A data collection and display system for a large-scale simulation

A. E. Brown, C. E. Phillips, J. S. Scandale, and D. P. Sparrow
Cornell Aeronautical Laboratory, Inc., 4455 Genesee Street, Buffalo, New York, 14221, USA

This paper describes a system of computer software (collectively called MONITOR) designed to acquire and post-process data generated during the execution of a large-scale hybrid computer simulation.

The user may, during the post-processing phase, interactively review the collected data on a CRT device. Selected graphical output may be produced on a line printer or x-y plotter.

The motivation for developing such a system is first given, followed by a description of the present capability and finally by a brief discussion of inherent drawbacks to the present system and recommended improvements.

(Received April 1971)

The problem
Designing, building and operating a large-scale digital or hybrid simulation is, in many ways, analogous to carrying out the same tasks on a physical system. One major difference, however, is the general inability to instrument the simulation to monitor its activities. In a physical system, one may attach, at various strategic locations, voltmeters, ammeters, pressure and temperature gauges and other transducers to monitor directly or indirectly the system's internal state. The need for an analogous instrumentation capability for large-scale simulation programs became apparent early in the development of a large air defence simulation for the United States Air Force. The simulation is hybrid and involves two digital computers, an IBM S/360-model 65 and a Honeywell DDP-116, in addition to two COMCOR CI-5000 analogue computers. The simulation is time based; time is incremented in integral units of the same size simultaneously for all simulated systems. Consequently, many of the variables of interest may be considered functions of time. There are approximately 10,000 variables of potential interest generated within the simulation program; all of these variables are stored in the IBM 360/65.

The most commonly required data format, for system performance analysis and checkout, is a plot of a variable as a function of time. With many variables having a large dynamic range, such as radar cross-sections and radar receiver inputs, scaling considerations will sometimes degrade the inherent accuracy in producing a graphical presentation of these data. Under such circumstances, it is desirable to 'spot check' the data in their most accurate form, that is, reduce the time span considered or print the data in full precision. These two considerations led to a basic requirement for a simulation monitoring system that would produce a variable's time history in either a plotted or printed format. It is also desirable at times, although less frequently required, to plot several variables against one another and also to plot the difference between two variables as a function of time. There is an inherent difficulty in manipulating and displaying the data after the simulation is complete. Namely, the presentation of one variable may suggest the investigation of another that has not been saved.

A candidate solution
One candidate design for such a monitoring system would centre around the transfer of all 10,000 variables of interest to a permanent recording medium, such as magnetic tape, at the end of each simulated time interval and then extract and process the desired data at the end of the simulation run. This approach, however, has several shortcomings. First, in order to prevent unacceptable degradation of simulation performance, the process should be double buffered. This leads to a buffer space requirement of 80,000 bytes or over 12% of the 650K bytes available for the IBM 360/65 program. Second, the amount of tape generated in a run would be excessive, e.g. for the simulation under discussion, a run for ten simulated minutes would fill approximately nine reels of magnetic tape at a density of 1,600 bits per inch.

These considerations led to a solution with sufficient flexibility, in principle, to handle the worst case, described above, that of monitoring 10K variables, but with the added ability to adapt itself to less complete data collection tasks. The basic collection philosophy that evolved allows the user to specify at model run time:

1. The simulated time at which data collection is to start and stop. If no times are specified, these default to model run limits.
2. The rate at which data are to be collected, that is, the number of model time intervals skipped between data samples. This number is useful in controlling the amount of tape generated. If no value is specified, the default value is 1 and no intervals are skipped.
3. The specific variables to be monitored.

The system developed, called 'MONITOR', is functionally divided into two phases, a data extraction phase and a post-processing phase that displays and manipulates the extracted data according to user supplied specifications. The programs written to support the extraction phase are collectively called the 'extractor', and those supporting the post simulation processing phase are called the 'post-processor'. These two programs are now described.

Extractor principles
The monitor extractor resides in main memory along with the simulation programs. Its function is to extract the selected variables from their assigned data blocks, move them into a memory buffer at the desired points in simulated time and then output the buffer to a permanent recording medium such as magnetic tape or disc storage.

