"MAD"-style lattice input format

MAD (Methodical Accelerator Design) is a charged-particle optics code originally written at CERN and extensively used in the high energy storage ring community for beam transport calculations. One of its most attractive features is an input hierarchy for specifying the focusing lattice — in fact, the MAD system is considered so attractive that authors of other codes such as TRANSPORT have created versions that will accept "MAD"-style input. Partly due to this generality and partly due to the desire to see how easily such an input formalism could be written in FORTRAN90 with its TYPE structures, pointers, and dynamic memory allocation, we decided to give HIBEAM a subset of "MAD" input capability. The basic structure of the HIBEAM "MAD" lattice file is first an optional number of comment lines, then a series of focusing element "class" definitions, followed by a "LINE" and "USE" specification that builds up and "instantiates" the actual focusing beam line out of the previously defined classes. A parser routine (contained in the source files *beamline_mod.f90* and *bl_parser.f90*) inspects the lattice input file and generates the beam line.

Each line in the file may be up to 132 ASCII characters, although for readability, a maximum of 80 characters is wisest.

A comment is always preceded by an exclamation point ("!") and may be at the beginning of the line, signifying that the entire line is a comment, or, alternatively, at the end of a line whose beginning contains lattice information such as a class definition. An unlimited number of comments may be in the file and may occur essentially anywhere, the beginning, the middle, the end, *etc*. However, remember, once the parser encounters an "!", all additional information on that line following the exclamation point is treated as a comment and, essentially, is tossed away. Usually, comments should be used to remind the HIBEAM user of the significance of this particular lattice file.

As with MAD, the general class definition format is

```
label:  keyword {, attribute , attribute ... }
```

The label is a simple ASCII name for the class being defined. The label must be terminated by a colon (and, obviously, should not contain a colon internally). If needed, one may use multiple lines for a given class definition by use of the ampersand (&) character at the end of every line but the last (as in Fortran 90 free-form style). The keyword indicates the type of element being defined in the class. At present, the following physical element types are pre-defined in HIBEAM: ***quad, drift, box, hyperb, wire, child***. As one would expect, ***quad*** refers to a quadrupole element; ***drift*** refers to a drift zone with a circular aperture and no external focusing; a ***box*** is a drift zone with a square or rectangular aperture. A ***hyperb*** class is a special quadrupole whose electrode surfaces are hyperboli; at present, this is a "hard-wired" element whose geometry is that of the "Q4" hyperbolic quad in the MBE4 combiner experiment. A ***wire***-type element is a focusing element composed of discrete conducting rods; the positions and voltages of the rods are specified in a "wire" input file (see Sec. 3-3 "Wire Input File Specifications" on page 24).

Once defined, a given class can then be used by subsequent classes. For example. a new class, such as a a syncopated FODo lattice cell, can use previously defined quadrupole and drift space classes. This allows the user to build up in a hierarchical fashion rather complicated beam lines.

A ***child*** class is a class which by default inherits all properties of a *previously* defined parent class, but may then change one or more of the attributes. Any physical focusing element class can be a parent class, including a previously defined child. For example, if one has a particular quad class (e.g. 'QF0') defined with various rod and aperture sizes and a given voltage, and one also needs an essentially identical quad class (e.g. 'QD0') which differs in only one or two properties (*e.g.* a -4.0 KV voltage rather than +4.0 KV voltage), it is most efficient to define QD0 with the child keyword giving its parent as "QF0", e.g.

```
QF0: quad,  length=0.3, voltage=4.0e3, aperture=0.1, &
     r_elem=0.04  !  focusing quad
```

```
QD0:  child, parent='QF0', voltage=-4.0e3    !  defocusing quad
QDhalf:  child, parent='QD0', length=0.15   !  half defocusing quad
```

There are a fairly large number of attributes (see Table 3-1 on page 14; a star indicates a given attribute is relevant to the given keyword). Some are applicable to many keywords, others to just one or two. The logic of the underlying coding is that once a label together with a permitted keyword have been found by the parser at the beginning of an input line, the remainder of the line is then internally restructured as a FORTRAN namelist (to give maximum flexibility with attributes that are needed by only some keywords and not by others) and then re-read by the parser. Most attributes are straight-forward, referring to physical properties of the element such as aperture size, voltage, length, transverse offsets, *etc.* A couple are rather specific, referring to some underlying FORTRAN code (*e.g.*, **`i_cap_pointer`** which indexes the pointer array contains the capacity matrix information for that class).

