A numerically intensive computing environment: IBM 3090 and the PS/2 Model 80

by R. F. Arnold
P. Halpern
G. R. Hogsett
B. T. Straka
C. Arasmith
J. McElroy

Recent advances in personal computer workstations, such as the IBM Personal System/2[1] Model 80 with its increased memory and CPU speed, loosely coupled with a host IBM 3090[2] Processor, can provide considerable computing advantages for executing and visualizing numerically intensive computing (NIC) applications. We have developed a prototype visualization environment which demonstrates effective use of this hardware. The user interface for the NIC application is written using Microsoft Windows[3] on the PS/2[1] Model 80 running DOS 3.3. The PS/2 Model 80 is connected to a host 3090 via a PC network. The user enters requests which are application parameters and selects graphic views for displaying the output results file. The entries are made through user dialog screens on the workstations. The user view of the system is such that it appears that it is running on the workstation. To achieve this transparency, file caches are used on both the workstation and the host. The cache on the host is in the form of graphic metafiles and numeric data. The cache on the workstation contains metafiles. Requests are monitored on the workstation to determine whether the results are in the local cache. When they are not, a request file is transferred to the host and checked against the host cache. The NIC application is run only when the requested result is not in either cache. In order to reduce the file size, the results file is converted to a metafile before being transferred to the workstation.

[1] Personal System/2 and PS/2 are registered trademarks of International Business Machines Corporation.

[2] 3090 is a trademark of International Business Machines Corporation.

[3] Windows is a trademark of Microsoft Corporation.

Introduction

The traditional IBM hardware configuration used for numerically intensive computing (NIC) consists of a mainframe or host CPU housed in a central computer facility often referred to as a glass house. Connected to the mainframe are a mix of IBM terminals which support text (3270) and graphic output (3279, 3179). Special high-resolution graphics display devices (e.g., the IBM 6090 Graphics System) may also be attached. The connection to the host for all of these devices is primarily through coaxial cable as terminals. The user interacts with the host-resident NIC application using input files. The user interface is primitive, if it exists at all, and is text-oriented. Output, which can be text or graphics, is viewed from files stored on the host. The viewing of output is based on computational demand.

This paper presents an alternative design and implementation for a NIC system which uses a workstation loosely connected to a host. In our system, program editing and compiling are accomplished traditionally. The user interface is graphically presented on a workstation and is organized around data viewing. Algorithmic as well as graphics-related computation is done on the host. However, this computation is subordinated to viewing and occurs implicitly on a demand basis. The application is "preloaded" with enough sample data and partial results that the user can postpone learning how to input or edit parameters and manage resources until familiarity with the viewing interface is attained and its relation to the underlying mathematics and physics of the problem is understood. By such a judicious splitting of application and user interface between workstation and host, an improved NIC working environment is achieved.

• *Advantages of accessing NIC through a workstation*
The advent of the personal computer (PC) accompanied by an original set of applications capable of reaching a set of new users has caused a revolution in the mix of software and hardware solutions available. The NIC user has been exposed to, and in many cases has become familiar with, PCs and PC applications such as word processors and spreadsheet programs. Thus, in terms of usage, the NIC user is being exposed to more responsive, better designed, and easier-to-use application interfaces on the workstation than was true on the 3270 terminal family. There is also much greater application development activity, and a larger number of productivity aids available for workstations than for the 3270 family.

Workstation user interfaces tend to be more visually oriented because data can be moved to the display device at lower expense. The application-specific control transactions have much better response times. As the workstation technology has proceeded, large local memory, high-resolution displays, and special graphics functions such as zoom, pan, and scroll have made local interpretation of data more efficient.

The NIC user working with the host through a workstation has greater flexibility in the placement of attachments, since the workstation requires a much lower connection bandwidth than a terminal. For example, a workstation can operate over a telephone line, compensating for the poor bandwidth by increasing amounts of local storage, computing power, and memory. In the extreme case, the computation may be run completely on the workstation for scaled-down problems suitable for testing of model assumptions and debugging. There also exists a large installed base of both stand-alone and networked workstations running applications which are unavailable on a host.

• *User interface and application function levels*
Structuring of applications on a workstation stresses current user interfaces. If the user must directly confront the multitude of files and file types along with the synchronization problems that NIC applications generate, he is likely to quickly abandon the effort. The application interface described here is structured and presented in three levels of increasing complexity. The user is encouraged to gain familiarity with each level before proceeding to the next. Upon starting the application, the user has the opportunity to explore graphic output and geometric input data of "previously run" instances of the problem. This is perceived as data viewing and navigation. The major portion of this for the application we consider is a group of geometric and temporarily organized graphic views of computed data. Our prototype application produces contour plots of pollutant concentrations.

The user can then explore the mathematics and physics of the problem by modifying the model and viewing parameter values. He can generate additional parameter sets, or he may establish new sets of initial conditions. After any of these changes, he can compare results. All of this is possible without the user directly encountering the file management activity being carried out on his behalf on both host and workstation.

The underlying driving mechanism for this system is the data-viewing facility. Computation is done only "on demand." But such systems, like storage hierarchies, need tuning and guidance. The approach used to achieve this is part of the "resource management" component of the interface. Extensive running of the program results in increasing amounts of disk storage allocated on its behalf both on the host and workstation machines. Procedures must be available to determine what is to be saved and what should be destroyed. However, the user can wait to learn any of this until he is thoroughly familiar with the first two layers of function.

• *System/application interface split*
The user's interface to the computer is traditionally split into a single operating system (OS) interface and some

number of application-specific interfaces. Examples of these OS interfaces include the VM/CMS and PC-DOS [1] command lines as well as the many full-screen interfaces such as Filelist and TopView [2]. The relationship between the system interface and the application interface can be quite straightforward. The application typically deals with a single file of a specified type. Its output is also a file of that same type. In the simplest case, the application program itself consists of a single nonmodifiable file. Most of the information in full-screen OS interfaces is a reflection of the state of the file system and the names, sizes, and dates of the files. However, the more sophisticated interface programs keep additional information, such as short command strings or affinity information, which allow a program to be associated with a specific file type. Additional complexity and function occur when the system interface allows for multitasking with user-controlled data sharing between tasks. In fact, few applications consist of a single program file. Even simple applications usually have one or more small files which are used to perform various configuration or personalization tasks when the application is started. Unless the designer is careful, he can make the learning of a new application quite difficult.

Applications which involve communications between operating systems increase the complexity by an order of magnitude or more. The number of file types introduced to manage function and performance is greatly expanded. Furthermore, the simple operating-system interface with which the user is familiar is likely to be inadequate. "Dumb terminal" emulation programs such as PROCOMM-PLUS[4] [3] or E-78 [4] allow the user to maintain his sanity by keeping almost all his data on one system (the remote one) and using a mostly stateless window to interact with that system. Such a strategy is practical, however, only when the effective screen bandwidth required by the application is not much larger than the bandwidth of the communications link. In this prototype, where the interest is graphics or image output with relatively low-bandwidth communications, this strategy doesn't work. We are faced with a situation in which there may be user data in several different formats, residing partially on one system and partially on another. Keeping track of all this with existing system interface tools is too difficult for the intended users of the prototype application. Operations which become very confusing are version management, checkpoint, and recovery. The simple file-management strategies associated with a text editor or a spreadsheet program are not possible when all the files are directly exposed to the user through the traditional file-management interfaces.

Our solution is to design the prototype application to look like a completely local application. Each instance of the application is represented by a single file. All other temporary or permanent files are subordinated to this one file. All resources, local or remote, are allocated on behalf of this file. For example, in the prototype the diffusion model program is called "DIFFUSE.EXE." It expects as an argument a control file of type DFU, say "SANJOSE.DFU."

The approach is to define a .DFU-type file which contains all the state information about a specific application instance. This includes information such as parameter sets, partial computation results, host server path information, and auxiliary data file names both on the workstation and on the host server. The point is to provide, in a single file, everything necessary to initiate, save, back up, or destroy an application instance. This strategy results in an interface that is both easier to learn and easier to use.

• *System concept and its implementation in NIC*
The system organization underneath the proposed structure is implicit in the description of the user interface. The several activities involved in writing and running an application are ordered with the hope of making the whole process easier for both the application writer and the user. For a user of the diffusion program (the example described later in this paper), it might be quite natural to start out with a three-dimensional visualization of what the pollutant concentrations might look like. From the point of view of a computer programmer, this would be distinctly unnatural; the visualization, if any, comes at the end of the process, not the beginning.

