Key elements of the user-friendly, GFDL SKYHI general circulation model

Richard S. Hemler
Geophysical Fluid Dynamics Laboratory/NOAA, P.O. Box 308, Princeton University, Princeton, NJ 08542, USA
Tel.: +1 609 452 6598; Fax: +1 609 987 5063; E-mail: rsh@gfdl.gov

Over the past seven years, the portability of the GFDL SKYHI general circulation model has greatly increased. Modifications to the source code have allowed SKYHI to be run on the GFDL Cray Research PVP machines, the TMC CM-5 machine at Los Alamos National Laboratory, and more recently on the GFDL 40-processor Cray Research T3E system. At the same time, changes have been made to the model to make it more usable and flexible. Because of the reduction of the human resources available to manage and analyze scientific experiments, it is no longer acceptable to consider only the optimization of computer resources when producing a research code; one must also consider the availability and cost of the people necessary to maintain, modify and use the model as an investigative tool, and include these factors in defining the form of the model code. The new SKYHI model attempts to strike a balance between the optimization of the use of machine resources (CPU time, memory, disc) and the optimal use of human resources (ability to understand code, ability to modify code, ability to perturb code to do experiments, ability to run code on different platforms).

Two of the key features that make the new SKYHI code more usable and flexible are the archiving package and the user variable block. The archiving package is used to manage the writing of all archive files, which contain data for later analysis. The model-supplied user variable block allows the easy inclusion of any new variables needed for particular experiments.

1. Introduction

SKYHI is a grid-point atmospheric general circulation model which was developed at the NOAA Geophysical Fluid Dynamics Laboratory nearly twenty years ago (Fels et al. [1]).The model is global, and extends in the vertical from the ground to about 80 km. SKYHI has been used to investigate various tropospheric, stratospheric and mesospheric phenomena, including sudden stratospheric warmings, the quasi-biennial oscillation, ozone depletion, gravity wave-mean flow interactions and chemical and tracer transport problems. A concise description of the model equations, numerics and physics may be found in Jones et al. [2]; a more detailed description is found in Hamilton et al. [3].

Jones et al. [2] describe changes made to the model in order to port it to the Thinking Machines Corporation CM-5 machine at Los Alamos National Laboratory. This paper concentrates on changes made to the model that significantly enhance the ability of the SKYHI user to quickly and easily modify the model code to initiate, run and analyze new scientific experiments, and to attach new features to the model with a minimum of difficulty. Section 2 discusses the development of the generic SKYHI code in which the efficient use of human resources is balanced with the efficient use of machine resources. Section 3 defines the basic structure of SKYHI and the current meaning of "modularity" as used by SKYHI. In Section 4, two of the important new user-friendly "modules" of SKYHI are presented, along with examples of their use. Section 5 discusses the parallel performance and scaling of SKYHI on Cray Research PVP shared memory and Cray Research T3E distributed memory machines, while Section 6 serves as a summary.

2. Balanced optimization

The SKYHI model code that existed in 1990 was the result of the efforts of many people over many years and several machines to produce a code which would run using a minimum amount of CPU time, memory and I/O time. As a result, the relationship between the model code and the analytical expressions it represents was difficult to see; model variables were often defined to minimize arithmetic operations, rather than to be physically- or numerically-meaningful quan-

Scientific Programming 8 (2000) 39–47
ISSN 1058-9244 / $8.00

Table 1
Desirable model resource optimizations

Machine resources	Human resources
Minimize disc usage	*Maximize* ease of code modification
Minimize memory usage	*Maximize* ease of code maintenance
Minimize I/O time	*Maximize* portability
Minimize system and user CPU times	*Maximize* ease of data analysis
Maximize ability to execute on a variable number of processors	*Maximize* ease of use by novice users
Minimize data archival requirements	*Maximize* ease of use by experienced users

tities. Memory usage was minimized by implicit and explicit equivalencing; certain common blocks were used to provide the current functionality of the stack. Specific coding practices needed for performance on previous platforms were still in place, even though they were no longer needed. The result was a model which could be run with scientifically-meaningful horizontal resolution on a machine with limited central memory, but which was difficult to modify and often failed in unexpected ways when perturbed, and in which code for specific model processes was scattered across many subroutines.