Each extractable variable is identified by a 32-bit word, each 8 bits conveying a different type of information to the extractor. In picking this scheme of identification, use was made of the fact that groups of variables are logically connected. For example, radar dependent variables, and aircraft dependent variables are stored in different data blocks in memory. To each of the data block types, a unique number was assigned. To

*The research reported herein was supported by the Air Force Avionics Lab., Wright-Patterson Air Force Base, Ohio.
locate a particular variable within such a group, a unique displacement number was assigned. Finally, to allow array specification, provision was made for the designation of two index values (in the FORTRAN sense). The resulting identifier then uniquely determines the location in memory of each variable capable of being recorded. It is tacitly assumed that connected variables are stored in contiguous locations in memory, similar to the result of a FORTRAN COMMON specification, although this restriction was circumvented in the sense that not every location in a particular block was recordable, these decisions being made at design time.

The identification word described above is input to the extractor in a hexadecimal format in the present MONITOR system. The format of the identifier is illustrated below:

<table>
<thead>
<tr>
<th>DATA BLOCK TYPE</th>
<th>VARIABLE</th>
<th>INDEX 1</th>
<th>INDEX 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and a specific example follows,

<table>
<thead>
<tr>
<th>0 8</th>
<th>0 E</th>
<th>0 2</th>
<th>0 1</th>
</tr>
</thead>
</table>

which when translated requests the extraction of the cosine of the pitch angle of the second missile launched from site number one.

Extractor design

The design of the extractor was influenced by the desire to keep both the execution time and the core requirements to a minimum while meeting the objective of providing a facility for recording a large number of simulation variables on a permanent medium. Given that each variable of interest has a unique identifier, the identifier must be decoded to locate the data and then the data must be moved to the buffer. The problem of determining the method of implementation is that of deciding whether:

1. To 'interpret' the list of identifiers at each desired time point, that is, decode the identifier and then move the data or alternatively
2. To look at the list of identifiers as a 'program' in a pseudocode language and compile them into machine code, i.e. perform the decoding process once, for execution at the desired times in the simulation to fill an output buffer.

The former method has a desirable low core overhead; space is required for the interpreter and the list of identifiers. However, as the number of variables increases, the time penalty in iterative interpretation is prohibitive. The latter method has an estimated core overhead almost twice that of the former; space is required for the 'compiled' interpreter. Here, the execution time required is minimised, relative to the execution time of the interpreter, as the number of variables recorded increases. Since the decoding phase of the interpretation is done during the compile phase, only the data movement into the buffer is required. Because additional core was deemed less expensive than drastically increased running time, the compiling method was chosen.

The extractor was designed as a two-phase program, consistent with the planned overlay structure of the simulation under S/360 OS. The input/initialisation phase reads the control data and variable identifiers and then 'compiles' the identifiers into groups of code and control blocks. The input phase is later overlaid with the execution/control phase. This phase supervises the execution of the generated code groups at the required times and also performs the output.

**Post-processor**

The post-processor phase operates in either of two available modes: the batch processing mode or the interactive graphic mode. In the batch processing mode, specifications are given to the Monitor system through card input while the interactive graphic mode utilises an IBM 2250 display device as an input/output medium. The post-processor input allows user designation of start and stop times distinct from those of the extractor, the time interval between data points, and the subset of variables from the extraction file to be used. In addition, operators defining plotting and/or printing options for the extracted variables may be specified. These options include for plotting:

1. A variable versus time.
2. The difference of two variables versus time.
3. One variable versus a second variable.
4. The difference of two variables versus a third variable.
5. Superimposition of up to 16 plots of different variables versus time.

The post-processor uses the input variable identifiers to produce sets of data values from the extraction file which are either (according to input control data) output on a high-speed line printer or scaled and output on an external plotting device. The plotting device currently in use is a Xerox LDX printer. The interactive graphic mode allows the utilisation of the IBM 2250 display unit as a communication device. The input which would be supplied on cards in the batch mode, is presented in the interactive mode through use of the display unit's attached light pen and alphanumeric keyboard. The absence of card input signifies the use of the interactive graphic mode. Upon the appropriate light pen signal, the graph or plot, which would have been output to the LDX printer in the batch processing mode, is displayed on the IBM 2250. The post-processing system then pauses allowing the user to view the display and decide upon the desirability or usefulness of the presented data. If it is desired to have a hard copy of the display produced, the user signals this intent with the light pen and the copy is produced on the LDX by the program. It then sequences to the next display.

A very important feature of this mode of operation is the ability of the user to make decisions concerning the presentation of the data while the system is operating, i.e. 'on line'. For example, in the case of a variable displayed versus time, the user can, after inspection of the display, decide to re-specify the limits on time between which the data will be presented, thus expanding or contracting a portion of the graph.