The two main successes in user interfaces for personal computing, WYSIWYG ("What you see is what you get") word processors and spreadsheets, were achieved by disturbing the "natural" order to put the end results up front in visual form. We intend here to pursue a similar strategy for NIC problems.

A program which achieved a breakthrough in usability by several techniques was VisiCalc[5] [5]. Here program editing, data entry, and output viewing are integrated into a single geometric model. The program structuring was greatly simplified by providing a fixed initial screen and keyboard monitor and completely eliminating the primary role of loop control logic. Loops perform two crucial functions—saving memory and permitting concise program statements. How spreadsheet programs avoid the use of loops at the programming interface requires a detailed explanation which is beyond the scope of this paper. A key piece of the solution, the "copy"

[4] Datastorm Technologies, Inc., 1987/1988.

[5] VisiCalc is a registered trademark of Personal Software, Inc. (VisiCorp.).

command, moves the function done by loop control variables into the program edit function. Using high-level computational primitives and subordinating computation to program editing also reduces the burden of the program structure.

These changes from traditional organization require the recognition that computing resources devoted to program editing, input, output, and other "support" functions far outweigh those required for execution of the user's algorithms.

For many NIC applications, the "support" functions also greatly overshadow the MIPS (millions of instructions per second) requirements for the algorithms. In many NIC applications the main MIPS-consuming item is visual output, which is the same as for spreadsheets. Although the resource ratios of these types of work may be similar to those for spreadsheets, there are some important differences that keep us from using the spreadsheet strategy directly.

While a spreadsheet-like geometric model of input and result data often exists, the complexity of the program at each point is such that a procedural language rather than just a collection of formulas is desirable for algorithm expression. The total amount of computation for all phases of the problem is much larger per user viewing event. The algorithmic computation cannot be done at edit time, and visual output computations seem too large to be done at viewing time. However, these differences do not make it less desirable to provide the friendly development and execution environment of a spreadsheet—they just make it more difficult. The components of a solution for the NIC problem include the use of decoupled MIP servers and exploitation of the high-performance direct-access storage available with them, along with the additional step of integrating the control of graphic and image output with the early steps of the process. Spreadsheets integrate data editing, program editing, algorithm computation, and numerical output viewing. Although they usually provide graphic output, the control of this function is not integrated. In a spreadsheet program, the user must selectively request all visual output after the algorithmic computations are completed. Each image is independently generated and discarded. Little attempt is made to understand or exploit the user's image-reuse behavior in general. Once one considers the integration of visual output with the remainder of the process, it becomes obvious that the user's interface to visual output control should pace as much of the entire process as possible. This is because it is closest to the most computationally intensive portion of the task, and it is also the point at which the greatest intelligence exists about user intentions. Furthermore, the user interface should be presented in a manner that allows for multiple instances of the application to run. This would at least include the possibility of multiple sets of parameters, but might also include changes in the algorithms themselves.

When the user "manually" deals with his files using the file management system under VM/SP[6] HPO [6] for example, he is presented with an unstructured mess of unclassified partial results and several different file types, generally with no well-thought-out file-naming scheme. Subordination of this detail was mentioned previously, when we suggested collecting everything into one file as seen from the user's point of view. A second reason for transparency relates to performance. It is only from a sufficiently broad point of control that significant computational repetition can be detected and avoided. A principal technique in solving data-intensive performance problems has been transparently managed staging. Expensive data accesses are eliminated by providing for easy reuse of data that have already been accessed. The same idea can be applied to computation. In our work, program editing and compiling are done traditionally. The user interface is organized around data viewing. Algorithmic as well as graphics-related computation is subordinated to viewing and occurs implicitly on a demand basis.

Representative NIC problem

The class of NIC problems which we chose to investigate initially includes those which had previously been developed and run in a batch or semi-batch mode. These tended to be primarily FORTRAN programs written for execution on a System/370[7] class machine. The user interacted with the host via a terminal (3277–3279) over a coaxial cable. The FORTRAN module required input and output files to be resident on the host, and the user kept track of all the necessary files. File management was primitive, often consisting of the user keeping notes as to which files were being used for particular runs of the NIC module and associating output files with parameters used.

NIC FORTRAN problems traditionally have a number of common characteristics. A large percentage of the instructions in the computation kernel are floating-point. Many computations require double-precision floating-point as well. The NIC programs require significant input data and produce vast amounts of output results. However, only a small part of the output may ever be reviewed, because the user is interested in obtaining some specialized insight from a small segment of the output. To obtain the desired information, the NIC computation

[6] VM/SP is a trademark of International Business Machines Corporation.

[7] System/370 is a trademark of International Business Machines Corporation.

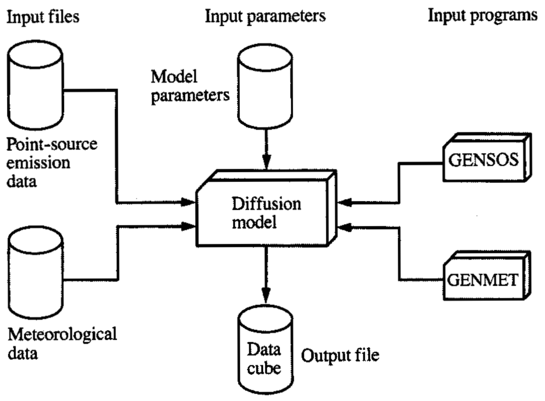


Figure 1

Diffusion model input/output flow.

is often executed repetitively until it reaches the state of interest to the user. This implicitly demands that NIC computations have access to large volumes of storage and memory. Once the NIC application is initiated by the user, there is little if any operating system interaction. The output results are usually analyzed off-line via a set of postprocessing programs, which perform the function of presenting to the user a visualization of the output. This usually takes the form of a graphic or pictorial representation of the results. After this is done, only the reviewed portion is retained, and any excess results can be and usually are discarded.

• *Atmospheric diffusion model*

We have selected the atmospheric diffusion of pollutants as our test application. The application mathematically models the solution of the time-dependent mass-conservation equation expressed in terms of concentration density values and transport variables.

Model formulation is based on numerical integration of the concentration equation over time, for a specified set of spatial mesh points which comprise a three-dimensional grid. This enables the temporal and spatial variation of meteorological variables and surface conditions to be accounted for in the model. However, local topographic features are not modeled.

• *Diffusion model input/output*

The values of concentration are obtained by numerically solving the mass conservation equation using finite-difference approximations. The model requires meteorological input data, such as wind velocity, to be supplied at each grid point of the three-dimensional spatial grid. The meteorological values supplied can be time-dependent, i.e., varying at each spatial grid point. The numerical solution of the finite-difference equations is obtained by marching in time steps of hundreds of seconds. The concentration levels at hourly intervals are saved to disk. This is done to permit comparisons of computed values with hourly-averaged observed data.

The finite-difference spacing or mesh size is normally fixed in the X (latitude), Y (longitude), and Z (vertical) directions. The horizontal grid spacings in the X and Y directions are normally uniform or fixed in size, and are typically composed of 40 and 30 grid points, respectively. The Z grid spacing can be variable and usually has 16 grid points. These values represent defaults which can be increased or decreased as desired.

The diffusion model requires meteorological and source emission data as input. Since the model is time-dependent, both of these sets of data can also be functions of time. The source data consist of point-source data (e.g., power plants) and area-source data (e.g., residential sources). Our implementation considers only point-source emissions. A most common format for the input data is to vary the source data every hour. The meteorological data are similarly required to vary every hour. In addition, the meteorological parameters are required at every grid point for every hour. The concentration values are simulated every hour over all the spatial grid points. Thus, the concentration data are in the form of a cube, and one cube is produced per hour of simulation. A typical model run would be for 24 hours of simulation time.

As is typical of many numerically intensive computations, the diffusion model has input in the form of files or programs which create the input data. A schematic representation of the input and output data required for the diffusion model is shown in **Figure 1**. The input for both meteorological and source emission data can be either obtained from observational information or created by executing the programs GENMET or GENSOS. These programs, which were written for this prototype study, simply create hypothetical fields which are useful for sensitivity studies of the model and for analyzing the effect of model parameters on the computed results. A third source for meteorological data is yet another model, which produces meteorological data. Such a model attempts to represent a dynamically consistent set of variables by solving the equations of motion using numerical approximations.

The output file from this model becomes the input to the diffusion model.

Originally coded in 1974, the diffusion model [7], as such, qualifies as a "dusty deck"—an application written for systems no longer in common use but capable of being run or emulated on current systems without significant changes in the original code. It was executed in batch mode under VM/CMS running on a System/360[8] Model 168. The model computed one data cube for every hour of simulated time.

