DOCUMENT RESUME

ED 201 325 IR 00_ _97

AUTHOR	Clymer, S. J.
TITLE	Software Partitioning Schemes for Advanced Simulation Computer Systems. Final Report.
INSTITUTION	Teledyne Brown Engineering, Huntsville, Al.
SPONS AGENCY	Air Force Human Resources Lab., Brooks AFB Texas.
REPORT NO	AFHRL-TR-80-42(1)
PUB DATE	Feb 81
CONTRACT	F33615-78-C-0013
NOTE	158p.
EDRS PRICE	MF01/PC07 Plus Postage.
DESCRIPTORS	*Computer Assisted Instruction; *Computer Programs; Databases; *Flight Training; Management Information Systems; *Mathematical Models; Simulated Environment; *Simulation

ABSTRACT

Conducted to design software partitioning techniques for use by the Air Force to partition a large flight simulator program for optimal execution on alternative configurations, this study resulted in a mathematical model which defines characteristics for an optimal partition, and a manually demonstrated partitioning algorithm design which implements heuristic controls based on the mathematical model statement. This report reviews the study objectives, background, approach, and results; defines the software partitioning problem environment, partitioning goals, and alternative approaches; presents the technical details of the software partitioning algorithm which was developed and manually demonstrated under this contract; addresses implementation considerations; and recommends a schedule of tasks for algorithm automation, verification and validation. A brief recapitulation of the study findings, related work, and areas of further study concludes the report. Appendices include user inputs and report formats. (CHC)

AFHRL-TR-80-42 (Part I)

AIR FORCE HUMAN RESOURCES LABORATORY

SOFTWARE PARTITIONING SCHEMES FOR ADVANCED SIMULATION COMPUTER SYSTEMS

By

S. J. Clymer
Systems Division
Teledyne Brown Engineering
300 Sparkman Drive
Huntsville, Alabama 35807

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85224

February 1981

Final Report

Approved for public release; distribution unlimited.

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235

NOTICE

When U. S. Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This final report was submitted by Systems Division, Teledyne Brown Engineering, 300 Sparkman Drive, Huntsville, Alabama 35807, under Contract F33615-78-C-0013, Project 1114, with the Operations Training Division, Air Force Human Resources Laboratory (AFSC), Williams Air Force Base, Arizona 85224. Pat Price was the Contract Monitor for the Laboratory.

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

MARTY R. ROCKWAY, Technical Director
Operations Training Division

RONALD W. TERRY, Colonel, USAF
Commander

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFHRL-TR-80-42 (Part I)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE PARTITIONING SCHEMES FOR ADVANCED SIMULATION COMPUTER SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) S.J. Clymer		8. CONTRACT OR GRANT NUMBER(s) F33615-78-C-0013
9. PERFORMING ORGANIZATION NAME AND ADDRESS Systems Division, Teledyne Brown Engineering 300 Sparkman Drive Huntsville, Alabama 35807		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62205F 61142304
11. CONTROLLING OFFICE NAME AND ADDRESS HQ Air Force Human Resources Laboratory (AFSC) Brooks Air Force Base, Texas 78235		12. REPORT DATE February 1981 13. NUMBER OF PAGES 462
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Operations Training Division Air Force Human Resources Laboratory Williams Air Force Base, Arizona 85224		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This report consists of two parts. Part I includes pages 1 through 152. Part II contains pages 153 through 460.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

multiple processors
software partitioning
real-time computational design evaluation
flight simulation
data base management
software design
computer systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The overall objective of this study was to design software partitioning techniques that can be used by the Air Force to partition a large flight simulator program for optimal execution on alternative configurations. The results were a mathematical model which defines characteristics for an optimal partition and a manually demonstrated partitioning algorithm design which implements heuristic controls based on the mathematical model statement.

DD FORM 1 JAN 73 1473

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

This report was prepared by the Systems [illegible], Teledyne Brown Engineering, Huntsville Alabama. The work was done under Contract F33615-78-C-0013 with the U.S. Air Force Human Resources Laboratory (AFHRL).

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

This report documents the Software Partitioning Schemes for the Advanced Simulation Computer Systems Study performed by Teledyne Brown Engineering (TBE) under Contract No. F33615-78-C-0013 for the Air Force Human Resources Laboratory (AFHRL). The report contains five sections. Section 1 introduces the study objectives, background, approach, and results. Section 2 defines the software partitioning problem environment, partitioning goals, and alternative approaches. Section 3 presents the technical details of the resultant software partitioning algorithm developed and manually demonstrated under this contract. Section 4 addresses implementation considerations and recommends a schedule of tasks for algorithm automation verification and validation. Section 5 concludes with a brief recapitulation of the study findings, related work, and areas of further study.

