GPSS/H 1979 - A Status Report

James O. Henriksen
Wolverine Software
P. O. Box 1251
Falls Church, VA 22041

ABSTRACT

A number of earlier papers (1,2,3) have described the design, features, and algorithms of GPSS/H, a new implementation of GPSS which runs on IBM 370 and 370-compatible hardware. Since publication of these papers, GPSS/H has been put into use at a number of commercial sites and colleges, as well as on the National CSS timesharing network. This paper describes the current status of GPSS/H and presents insights gained from application of GPSS/H to "real world" problems. Finally, ongoing developments in GPSS/H are described, and long range plans are outlined.

CURRENT STATUS OF GPSS/H

As of this writing, all GPSS/V features except Jobtapes, READ/SAVE, and AUXILIARY storage allocation have been implemented. The Jobtape and READ/SAVE features will be completed by the end of 1979. The AUXILIARY storage allocation of entities will never be supported, because inclusion of this feature would have a deleterious effect on run-time performance, slowing down the execution of all models, even those which do not use the feature.

In addition to the features of GPSS/V, a number of language extensions have been implemented. First, the syntax of the GPSS language has been expanded to remove many arbitrary restrictions and to allow more expressive notations. Principle features of the expanded syntax are summarized as follows:

1. Operands of all Block statements and nearly all control statements can be coded as expressions.
2. Standard Numerical Attribute (SNA) references can be written in a subscript notation, e.g., "FN(X$BASE)".
3. All entities, including Transaction Parameters and Random Number Streams, can be symbolically referenced. The use of symbolic Parameter names greatly enhances the readability of a model. For example, "PF$SIZE" conveys much more meaning than "PF3".
4. Symbols can be up to 8 characters in length.
5. Statements can be entered in a simple free format and can be continued across as many input records as desired.
6. Standard Logical Attributes (SLA's) can be used outside Boolean Variables. When so used, SLA's assume integer values of 1 and 0, corresponding to true and false values of the tested condition, respectively.
7. Extended flexibility is provided for Macro usage. For example, GPSS/H allows Macros to be used for constructing sequences of control statements, while GPSS/V allows use of Macros only for constructing sequences of Block statements. The GPSS/H compiler performs a very general text substitution in expanding Macros, allowing any part of a statement, including the comment field, to be specified as a Macro argument. By contrast, GPSS/V allows text substitution only for complete Block operands.
8. Specifications for the Y-axis of a graph may be given as floating point expressions, facilitating the plotting of floating point statistics.
9. The BCLEAR, BRESET, BRMULT, and BSTORAGE Blocks have been implemented to provide in Block form, the capabilities of the CLEAR, RESET, RMULT, and STORAGE control statements, respectively.

In addition to the syntactic extensions listed above, the following general improvements (over GPSS/V) have been made:

1. The allocation process for entities has been greatly enhanced. For example, if the compiler encounters a statement of the form "SEIZE 100", it automatically allocates at least 100 Facilities. No members are allocated for completely unused entity classes. Due to the automatic allocation performed by the compiler, the programmer does not need to supply REALLOCATE statements, unless entities are referenced by indices computed during model execution, in which case the compiler cannot determine the appropriate allocation.

CH1437-3/79/0205-0209$00.75
1979 Winter Simulation Conference

2. Compilation diagnostics are given in English (or a programmer's approximation thereto), and, whenever possible, errors are flagged with a "+", indicating the exact location of the error in the source statement.
3. The cross-reference listing for entities includes references to entities made with constant operands, e.g., "SEIZE 2" causes a Facility-type cross-reference item to be included for the constant "2".
4. The number of Random Number Streams in a model can be REALLOCATEd. The default number of streams is 8, as in GPSS/V, but more than 8 streams can be requested.
5. A Transaction may belong to an unlimited number of Queues, without loss of statistics beyond the fifth Queue to which the Transaction belongs.
6. GPSS/H allows Transaction priorities in the range $\pm$ 2**31-16 (very big).
7. Antithetic variates can be requested for Random Number Streams.
8. Standard model output includes a summary of the utilization of COMMON storage, facilitating the REALLOCATion of COMMON to reasonable values.
9. The TBW, TCW, and TDW SNA's have been added to allow access to weighted Table statistics.

The preceding two paragraphs have provided only a brief summary of the GPSS/H system. For a more detailed description, interested readers should consult the GPSS/H User's Manual (4).

GPSS/H PERFORMANCE

Before we can discuss the run-time performance of GPSS/H, some background information must be presented. GPSS/H implements GPSS statements in three ways. Simple computational statements, such as the SAVEVALUE Block, are compiled into sequences of machine instructions which carry out the statement in its entirety. Syntactically simple statements which require a significant amount of computation and logic, e.g., the SEIZE Block, are compiled into simple subroutine calls. Syntactically complex statements which have lots of variations and require significant amounts of logic and computation, e.g., the UNLINK Block, are compiled into sequences of machine instructions which interact with simulator subroutines on a co-routine basis, to carry out functions requested for each instance of the statement.