At the same time, the shrinking budget for basic research and the desire to reduce the Federal workforce was shrinking the number of GFDL programmers and scientists available to manage the code and to run scientific experiments. To augment the dwindling permanent staff, more visiting scientists were being invited to GFDL, usually for periods of a few years. In order for these people to make productive use of their time at GFDL, it was essential that they quickly become able to use the model in their studies, and not spend a lot of time attempting to attach their experiment to SKYHI. The existing code structure of SKYHI made this process difficult.

With the coming of the Cray Research Y-MP machine in 1990, the available computing power at GFDL increased significantly. This allowed the "machine deoptimization" and "user optimization" of the SKYHI code to begin, without resulting in a reduction in model throughput for the SKYHI user community. Code constructs which may have been machine efficient at some point in the past but which were difficult for users to understand and modify correctly were replaced with more easily understood code, the first step toward balanced optimization.

Balanced optimization attempts to optimize not only the use of machine resources, but also the human resources required in scientific investigations. Table 1 lists several of the important code qualities that need to be optimized in a research code designed to be used by a variety of users. The ideal code would be optimal in all of these user and machine resource dimensions; since no real code is ideal, it is desirable to produce code in which none of these features is out-of-balance with the others. Typically these features compete with each other; for example, a code which is highly I/O efficient is likely to be less memory efficient than it could be if the I/O were not so efficient. The strategy then is to produce a generic production code in which there exists a balance between these features.

Table 2
Lines of code devoted to various model functions

Function	Lines of code
Backbone code	6000
Dynamics	14000
Physics	24000
I/O, Archiving	21000
Diagnostics	7000

This generic code is the form that is maintained and made available to users. It is written so that it allows easy access to the model to a broad spectrum of users with diverse research interests, each of whom may wish to view different parts of the model as a black box, and who may not need or want all of the features that are provided. Individual users then optimize the generic code as they must in order to reach their scientific goals.

3. The structure of SKYHI and modularity

The SKYHI model consists of nearly 300 subroutines and 200 include files containing parameters and common blocks, all written in Fortran. It totals nearly 72000 lines of code, of which about 25000 are comments. A pre-processor is used to selectively activate desired portions of the code for particular experiments and to insert the include files where desired. Table 2 indicates the approximate number of lines of code associated with each of the major model functions.

Figure 1 shows the four basic sections of the SKYHI model. The model's initialization phase consists of setting up tables, initializing constants, reading input data – all the things that are only done once during a model run. After completion, the time-step loop is entered. Some of the calculations in this loop are not a function of gridpoint, but only a function of the time or timestep. This is called the time-dependent phase. After it is executed, the chunk loop is entered. This loop extends over the model domain, which is broken

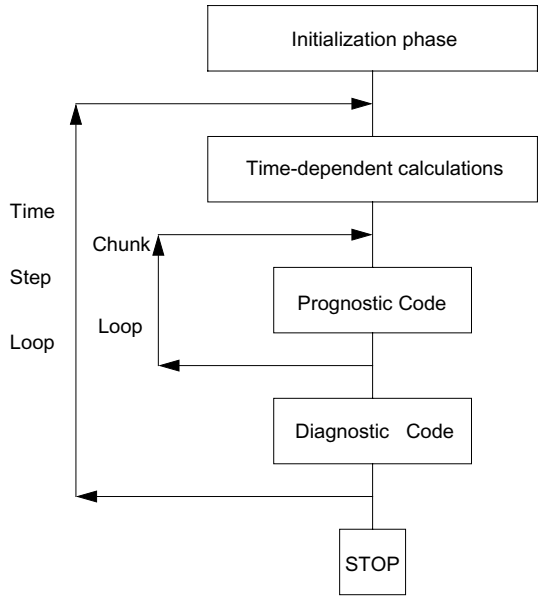

Fig. 1. The basic structure of the SKYHI model.

up into rectangular portions in the horizontal, referred to as chunks. These chunks may be executed either sequentially or in parallel. Within this prognostic section the model equations are time-integrated, contributions from the chunk to any integrands being calculated are computed, and any data from the chunk which are needed in archive files are written. After the domain chunks are processed, the processing of global integrals is done in the diagnostic code section to complete the time step.