TABLE 3-3 Defined keywords and attribute variables in HIBEAM MAD-style lattice input

Attribute \ Keyword	Type	Units	drift	box	quad	hyperb	wire
`aperture`	real	meters	✪		✪		
`error_type`	String		✪	✪	✪	✪	
`gradient`	real	volts/meter			✪		
`i_cap_pointer`	integer		✪	✪	✪	✪	
`length` (*alt.:* `l`)	real	meters	✪	✪	✪	✪	✪
`n_cap_nodes`	integer		✪	✪	✪	✪	
`offset_x`	real	meters	✪	✪	✪		
`offset_y`	real	meters	✪	✪	✪		
`parent`	String		✪	✪	✪	✪	✪
`r_elem`	real	meters			✪		
`voltage`	real	Megavolts			✪	✪	
`width_x`	real	meters		✪			
`width_y`	real	meters		✪			

As previously mentioned, the "base" classes provide a foundation, upon which more complicated classes and, eventually, the entire beam line lattice (which itself is a class) is built. To define a multi-element class which is known as a *sub-beam line* in the MAD syntax, the keyword **`LINE=`**

must be used after the colon-terminated class name. Beam lines may contain both normal physical elements (*e.g.*, quads) and previously defined beam lines. All of these should be separated by commas. Chapter 4 of the *User's Reference Manual* for MAD gives a fairly complete summary of how to build up beam lines. As of now, HIBEAM allows one to include parentheses-delimited sub-lines and repetition.

```
! llat_j.lat ::  input lattice file for long transport test
!                x offsets set to 0.4mm ;  rod radius=4.0 cm
!              sigma0=60 degrees

 Q0:  quad,  l=0.2, aperture=0.04, i_cap_pointer=1, &
       r_elem=0.036,  offset_x=0.4e-3, offset_y=0.4e-3

 QF: child, parent='Q0', voltage=46.8e-3  ! focusing MBE-4 quad
 QD: child, parent='QF', voltage=-46.8e-3  ! de-focus MBE4 quad
 HQD: child, parent='QD', l=0.1        !  half de-focusing quad

 DO:  drift, l=0.2,  aperture=0.04, n_cap_nodes=128, &
       i_cap_pointer=2,  r_elem=0.04    !  drift in MBE4 cell

 BASE: LINE=  DO, QF, DO, QD

 TENB: LINE=  10*BASE

 BLINE: LINE= HQD, 15*TENB

 USE, BLINE
 END
```

FIGURE 3-1. Lattice input file for FODO array begun with half-length defocusing quad

Figure 3-1 on page 21 gives an example of a relatively simple FODO lattice definition. After some comment lines, the basic quad building block "Q0" is defined with its length, aperture, rod size, and offset errors. Then, its children, QF and QD, are defined with their appropriate voltages. Then a "grandchild", HQD, is defined whose sole difference from its immediate parent QD is that it is a half-length element (one needs this for matching a beam with da/dz=db/dz=0 at the middle of a full quad). Finally, the final elementary building block of a drift cell is defined.From these elementary elements, a base FODO cell (i.e. a sub-beam line), named "BASE", is made from the full quads and drifts. Then a 10-cell block (also a sub-beam line) named TENB is made from repeating the BASE cell ten times, and finally, the final beam line ("BLINE") is built from repeating the ten-cell block 15 times, preceded by the half-length defocusing quad.