1.1 OBJECTIVES

The overall objective for this study was to design software partitioning techniques that can be used by the Air Force to partition a large flight simulator program for optimal execution on alternative multiple processor configurations. In particular, the Air Force needs a software partitioning algorithm for use in conceptualizing, manipulating, and evaluating candidate flight trainer computational designs. Major design objectives pursued by TBE in deriving the software partitioning algorithm included emphasis on potential automated steps, manual feasibility demonstration, and recommended implementation steps for its use by the Air Force.

1.2 BACKGROUND

It has been evident for some time that significant increases in computer system performance may be realized by using two or more smaller processors connected in parallel, as opposed to one large processor. This concept has been utilized in many real-time flight simulators where each of several computers performs a specific task. Future trends are toward further expansion of this concept to include not only tasks that may be executed in parallel but also tasks that must execute serially because of temporal relationships. This causes many multiple processor configurations to be applicable to flight training simulators and complicates the problem of allocating the software among the processors.

Typically, the design of a computer system is an iterative procedure. Certain portions of the hardware and software can be designed independently, but the remaining portions must be designed interactively. With the rising cost of software, it has become more and more important to know the effect of computer hardware design on the design of the software as well as the effect of the software design on the selection and interconnection of the hardware to develop the optimum design for the computer system.

This study has pursued the development of an algorithm that will facilitate the partitioning of both parallel and sequentially dependent tasks to a given hardware configuration. The algorithm has the potential of being automated.

1.3 APPROACH

This study was comprised of three phases: Phase I - Literature Search, Phase II - Simulator Analysis, and Phase III - Algorithm Design and Demonstration. This three-phased approach provided a logical sequence of research and analysis that resulted in the delineation of the partitioning technique presented in this report.

The Phase I literature search focused on current documentation in two major technical areas. The first area concerned flight training simulator computational subsystem designs. The second area addressed software partitioning schemes for allocation of parallel and serial application tasks to advanced multiple processor configurations.

The Phase II effort was subdivided into two parts. The first part was the analysis of literature collected to properly identify the software partitioning goals with respect to flight training simulator designs. The second part was the selection and expansion of the specific approach for the techniques to be applied in the algorithm design to achieve the design goals. Partitioning approaches considered included manual allocation schemes, real-time dynamic task allocation schemes, and a mathematical goal program statement of the allocation problem. The mathematical goal program model approach was selected because of its potential for systematically obtaining optimal partitions and related quantitative measures in an automated mode, which are responsive to alternative candidate design features. The features and measures that can be modeled are described in Section 3 in terms of the mathematical model, algorithm design, and algorithm feasibility demonstration. Model measures include task sizing and timing; processor utilization; memory storage, retrieval, and sizing; and real-time task constraints and relationships.

Some problems were encountered in pursuing the Phase III design to implement the mathematical goal program model when allocating a large number of tasks and data blocks to a large number of processors, memories, and peripherals comprising the candidate configuration. It became evident that a heuristic goal program algorithm needed to be designed that interfaces with a linear program optimizer to obtain "good" task partition allocations for large partitioning problems. TBE's Input/Output Requirements Language (IORL) supplemented with flowcharts was used to delineate the algorithm design and provide the steps for performing a manual demonstration of the algorithm's feasibility.

1.4 RESULTS

One of the most important results of this study was a mathematical model defining partitioning parameters and measurements. From these parameters, a set of guidelines has been recommended for the establishment of a centralized automated flight training simulator computational design data base repository for the Air Force. These design parameters address five major areas, including flight training simulator computational interface requirements, baseline software task/data descriptions (independent of hardware implementation), candidate hardware configuration specification, a technology data base, and (most important) design evaluation user interface data options. These parameters along with the partitioning mathematical model provide steps for the implementation of an automated partitioning algorithm for real-time simulators. Detailed recommendations for algorithm implementation are provided in Section 4.

Section 5 expands TBE's findings, including related aspects of our Advanced Multiple Processor Configuration study contract encompassing areas for further research and development. In the multiple processor area, the impact of heterogeneous processor configurations and potential reconfiguring capabilities is currently being investigated. A major area for future study is the impact of higher order architectures on partitioning allocation.

