# NGPSS/6000: A NEW IMPLEMENTATION OF GPSS

Karen Ast, Jerry Katzke, Jim Nickerson, Julian Reitman
Nordan, Division of United Aircraft Corporation
Norwalk, Connecticut

Lee Rogin
Naval Air Development Center
Warminster, Pennsylvania

## Abstract

A version of the GPSS simulation language, compatible with GPSS V, has been implemented for the CDC 6000 series and is operational at the Naval Air Development Center. In addition, the design goals of NGPSS/6000 were thorough assembly debugging, reduced core requirements, decreased running time, unrestricted use of matrices, and language flexibility to allow flexibility for future enhancements. This implementation seeks to aid both the user unfamiliar with the language with more complete diagnostics; while enabling the experienced user to build very large-scale models and associated data banks. Provisions have been made for future language extensions toward real-time operation, on-line debugging, and a faster execution module for debugged models.

## INTRODUCTION

The implamentation of a GPSS for the CDC 6000 series provided an opportunity to reevaluate the organisation and structure of the language. The goal was to preserve the language syntax in its entirety and to simplify the implementation, obtain rapid execution and enable flexibility for future modifications. In addition, as part of this undertaking, it was felt that the resultant language should be internally consistent, for example, matrices can be referenced anywhere within the language.

The first major departure was to implement NGPSS as a two-phase processor consisting of a translator, the assembly phase, and an interpreter, the execution phase. The assembly phase was written primarily in COBOL with COMPASS routines to carry out operations which would have been awkward in the higher level language.

COBOL was chosen for this phase for:

- character manipulation capabilities required to perform syntax checking
- file manipulations used for input, output and cross reference symbol table generation
- sort capabilities used to structure the numeric entity table
- ease of debugging
- to establish a degree of machine independence

The execution phase consists of three distinct segments. The first is an input module which processes the NGPSS control cards and loads the entity area into core, the second is the execution module which performs the actual GPSS simulation and the third is an output module which generates standard NGPSS statistical output. These and any HELP block subroutines are dynamically loaded into core via the SCOPE 3.3 segmentation loader as needed. The first two modules were coded in CDC assembler, COMPASS, to minimize central processor and core requirements. FORTRAN was not used for the following reasons; awkwardness of partial word manipulations, additional core required for system routines, and slower processing of the relative addresses required for entity operations. The output module was written in FORTRAN. Although core and speed are important, they were not as critical as the ability to trans-

form binary values into their character representations.

The two phases of NGPSS/6000 are run as independent operations with communication via a temporary file. The assembly phase is organized in an overlay structure to reduce CPU time. The execution phase is organized into segments because of the reallocatable entity area and the variable number of subroutines. The incompatibility of these loader techniques was resolved by use of a two step processor. The advantage of this type of organization is that assembly and execution can be run independently.

Consistent with the Navy and Norden experience in the building of large scale GPSS models, this implementation of the GPSS language emphasizes the use of matrices to control and operate the GPSS logic. In this manner, a logic system representing the general case can be used with a variety of matrices providing the data to separate the model logic and data libraries. This generalization is emphasized in this implementation by complete accessibility of matrix references anywhere within the GPSS language.

The assembly phase requires 18,700 decimal words of core plus an area for the GPSS symbol table. A typical model can be assembled in default core, 24,576 decimal words. The execution phase requires 7,600 decimal words of core for the system and 3,709 words of core for the system loader and loader tables.

Besides the above considerations, the following features have been added to enhance the language:

1) The restrictions that have been imposed on the usage of matrix and floating point Standard Numerical Attributes (SNA's) have been lifted. They can therefore be used in function arguments and follower cards, GENERATE block arguments, SPLIT block arguments, etc.
2) The number of matrix rows and columns and the transaction number have been added as legal SNA's.
3) A new auxiliary field has been added to the SPLIT block to permit the option of new transactions to no longer be considered as part of an assembly set so that an ASSEMBLY block does not operate on this transaction. This field "GEN" creates a new transaction copy without adding it to the assembly set of the parent transaction.

4) Unary minus operators and a continuation card have been added to arithmetic variables.

ASSEMBLY

Extensive investigation during the assembly design phase suggested the need to eliminate many of the syntax constraints of previous GPSS implementations and provide the user with additional debugging aids and more thorough error checking.