Note: The actual FORTRAN coding for the repetition function is very simple and works by building up the beam line string via literally repeating the ASCII strings for the sub-elements. Therefore, in order not to exceed the maximum string length of 512 characters, it is necessary in a long lattice with (*e.g.* 20 or more elements) to use sub-beam lines to build up the final beam line. In the present

example, the string representation of "BLINE" is "HQD, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB, TENB". TENB itself is "BASE, BASE, BASE, BASE, BASE, BASE, BASE, BASE, BASE, BASE". It should be rather obvious that if we had not defined TENB but had rather defined BLINE as "HQD, 150*BASE", the expanded string representation of BLINE would exceeded the 512 character limit. It should also be apparent that it is not wise to pick a long name for a sub-beam line that will be repeated a number of times (*e.g.* "OLD_MBE4_FoDO_CELL" would not be a good choice...). The coding for the MAD lattice routines is certainly not bullet-proof and one may encounter abnormal behavior --- *caveat emptor.*

Once the final beam line is defined, it becomes instantiated by the "USE" command (as is true in the MAD program itself). One should then terminate the lattice file with an "END" statement (although, as presently written, once a beam line becomes instantiated by a "USE" statement, no additional lattice input file statements are actually parsed or used).,

```
! input lattice file for MBE4 combiner expt:  mbe4_85mad.lat ;
!  P. Seidl voltages 4/7/97 ;  adjusted L45B drift to 8.36 cm 4/25/97

 QF:  quad, voltage=7.2e-3, l=0.107283, aperture=0.02703, &
          r_elem=0.02471,  i_cap_pointer=3     ! focusing MBE4 quad
 DO:  drift, l=0.178804,  aperture=0.03    ! long drift in MBE4 FoDO cell
 Do:  drift, l=0.06437,   aperture=0.03    ! short drift in MBE4 FoDO cel

 LS1: drift, l=0.015, aperture=0.015
 Q1:  quad,  l=0.05,  aperture=0.011, r_elem=0.01257, voltage=-2.516e-3
 L12: drift, l=0.12,  aperture=0.015
 Q2:  quad,  l=0.10,  aperture=0.012, r_elem=0.01371, &
                      voltage=+2.3737e-3, i_cap_pointer=2
 L23: drift, l=0.15,  aperture=0.015
 Q3:  child, parent='Q2', voltage=-2.4336e-3
 L34: drift, l=0.11,  aperture=0.015

 Q4:  hyperb, l=0.10, aperture=0.011, r_elem=0.01257, voltage=+2.0109e-3
 L45A: drift, l=0.05,    aperture=0.015
 L45B: drift, l=0.08358, aperture=0.038
 Q5:  wire, l=0.0803,  aperture=0.035

 L56:  drift, l=0.06375, aperture=0.03
 Q6:  child, parent='QF',  voltage=-5.988e-3
 Q7:  child, parent='QF',  voltage=+8.681e-3
 Q8:  child, parent='QF',  voltage=-8.013e-3
 Q9:  child, parent='QF',  voltage=+7.104e-3

 MBE4C: LINE= LS1, Q1, L12, Q2, L23, Q3, L34, Q4, L45A, L45B, Q5, &
              L56, Q6, Do, Q7, DO, Q8, Do, Q9
 USE, MBE4C
 END
```

FIGURE 3-2. A lattice file for the MBE4 combiner experiment -- parameters are current as of November 1997.

A more complicated example, Figure 3-2, shows the lattice input file for the MBE-4 combiner experiment. Here, nearly every quad and drift zone is unique, differing in length or voltage or aperture, and must have its own class defined. Hence, there is no need to define sub-beam lines in order to exploit repetition. The Q5 wire zone in reality has only one parameter --- its length --- that can be set in the lattice file. The rest of its parameters are set in the wire input file. Similarly, the Q4 hyperbolic quad actually only has three parameters --- length, voltage, and offset error --- which can be set, since the aperture and individual electrode geometry is presently "hard-wired".