The run-time performance of GPSS/H, relative to the more traditionally implemented GPSS/V, depends on the distribution of model statement types within the three categories of statement implementation strategies described above. This distribution is, in turn, dependent on the nature of the application being modelled. For example, student programming exercises often emphasize basic language concepts and, accordingly, tend to contain a high percentage of statements that are implemented as subroutine calls. By contrast, real-world industrial, commercial, or military simulations are often characterized by large amounts of data required to represent the system being modelled. Such models often contain a large amount of purely computational (bookkeeping) statements, which are carried out by means of of tailor-made, compiler-generated machine instructions. The best relative performance of GPSS/H over GPSS/V is obtained in such models, because the compiler-generated machine instructions of GPSS/H are much faster than the classical interpretive approach employed by GPSS/V. The following table illustrates the performance of GPSS/H relative to GPSS/V on a variety of applications:

Application	Characterization of Application	Performance Enhancement (GPSS/H:GPSS/V)
Monte Carlo simulation of an electric power network	Purely computational (does not use any ADVANCE Blocks)	9:1
Military logistics model	Highly computational	6:1
Automotive machining line	Fairly computational, lots of complicated synchronization	6:1
One-line single-server	Pedagogical model, no computation	4.5:1

The results presented above represent model execution times only. In order to present a complete picture of GPSS/H performance, compilation times must be considered as well. For most models, the GPSS/H compiler runs about 2 to 2.5 times as fast as the GPSS/V assembler, i.e., programs are translated into internal form in less than half the time. Since total run times may be expressed as the sum of compilation and execution times, the performance of GPSS/H, relative to GPSS/V, for entire runs depends on the ratio of compilation time to execution time. In typical industrial, commercial, and military models, run times are much larger than compilation times, so the relative performance of GPSS/H approaches the run-time performance illustrated in the table above. For academic models, where the ratio of execution time to compilation time is typically much lower, the relative performance of GPSS/H is somewhat lower, typically 3:1 (versus GPSS/V).

USER REACTION TO GPSS/H LANGUAGE EXTENSIONS

Users of GPSS/H can be divided into two broad categories, with respect to use of language extensions. One group is very enthusiastic about the language enhancements available in GPSS/H and makes extensive use of them. The other group is comprised of organizations which feel they must maintain compatibility with GPSS/V. Such users, while they may appreciate language extensions, are barred from making any use of them. The essence of this division of the GPSS/H user community is the issue of transportability. Most of the users of GPSS/H are using it for highly proprietary purposes, with the intended user community for models developed being very limited. For such users, transportability is not an issue. However, other users produce models which are widely distributed, and for them, transportability is a vital issue.

USER REACTION TO INTERACTIVE DEBUGGING FACILITY

All present users of GPSS/H have an interactive computing capability (MV/370, MVS/TSO, VP/CSS, and MTS operating systems). In the near future, GPSS/H will be installed at a number of batch-only installations. Among current users, the acceptance of the interactive debugging facility of GPSS/H has been very high. It has often been the case that considerable prompting by the author has been required to convince users of the efficacy of using interactive debugging techniques. However, once users have become familiar with the advanntages of interactive debugging, they soon hope they never have to go back to a pure batch computing environment.

The effectiveness of the interactive debugging facility rests on the principle of trading modest amounts of relatively inexpensive computer time in exchange for large amounts of expensive human time. Rather than sitting around for hours poring over dumps and mountains of trace output, users find it far more effective to sit down at a terminal and carefully step through a program, in order to find an elusive bug. When a user carefully steps through a model, displaying selected model data on his terminal, he participates in the execution of the model and gains insights into the operation of both his model and the underlying system being modelled.

ONGOING GPSS/H DEVELOPMENTS

At present, the following developments are underway:

1. Jobtapes are being implemented.
2. The READ/SAVE feature is being implemented.
3. Character variables are being added to the GPSS language. The availability of character variables will greatly improve the abilities of GPSS/H to communicate with user. For example, it will be possible to construct tailor-made messages for typing on a terminal, and it will be possible to create far more sophisticated summary reports than are possible with the relatively limited standard GPSS/V Output Editor.
4. Generalized input/output is being added to the language. This will allow GPSS/H models to read and write data files and to read from and write to interactivee terminals.
5. IF-THEN-ELSE statements, GOTO statements, and a DO loop capability are being added to the control statement language. These additions will have two major benefits. First, far more sophisticated control of model execution will be possible. For example, it will be possible to iteratively execute an entire model, with parameters for each run dynamically computed as a function of the results of preceding runs. Second, because the new statements will also be allowed within the context of a REPORT section, the capabilities of the Output Editor will be greatly enhanced. The combination of new control statements and a generalized input/output capability will give GPSS/H a report generation capability nearly as sophisticated as that of Simscript II.5.