Each hour of concentration data was stored in separate, sequential, formatted files. These were modified so that any one simulation run was stored on a direct-access unformatted file, thus reducing the number of output files. These data files compose the results data cache on the host.

The diffusion model currently runs on an IBM 3090 Model 300 Processor with Vector Facility [8]. The CPU time for computing 24 cubes of data is two minutes. The total size for these cubes is 4.7 megabytes of data.

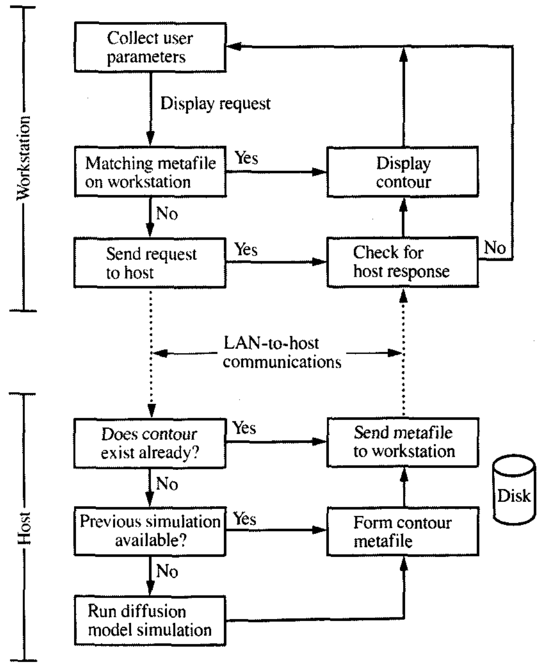
To obtain a visual representation of the output, the NCAR graphics package [9] is used to produce two-dimensional contours of concentration. The graphic representation of *XY*, *XZ*, and *YZ* sections through the data cube is converted to graphic metafiles. The total CPU time for creating metafile contour plots for 24 hours of data is 92 minutes. The total size of the files generated is 8.6 megabytes of data. Thus, we conclude that the main computational and storage requirement is generated by graphics postprocessing. There are two main problems associated with the host implementation: The user has a significant file-management responsibility forced on him, and the rapid interaction with simulation output is infeasible even on a 3090 processor because of severe performance limitations, even if all cross sections in a particular cube of data are not computed.

Implementation of solution

• *Host and workstation system overview*

Figure 2 shows the overall flow for the system. The heart of the design is the split between the user interface and the computational portions of the application. The user interface code is resident on the workstation. Its primary function is to collect user parameters for running instances of the application and for viewing output. The user-entered requests are of two basic types—model parameters and viewing parameters.

The new user initially gravitates toward varying the viewing parameters and displaying previously computed results. Experienced users change both model parameters and viewing parameters. The selected output is stored on



Workstation
Collect user parameters
Display request
Matching metafile on workstation
Yes
Display contour
No
Send request to host
Yes
Check for host response
No
LAN-to-host communications
Host
Does contour exist already?
Yes
Send metafile to workstation
Disk
No
Previous simulation available?
Yes
Form contour metafile
No
Run diffusion model simulation

Figure 2

Overall system diagram.

either the workstation or the host, depending on whether the particular view has been previously displayed. Output is created and stored on the host in raw data format and in graphic metafiles. A metafile is a standardized file for storing and transporting graphic data and control information in device-independent form. We chose the computer graphic metafile format [10] to transfer from the host to the workstation because of the device-independent output, small file size, and speed of interpretation. Thus there is a cache of previously displayed metafiles on the workstation. As shown in Figure 2, the user's display requests are compared against a table describing the metafiles in the workstation cache. When a match is found, the desired contour is displayed. When no match is found, the request file is sent to the host. The request file contains both viewing and model parameters. The workstation–host connection is a local area network. The file is received at the host and read into a disconnected VM/CMS virtual machine. Once the file has been read by the host, its data manager (DM) is activated. The DM checks whether the requested view already exists. If the view is in the metafile cache, it is

[8] System/360 is a trademark of International Business Machines Corporation.

sent to the workstation. The workstation meanwhile is polling for host response in the form of a sent metafile. When the view does not exist in metafile form, the DM searches the data cubes. When a match is found as to model parameters, a metafile is created with the specified view and sent to the workstation. If no match is found, the application (in our case the diffusion model) is executed, and a data cube and the requested metafile are produced and sent to the workstation.

- *Host virtual machine configuration*

The current host for our test application is an IBM 3090 Model 300 Processor running VM/XA[9] SP1 [11] and a guest system, VM/SP HPO 4.2 [6]. The diffusion model and all its associate input files are resident in a virtual machine, which also contains the data cube and metafile caches. The virtual machine runs in disconnected mode using the GONE EXEC program, an IBM internal-use-only facility which permits a virtual machine to be disconnected from VM yet have messages trapped and written to a log file. It also provides a user exit which is called by GONE whenever a reader file is received. The user exit is a function called PROCFILE EXEC written in REXX. This user exit parses the file type of the incoming file searching for a type which indicates that it is a workstation request file. Upon finding a file of that type, the user exit copies the file onto a designated disk. The request file is now ready to be used by the DM. The DM is executed using another EXEC called RUN6. After RUN6 finishes, the user exit code sends the result metafile to the requesting workstation. The user exit then returns to GONE, which waits for the next reader file. The identification of the sending workstation is included in the request file and is used in returning the result metafile. Thus, the host server may serve several workstations in a serial manner.

- *Data manager, data cube, and metafile cache*

The function of the data manager (DM) is to act as the file manager for the two sets of data caches on the host. The DM is written in the C language. This choice was made because of the availability of C on both the workstation and host, and the need to replicate a large portion of the DM on the workstation. The diffusion model, when requested, produces a results file which contains concentration data for each spatial grid point as a function of time. This is a direct-access data file that contains not only the concentration data but the set of model parameters which produced the data. Such files constitute the data cube cache. One file can contain many cubes of data, each for a different time but with the same model parameters. The metafile cache consists of files each containing a view of a cube that has previously been requested by the user via the request list. The views are themselves functions of the geometric section of a cube, for example an *XY* cross section.

- *Workstation-to-host communications*

The communication for the prototype takes place "behind the scenes" so that the user has the impression that the entire application is running on the workstation. The workstation–host communication assumes the availability of a narrow-bandwidth communication mechanism. A telephone line or narrow-bandwidth network is commonly what NIC users have available to access a remote host.

In the prototype, the request files and metafiles are sent and received over an IBM PC Network [12]. The workstation program uses a library of routines, WECOMM, which were developed to interface with an internal package, PVMCOMM, that interfaces with NETBIOS and the PC Network adapter. The PVMCOMM program uses the PASSTHRU facility (PVM) and the Inter-User Communication Vehicle (IUCV) to communicate with a disconnected CMS virtual machine named NET2LAN. The purpose of this disconnected machine is to forward files between any virtual machine on the host and any PC on the PC Network. In the case of our prototype, the request files from one or more workstations are forwarded to a disconnected virtual machine, NICSERV, which runs the diffusion model simulation. NICSERV then receives and processes the requests and sends the results back to the appropriate workstation.

When a request file is sent, a timer interrupt process is initiated on the workstation. The timer process periodically causes the user interface to query the network for results. Upon receipt of a metafile, the user interface updates the local cache. The metafile is displayed if the viewing window is still active. If the user is currently analyzing other views, the new metafile is placed in the cache but not displayed.

- *User interface development and components*

The system concepts and rationale for using a graphically based interface have previously been discussed. We have chosen Microsoft Windows [13] as the facility to create the user interface on the workstation. The user interface is a set of screens which contain text and data fields defining the input parameters for the diffusion model. Through these screens the user interacts with the model, requesting model runs, results, and views of these results.

Windows is a facility which runs as a shell around PC-DOS [1]. User interfaces written in Windows are multitasking. The user can display and run several Windows programs or multiple instances of the same

[9] VM/XA is a trademark of International Business Machines Corporation.

program simultaneously. The Windows screen, because it is a graphic interface, permits a user interface to have both text and graphic representation. Windows programs will run on any of the standard displays for which there is a Windows device driver available, and such a driver is device-independent.

The structure of Windows is such that, while it runs as a shell around PC-DOS [1], it shares with DOS the managing of hardware resources. It controls the display, keyboard, mouse, printer, and serial ports. From the software aspect, it directs the memory management, execution, and scheduling of the application. The interface itself is a Windows application. It is written using some of the more than 450 function calls which comprise the Windows programming facility.