2. SOFTWARE PARTITIONING

To develop the software partitioning algorithm design goals, TBE addressed the definition from both general system software design and particular flight training simulator software design viewpoints. This section supplies the basic definition of the software partitioning environment, the design goals selected for flight training simulator software partitioning features, and alternative approaches considered during this study.

2.1 PARTITIONING ENVIRONMENT

To fully appreciate the software partitioning environment and its associated steps, one must first examine its relationship with the system life cycle. Then, flight training simulator system life-cycle peculiarities must be considered. The questions posed by this study in both these areas concerned the identification of the software application task features that are peculiar to advanced real-time simulation computational systems and that influence the software design partitioning process. The system and flight trainer life cycles are now described for the general system, followed by a description of the flight training simulator software partitioning features. Emphasis was placed on identifying software features that characterize an optimal partitioning scheme and that account for alternative candidate configurations and provide partitions that meet real-time load balance constraints.

2.1.1 System Life Cycle

Figure 1 depicts the major phases of a system development effort. The development phases that directly relate to or influence software partitioning include subsystem interface requirements, subsystem functional specification, and subsystem detailed design. In addition, during the operational maintenance of the system, any changes that are deemed necessary (to either correct for a design deficiency or oversight, or to implement an expanded capability) imply that a repartitioning of tasks may be needed to accommodate the required change. This phasing relationship to partitioning holds for any system, whether it is an aircraft, computer center, air defense system, ..., or a flight training simulator system.

For purposes of this study, the detailed design phase was selected as the major area where software partitioning parameters become known. Prior to this phase, a system partitioning is generally performed to denote the major subsystems and their respective interface functions. After the detailed design phase, actual hardware is procured from which prototype build implementation is initiated. Therefore, the detailed design phase has the greatest influence on mapping software tasks to hardware and vice versa.

