OPTICOOL – A Multiprocessor Optimizer for Muon Cooling Simulation Codes

Steve Bracker
s_bracker@hotmail.com

Version 1.1 January 8, 2001
Version 1.2 March 3, 2001 MuCool Note 0197

Introduction

Muon cooling simulation studies [1] are time-consuming; simulating a single apparatus configuration may take many hours. When the user is searching for an optimal apparatus configuration, it is often necessary to simulate one configuration, examine the results, choose a new configuration to test, compose new simulator control files and restart the simulator... and do this many, many times.

OPTICOOL is a package that "wraps around" an existing simulation code. It aids the user in two ways: (1) OPTICOOL brings the computing power of many processors to bear on the simulation task, reducing the clock time required to find a nearly-optimal apparatus configuration, and (2) with initial guidance from the user, OPTICOOL chooses new apparatus configurations to examine, composes new simulator control files and initiates new simulations without further user intervention.

In the ideal case, the user sets up OPTICOOL and the simulation package of choice. He specifies those apparatus parameters that can be modified – the **optimization variables** – and specifies how certain simulation results – the **merit variables** – are to be combined into an overall figure of merit for each apparatus configuration examined. Having told OPTICOOL what processors are available for use and how precisely the optimum needs to be located, the user starts the optimization process, walks away, and comes back when OPTICOOL has tested enough apparatus configurations to have homed in on a close-to-optimal configuration, characterized the optimum, and halted. OPTICOOL controls all of the processors from a unified user interface, and provides a running progress report of its activities. Even in the less-than-ideal case, when human judgment must still be applied at intermediate stages of the optimization, OPTICOOL may still be of substantial benefit, reducing the clock time required to obtain intermediate results and allowing the user to guide the process only when enough work has been done to make intervention productive.

Overview and Terminology

One **simulation run** of a **simulation program** examines the performance of one **apparatus configuration**. The apparatus configuration to be examined is specified by setting the values of a set of **configuration variables** – absorber lengths, magnetic field strengths, cavity phases, etc. During OPTICOOL's search for an optimal apparatus configuration, some configuration variables are held fixed, while others – the **optimization variables** – are allowed to vary within prescribed **optimization variable bounds**.

Each run of the simulation program reports the values of a set of **simulation output variables**. A subset of these – the **merit variables** – may contribute to determination of a **configuration merit** – a single figure-of-merit which specifies whether the apparatus configuration now being considered is better or worse than other competing configurations. By definition, the **optimal configuration** is that apparatus configuration, from among all those examined, which has the highest configuration merit.

The configuration variables are specified to the simulation program through one or more **simulator input files,** in a format determined by the simulation program. OPTICOOL is taught how to compose simulator input files, changing the optimization variables as needed. The user must compose a **simulator input file template** that looks very much like an ordinary (non-OPTICOOL) simulator input file. However, for those configuration variables which are optimization variables (hence subject to change during optimization), the actual value of the variable is replaced by a **substitution marker**. Prior to each simulation, the optimizer determines the values of optimization variables, reads in the simulator input file template, and replaces each substitution markers by an actual value, thereby producing a simulator input file that the simulator can understand.

During each simulator run, the simulator may produce a number of reports characterizing the behavior of the apparatus configuration. One of these reports – the **merit output file** – reports the values of all the merit variables – values that may enter into the calculation of the configuration merit. Following specifications supplied by the user, OPTICOOL uses the reported values of the merit variables to compute the final configuration merit, compares the configuration merit to the merits of apparatus configurations previously studied, chooses new values of the optimization variables, composes new simulator input files, and starts a new simulator run. This continues until pre-defined **optimization stop criteria** are satisfied, at which point OPTICOOL proceeds to **characterize the optimum** according to user instructions. When characterization is complete, OPTICOOL halts.