The user interface for the prototype uses the recommended directory substructure of Microsoft C [14] and Windows Development System [15]. There are four subdirectories, BIN, INCLUDE, LIB, and WINDOWS, within the root directory, along with the expected contents. The LIB and INCLUDE directories each have an additional subdirectory, APP, which holds application-specific library and "include" code. In our case, other application-specific code and data are stored in the METAPOOL, WINUTIL, DIFF, and DIFFCOM directories.

It has been our intent to make the code as application-independent as possible. Code judged to be completely application-independent is placed in the subdirectory WINUTIL and managed as a library. Application-specific code is divided into two main file sets, DIFFCOMM and DIFF. DIFFCOMM contains the host communications, as well as the staging code for the workstation metafile cache. A goal is to make this code increasingly independent of the specific details of the diffusion model, so that it can eventually be shared by different models, all of which use, however, the same host/workstation staging strategy. The second application code, DIFF, contains code and data files specific to the diffusion model itself. The DIFF and DIFFCOMM programs define and use two additional data resources, one for the support of the Data Entry Dialog (DEDialog) facility (described below) and a second for "Help" text. The METAPOOL directory contains the metafile cache on the workstation.

The WINUTIL library is composed of a DEDialog file interface and metafile interpreter modules. The normal Windows development procedure involves writing a separate message-handling "window procedure" for each screen. To simplify the development effort, we have built a general screen-handling facility which is data-driven and allows the developer to substitute a relatively small data structure for those procedures in which the screens are mostly for parameter entry. The code to support this is called "DEDialog." For situations where the function provided by DEDialog is inadequate, it is still possible to use the applicable portion of it by "piggybacking" one's own message handler on the default one.

The interface to the user session files (DFMs) consists of code which is common to several of the sample Windows programs. It involves picking a file from a list to load, directory navigation, etc. A version of this code has been packaged as a modular unit.

The current state of our system downloads and "plays" (interprets and displays the images from) Computer Graphics Metafiles (CGM, an ANSI standard). The metafiles are created on the host using a CGM driver from the NCAR graphics package [9]. The host driver creates a binary encoded form of the CGM standard. The interpreter on the workstation translates the graphic orders into Windows GDI calls. This code is independent of the diffusion model. The speed of interpretation, file size, and device independence of the CGM have suggested this to be a reasonable approach to providing the filled contour graphs.

The Windows development system provides a powerful general-purpose method for the development, linking, and use of static data. The tool that provides this capability is the resource compiler. It makes use of static data of many kinds, which usually include dialog box definitions, icon bit maps, menu definitions, and ASCII text, at the minimum. The DIFF system uses this mechanism to establish dialog box definitions, icons, bit map symbols, menu definitions, and text.

The program data start out in many small files. The resource compiler is called to process an ASCII source resource file which consists of lists of external data files, their resource type, and the internal symbols by which they will be identified. The resource file may also contain definitions for some number of the built-in Windows resources. The result is a single binary file which consists of all of the data resources along with an internal directory. After the normal link-edit step, the resource compiler is called again with a special parameter which causes this binary resource file to be appended to the normal .EXE file.

At execution time, library routines will dynamically load and locate these data resources on an as-needed basis. The space occupied by the resource data can be shared and recovered in a largely transparent manner. This results in much smaller actual memory requirements than if the same data were statically linked with the program, and much better performance than would be possible if the data were accessed from a separate disk file every time they were needed.

• *User's view of the diffusion model environment*

In order to run the diffusion model the user must have, in addition to an installed version of Microsoft Windows

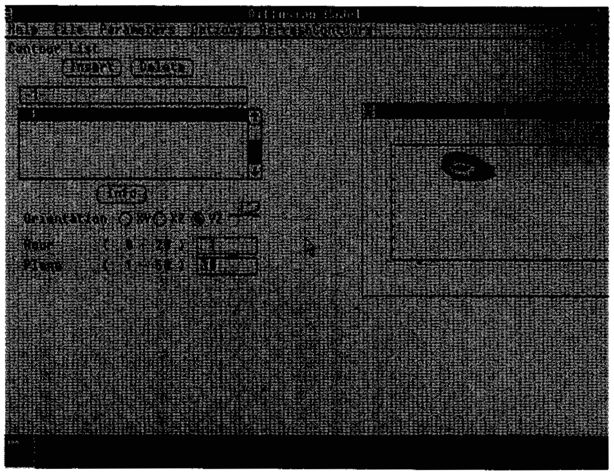


Figure 3

Main diffusion model screen.

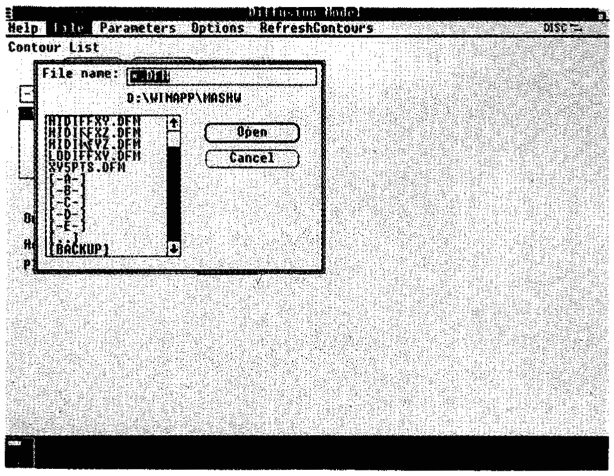


Figure 4

Menu of previously saved session files.

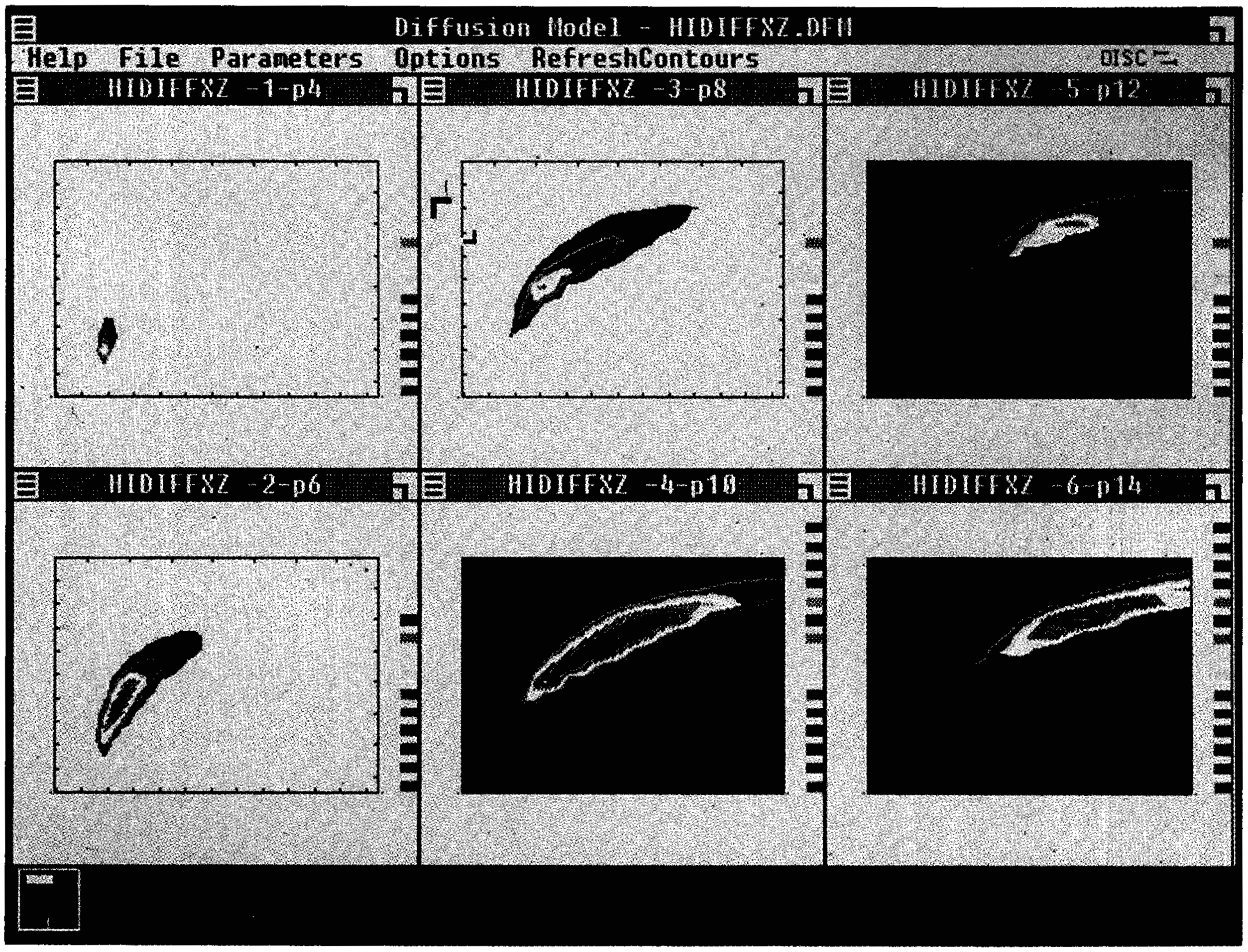


Figure 5

Contours displayed from a selected session file.