An anomaly in a graph might suggest the further investigation of other variables, or conversely might obviate the necessity for viewing other relationships. Singularly useful is the ability to superimpose plots; the user can compare the behaviour of analogous data items and thus evaluate the performance of the simulation.

**MONITOR system use**

The simulation monitoring system described in the previous paragraphs has proven to be indispensable in the development and checkout of the simulation that prompted its creation. It has been used in the batch and interactive mode in both simulation checkout and in demonstrating the capabilities of an otherwise 'opaque' simulation to the sponsoring agency.

Two examples of the output provided by the LDX printer are given as Figs. 1 and 2.

Fig. 1 is a plot of seven aircraft flight paths. It is a set of seven superimposed plots, each of which is a plot of the aircraft X co-ordinate versus the Y co-ordinate.

The second example is a difference plot versus time, and illustrates the initiation of automatic tracking in elevation of a radar at site No. 1, on aircraft No. 1. The difference is between the elevation angle to the aircraft (ELN) and the elevation angle of the radar antenna axis (ETLABS).
Present limitations and recommended improvements

The development of the simulation monitoring system described in this paper took place at the same time the main simulation was being developed. Consequently, the system lacks a number of desirable features because their usefulness could not be foreseen. In addition, the described monitoring system was designed specifically for a particular simulation. The improvements that can be recommended can therefore be divided into two categories: those that would add to the system's use with the simulation for which it was designed, and those that would generalise on the concepts described and allow the development of a general purpose tool for use by a large class of computer problems.

These two categories of improvements are discussed below.

Improvements to the 'particular simulation' oriented system

The present system does not allow comparison plots to be made using data from different simulation runs. This is a serious limitation as one frequently wishes to compare the effect of a change in input and/or program on the generated data. This present limitation arises from the fact that the data output from the extractor phase is not uniquely identifiable to the post-processor phase (a program identifiable label is not written on the recording medium), and the fact that only one data file is available to the post-processor at a time. The ability to compare data across runs would then rely on the generation of an identification record on the recording medium, and an expansion of the number of source files from which the post-processor could work.

Presently if one wishes to examine the same set of variables generated by more than one simulator run, the post-processor programs must be loaded and input data provided once for each data set. A more convenient way to perform this operation would require the addition of a looping mechanism to the post-processor and a means of specifying the source(s) of data on consecutive passes. This would allow the program to be loaded and variable data input once per exercise.

The present package was structured around the concept of data in the form of a four-byte floating point number as implemented on the IBM S/360. Some of the data generated by the simulation does not fit the format. For example, some calculations are done on fixed point numbers, in both 16- and 32-bit format. A second example is the generation of logical information that is stored in individual bits of 'flags' words. The ability to extract and process this type of data is highly desirable. Sometimes it is data in one of these forms that provide the clue necessary to eliminate a particular problem, or otherwise explain an anomaly in the presently available floating point output.

As will be noted by the two output examples given in Figs. 1 and 2, the plot axes are not labelled with the units of the variable(s) plotted. The addition of this information to the plots would make them more easily usable to personnel not as familiar with the simulation as the designers and builders.
Generalisation of the system

Because of the proven usefulness of the system described in the development of a particular large-scale simulation, it is thought that the concepts employed, if generalised, would be useful as an output and diagnostic tool for a large class of computer problems. To make the system generally useful, it must be usable by a large class of computer users. This suggests the need for two additional facilities: a high level user-oriented language to replace the presently used machine-oriented pseudo language for ease of requesting different extraction and post-processing facilities; and a generalisation of data types that can be handled.

The language would be usable for requesting both data extraction and post-processing. It would allow the specification by algebraic and/or logical expressions of,

1. The variables to be extracted.
2. The times of extractions, and
3. The extraction interval,
and to the post processor,

1. Algebraic and/or logical combinations of data to be plotted.
2. Functions of extracted data to be plotted.

Data type extensions would include:

1. Character (byte).
2. Address pointers (3 bytes).
3. Arrays (in the FORTRAN and/or PL/I sense).

Conclusions

The design and checkout of a large-scale simulation requires an extensive performance monitoring system. Such a system has been implemented to support the development of a particular simulation and has proven its value.

The concepts, upon which the monitoring system was designed, may be generalised to produce a facility useful in monitoring the performance of a large class of computer problems.

A high-level language would be useful in making the facilities of a generalised system usable by a large class of personnel other than the simulation designers.