We may consider the configuration merit as a scalar function – the **merit function** – over a bounded N-dimensional **configuration space** whose axes are the N optimization variables. The optimizer's task is to find the N coordinates of the point in configuration space where the merit function takes its maximum value. The shape of the merit function may be a simple smooth hill whose summit is found somewhere well within the bounds of the configuration space; in such cases, the optimization will almost certainly converge in a fairly straightforward manner to the true optimum. Of course it is possible to imagine far more pathological merit functions, in which the true optimum is surrounded by deep valleys and false summits. In such cases, only a fine-grained search over the entire volume of the configuration space will offer substantial assurance that the true maximum has been located – a search which will take forever or even longer unless there are very few optimization variables. The user's task is to try to use physics insights to define the optimization task in a manner that makes the shape of the merit function as much like a simple hill as possible. One cannot expect to succeed every time on the first try.

The user controls the optimization process by composing the **optimizer control file** and providing it as input to OPTICOOL. Among other things, the optimizer control file specifies:
- the names of the **processors** to participate in the optimization, and the role each is to play
- the **optimization variables –** names, initial values, boundary values etc.
- the **substitution markers** – names, associated optimization variables, data formats, etc.
- the **merit variables** – names, how each contributes to the configuration merit, etc.
- how exactly the optimum must be located, and how the optimum is to be characterized once found.

OPTICOOL produces a **terminal screen display** which summarizes progress, a **report file** which details the results obtained, and a **diagnostics file** which produces extensive information about the optimization process. The last is usually of interest only if troubles are encountered. The reports normally produced by the simulator are also generated, but are normally not kept.

Optimization Strategies and Caveats

There is no magic in OPTICOOL. It does not substitute for physical insight; at best it may give some practical help to the user as he tries to develop such insight with respect to a specific problem.

OPTICOOL's primary optimization strategy is a modified variable-size simplex algorithm. It is best characterized as a semi-smart hill-climbing algorithm. As such, it will proceed to the optimum of a non-pathological merit function in a fairly efficient manner, and will be fairly easily led astray by a pathological merit function. Simplex algorithms are widely used in all kinds of optimization problems, so their strengths and weaknesses are well understood and documented. A particularly detailed (to the point of tedium) explanation, focused on optimization of industrial-scale chemical processes, is found in reference [2]. In short, the algorithm strides through configuration space, calculating the configuration merit at each step, switchbacking its way uphill toward the optimum, decreasing its stride length as it approaches the optimum, and stopping when the volume of configuration space still being examined is sufficiently small.

Once the optimum has been located, OPTICOOL will define **optimum characterization lines** in configuration space which pass through the optimum and are parallel to the axes. User-specified lists of configurations along these lines are simulated. If the merit decreases along the characterization lines as you move away from the optimum, you may feel somewhat reassured (though obviously not completely certain) that the optimum you have located is the one true optimum.

For optimizations involving only one or two optimization variables, it is feasible to do a **lattice optimization**. The program simulates configurations at a regularly spaced lattice of points within configuration space. The best point is chosen, the lattice granularity is reduced, and another lattice is computed, centered on the previous best point. In most cases, two stages are enough to locate the optimum to within the required accuracy. The chances of being fooled by a false optimum are much reduced (but not eliminated) using this kind of search.

Although it is easy to make up scenarios in which merit functions will be complex and fool OPTICOOL – merit functions that look like double-slit interference fringes come to mind – I suspect that such examples are rare in the real world of practical muon cooling designs.

How many optimization variables might one employ? OPTICOOL is currently written to permit up to 20. This limit is simple to change, but in any case, I suspect that ten optimization variables are probably a practical upper limit. By including many optimization variables upon which the configuration merit is only weakly dependent, one soon finds that one configuration's merit looks very much like the next, and that horrendously long simulation runs – propagation of millions of beam particles – are needed to detect the direction of "uphill" in the merit function. It is often best to concentrate initially upon working with a few optimization variables that are known to be important, and to use later optimization runs to assess the configuration merit's dependence on other variables.