[13], the two executable files DIFF.EXE and DIFFCOMM.EXE and a metafile cache. The first execution of the program DIFF.EXE starts the auxiliary program DIFFCOMM.EXE, which runs in parallel with DIFF.EXE and handles host communications. A second concurrent invocation of DIFF does not start a second DIFFCOMM. DIFFCOMM coordinates the use of the stage table and metafile cache by multiple instances of DIFF.

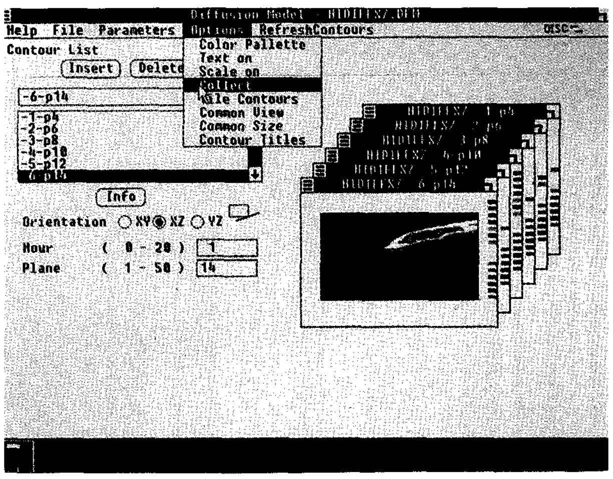
During the execution of DIFF, the user has the opportunity to load and save named files of type .DFM. These are session "state" files and consist of one complete set of diffusion model parameters plus some number of "view" definitions of the output data for those parameter values.

A number of control files are required in order to execute DIFF. The DIFFCOMM program, when started, looks for a subdirectory called METAPOOL in the root of the default drive. If the subdirectory is not found, it is created. Normally the METAPOOL directory contains a cache of one metafile (default view) of a data cube generated previously by the diffusion model. This directory contains two special files, DIFF.SYS and METAPOOL.TAB, as well as a large number of contour metafiles from the host, recognizable by the extension .MTB. The DIFF.SYS file contains communications parameters specific to the user and installation. The METAPOOL.TAB file is a directory of the currently downloaded metafiles and their parameter sets.

• *Integration of user interface and visualization*

The Windows-created main user interface screen for the diffusion model application is shown in **Figure 3**. On the right is a default contour display of a section of the data cube. The current hour and viewing plane number are displayed in the data areas in the lower left portion of the screen. The viewing orientation is shown by the filled circle ("radio button") and the icon directly above. The list box window below the Insert and Delete push buttons can be scrolled, and contains the names of contour views currently being displayed. The push buttons are for adding to or deleting from the list displayed in the window. A user can store views as a group and redisplay these instances at a later session. At the top of the screen are the labels for additional pull-down menus. The state of the connection to the host is shown by the icon and message in the top right corner of the screen. At this time, the host is disconnected from the workstation. The icon of the diskette at the bottom of the screen represents Windows DOS Executive. The location of the mouse is represented by the arrow icon.

To activate the DOS Executive application, the push or radio buttons and the pull-down menus, or to enter data, the mouse (icon) must be placed on the appropriate area



Orientation ○ XY ● XZ ○ YZ
Hour (0 - 20) 1
Plane (1 - 50) 14

Figure 6

Collected contour windows.

of the screen and one or more buttons pressed. For example, the data entry areas are activated by placing the mouse icon in the data area and clicking the mouse button. Data can now be written into the area. The data are verified for type, magnitude, and form. When incompatible data are found, a message appears on the screen showing the acceptable type, form, and maximum and minimum values permitted.

Figure 4 shows the screen obtained when the File pull-down menu is selected and the Open item is chosen. The window on the left can be scrolled, and contains the file names which were previously saved with different instances of the output from the diffusion model. The Open and Cancel push buttons are for opening or canceling the selected file. The file is selected by placing the mouse on the file name and clicking the mouse button. The file name is displayed in the File name data area. A double mouse click results in the contents of the file being displayed on the screen, as shown in **Figure 5**. In this case, the HIDIFFXZ file was chosen; it contains six metafiles. These views are in the *XZ* orientation of six different planes (4, 6, 8, 10, 12, 14).

The Option pull-down menu contains several selectable items. Collect Item arranges the metafiles, as shown in **Figure 6**. Color Palette permits changes to the contour or color relationship of the view selected. Selection of Text On causes the metafile interpreter to paint textual information with the contour. The Scale On selection causes a labeled scale to be placed around the contour window. When it is desirable to spread the contour out across the entire screen, as in Figure 5, the Tile Contours item is selected. The Common View item,

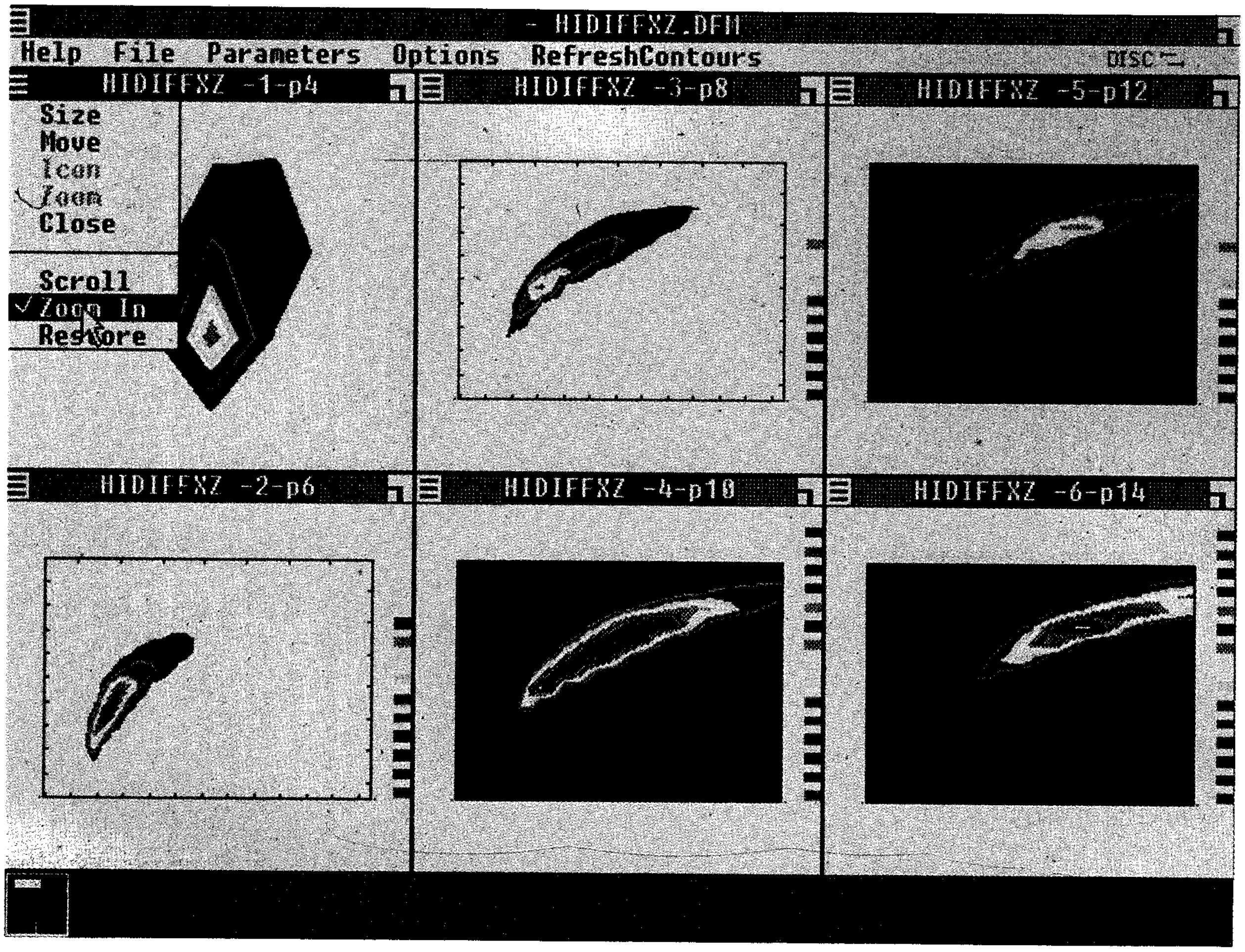
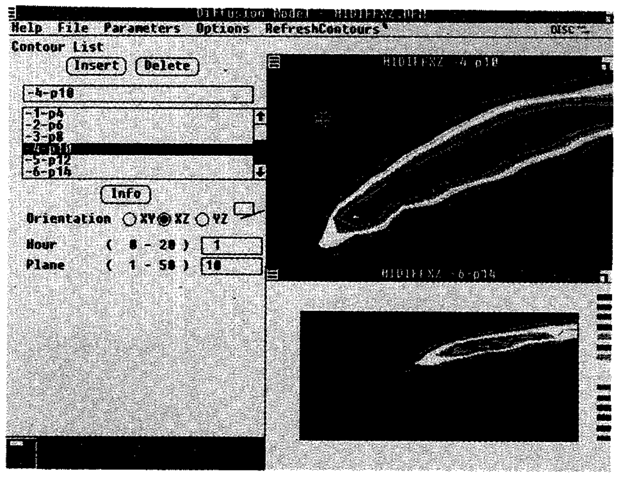