NGPSS/6000 permits a more extensive use of both symbols and matrices. A block or EQUed symbol may be used anywhere a constant is legal; for example, to specify the rows and/or columns on a matrix definition statement. The extensions to the use of symbols in conjunction with use of the SYN statement gives the user greater flexibility than was previously available. For example,

```
ROWS    SYN     10
CLMNS   SYN     20
1       MATRIX MH,ROWS,CLMNS
```

permits the user to vary the number of rows and columns in multiple MATRIX definition statements by simply changing two cards.

- Another extension for the user of symbolics is in the READ mode where the use of symbols is not limited to those defined in that run. Any symbol defined in the SAVE run is automatically defined and can be used in the READ run; in addition, new symbols may be used and will be assigned the next available number for the particular entity type.
- Because all GPSS variable statements are decoded and translated into Reverse Polish notation, there is no limit on the number or depth of parenthesis.
- Additional optional fields were added to the MATRIX definition statement allowing the user to define, EQU, and RESTORE a matrix within one statement.

Numerous debugging aids have been added to the GPSS language. These include an optional cross-reference of all numerically referenced entities in addition to the standard cross-reference of all symbolically referenced entities, a second symbol listing of all symbolically referenced entities in entity number order within entity type, and a summary of the entity allocation. The first scan of function follower cards has been changed so

that the search for follower points terminates at the first legal card type; thus, an error in the number of function points does not cause the entire model nor any part of it to be "lost" in the search for the follower points.

PASS2 of the Assembly Phase was completely redesigned, especially that portion of it which scans and decodes block arguments. Block argument decoding is completely table driven and thus the arguments are examined according to the type of argument expected and more complete error checking is possible. The error message and the argument in which the error occurred are printed in addition to the error number; this eliminates any possible confusion concerning in which argument a particular error was found.

Additional Assembly Phase control options have been incorporated in NGPSS/6000. All comments -- both comment cards and comments on any statement -- can be eliminated from the Assembly Listing; this is for use in particular with classified models. Since a free-format scan of the input images requires extra system time, this preliminary scan is performed only when requested. In addition, the free-format input may be listed before the Assembly Listing and a complete source file of the model in fixed format is written so that the user may assemble from the fixed-format file in future runs.

INPUT

In NGPSS/6000 the functions of the Input Phase have been greatly reduced. This was done by moving many of the tasks previously performed by input, particularly in the processing of blocks, to assembly and others to execution. This reduced much of the overlapping of tasks between assembly and input, and input and execution that are present in other implementations of GPSS and lead to a loss of efficiency in running time and core size. For instance, the source code is scanned and decoded, in assembly, and a binary file rather than a coded file is passed to input. Thus, rather than having to scan and decode an intermediate file, the input module, in the case of blocks, has only to store the binary information in the entity area and GPSS common. Similarly, in the case of GPSS control cards such as INITIAL, CLEAR and RESET, the necessary fields have all been scanned

and decoded in assembly and only have to be processed in input. Many of the errors are now caught in assembly rather than in the input phase, thereby reducing wasted processing time. A great deal of overlapping of tasks between input and execution has also been eliminated by having all evaluation of SNA's done in execution. This eliminates the need of having all of the SNA evaluation routines present in the INPUT module, thus saving some core. For instance, rather than having the first transaction of a GENERATE block or JOBTAPE created in INPUT, they are done in execution.

Aside from the elimination of all tasks from input that resulted in an overlapping of efforts between modules, a number of other improvements have been made:

1) The READ/SAVE feature executes faster and more efficiently since all pointers to the entity area and GPSS common are stored as relative addresses. Thus, when a saved model is READ the pointers do not have to be adjusted as they would have to be if they were stored as absolute addresses.

2) The INITIAL card has been changed to permit the initialization of Random Number Generators and the character initialization of matrices. Since in NGPSS/6000 the number of random number seeds is reallocatable, the user can selectively initialize whichever ones he wishes to. Character initialization is very useful for printing out reports.

3) At the end of model execution, input prints out the maximum amount of GPSS common and the size of the entity area used during the run. Thus, the user always knows exactly how much common he has available for future additions to his model, or how much common he can eliminate from his model so that it will require less core.