If you have 100 cooling cells in your apparatus, and each cooling cell has a cavity phase to define, don't even think of defining 100 optimization variables, one per cell. It is overwhelmingly likely that the optimum phase varies only slowly and simply with cell number. Parameterizing phase as a simple function of cell number and using coefficients of the function as optimization variables is far more practical. It is physically implausible to suppose that the best configuration is one in which most cavities have a 15-degree phase, but cavities whose cell number happens to be a Fibbonacci number will need to have 30 degree phases; there is no point in setting up the optimization schema to be so general.

Processes, Processors and Processor Interconnection

Although it is possible to run OPTICOOL on a single processor, much of its power rests on its ability to make efficient use of "farms" of processors. Given a simulation of reasonable complexity, fairly large numbers of processors (up to several dozen) may be put to beneficial use. Eventually, of course, the time required to communicate between processors becomes significant compared to the simulation time on each processor, and at that point there is little benefit in adding still more processors. There is no simple answer to how many processors is too many to utilize efficiently; it depends strongly on the complexity of the apparatus configuration being examined, the sophistication of the model employed to propagate particles through various regions of the apparatus, the shape of the merit function, etc. OPTICOOL has never run with fifty processors, but I think it is safe to say that fifty processors could be put to beneficial use on ICOOL simulations of apparatus having several hundred regions (e.g. 50 fairly simple cells), and substantially smaller GEANT-based simulations. Ten processors can be used efficiently with virtually any cooling configuration complex enough to show any promise of success.

OPTICOOL consists of **two types of process**: one instance of a **Boss process** and one or more instances of a **Worker process**. As seems only fair, the Boss manages the optimization process and tells the Workers what to do; the Workers do all of the hard work, the simulations themselves. The user communicates with OPTICOOL only through the Boss, via terminal i/o, input files, and report files. All terminal i/o is of the simplest kind, so that remote operation of the entire system via telnet is straightforward. The Boss runs the optimization algorithm in accordance with the user's instructions, defines the simulations to be performed, farms the simulation work out to the Workers, collects the simulation results from the Workers, computes the overall configuration merit for the current apparatus configuration, decides upon a new simulation to be performed, orders the Workers to do it . . .   The Worker code is little more than a simple wrapper around the simulation code. The Boss supervises the whole optimization process, but performs no simulations itself.

In the usual case, each processor (a physical computer) runs one instance of the Worker process, and one processor runs, in addition to its Worker process, the single instance of the Boss process. The Boss uses very little computing time except for a few moments between one simulation and the next.

All processors participating in an OPTICOOL run must be able to communicate via a network. At present, OPTICOOL uses Linux implementations of remote shell (rsh) and remote copy (rcp) commands. The Boss uses rsh commands to send instructions to the Workers. The Boss uses rcp command to send disk files (typically simulator input files) to the Workers, while the Workers use rcp commands to send files (typically simulator results files) to the Boss. As in any well-conceived autocracy, Workers do not communicate with other Workers.

Other than network-compatible versions of Linux, there is no multiprocessor management software (farm management software) needed. However, the processors must not be isolated from one another by unreasonable security barriers; they must be able to rsh and rcp freely with one another, and must not impose unreasonable limits on the size and frequency of such transactions. I have found it necessary to modify a few default Linux configuration file settings to provide adequately open interprocessor communications [3]. Doing so normally requires superuser (root) access to the computers. (However, OPTICOOL runs in a normal user account.) Where computers languish under the lidless eyes of paranoid security mavens, an occasional discrete poisoning may prove helpful.

How is the simulation work divided amongst Worker processes? In two different ways. At some points in the optimization process, there is only one apparatus configuration being examined at a time. In this case, all Workers simulate the same configuration, each propagating its own unique list of beam particles. Upon completion, the merit variables from the various Workers are gathered and combined by the Boss. At other points, there are many apparatus configurations being examined at the same time. In this case, each configuration is simulated by one Worker. It is this aspect of the beam simulation problem -- that the overall simulation workload can be divided in two straightforward ways amongst many Workers – that minimizes the interprocessor communication requirements and simplifies the design of the optimization process.