Within the chunk loop, the prognostic code takes the form of triply-nested do loops, with k as the outer and i as the inner loop. Model arrays are dimensioned (i, j, k), corresponding to (longitude, latitude and height), so that this loop structure provides vectorization over the i loop, which in most production experiments is the longest.

Part of the ongoing effort to make SKYHI more user-friendly is the separation of the source code into "model" code and a number of modules. In SKYHI at this time, a module is defined simply as code needed to perform a specific model function which has been isolated from the rest of the source. That part of the code not yet contained within a module is referred to as the "model". The following guidelines must be met for a portion of code to be called a module in SKYHI:

1) No "model" include files, common blocks or subroutines may appear in any module.
2) No module include files, common blocks or subroutines may appear in the "model".
3) All communication between "model" and a module must be through argument lists.
4) Module to module communication must occur by going through the "model".
5) Modules may have an interface to the "model" in each model section defined above.

These conditions assure that the communication between model and module is explicitly defined, via a subroutine call argument list. It makes it obvious to a user what model variables must be supplied to the module, and prevents the inadvertent modification of model variables that could be brought into a module via a model include file or common block. Experience has indicated that many argument lists may be shortened by some code reorganization, which also invariably produces a more understandable and more modular code.

At the moment, SKYHI contains several physics modules that follow these coding guidelines. These include a radiation package, a cloud package, a surface albedo package, an astronomy package and an ozone package. These modules have been carved out of the previous SKYHI radiation code by isolating the code which is related to these processes. A modular structure allows easy implementation of alternative parameterizations and also allows the output from these modules to be passed back to the model and then used as input to other parameterizations, e.g., the stratospheric chemistry package associated with SKYHI requires astronomy package output. More processes will be pulled out of the "model" and made into "modules" as time permits.

Several modules have been added to SKYHI recently, following the coding guidelines presented above. These added modules include two optional tracer transport packages (advection plus subgrid-scale diffusion) and a particle trajectory package. An interface for each module was created in each of the four model sections (initialization, time-dependent, prognostic, diagnostic), and the package code which should be executed in each model section is accessed through this interface. As time allows, the modules in SKYHI will be made into Fortran 90 modules, further increasing the ease with which new Fortran 90 code written by others may be incorporated into SKYHI.

4. The archiving module and the user variable module

Two new higher-level modules have been created in SKYHI. These modules handle two functions which

are essential for users who must significantly alter the existing code for their experiments. Both of these features help to isolate the user's code from the SKYHI model code and so reduce the chances of the user inadvertently modifying the model code.

4.1. The archiving module

The archiving module controls the writing of data files that will be analyzed offline. Five file types are currently recognized: restart, point data, column data, slab data and reduced data. A restart file is written at the end of a job, and must contain the time-dependent variable fields and any other data that are necessary to allow the model to be started again at a later time, as though it had never been halted. Point data, column data and slab data files contain the desired model variables at a specified set of individual grid points, grid columns or grid planes, respectively. In the limit, the slab data files will cover the entire model domain. Reduced data files contain non-gridded data; they may contain integrals over the globe or some portion of it, or any other grid-independent data set.

All of these files (with the exception of the restart file) may capture either a "snapshot" of the data or may be used to contain some type of time-averaged representation of the data. Standard forms for each of the five file types are provided with the model code as a template to be used in creating customized files. Three standard forms are provided for each file type: the form for the standard version of the file; the form which will produce a time-averaged standard version of the file, and a form which may be customized by the user to provide whatever set of variables is desired, either as a snapshot or as a time-average.

User control of the archive files is provided through a combination of pre-processor options, namelist variables and model parameters. Pre-processor variables define how many file forms of each file type are present in the code. If the user adds code for a new file form, then the variable for that file type would be increased from the default value. If the model is to be integrated without writing any files of a given type, then the variable for that type is set to zero, and all the code and storage associated with the data files of that type will be removed from the model source.

If any data files for a given file form are to be written, then the user must specify namelist variables that define the temporal characteristics for each of the file forms of that file type. These variables and their use are described in Appendix A.