4) A permanent disk-resident matrix data reference library whose data can be accessed randomly was added. Its implementation required no change to the GPSS language structure, merely the addition of 3 control cards; one to create the data library

(KREATE); another to store data in a permanent library (MSTORE) and a third to retrieve it (RESTORE). In addition, the user can retrieve a matrix from a permanent library by use of the E field of a MATRIX definition card. By use of the data base feature the cost of inputting data into a system can be reduced since the data need only be input once and then can be accessed randomly rather than sequentially. Also, the data base can be accessed and modified independent of a model and thus a given model can be run with a variety of data or vice versa.

In NGPSS/6000 the input phase has been streamlined to provide maximum overall efficiency in terms of running time and core size by moving a number of its functions to other modules, adding some new control cards and expanding on the functions of others.

EXECUTION

In the design of the execution phase, the following factors were established as critical considerations:

1) Central processor usage
2) Total memory needed during execution
3) Ease of program modification

One of the major criticisms of existing versions of GPSS has been the total CPU time needed for execution of a large scale model. The objective was to decrease this requirement while staying within memory constraints. Also needed was a system in which enhancements could be made to the language with a minimum of program revision.

In the internal structure of the execution phase, there are two areas which are critical to execution time. The first is the evaluation of block arguments and the second is the structure of the routine which scans the current events chain. Since GPSS is an interpreter each block argument must be evaluated every time the given block is executed. To increase the speed of execution the assigning of default values to missing arguments was given to the assembly phase where each block is processed only once. Also, within execution, space was reserved for the maximum number of arguments for each block. This meant that the arguments could be evaluated as needed and the amount of duplications in argument evaluation could be decreased. The core allocation for

matrix arguments was also reassessed. NGPSS/6000 generates a matrix packet for each matrix argument thus nulifying the need for multiple words for each block argument. (See Figure 1 for core allocation of a typical block.)

Because GPSS is a discrete system simulation language, it must be structured so that every active transaction is operated upon after each change in the status of the system. If many transactions are moving at any time, then a great deal of overhead is used in scanning these events. To decrease this overhead the chains use relative addresses and are linked both forward and backward (displacements off the base). The descriptive transaction bits have been moved from the transaction common area to the fixed area to allow for ease in checking to see if the transaction is active. These features along with the incorporation of all chain headers into a common area compensate for the lack of partial word operations on the CDC 6000.

In order to decrease core requirements and use the CDC 6000's 60-bit word, the GPSS entity area has been redesigned. By compressing these areas from 1 to 4 words of core have been saved per entity. Variable statements have also been redefined. Instead of constructing a pseudo object code, NGPSS/6000 uses a Reverse Polish notation. This type of expression removes the need for temporary storage space within each Variable; instead one common area is used by all Variables as an area for storage of temporary calculations.

The final objective was probably the easiest to accomplish. This was done by completely modularizing the execution phase, for example, one routine to evaluate all block arguments, one routine to calculate the address of all entities. By doing this, the addition of a new block means only coding the block and inserting its block mask into the branch table.