It is worth noting in passing that when a single apparatus configuration is simulated by several Workers at one time (using unique beam particle sets), comparing the merits returned from the various Workers gives immediate insight regarding the statistical significance of the simulated merit function and the usefulness of comparing it with merits computed for competing apparatus configurations. If five Workers compute merits of 0.85, 0.90, 0.95, 1.00 and 1.05 for configuration A, and one needs to compare the merit of configuration A (merit(A) = 0.95) with that of configuration B (let us suppose merit(B) = 0.93 with similar scatter), the reliability of the merit comparison (e.g. configuration A is better than configuration B) is clearly very low. But it is exactly this kind of comparison that optimizers use to decide what configurations to try next. Either one must stop (the optimum is located to sufficient precision), or compute the merit function more accurately; in general, this will be accomplished only by propagating many more particles, perhaps at great expense in computing time. In situations where the merit function is changing rapidly as one steps along in configuration space, one may save a lot of time by computing merits to only the accuracy required to decide upon the next configuration to try.

Musings on the Configuration Merit

It may seem a bit confining to have the merit of an apparatus configuration characterized by a single number. After all, a muon cooling apparatus must embody many virtues. It must, at least, reduce several components of the beam emittance, preserve a reasonable fraction of the incident muons, and do so using apparatus which can be built at reasonable cost. However, whether the optimization is done “by hand” (i.e. with human judgment exercised at the start of each simulator run) or by an automatic optimization procedure of some sort, one must finally be able to say “all things considered, apparatus configuration #23 seems to be better than the alternatives, so it is the one we will build”. In requiring a real specification for how one goes about the “all things considered”, OPTICOOL is merely forcing the user to be explicit about defining tradeoffs that might otherwise be left fuzzy. It’s hard to imagine this being a bad thing.

By default, OPTICOOL accepts a table of simulator results that might enter into the final figure of merit – the so-called merit variables. The first thing it does is to transform the value of each merit variable into a **merit component**. OPTICOOL tries to be a little bit intelligent about how it goes about this.

For some merit variables, the merit increases as the merit variable’s value increases; in other cases, the opposite is true. In principle, the relationship could be more complex than this, but in fact it is difficult to find a good example in the muon cooling task; you can never get an emittance too low or a muon survival fraction too high.

On the other hand, it is certainly not true that merit is expected to vary linearly with the merit variable’s value. Consider the muon survival fraction. One might well decide that anything below 60% is unacceptable, anything above 90% is good enough, and that merit rises fairly smoothly from “unacceptable” to “good enough” between 60% and 90% survival.

To make this qualitative description of merit specific enough to calculate, OPTICOOL provides a **merit transfer function**, as shown. It ranges from user-defined minimum merit Mmin (typically zero) at -∞ to maximum merit Mmax (typically 1) at +∞.

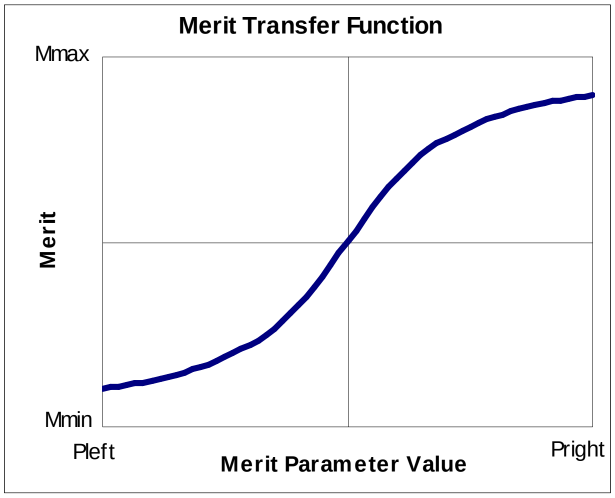
The user can specify the values of the merit variable – Pleft and Pright – between which the merit rises steeply from just above Mmin to just below Mmax. In cases where merit increases with increasing merit variable value, Pleft < Pright. In cases where merit increases with decreasing merit variable value (e.g. an emittance), Pleft > Pright.