Data for each of the file forms of the grid-dependent file types is collected in an array dimensioned by (i, j, n) where i and j refer to the horizontal grid points in x and y respectively, and n is the sum over all variables of the number of k levels at which each variable is to be written. The beginning and ending spatial location indices of these arrays and the value of n for a given file form are specified as parameters in the model code.

Appendix B defines the recommended procedure for a user to follow when using the archiving module.This procedure has been successfully followed by SKYHI users. The naming conventions and supplied code allow even an inexperienced user to successfully create new archive file forms by making it evident what code changes must be made to the default code to create a new file form. Equivalent variables for different file types have names which differ only by the unique letter associated with that file type. Equivalent variables for different file forms of a given file type have names which differ only by the file form number, which is (are) the last digit(s) in the variable name. For example, the parameter defining the record length of the data record is ItRECLENa where t is the file type (r, t, s, h, or i) and a is the file form number. Thus the variable names are very recognizable, and contain the file type and file form information within them. The creation of control code for new file forms simply requires the duplication of existing supplied code, the location of the file-form-specific variable names, all of which contain the file form number, and the replacement of the old file form number with the new in these variable names.

The archiving package has also been incorporated into the GFDL Limited-Area Non-hydrostatic model (LAN) without difficulty, and should be insertable into any atmospheric model which has the proper interfaces. New and useful Fortran 90 constructs will soon be incorporated into the package and any problems or lack of clarity reported by users will continue to be addressed. Additional capabilities are also planned, including the ability to create time-averaged files that span jobs.

4.2. The user variable module

The user variable module allows the user to easily add variables to the model. The model contains an array ooo (ist:$iend$, jst:$jend$, kst:$kend$, n, m), where the first three dimensions are the (i, j, k) spatial indices, n is the number of ooo variables, and m is the time level index, indicating lag, mid or lead time. This variable is available to the user to contain n variables of his choos-

ing. If no user supplied variables are desired, then a preprocessor option is set, and all the code and storage associated with this variable is removed from the code. When user-defined variables are desired, model-supplied code will handle all aspects of their integration except for the calculation of the specific physics-chemistry-source-sink terms relevant to the variable in question, which obviously must be supplied by the user.

These user variables may be fully prognostic, semi-prognostic, or diagnostic. Fully prognostic variables have a time tendency resulting from transport and may also contain physics-chemistry-source-sink terms. The transport is handled by the model, as specified by the user from a series of options that are offered; there is no need for the user to provide code to transport these variables, unless a scheme is desired that is not offered by the model. Semi-prognostic variables have a time tendency resulting from source / sink terms, but are not transported. Diagnostic variables are defined on the basis of other conditions, and do not have an explicit time tendency equation. These variables may be integrated with a timestep either smaller or larger than the model timestep, if desired.

The user must customize the general user variables so that they become the variables that are desired. The model contains the chemical species nitrous oxide as a sample ooo variable, the treatment of which the user can follow for his own variables. Interfaces between the model and user variable module are provided in each model section; the user must determine the variables that will become the argument list between model and module.

Different types of variables have been coupled with SKYHI using the user variable code block. The experience gained in coupling these diverse types of variables to SKYHI has produced a more general structure of the user variable block, so that all of these variable types, each with their own special requirements, may be handled properly. The result is a more robust module, one that is more likely to be able to handle the next set of variables thrown at it than it was previously.

Several different uses have been made of the user variables in a chemistry context. The investigation of tracer transport by different transport schemes has been done very neatly by setting up an experiment with several initially identical copies of a given species, each of which is integrated with a different transport scheme, all within the same model run. The data for all the schemes are then present in the same data files, simplifying the analysis effort. A stratospheric chemistry package with thirty-seven chemical species has been attached to SKYHI and run without difficulty. In this case, some variables are prognostic, some semi-prognostic, some diagnostic, and some may be prognostic or diagnostic, dependent on the time of day. The option to have variables change between prognostic and diagnostic was not originally present and required a generalization of the original code.

Experiments that have investigated the vertical diffusion scheme in SKYHI have employed a simple radon tracer with a surface source. Multiple versions of the diffusion scheme may be tested in the same job by starting multiple identical copies of the tracer, each as a different user tracer variable which has a different diffusion scheme. In such a case the model needs to be run only once in order to compare n different diffusion formulations, rather than n times.