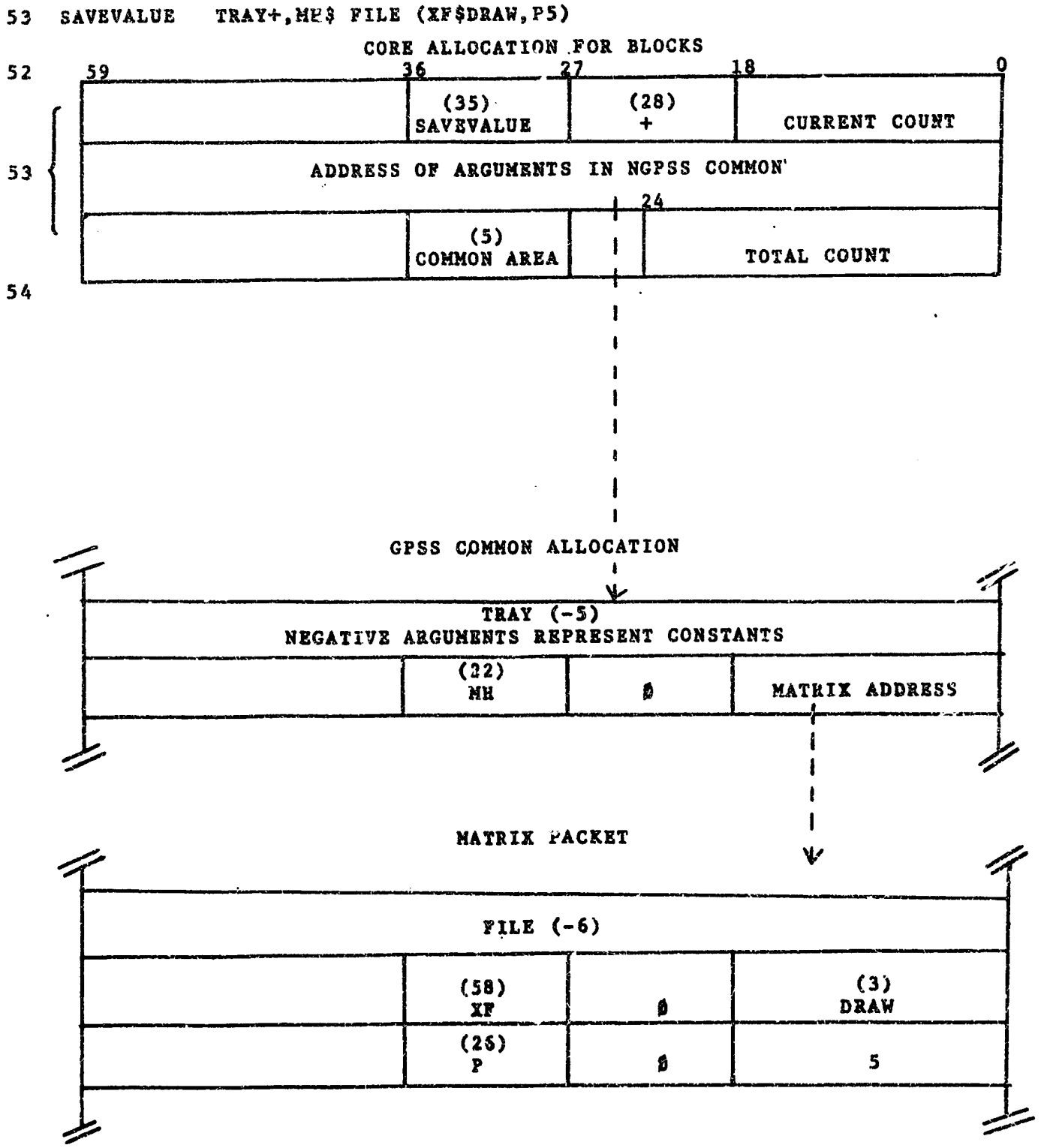
OUTPUT

The Output Phase, which prints out the Standard GPSS Statistics, contains a number of minor improvements:

1) Block Symbols are printed out with the block counts where they exist.
2) A random access file is used to retrieve entity symbols thus cutting down on retrieval time.

FIGURE 1

CORE LAYOUT FOR A TYPICAL NGPSS BLOCK



3) Only matrix savevalue rows and columns that are non-zero are printed out.

## TESTING AND SUMMARY

The testing of NGPSS/6000 has involved a series of models originally coded in GPSS/360 and GPSS V. Since the operation of the pseudo random number generators are machine dependent, a set of models were converted to contain a series of GPSS Variable statements which provided the random number sequence. The running of these models has produced matching statistics and transaction data limited only by the round-off errors in the calculation of utilization factors. As a result of these tests, it is felt that the objective of being able to run GPSS models on a CDC 6000 computer has been achieved.

## REFERENCES


1. IBM General Purpose Simulation System V User's Manual, SH-20-0851, November 1970.

2. GPSS/NORDEN Simulation Language, Form 910-1, National CSS, Inc. March 1971.

3. User's Guide to Interactive Simulation (A Superset of GPSS/360), Norden Report #4300 R 0001, August 20, 1970.

4. User's Guide to Conversational GPSS (GPSS/360-NORDEN), Norden Report #4269-R 0003, December 1969.

5. Approaching a Universal GPSS, Jerry Katzke and Julian Reitman, March, 1972.

6. A Computer-Aided Environment for Systems Design, Julian Reitman, October 25, 1971.