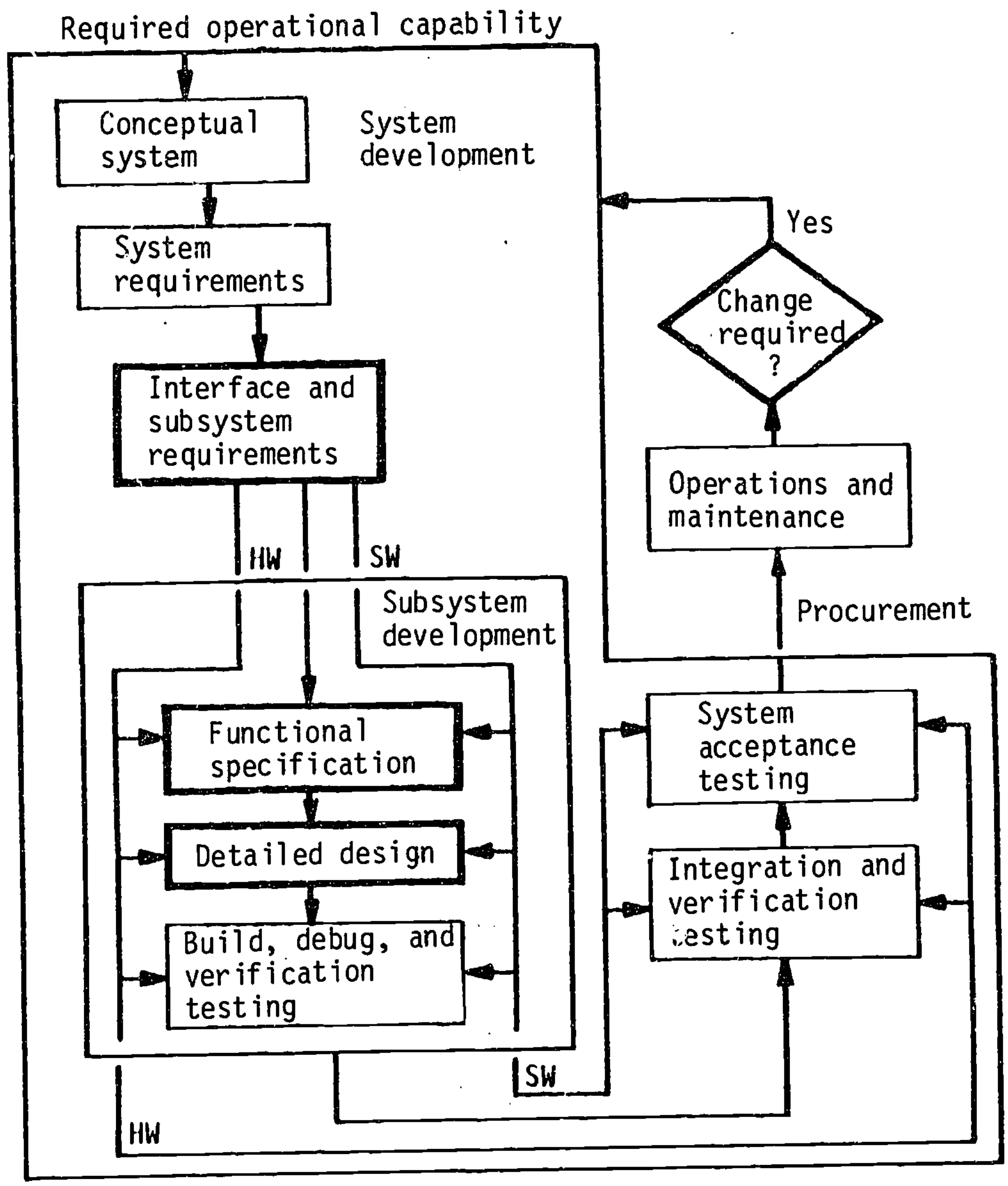


Figure 1. The system life cycle addresses partitioning at subsystem, function, and detailed design phases for new and/or modified system development efforts.

The design of a multiple computer system traditionally has begun with the hardware selection. Once the computer system has been selected, the development of software begins. During development and even after the system is installed in the field, there are various modifications to both the hardware and the software. Because software has traditionally lagged the hardware development activities, the hardware has had a direct influence on software partitioning. As the details of the software tasks become known, projected hardware resources are typically found to be inadequate, which necessitates the acquisition of additional processors and/or memories to meet system interface requirements. A software partitioning algorithm must be able to address software application design parameters, which are independent of a particular hardware configuration, to permit a variety of design tradeoffs to be evaluated for alternative candidates prior to the exact configuration selection.

Once a system enters the operational phase, maintenance becomes the prime cost factor (indeed, maintenance cost is the largest cost of the system life cycle). Change and configuration controls are necessary for a system or subsystem of any significant size. As technology advances, new software and hardware architectures may need to be implemented. A tradeoff must be made to decide whether to convert or totally redesign existing software. A software partitioning algorithm should provide useful information regarding allocation of current baseline software design tasks to the new or modified hardware architecture. As with design development, software partitioning in the operational maintenance phase addresses the design details of any proposed changes.

The key factor for flexible software partitioning (from the system life-cycle viewpoint) is the ability to define software design attributes in terms of the dependent application software task/data flow relationships. The software attributes should remain independent of, but be mappable to, a particular processor architecture. The proliferation of requirements languages (RLs) and higher order languages (HOLs) is a testament to this emerging philosophy in the DOD community. The distinction between an RL and an HOL is that RLs are not currently automated to the extent of target machine code generators for the RL. An HOL such as JOVIAL, HAL-S, or PL-1 supports interpretation, data management, and code generation from machine-independent HOL source code to an intermediate level language that can then be specifically translated to any one of the languages supported by different target machines. Once the tasks have been defined in a suitable RL and HOL, the problem still exists as how they can best be partitioned or allocated to the candidate architecture. Once allocated, the resulting partition should be evaluated in terms of predicted performance and cost/risk assessments by a software partitioning model. Iterative feedback from this performance evaluation model can then be used to perturb the partition based on performance penalties to derive a well-balanced software execution sequence.

2.1.2 Flight Trainer Life Cycle

In addition to problems associated with the general system life-cycle environment, the simulation training system environment offers special considerations and problems with respect to software partitioning. Aircraft systems are continuously being upgraded, and this causes changes to training requirements. Manual interfaces change when new or modified weapons systems, embedded onboard computer systems, and operational tactical policy changes are introduced. These problems are really no different from problems encountered during the maintenance phase of the actual system. The key issue is when and how actual system changes are received, evaluated, and introduced into the training requirements.

Actual system test and performance measurement tools can and should provide useful inputs for simulator training software required to support the new/modified devices. In the case of embedded computer systems, simulated training scenarios could provide additional reliability tests of the actual onboard computer systems as well as the prime goal of training personnel. As a result of these considerations, the partitioning algorithm should facilitate modular design definition input changes and permit new technology configurations to be introduced as needed to support a given evaluation. This should also include the ability to fix allocations of certain functional tasks, such as a set of onboard computer tasks, while permitting others to be allocated by the partitioning algorithm.