For the example above, Mmin might be 0, Mmax might be 1, Pleft would be 0.6 (60% muon survival), and Pright would be 0.9 (90% muon survival). Though it is plotted only for merit variable values between Pleft and Pright, the merit function is well defined for all values of the merit variable.

This procedure is carried out independently for every merit variable participating in the calculation of the overall configuration merit. When all of the merit components have been calculated, they are joined together in a user-specified manner to determine the overall configuration merit.

Merit components may be combined by either addition, or multiplication, or both. Imagine two merit variables which might be thought of as similar virtues of about equal importance. Then you can set Mmin = 0 for both, Mmax = 1 for both, and add the two merit components together to determine the total virtue. If one merit variable is twice as important to you as the other, set its Mmax = 2. On the other hand, if you have a merit variable which is of overriding importance – muon survival fraction might be a good example – then it should be multiplied into the other merits, so that no configuration having a poor score in this variable will end up with a good overall merit, no matter what the configuration's other virtues. OPTICOOL allows one to combine merit components into partial sums, and then multiply the sums together to obtain a final figure of merit. So Mfinal = (M1 + M2) * M3 * (M4 + M5 + M6) is a possible formula, where Mfinal is the final configuration merit and M1 . . . M6 are merit components.

I anticipate that this method of combining merit components will satisfy most users. For exceptional circumstances, one always has the option to calculate the configuration merit directly, either in the simulator or in a private version of OPTICOOL, where all the generality of a general purpose programming language may be employed.



How do the merit variables get from the simulator to OPTICOOL? Through a file written by the simulator. This is the only substantial change that must be made to the simulator to embed it within OPTICOOL. Upon completion of every run, the simulator must write a file, one merit variable per line, specifying the name and value of each potential merit variable. For a given optimization, the user may instruct OPTICOOL to ignore some of the merit variables listed in the file, but naturally all those variables OPTICOOL is instructed to use must be present in the file and must have valid values.

Bounds on Optimization Variables

Since the optimization variables represent apparatus construction parameters – lengths, magnet currents, etc. – there are practical limits to all of them. Some of the limits are hard limits; there is 50 cm available for an absorber, and it can't be any longer than that. The window cannot be thinner than 150 microns for safety reasons. Other limits are soft; although it is preferable to stay within the stated limits, it is permissible to stray beyond them if a compelling case can be made.

In OPTICOOL, every optimization variable has an upper and lower bound, as well as a parameter specifying how rigidly the boundary is to be enforced. For each configuration, a **boundary merit** is computed. It is 1 if every optimization variable is within the stated bounds. It is decreased as optimization variables stray further and further outside the stated boundaries. For boundaries with rigid enforcement, the optimization variable can transgress the boundary by only a tiny distance before the boundary merit approaches zero. As boundary enforcement for a given variable becomes more relaxed, the impact of straying a given distance across the boundary is diminished.

At the completion of each simulation run, the boundary merit for the configuration (which does not depend in any way on the output from the simulator) and the simulated merit (which depends only on the output of merit variables from the simulator) are combined to form the final figure of merit for the configuration. In this way, intolerable boundary violations are "punished" by telling the optimizer that it has just stepped into a configuration of very low merit – and it had best reverse course at once. Note that this reversal is desirable whether the low merit is the result of a serious boundary violation, an intrinsically lousy configuration, or both. When an optimum does truly lie beyond a boundary, the optimizer will push against the boundary repeatedly, but it soon learns not to make large leaps into forbidden territory.

It is important to avoid letting the boundary merit ever go to zero within a region, no matter how absurd the configuration. The optimizer must always know what direction to travel to repent of its reckless explorations.

In this illustration, an optimization variable boundary for variable X lies at +1. Boundary merits for four different boundary enforcement levels are drawn, ranging from the very relaxed S=1 to the very rigid S=20. (S is the rigidity parameter specified by the user for each boundary.) If boundary enforcement is very relaxed, then the optimal configuration located by OPTICOOL may lie well outside the prescribed boundaries. With boundary enforcement as tight as 20, it is very unlikely that the optimum will be found in forbidden territory.