Another use of the user variables has been with a cloud ice parameterization scheme. Here there was a need for twenty-three diagnostic variables, and the presence of the user variable block allowed the easy inclusion of that many new variables. It is anticipated that cloud microphysics and atmospheric aerosols may soon be examined in SKYHI and it is likely that both of these variable sets will be handled within the user variable block.

None of the above-cited cases could have been run using the 1990 SKYHI code without extensive and unique recoding for each specific experiment, which for similar studies in the past has taken inexperienced users of SKYHI many weeks to months of effort. The presence of the archive package and the user-variable code block confines the required code generation for these experiments to the definition of the model interfaces to the new code, which has been done even by new users within a week. The ability to easily create new archive files specific to a new experiment also greatly reduces the analysis effort. The price paid for this enhanced ability is a slight degradation in CPU performance (less than 5%), primarily resulting from enforcing clean interfaces between the model and these new modules. This is a small price to pay for the savings in human resources that are realized.

5. Performance on parallel systems

One of the optimization dimensions of Table 1 is the ability to execute the model in production mode on multiple processors. This feature is essential to avoid limiting the numerical experiments which may

be undertaken and the platforms upon which the model can be run.

Two major changes were necessary to allow the code, which had been running on a single processor, to be run on multiple CPUs of the Cray Research PVP machine. The original unitasked code executed the model one latitude row at a time (the chunk loop of Fig. 1) because of memory constraints, marching from south to north, providing a natural coarse-grained, one-dimensional data decomposition scheme for parallel execution. In unitasked mode, this decomposition allowed the center and northern row variable fields and the fluxes that had been calculated at the northern boundary of a grid row to be saved and used as the southern and center row variable fields and southern boundary flux of the next row. Thus each new row required only the reading of the new northern row of data from disk and the calculation of the northern boundary fluxes. However,with multitasked execution, each processor must calculate fluxes at both boundaries, and read all the data it needs from disc, since it is not certain that it will have the data from the previous row. These two changes result in a 10–15% increase in CPU time for SKYHI, but allow the model to be run on parallel systems.

The scaling performance of SKYHI on the GFDL Cray Research T932 machine during a dedicated test time is summarized in Table 3 for both one degree latitude (N90) and for one-third degree latitude (N270) resolution. These numbers were obtained by running the model for several timesteps without archiving any data, and do not include the time spent in the initialization section of the model, which is primarily spent reading the initial data and is not parallelized. It is seen that scaling for both resolutions deteriorates above 9 processors. This decay likely reflects the single-threaded nature of access to the model data stored on the solid-state storage device (SSD), meaning that as the number of processors increases, processors must wait longer to get the data they need. A further degradation of performance occurs on 24 CPUs, which presents an inherent load balancing problem for 180 or 540 chunks, since some CPUs have more chunks to integrate than do others.

Thus, on the Cray Research PVP machine, a one-dimensional domain decomposition is adequate, since the number of processors which can be efficiently used on a problem will be limited by the single-threaded SSD access time before any load balancing problem resulting from the limited number of latitude rows becomes important. Currently only the one-third degree latitude SKYHI experiment is being run multitasked in production mode on the GFDL T932; lower resolution runs are run unitasked to take advantage of the CPU savings discussed above. The one-third degree experiment is multitasked 4 ways; on 4 processors the code is still parallel efficient and the machine resource usage is balanced. The model at this resolution requires 12% of the total system memory and 15% of the total system SSD, so that 4 processors (15% of the total of 26) is reasonable.

Table 3
SKYHI PVP scaling characteristics

Resolution	Number of processors	Wall clock time (seconds)	Scaling	Parallel Efficiency
N90	1 (unitasked)	334	–	–
N90	1	369	1.00	1.000
N90	3	125	2.95	0.983
N90	9	44	8.39	0.932
N90	18	26	14.19	0.783
N90	24	24	15.38	0.641
N270	1 (unitasked)	267	–	–
N270	1	308	1.00	1.000
N270	3	104	2.96	0.987
N270	9	35	8.80	0.978
N270	18	23	13.39	0.744
N270	24	23	13.39	0.558

The SKYHI code running on the Cray Research PVP machine has also been modified to run on the Cray Research T3E distributed memory machine. Preprocessor options are used to select either the shared memory or distributed memory version of the code. These code versions differ by about 1200 lines, primarily in code involving the T3E domain decomposition and assignment of data to processors, the storage of the time-dependent grid point data in local memory rather than on the SSD, communication of data between processors (shmem calls), and the data archiving process. The code remains that which has been optimized for the vector machine; no specific T3E optimizations have been made. As a result, raw performance of this code on the T3E is about 36 Mflops/pe, compared to 500 Mflops on a single T90 CPU.