Figure 7

All six contours displayed using Tile Contours menu item.


Diffusion Model - HIDIFFXZ.DFH
Help File Parameters Options RefreshContours
DISC
Insert
Delete
Info
HIDIFFXZ -4-p10
HIDIFFXZ -6-p14

Contour List
-4-p10
-1-p4
-2-p6
-3-p8
-4-p10
-5-p12
-6-p14
Orientation ○ XY [✓ ● XZ] ○ YZ
Hour (0 - 20) 1
Plane (1 - 50) 10

Figure 8

Greater detail shown by use of Zoom In menu item.

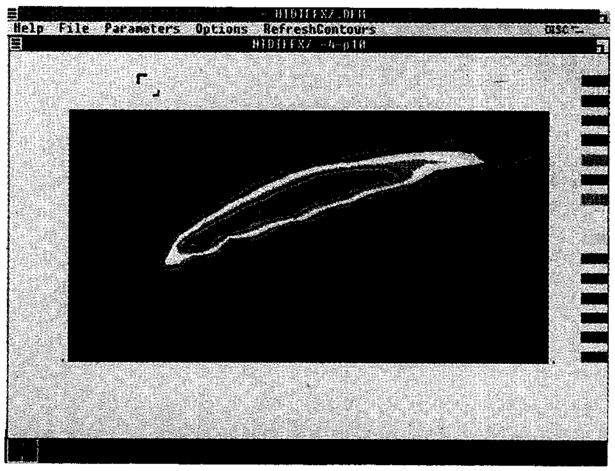


Figure 9

Resized contour window containing zoomed view.

when selected, causes all contour views to be zoomed or not zoomed at the same coordinate, by the same amount. The currently selected contour is used as the base coordinate. The Common Size item is similar to Common View, but deals with the size of the window in which the contour is drawn. The title bar from each view can be removed by selecting Contour Titles, thereby permitting easier comparisons among views.

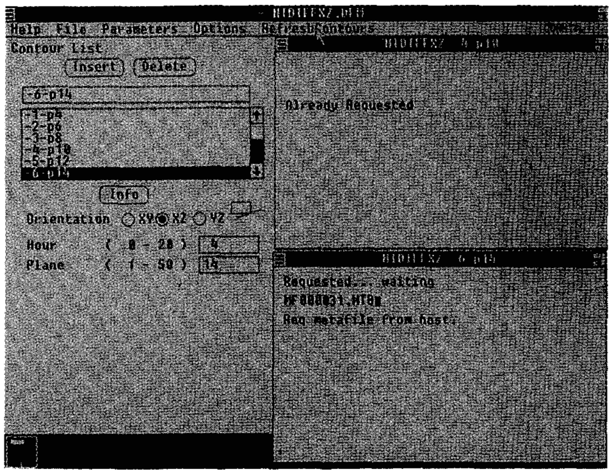
Figure 7 shows six contour views using the Tile Contours selection. The pull-down menu in the upper left corner displays the eight options available to control the contour views in the windows. The last three options were added to the standard Windows functions. The Scroll option permits a zoomed contour to be moved within a window. The Zoom In option allows the user to mark an area in the view for zooming. The view can be redrawn to its initial size by selecting the Restore item. Comparing two views using the Zoom In and Scroll options is shown in **Figure 8**. View HIDIFFXZ-4 has been zoomed and scrolled, while view HIDIFFXZ-6 remains in its initial form. The remaining views have had their windows closed. Reopening the views is done by double clicking on the list entry in the Contour List window.

Figure 9 shows the zoomed single-contour view, with the window resized to cover a large portion of the screen. The zoom icon is shown above the view. The user can zoom on an area within the contour view by moving the mouse to the desired location, pressing the mouse button, and moving across the selected area. Releasing the button defines the area for zooming, and the chosen area is redrawn in zoomed mode.

Up to this point, the manipulation of results concentrated on showing views that were saved during previous work sessions (.DFM files) or individual metafiles that were resident in the workstation metafile cache due to prior requests. In the case where the requested view is not part of the workstation cache (**Figure 10**), the request file containing the model parameters is transferred to the host. The message in the lower window indicates that the request was sent, and the metafile name is displayed. The message in the middle window indicates that the window has already been repainted after the initial request, or the metafile has been requested a second time because of the time elapsed since the previous request.

There are four selections on the pull-down menu item Parameters: Physical, Algorithmic, Geometric, and Point Source. The screen containing the physical parameters is shown in **Figure 11**. The user is also presented with acceptable ranges and units for his input. In addition to the input fields and radio buttons, push buttons labeled Ok, Cancel, and Info are present. When the user has completed his choices, the Ok button is selected, and if the input values are within the acceptable range, the user is returned to the main screen. When the inputs are outside the specified range or an incorrect type is entered, a window containing a message to that effect appears, as shown in **Figure 12**. The window also contains a push button for canceling the input and returning to the currently active screen with the input reset to its previous value.

Two types of help are available for the user. The first type describes how to navigate through the various



-6-p14

-1-p4
-2-p6
-3-p8
-4-p10
-5-p12
-6-p14

Orientation ○ XY ● XZ ○ YZ

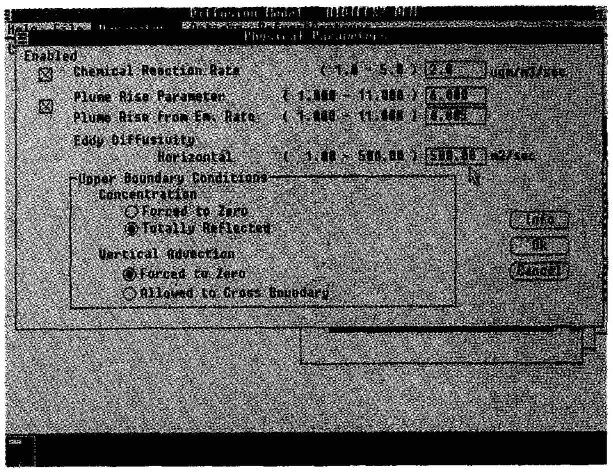
Hour (0 - 20) 4
Plane (1 - 50) 14

Already Requested

Requested... waiting
MF000031.MT0
Req metafile from host.

Figure 10

Contours requested from host.



Enabled

☒ Chemical Reaction Rate (1.0 - 5.0) 2.0 ugm/m3/sec

☒ Plume Rise Parameter (1.000 - 11.000) 6.000

Plume Rise from Em. Rate (1.000 - 11.000) 6.005

Eddy Diffusivity

Horizontal (1.00 - 500.00) 500.00 m2/sec

Upper Boundary Conditions

Concentration

○ Forced to Zero

● Totally Reflected

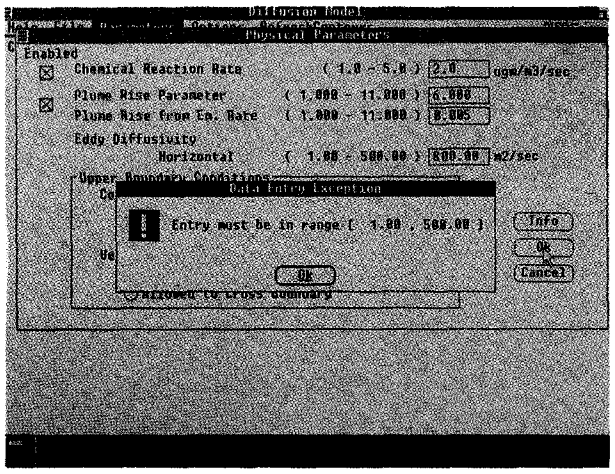
Vertical Advection

● Forced to Zero

○ Allowed to Cross Boundary

Figure 11

Physical parameters window for the diffusion model.