The rigidity of boundary enforcement should reflect the cost of implementing a solution that lies outside the boundaries. This varies between physical impossibility (very rigid bounds) to slight additional costs (very relaxed bounds).

Using OPTICOOL

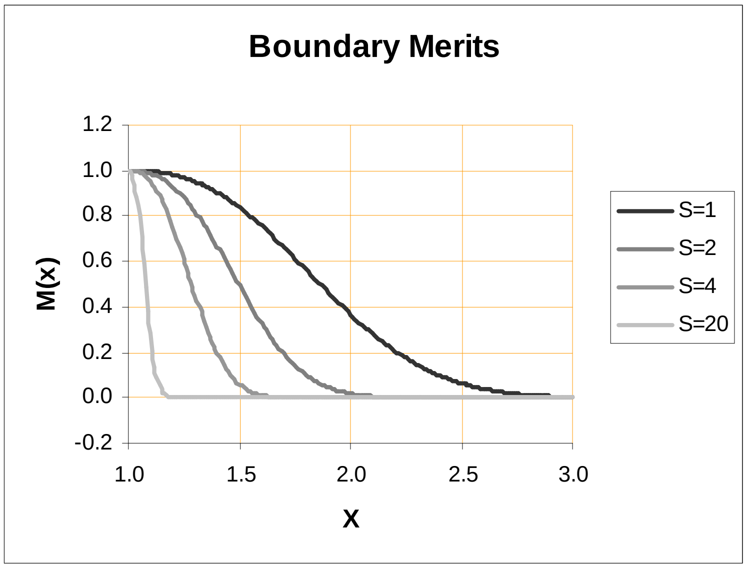
If you are starting right from the beginning, you have a fair bit of work to do before starting your first optimization run. By “starting at the beginning”, I mean no computers, no experience with Linux, ICOOL (or whatever simulator you are using) and OPTICOOL, and not much careful thought – the kind of thought that translates rather directly into an ICOOL control file, for example – about the apparatus you are hoping to study. Few potential users are in this position.

There are many alternatives to an unassisted cold start. If you don’t have Linux computers readily at hand, see if you can use someone else’s computers. For example, there is currently lots of time available on the Ole Miss Linux cluster, and you can operate it from just about anywhere using Telnet. This is by far the preferable option if you are running a small problem, or aren’t sure whether OPTICOOL will be useful to you or not.

If you’re not wanting to get into Linux system installation, you can drop-ship your new computers to someone who can install and configure Linux, ICOOL, and OPTICOOL, and send the computers on to you, ready to connect to your network and use; in such cases, it’s always best to select hardware in consultation with the people who will be doing the software installation. Alternatively, you can go to an established OPTICOOL site with a computer or two of your own and get started with guidance from someone who has done it before.

If you decide to install Linux yourself, I strongly recommend paying the big bucks (about $50-$100) and buying a commercial Linux distribution like Redhat Linux. You should also make sure you get a copy of ***Setting Up Redhat Linux 6*** and ***Setting Up Linux Workers 01*** (local documents) which describe some changes to default Linux installation options and configuration files that are required to make OPTICOOL run smoothly.

Insofar as possible, you should set up the OPTICOOL accounts and disk directory structures so that they exactly parallel those used here; it makes debugging and distribution of new software much easier. We have a test version of



ICOOL which generates fake simulation output with known results; you should be sure to test your new OPTICOOL installation with it.

Before trying to run a new apparatus configuration in OPTICOOL, run it in stand-alone ICOOL (or whatever your simulator of choice may be). You may find out one of several things: (1) that the proposed apparatus is too hopeless to be worth optimizing, (2) that it is so robust to configuration changes that there isn't enough optimization to do to justify setting up OPTICOOL, or (3) that OPTICOOL can probably help you, and you now have enough practical experience simulating the new apparatus to set up the optimizer rather easily. I wouldn't use OPTICOOL until I had made several successful ICOOL runs of the proposed apparatus and had developed some "feel" for what configuration variables and merit variables were important.