The major source of load imbalance in SKYHI is the polar Fourier filter. All latitude rows which are filtered take about the same time to execute, as do latitude rows which are not filtered, with the difference in time between filtered and non-filtered rows being about 15%. This information may be used in deciding how to assign latitude rows to processors, and so better balance the load across processors, in contrast to a simple round-robin assignment of rows to processors.

Table 4
SKYHI T3E scaling characteristics

Number of processors	Wallclock Time (seconds)	Scaling	Parallel Efficiency
15	266	15.0	1.000
30	135	29.55	0.985
45	93	42.9	0.953
60	72	55.42	0.924

For example, the best balanced load for an experiment with 180 latitudes was obtained using eight processors (in contrast to nine, ten or twelve), even though the eight processors were not all responsible for the same number of latitude rows. However, as the number of processors approaches the number of latitude rows, the ability to balance the load decreases. Ultimately, it is essential that a two-dimensional decomposition be employed, to allow better load balancing on a greater number of processors.

Scaling results for a one-degree latitude version of SKYHI with 160 vertical levels run on the 512 processor T3E at the National Energy Research Scientific Computing Center (NERSC) are shown in Table 4. As in Table 3, these results are from several timesteps of integration, without archiving data, and do not include the time spent in the initialization section of the model. This resolution requires a minimum of fifteen 32-Mw T3E processors in order to have enough memory per pe to integrate the model with a one-dimensional domain decomposition. The degradation of performance with increasing number of processors seen here reflects the reduction in the ability to balance the load as the number of processors approaches the number of chunks of parallel work (180); the efficient use of a greater number of processors requires a two-dimensional domain decomposition.

6. Conclusions

The increase in the relative value of human resources compared to machine resources at GFDL in recent years means that the definition of code optimization must be changed to include human factors in addition to the traditional machine resource usage. A meteorological model used in research must be structured so that investigators not familiar with the details and history of the model may quickly learn enough about it in order to use it productively in their scientific research. This user-friendliness will usually come at the expense of machine performance, a condition which must be accepted in order to optimize the total scientific productivity of the model and of the scientists who use it.

A user-friendly model requires at a minimum that the code is "modular", meaning that the different model processes communicate with the rest of the model in clearly specified ways, as opposed to being intertwined. In this way investigators may easily examine individual parts of the model, without having to extract the process of interest from a dense ball of code, a process which usually proves to be both difficult and time-consuming, even for those experienced in handling the code.

The restructured GFDL SKYHI general circulation model has also addressed two specific topics which in the past have inhibited investigators in their productive use of SKYHI; the ability to easily define new data files for later off-line analysis of the model output, and the ability to easily add new variables to the model for specific investigations. The archive module and user variable module described here have a standard format and are flexible to user needs. These packages will continue to evolve in response to user desires and complaints, becoming even more user-friendly over time. Inclusion of Fortran 90 constructs in these packages should result in some performance improvements and ultimately cleaner code, albeit code which will look less familiar to the current user community.

SKYHI has been successfully integrated in production mode on the Cray Research T932 PVP machine in both unitasked and multitasked modes using one-dimensional domain decomposition. Parallel performance scales well with number of processors (if obvious load imbalance configurations are avoided) to the point where the single-threaded nature of SSD access limits performance. The same PVP-friendly source has been successfully run in production mode on the Cray Research T3E machine, also using one-dimensional decomposition.