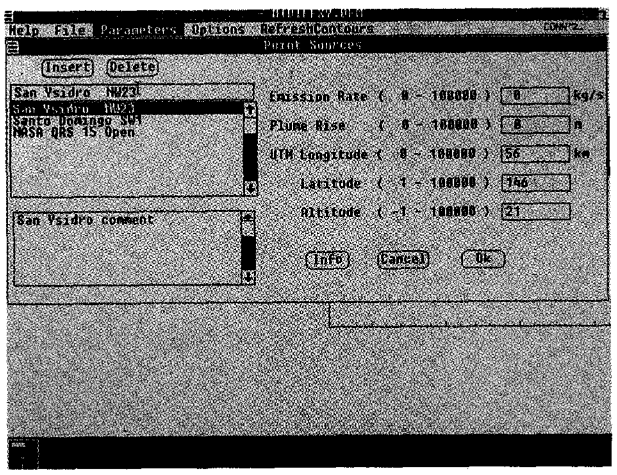


```
Enabled
[X] Chemical Reaction Rate        ( 1.0 - 5.0 )       2.0     ugm/m3/sec
[X] Plume Rise Parameter          ( 1.000 - 11.000 )  6.000
    Plume Rise from Em. Rate      ( 1.000 - 11.000 )  8.005
    Eddy Diffusivity
              Horizontal          ( 1.00 - 500.00 )   800.00  m2/sec

Entry must be in range [ 1.00 , 500.00 ]
```

Figure 12

Invalid data window.



```
San Ysidro  NW23          Emission Rate ( 0 - 100000 )  0    kg/s
San Ysidro  NW23          Plume Rise    ( 0 - 100000 )  0    m
Santo Domingo SW1         UTM Longitude ( 0 - 100000 )  56   km
NASA QRS 15 Open               Latitude ( 1 - 100000 )  146
                               Altitude ( -1 - 100000 ) 21
San Ysidro comment
```

Figure 14

Point sources parameter window.

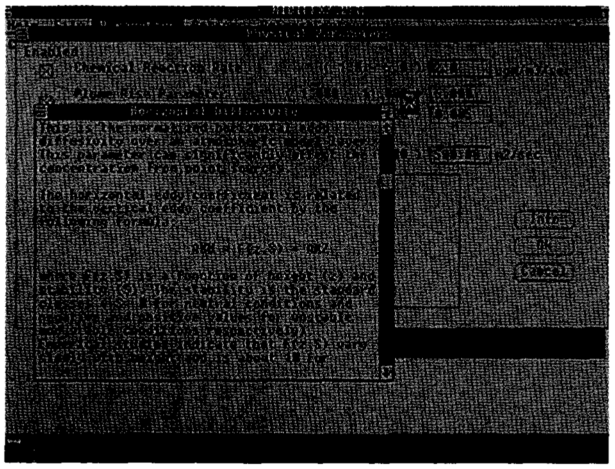


Figure 13

Selection of "Info" displays a poster window.

windows and how to initiate actions using the mouse or keyboard. A second category of help is associated with the mathematical and physical approximations used in developing the diffusion model. This second type of help is called poster windows. **Figure 13** has a poster window overlying the physical parameter window. Each input field or button on a window has associated with it a poster window. The user first points to the field in question, then presses the Info button, and the poster window is displayed. In addition, each parameter screen has an additional poster window which describes, in general terms, the function performed by each parameter window. These poster windows are activated by placing the mouse on the upper left-hand corner of the currently active window and selecting Info. The poster windows can be created or modified during execution of the application. The next linkage of the application code results in the new text being incorporated. In the case of the diffusion model, the accompanying technical reference manual [7] was decomposed into its components and made into poster windows. The prototype code contains a detailed discussion of that procedure.

A second parameter window containing both data input areas and a list box is displayed in **Figure 14**. The Point Sources window permits the user, through the list box, to enter or delete the name of a pollutant source and some character information about the source. This is accomplished by using the Insert and Delete push buttons above the list box. The data entry areas are similar to those on other screens.

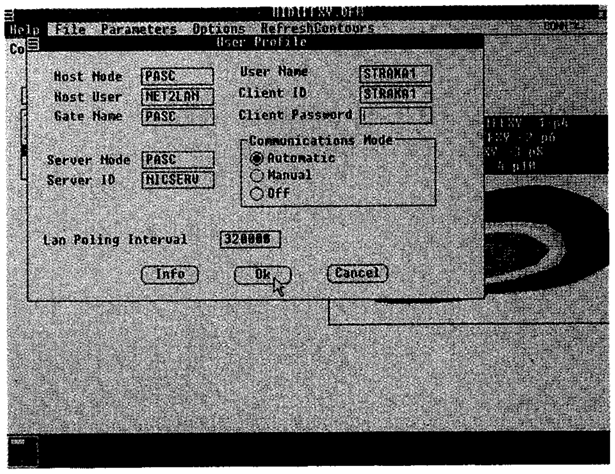
After the user has made all his changes to parameter screens, the updated views are requested by selecting the Refresh Contours option on the main window. The local cache is searched for matches; if any are not found, requests are sent to the host.

The final pair of screens shown in **Figures 15** and **16** concern communications between the workstation and host. These are accessed by selecting the Help pull-down menu on the main screen. There are four selections on that menu; Help, About, Install, and Manual. The Help menu provides a general overview of the diffusion model designed for the application specialist. It reviews the mathematical and physical theory and approximations made in developing the model. The Install menu item causes the User Profile window to be displayed, as in Figure 15. It shows the communication parameters. The significant area is that for the Communication modes. The default is the push button for Automatic, which shields the user from the underlying staging. Figure 16 displays a portion of the Manual Communications screen, which permits the user to manually connect, disconnect, and query the input queue on the network machine. In the figure, the Query has been selected and is displayed as a separate window. The queue has two metafiles (.MTB) which have been sent from the host in response to requests initiated at the workstation. It also contains one file not related to the diffusion model application. The DIFFCOM code ignores the unrelated file, reads the .MTB files, and updates the corresponding contour windows at the workstation. In the case where the user has saved a DFM file and exited the DIFF program with a pending request, the DFM file will either be read at the next instance of a DFM request, or purged if another DFM has been selected.

Conclusions

We have presented a novel approach for carrying out NIC using the IBM 3090 Processor and a workstation. The major thrust of our approach is making it easier for the NIC application user to compute and display visual representations of his output results. To achieve this, the application was structured so that the user interface is on the workstation, and the application, with its associated visualization of results, is computed on the host and displayed on the workstation. Such a split of the application leads to the need to cache the results both on the host and the workstation. Caching is necessary in order to achieve the traditional interactive response associated with using a workstation. As a result, the actual computations are done only on demand, i.e., when the result requested is not either in the workstation or on the host cache. The results are staged at the workstation in the form of metafiles, thus reducing file storage and transmission time from the host. On the host, both data and metafiles are available. As a consequence of our system design, the application looks to the user like a completely local application. In addition, the system design is such that the user is shielded from the complex file structure and his files are managed for him on both the host and workstation.

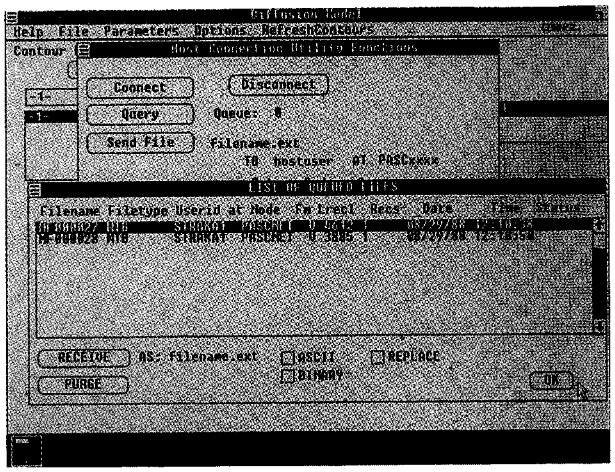
Our structuring of the application stresses the current trend in user interfaces, namely that of using graphics screens. The user interface has several levels of increasing complexity. The NIC application user is encouraged to begin by using the system at the least complex level, and is then naturally led toward gaining familiarity with the



Host Node PASC
Host User NET2LAN
Gate Name PASC
Server Node PASC
Server ID NICSERV
User Name STRAKA1
Client ID STRAKA1
Client Password
Communications Mode
(● Automatic) ○ Manual ○ Off
Lan Poling Interval 320000

Figure 15

Communications parameter window.