I upgrade OPTICOOL from time to time, and users should inquire about upgrades. Suggestions for improvements are always welcome, and many users will find it fairly straightforward to tailor and enhance OPTICOOL to better serve their needs. Improvements of general utility can always be incorporated in the next standard release.

If you decide to download and install OPTICOOL yourself, you'll probably find that installation is pretty straightforward. The more your computer system resembles ours, and the more thoroughly you have checked out your operating system, your g77 compiler, and your network, the easier it will be. You will receive some source files and some data file templates. You will run a couple of small command files to compile and link the Boss and Worker code, and distribute the Worker code (which embeds the simulator) to all the Workers. A couple of things won't work because I forgot to tell you something or send you something; before you get all hot and bothered, you'll give me a call and we'll get it sorted out.

Performing an Optimization Run

Among the things that must already be in place:

1. You must have the computers on and booted, connected to the network, and able to talk to one another. You must be able to start a terminal session on the processor which is to run the Boss process. If in doubt, it is not a bad idea to log onto the opticool account on the Boss, and ping each Worker processor from the Boss processor to confirm at least minimal connectivity.

2. You must have prepared and distributed to the Worker processors the Worker program, which has the simulator embedded in it as a subroutine. The embedded simulator must produce a Merit Output File which returns values for all of the Merit Variables that might enter into your calculation of Configuration Merits.

3. You must have a valid Boss process ready to go on the Boss processor. The Boss process does not normally change from application to application.

4. You must have prepared simulator control files describing the apparatus, the beam, etc. Ideally they have already been tested using a stand-alone simulator run. These files will be modified slightly a bit later.

Here is what you do to carry out the run.

1. On the Boss processor, log in to the ***opticool*** account, either using a local terminal or via telnet. The ***cdbd*** command will take you to the oc/bd (opticool boss-data) directory where most of the upcoming work will be done.

2. Make a list of the **optimization variables** you will be using, the bounds on each, and the rigidity of the bounds. Decide how thoroughly you want the optimum configuration characterized in this variable. Often this list is just a slight modification from a previous run; for example, you may be allowing a new configuration variable to be varied and optimized, or assigning a fixed value to a former optimization variable.

3. Make a list of **substitution markers** associated with the optimization variables. In simple cases, there is a 1:1 correspondence between substitution markers and optimization variables, but it is sometimes useful to associate more than one substitution marker with a single optimization variable. An example: two devices must fit side by side in a fixed space, so that centimeters added to one must be subtracted from the other.

4. Make a list of **merit variables** contributing to the overall figure of merit for the configuration. Make sure that they are all present in the Merit Output File produced by the simulator. Often this is a slight modification from a previous run.

5. Make a list of all the **processors** that will be participating in the optimization. Usually this does not change from run to run.

6. Based on these lists, compose the optimization control file **OptCont.dat**. It lives in the oc/bd directory. Files from previous runs may be copied and modified as required. It is <u>much</u> easier to modify an existing file, even one from a different optimization package, than it is to compose one from scratch.

7. For each **simulator input file** that specifies the value of at least one optimization variable, edit the file, replace the numerical values for optimization variables with substitution markers, and store the file under a new name, for example for001.template instead of for001.dat. The templates must be in the oc/bd directory. Make sure that the file is listed in the FileTransfer section of OptCont.dat; this is how you insure that when new simulator input files are prepared by the optimizer, they are distributed to all of the Workers.

8. Prepare and list in the **FileTransfer** section of OptCont.dat any additional input files that must be distributed to the Workers. Place the files to be distributed in the oc/bd directory

9. Define the few additional **control parameters** in OptCont.dat. Save the file.

10. Issue the **bb** command. This will distribute files, start workers, perform the optimization task until done to specification (or fatal error) and return reports.

11. Examine the **report** file in the oc/bd directory. In case of trouble, examine the **diagnose** file.

References and Notes

[1] There are a number of simulation programs and packages in use by people studying the muon cooling problem, but two of them – **ICOOL** (Rick Fernow et. al.) and **DPGEANT** / **GEANT4** (Paul Lebrun et. al.) – seem to be the most widely used and publicly accessible. As the name implies, OPTICOOL was developed around early implementations of ICOOL.