At this time, three major areas are negatively impacting the performance of SKYHI on parallel systems. Single-processor performance is an issue; whether improvements in cache size and system software and utilities will improve performance significantly or whether major code redesign is essential remains unclear. Motivation for such code redesign will naturally increase when SKYHI is no longer primarily executed on a PVP machine, thus allowing cache-friendly coding techniques to replace the vector-friendly techniques now in place. The lack of parallel I/O significantly reduces the scaling efficiency shown in Table 4 in production runs; when data are read or written, a single processor does the i/o while the remaining processors wait. Fi-

nally, the need for a two-dimensional domain decomposition to produce more, smaller chunks and therefore permit better load balancing is obvious as the number of processors employed on a problem increases. The longitudinal data dependencies in SKYHI which must be handled in order to allow such a decomposition have been identified, and it remains to develop a mechanism to deal with them, within the context of a user-friendly model.

Acknowledgments

I would like to thank S. Fan, C. Kerr, J. Mahlman and D. Schwarzkopf for reading the paper and offering valuable comments. Thanks also to the SKYHI user community who have provided me with the feedback necessary to produce a model which is becoming more user-friendly, and to J. Mahlman who recognized the need for a user-friendly SKYHI and who provided me the opportunity to work to that end.

Appendix A: Temporal control of archive file forms

Six variables are used to control the temporal characteristics of the archive file forms:

(1) the number of times which data is to be written to a file before closing the file and opening another one;
(2) the number of seconds between writes to the file;
(3) the number of time levels that are to be averaged to generate the data that is to be written;
(4) the amount of time to be counted toward the number of seconds between file writes at the beginning of the run;
(5) the time in the run at which the file clock is to start;
(6) the time in the run at which the file clock is to stop.

Variable (1) allows one to write multiple files of a given file form during a job. File names are created automatically following a simple pattern. By making the value of (2) larger than the length of the job, the writing of the particular file form may be turned off. The mechanism to define the time-averaging characteristics are defined by (3); if a snapshot file is wanted, then variable (3) is set to 1. Otherwise, the combination of (2) and (3) determine the frequency of data sampling. Variables (5) and (6) allow one to write a file during a specified period of an integration, and (4) provides a means to write files at the frequency given by (2), but with an offset in time from the start of the job. These six variables are named in a consistent way for each of the file types, differing only by a single letter, which indicates the file type.

Appendix B: Suggested procedure to use the archiving module

The following process is recommended when using the archiving module:

I) Decide which archive files are to be written during the experiment and define their desired characteristics. The characteristics are found in the namelist and parameter file associated with the given file type.
II) For each file type, choose one of the following options:

A) If no files of this type are desired:

1) Set the preprocessor variable defining the number of file forms of that type to zero.
2) Remove the namelist variables associated with that file type from the namelist.

B) If you desire either a subset or all of the default file forms:

1) Leave the preprocessor file form number variable at the default value.
2) For those file forms that are not being written:

a) Set the seconds between file writes to be larger than the run time of the job.
b) Set variable (3) of Appendix A to be 1, reflecting a snapshot file.
c) Set the spatial index parameters for the file form to all be 1, thus setting the size of the array which will hold the data to be of length 1.

3) For those file forms to be written:

a) Set the namelist and parameter file to contain the desired file characteristics.
b) If the file form containing a subset of the standard file is to be written, modify the subroutine defining the file contents so that it will contain the desired variables.

C) If a new file form is to be added:

1) Set the preprocessor file form number variable to the proper value.
2) Add the code to define the new file form, following the existing code pattern. Code mods must be made to six to ten source files, dependent on the file type.
3) The mods involve duplicating code and changing file form numbers in the variable names to the number of the new form. Additionally the user must define the variable names to be placed in the file, following the provided patterns.
4) Modification to the script will be necessary to save any new files generated and to assign the file characteristics, if desired.

References

[1] Fels, S.B., J.D. Mahlman, M.D. Schwarzkopf and R.W. Sinclair, Stratospheric sensitivity to perturbations in ozone and carbon dioxide: radiative and dynamical response, *J. Atmos. Sci.* **37** (1980), 2265–2297.

[2] Jones, P.W., C.L. Kerr and R.S. Hemler, Practical considerations in development of a parallel SKYHI general circulation model, *Parallel Computing* **21** (1995), 1677–1694.

[3] Hamilton, K., R.J. Wilson, J.D. Mahlman and L.J. Umscheid, Climatology of the SKYHI Troposphere-Stratosphere-Mesosphere general circulation model, *J. Atmos. Sci.* **52** (1995), 5–43.