Queue: 8
Filename.ext
TO hostuser AT PASCxxxx

```
Filename Filetype Userid at Node  Fm Lrecl  Recs   Date     Time  Status
[illegible]
MF000028 MTB      STRAKA1 PASCNET1 V 3885 1        08/29/88 12:10:58
```

AS: filename.ext ☐ ASCII ☐ BINARY ☐ REPLACE

Figure 16

Manual communications window.

most difficult level. He initially navigates visually through previously computed results, becoming accustomed to views of existing instances of the problem. Once this base has been established, he edits existing model parameters resulting in execution of the NIC problem, and views results based on his previously acquired knowledge. At this level, he can make use of poster help screens which provide detailed descriptions of the physical and mathematical approximations used in the model. Poster panels can serve as documentation for developing papers written for professional journals. The most complex level involves application performance and file management and need not be used.

We have attempted to define an environment in which the NIC user is encouraged through a careful design of the user interface to execute his "dusty-deck" application in the traditional host environment. Considerable host computation in the form of visualization is also carried out on the host. The additional data storage resulting from the visualization is also maintained on the host DASD. The rapid response of a graphic user interface is placed on the workstation, where it can best be realized.

References and note

1. Disk Operating System Version 3.3 IBM Personal Computer Software, 1985; IBM Product No. 5871-AAA.
2. Top View, IBM Personal Computer Software, 1984; IBM Part No. 6024131, Feature Code 4131.
3. PROCOMM-PLUS, Datastorm Technologies, Inc., Columbia, MO 65201, 1987/1988.
4. IBM 3278/3279 Emulation Adaptor, 1984; IBM Part No. 53F6425, Feature No. 5050.
5. *Visicalc Guide*, 1982; VisiCorp, 2895 Zanker Rd., San Jose, CA 95134.
6. *IBM Virtual Machine/System Product Release 4, Product Introduction Manual*, 1984; Order No. GC19-6200; available through IBM branch offices.
7. C. C. Shir and L. J. Shieh, "A Generalized Air Pollution Model and Its Application to the Study of SO_2 Distributions in the St. Louis Metropolitan Area," *J. Appl. Meteorol.* **13**, No. 2, 185–204 (1974).
8. W. Buchholz, "The IBM System/370 Vector Architecture," *IBM Syst. J.* **25**, No. 1, 51–62 (1986).
9. F. Clare, L. Henderson, S. Henderson, B. Horner-Miller, J. Humbreet, and D. Kennison, "The NCAR GKS-Compatible Graphics System," *NCAR/TN 267-1A*, Technical Note, 1986; available from the National Center for Atmospheric Research, Boulder, CO 80306.
10. The Computer Graphic Metafile (CGM) format is defined by the American National Standards Institute (ANSI) in its publication ANSI X3,122-1986; American National Standard for Information Systems—Computer Graphics—Metafile for the Storage and Transfer of Picture Description Information (1986).
11. *Virtual Machine/Extended Architecture System Product, General Information*, 1987; Order No. GC23-0362, available through IBM branch offices.
12. IBM PC Network Program, IBM Personal Computer Software, 1984; IBM Product No. 6361559.
13. *Microsoft Windows, Software Development Kit 1.03*, Microsoft Corporation, Redmond, WA, 1987.
14. *Microsoft C Compiler*, Microsoft Corporation, Redmond, WA, 1986.
15. *Microsoft Windows Operating Environment 1.03*, Microsoft Corporation, Redmond, WA, 1987.

Received October 18, 1989; accepted for publication January 17, 1991

Richard F. Arnold *IBM Palo Alto Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.* Dr. Arnold holds a Ph.D. in computer science from the University of Michigan, and served on the faculty of the Electrical Engineering Department there. He joined the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, in 1965 after a series of temporary assignments at IBM in Poughkeepsie, New York, and San Jose, California, starting in 1958. He worked in the systems architecture areas in Poughkeepsie, Yorktown Heights, and Menlo Park, California, and was manager of storage architecture in San Jose for disk products. In recent years Dr. Arnold has done work in experimental data organizations at the Palo Alto Scientific Center; he is currently working on tools for integrating traditional host applications with current workstation technology.

Paul Halpern *IBM Palo Alto Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.* Dr. Halpern has been with IBM since 1968, and is currently a staff member in Technical Computing Systems at the Palo Alto Scientific Center. He received his B.S. degree from the City College of New York in 1961, an M.S. degree from New York University in 1963, and a Ph.D. in atmospheric science from the University of California, Davis, in 1975. He has authored more than thirty reports and scientific publications in the fields of numerical weather prediction, air pollution meteorology, radiative transfer, solar energy harvesting, real-time data acquisition, digital image processing, and graphics user interfaces. He is currently working in the area of numerically intensive computing and visualization. Dr. Halpern is a member of the American Meteorology Society and the American Geophysical Union and an Associate Editor of *Environmental Software.*

Gerald R. Hogsett *IBM Palo Alto Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.* Mr. Hogsett is a member of the technical staff at the Palo Alto Scientific Center. He received his B.S. in electrical engineering from Stanford University, Stanford, California, in 1962, and was a Project Engineer at the NASA/CalTech Jet Propulsion Laboratory, Pasadena, California, from 1962 to 1963. In 1963 he joined IBM, where his work assignments have included the IBM Electronic Circuit Analysis Program, a joint effort with Lockheed Aircraft Corporation in the development of CADAM, and the IBM Electronic Circuit Analysis Program II. In 1974, Mr. Hogsett joined the IBM Scientific Center in Palo Alto, and participated in a large database development project. In 1985, he accepted a temporary assignment to the IBM Development Laboratory in Hursley, England, where he developed the GDDM-REXX Program Product. Returning to Palo Alto in 1987, Mr. Hogsett joined the Visualization Systems Group of the Numerically Intensive Computing Department of the Scientific Center. Mr. Hogsett's emphasis and expertise lie on both sides of application programs (interfaces to systems and interfaces to the user), and in system integration. He has concentrated his efforts in extending the availability of graphics and graphics display support systems.

Barbara T. Straka *IBM Palo Alto Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.* Ms. Straka is a scientific center staff member in the Visual System group, Numerically Intensive Computing Department, of the Palo Alto Scientific Center. She joined IBM in 1984 to work on scientific programming environments and graphics and is now involved in scientific and distributed visualization and user interfaces. Ms. Straka received her B.A. in mathematics for UCLA in 1966 and her M.S. in computer science from American University in 1979. From 1966 to 1970 she was an engineer/scientist at McDonnell Douglas Corporation, and from 1970 to 1974 she was a member of the technical staff at MIT Lincoln Laboratory. From 1974 to 1978 she was a systems analyst in the Computer Center at Jackson State University, Jackson, Mississippi, and from 1980 to 1984 she was an assistant professor in the Computer Science Department at Jackson State. In the summers of 1982 and 1983 she participated in the NASA/ASEE Faculty

Fellowship Program at the NASA Goddard Space Flight Center in New York. During the summer of 1984 she participated in the Summer Institute at Lawrence Livermore National Laboratory, University of California, Livermore. She is a member of the Association for Computing Machinery (ACM) and the Association for Women in Computing (AWC), and an affiliate member of the IEEE Computer Society. She was president of the Mid-South regional affiliate of the International Association for Computing in Education (IACE/AEDS) from 1982 to 1984.

Connie Arasmith *Lockheed Missiles & Space Co., Inc., Palo Alto, California 94304.* Ms. Arasmith held a supplemental position at the Palo Alto Scientific Center from 1987 to 1988; since 1989 she has been located at the Lockheed Missiles & Space Company, Inc., in Palo Alto. She received her M.S. in computer science and engineering from Santa Clara University, Santa Clara, California, in 1988, and her B.S. in industrial engineering and operations research from Virginia Polytechnic Institute and State University in 1978.

J. McElroy *Tredennick, Inc., 4040 Moorpark Blvd., San Jose, California 95150.* Mr. McElroy worked as a supplemental employee at the IBM Palo Alto Scientific Center from 1986 to 1989; he now works at Tredennick, Inc., San Jose. He received his B.S. in computer engineering from San Jose State University in 1989.