[2] Frederick H. Walters et. al., ***Sequential Simplex Optimization***, CRC Press, 1991. ISBN 0-8493-5894-9 Chapter 7, "Additional Concerns and Topics", describes a number of ways that optimizations can go wrong; it's worth

reading, even though the more amusing portions (see "Safety Considerations") are not directly applicable to optimizations of simulations.

[3] Available on request: ***Setting Up Linux Workers 01.doc*** which describes, among other things, how to open up trouble-free lines of rsh/rcp communication between the computers.

Appendix A: A Complete Optimzation Control File

```
! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

WorkerNames

! Specify the processors to participate as workers in the simulation,
! and the speed of each (MHz CPU). Note that it is customary to run
! a worker instance on the boss processor, but to reduce its nominal
! speed by 10% or so to compensate for the time spent running boss
! code.

  linuxfarm1    425
  linuxfarm2    475
  linuxfarm3    475
  linuxfarm4    475
  linuxfarm5    475
  !!linuxfarm6    500
  !!BigGee        350
  !!alethia       350


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

OptimizationVariables

! Define the optimization variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a lower bound
!  --an upper bound
!  --a number specifying how steep the merit curve is at the boundaries
!  --the number of summary configurations to be computed
!  --an optional comment at the end of the line

  !Length of absorber (per cell, meters)
  absorberLength   0.30  0.46   10.  11

  !Current in solenoid (amps)
  solenoidCurrent  1400.  2000.  25.  11  very tight bounds

  !RF phase in accelerating cavities (degrees)
  cavityPhase  10.  24.  5.  11  (loose bounds for now)

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
InputSubstitutions

! Define the input file substitutions. Each has:
!  --a substitution marker e.g. {abc}
!  --an optimization variable name
!  --an output format specifier
!  --a multiplier M
!  --an addend A

! For every occurrence of the marker in a ghost input file, the value
! M * optValue + A is substituted, formatted as specified.

  {a}   absorberLength     f8.3   1.0  0.0
  {s}   solenoidCurrent    f8.1   1.0  0.0
  {cp}  cavityPhase        f8.3   1.0  0.0

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

MeritVariables

! Define the merit variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a merit combination code, M, A1, A2 ... A9
!  --merit variable value at normalized merit = 0.1
!  --merit variable value at normalized merit = 0.9
!  --minimum value of the merit function (usually 0.0)
!  --maximum value of the merit function (usually 1.0)

  !Fraction of muons that make it through the apparatus
  MuonSurvivalFraction  M  0.85  0.95  0.0  1.0

  !Bunch Length in meters -- note this is good-when-low
  BunchLength  A1   10.0  5.0  0.0  1.0

  !Transverse Emittance -- note this is good-when-low and contributes twice as
  !much as the bunch length
  TransverseEmittance  A1    0.35  0.20  0.0  2.0

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

FileTransfers

! Define the files to be sent from boss to workers. All files are
! sent from the bd (boss-data) area of the boss to the wd
! (worker-data) area of each worker.
!
! If a single filename is specified, then the file is transferred
! unmodified and with the same name to the designated worker.
!
! If two filenames are specified, then the input file (first name)
! is subjected to variable substitution, producing an output
! file (second name) in the boss data area. The output file is then
! copied to the designated worker and deleted from the boss data area.

  ! Primary simulation control file for ICOOL
  for001.ghost for001.dat


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

Parameters

! Define various opertating parameters controlling the optimization
! process. Each line consists of a name and a value. All parameters
! must be explicitly specified; there are no defaults.
!
! nPartSim -- number of particles to propagate each optimization
!  simulation
! nPartSum -- number of particles to propagate each summary
!  simulation
! stopCriterion -- how small a volume you wander within before you
!  stop simulating, in normalized coordinates
! maxOptSims -- maximum number of optimization simulations

  nPartSim  500

  nPartSum 1000

  stopCriterion 0.01

  maxOptSims 50
```