SHORT TERM PLANS FOR GPSS/H

Over the next several years, a number of improvements will be made to GPSS/H. First, a number of excellent suggestions made by early users of the system will be incorporated. Most of these suggestions involve improved capabilities for, and improved control over, statistical output. Second, a number of improvements are planned for the interactive debugging system. Among these improvements will be a checkpoint/restore feature and the capability to flag Transactions in such a manner that whenever they are picked up in a scan of the Current Events Chain, control returns to the interactive debugging system. If a request is made to flag a Transaction prior to creation of the Transaction, the request will be automatically queued. Suppose, for example, that a simulation is terminated due to an error caused by Transaction number 12345. (GPSS/H numbers all Transactions in a model consecutively, without reusing previous ID numbers.) A subsequent run could be made under the interactive debugging system, requesting that Transaction 12345 be flagged. When Transaction 12345 is created, it will be flagged, and each time it attempts to move through the model, control will return to the interactive debugging system. Thus, the life history of the Transaction which caused the error in the original model can easily be determined. The checkpoint/restart feature will allow

the current status of a model to be saved at any point and to be restored at any subsequent point. The checkpoint/restore feature will help compensate for one of the common frustrations of interactive debugging, namely the "Whoops! I went too far." problem. Sometimes, in the process of debugging a program, after having spent a great deal of time to set up breakpoints and having spent a great amount of time stepping through a program, a user loses control of model execution without finding the problem he was looking for in the first place. With the checkpoint/restore feature, the programmer will be able to periodically save the current status of the model, so that in the event of an unforeseen disastrous error or loss of control, he can restore the previous status of the model and try again, without having to rerun the entire model to get back to the point of departure.

LONG RANGE PLANS FOR GPSS/H

Long range plans for GPSS/H call for introduction of a completely new version of the GPSS language, possibly as early as 1982. One might naturally be tempted to ask "Why a totally new version?" In the author's opinion, the GPSS language has a number of critical shortcomings, but the language is already too big to correct these shortcomings by adding more features. The GPSS language is now almost twenty years old. The time has come to draw on those years of experience and to create a new language which, while retaining the best features of GPSS, includes capabilities expected in modern programming languagesa and excludes those GPSS features which have become archaic. In the earliest versions of GPSS (5), a "program" took the form of a fixed format card deck, with each card specifying the name of a simulator subroutine to be called and fields of the card specifying numeric arguments to be passed to the subroutine. Thus, GPSS "programs" were really "simulator input decks." Over the years, enhancements to GPSS have provided both a more powerful simulator and a more powerful superstructure for invoking the capabilities of the simulator. Improvements to the superstructure have made GPSS look more and more like a simulation language, but it is still basically just a simulator with an elaborate superstructure.

The proper viewpoint to be taken in designing a new version of GPSS would be to structure it as a programming language with powerful simulation features, rather than as a simulator with a language-like superstructure. To some, this may be a subtle distinction; illustrations are in order. Consider the most fundamental statement in most programming languages, the assignment statement. In GPSS, there is no assignment statement. Rather, we have the ASSIGN, INDEX, LOGIC, MSAVEVALUE, and SAVEVALUE Blocks, each of which manipulates a different class of entities. In GPSS, there is no capability for separate compilation of routines; indeed, there is no true subroutine capability in the language. Similarly, GPSS lacks generalized input/output capabilities. The basic units of traffic in a GPSS simulation, Transactions, have a very limited structure. Transactions compare very poorly to the temporary entities of Simscript II.5 or to generalized data structures of non-simulation programming languages. All of the deficiencies cited above have a common denominator: they can be incorporated into a programming language of modest size.

The underlying simulator of GPSS is extremely powerful. The ease of use of Transaction-based simulation stands as a remarkable achievement for Geoffrey Gordon some twenty years after his development of the GPSS language. However, programming languages have come a long way in the last twenty years. The time has come to marry current technology of programming languages with the powerful simulator devoped by Gordon and extended by many others.

BIBLIOGRAPHY

1. Henriksen, James O., "Building a Better GPSS: a 3:1 Performance Enhancement," Proceedings of the 1975 Winter Simulation Conference, 1975.

2. Henriksen, James O., "An interactive Debugging Facility for GPSS," Proceedings of the 1977 Winter Simulation Conference, 1977.

3. Henriksen, James O., "An Improved Events List Algorithm," Proceedings of the 1977 Winter Simulation Conference, 1977.

4. Henriksen, James O., The GPSS/H User's Manual, Wolverine Software, P.O. Box 1251, Falls Church, VA 22041.

5. Gordon, Geoffrey, "A General Purpose Systems Simulation Program," Proceedings of the Eastern Joint Computer Conference, New York, NY: Macmillan Publishing Co., Inc.