One last example (Figure 3-3 on page 23) is given to show the role of parentheses in a MAD-style input lattice file. Here rather than define a separate sub-beam line for the FODo cell, multiple repetitions of the cell are achieved by enclosing its defining elements in parentheses and then using an integer repetition factor in front of the parentheses. Parentheses may be nested essentially unlimited levels deep. In practice, however, if one needs to use more than three levels, it is probably best (for the sake of clarity) to define some sub-beam lines to hold one or more of the deeper levels. For "normal" HIF beam lines, one will rarely need to use parentheses.

```
QF:  quad, voltage=7.2e-3, l=0.1, aperture=0.027, r_elem=0.024
QD: child, parent=-QF-, voltage=-7.2e-3
DO:  drift, l=0.178804,  aperture=0.03
Do:  drift, l=0.06437,   aperture=0.03

Diag1: drift, l=0.05
Aper1: box, aperture=0.01, l=0.001

DLINE: LINE= 3*(QF, D0, QD, Do), (Diag1, Aper1), 2*(QF, D0, QD, Do)

USE, DLINE
END
```

FIGURE 3-3. Example showing use of parentheses-delimited elements

"Old" Lattice Input File Form

The original "Hahn" version of HIBEAM used a namelist (named ***lattice***) to input large arrays to specify the beam line focusing lattice. Essentially, there are eight arrays: (1) **ielement**, an integer array that specifies the type of element (see Table 3-4 for a key); (2) **zl**, a real array that specifies the element length in meters; (3) **yapt**, a real array specifying the clear aperture radius (in meters); (4) **qr**, a real array specifying the rod radius (in meters) of each focusing element; (5) **vq**, the element voltage (in megavolts) with positive values corresponding to focusing in the x-plane; (6) **vq6**, the voltage at r=yapt corresponding to the dodecapole component (7) **qofx** and **qofy**, the exact x and y offsets of each element from the nominal center of the beam line. One should

note that the preferred way to input lattice quantities is in "MAD" style format and that a number of element types (*e.g.* dipoles) that existed in HIBEAM prior to summer 1996 have been disabled.

TABLE 3-4 Correspondence between "ielement" and physical type of element in the "old" form of the lattice input file

ielement	description	ielement	description
0	circular aperture drift zone	1	accelerating gap
2	quadrupole - no dodecapole	3	quadrupole - dodecapole permitted
4	rectangular pipe (i.e. a box)	6, 7, 8	obsolete, non-functional types
97	wire cage	98	MBE4 combiner hyperbolic quad

Sec. 3-3 *Wire Input File Specifications*

The ***wire*** input file details the positions and voltages of the individual rods comprising a "squirrel cage" focusing element, in addition to some geometry information concerning the alignment of the cage. While to date this type of focusing element has been defined only for the MBE4 combiner experiment, in principle the input file structure is general enough that any element composed of discrete rods should be representable by a wire input file. Since all focusing electrodes in the capacity matrix field solver are in practice represented by discrete nodes, which are equivalent to wire rods, one could use a wire input file to represent some quad or dipole with a peculiar shape.

TABLE 3-5 Variables in Namelist *in_wire*

Variable Name	Type	Units	Default Value	Explanation
`bias_voltage`	real	volts	0.0	offset bias for wires
`conv_angle`	real	degrees	6.0	convergence half-angle of beamlets
`conv_factor`	real		0.	upstream/downstream convergence ratio of wires
`delx_w`	real	meters	18.704e-3	x-offset of left and right beams at element **imerge**
`dely_w`	real	meters	26.198e-3	y-offset of top and bottom beams at element **imerge**
`node_per_wire`	integer		2	number of nodes for each wire
`nw`	integer		72	number of rods in cage

TABLE 3-5 Variables in Namelist *in_wire*

Variable Name	Type	Units	Default Value	Explanation
pwire	real	volts		array containing individual wire voltages
wire_diameter	real	meters	1.5e-3	transverse separation between nodes of a given wire
wire_length	real	meters	-1 -- *see text.*	length of wire zone *(inoperative)*
xwire	real	meters		array containing x positions of wires
ywire	real	meters		array containing y positions of wires

The wire input file is one large namelist, setting general properties of the wire zone and specific positions and voltages of the individual wires. Most of the input parameters in Table 3-5 are reasonably self-explanatory. Note that one is required to input the total number of wires (**nw**) in the cage and also the number of capacity matrix nodes (**node_per_wire**) to represent each wire --- 2 or 3 is a good value unless the wire diameter is much larger than a field grid zone, in which case one might want to use 6 or more nodes per wire. However, the present coding currently limits **nw** to 256 and **node_per_wire** to 4 by an easily-changed parameter statement in the source file *hib_wire_mod.f90*. Moreover, at present, the actual length of the wire zone is set in the lattice input file so one should not set the **wire_length** variable in the wire zone input file. Similarly, at present, the convergence factor of the cage is set at the MBE4 combiner nominal value of 1.4 over the approximately 78.6-mm length. One can override this value by setting the variable **conv_factor**.

The last three variables in Table 3-5 describe the placement of multiple beamlets in the MBE4 squirrel cage. Normally, HIBEAM is run with one beam from the beginning lattice element up to the beginning of element # **imerge** (which is set in the ***ztime*** namelist). At that point, the single beam is replicated into 4 separate beams., Each of the four beams is given the appropriate rotation for each of the various cages, and then offset by **delx_w** and **dely_w** in transverse position and given a uniform dipole dx/dz or dy/dz corresponding to **conv_angle** (note this is in degrees, not radians) for the side/vertical cages respectively. For the MBE4 combiner experiments, this convergence angle is 6 degrees relative to the central axis.

Sec. 3-4 *Output File Formats and Specifications*

At present HIBEAM can make three different types of output files. The first is in the form of a NCAR graphic CGM metafile that can be viewed by the *ictrans* family of codes (available on the

NERSC Crays) and *gplot*, available both at NERSC and on the local SUN workstations. As described in Sec. 3-1 "Main Input File Structure and Variables" on page 13, most of the graphics output specifications are set in the namelist ***ztime*** in the "main" input file. Normally, one will want some combination of phase space plots and field plots. History plots (*e.g.*, transverse emittance versus z) are automatically produced --- normally the user will only set the variable **`dzhist`** to control the spacing of the individual z locations in the history arrays. Once the cgm file is created, one can use the *ictrans* program to output individual frames into other graphics formats such as Postscript.

The second form of output file available from HIBEAM is a simple ASCII text file containing columns of "history" information of envelope quantities such as I, X_{rms}, $X_{centroid}$, ε_{rms}, *etc.* versus z. One uses the switch **l_print_hist** to make HIBEAM write this file, which will have the name *run_name*`.txt`, where *run_name* is the ASCII string chosen for the run name on the execute line (see Sec. 2-2 "Running HIBEAM directly from a terminal window" on page 8). Note that if *run_name*`.txt` already exists on disk, the old information will be overwritten by the new.

The third type of output file is a particle "dump" file, which is a snapshot at a given z of the transverse coordinates (x, y, x', y') of all the individual macroparticles in the HIBEAM run. The locations of the dump are controlled by **`zpardump`** while the format is chosen by **`dump_type`** where the allowed types are 'BINARY', 'ASCII', and 'HDF'. Unless one is trying to export the phase space information to a bizarre hardware platform, in which case ASCII is probably the most portable format, one should probably set **`dump_type`** to 'BINARY" as this is the most compact. More importantly, at present HIBEAM can restart from a previously written dump file if and *only* if this file was written in binary format. In order to restart, the variable **`par_dump_file`** in the namelist ***hinit*** in the main input file should be set to the dump file name. Note that the z-location at which the old dump file was written will supercede the value of **`zstart`** in the *ztime* namelist - *i.e.* the new run will start at where the old run left off..