AFHRL supplied a benchmark problem and the detailed design documents and source code listings from the Advanced Simulator for Undergraduate Pilot Training (ASUPT, now known as Advanced Simulator for Pilot Training (ASPT)). These documents were analyzed to obtain estimates on the complexity and sizing of flight simulator software partitioning. This analysis identified 50 major application (both real-time and support) tasks (some of which would be duplicated to support multiple training stations, instructor consoles, weapon systems, and aircraft models). The results of this analysis were presented at an interim briefing.

It should be noted that a task is related to the application. Its ultimate operational realization may be software, firmware, hardware, or a combination of these, depending on the selected design configuration. The tasks being considered for the partitioning algorithm are related to the computational subsystem of real-time flight training simulators.

Further analysis revealed that the trainer computational subsystem is really comprised of a set of smaller functional subsystems, such as simulator facility control, visual computational support, and simulated aircraft mathematical models; thus, the number of processors and number of tasks for which selected software functions are being allocated is reduced to approximately 30 tasks to three processors using a common, shared multiport memory. In summary, flight trainer computational configurations have both a functional partitioning of processors and a task partitioning within each functional processor group.

2.2 DESIGN GOALS

Software partitioning of tasks to alternate candidate multiprocessor configurations must be a systematic process based on measurable evaluation goals. The selected design goals for the partitioning algorithm developed are as follows:

(a) With software system task flow inputs given, partition tasks to a user-specified multiprocessor hardware configuration subject to input constraints

(b) Identify interdependencies among the tasks that require communication links

(c) Incorporate dynamic performance evaluation feedback to determine the best partition to preclude system deadlocks and account for critical path task precedence orderings

(d) Provide a means of balancing the processing load as a function of processor utilization, which is evenly distributed among the processors such that no one processor is saturated while others remain idle for appreciable periods of time

(e) Provide cross reference of task(s) assigned to each processor and processor(s) assigned to each task

(f) List critical constraints when a valid partition is not obtainable

(g) Provide a development cost estimate as a function of task sizing and instruction mix, which is related in terms of assigned candidate processor language compilers and debug tool measures.

In deriving this set of goals, several issues have been discussed pertaining to the evaluation environment in which the partitioning algorithm is to operate. The baseline set of questions was:

(a) At what point(s) in the system development cycle is the algorithm to be used?

(b) What timeframe and computer resources are anticipated for candidate evaluations?

(c) To what extent will the system requirements be formatted? In what format?

(d) To what extent will the alternative candidate design configuration be documented? In what format?

The answers to these questions relate directly to the level of software partitioning and types of system parameters that can be modeled, allocated, and measured. In summary, there are no definitive answers to these questions since each flight trainer evaluation tends to be tailored to specific needs. This does not mean that systematic methodologies and standards do not exist, but they do differ from one project to another. The potential use of an automated partitioning algorithm will require systematic collection and development of flight trainer requirements, software specifications, and candidate configuration inputs. This contract has concentrated on the definition of partitioning algorithm logic in terms of design inputs which are transformed via technology data and user evaluation options to assist and assess the partitioning of tasks for a given candidate configuration.

2.3 ALTERNATIVE APPROACHES

Software partitioning to date has been primarily a manual process based on experience gained in development of previous flight simulators. The designer community continually evolves and improves partition allocations using projected resource requirements and implementing the partition to see how well it performs. In some cases, real-time allocation is determined by a master computer using a predefined assignment scheme that incorporates certain dynamic application considerations. These schemes, whether manual or partially preprogrammed controlled, are not easily automated, since they generally require that a specific system allocation be implemented for a given configuration. Manual projections are limited to a few alternatives for a given type of configuration, but they must be redone for alternative configurations.

In surveying potential automated models to meet the design goals, the basic problem to be solved is one of distributing the software system tasks and related data blocks to a candidate hardware architecture network such that a representative stressing simulation load is handled. In general, this type of problem is typical of mathematical programming problems addressed in an operations research (OR) environment. Within this field, there are a variety of algorithms. The following are some of the more familiar:

1. Transportation problem of product transport from production locations to warehouses and customer distribution centers to meet customer demand at minimum cost.

2. Traveling salesman optimal route determination to service customers

3. Knapsack packing of items required for a camping trip to be distributed evenly among campers

4. Capital budgeting problem of choosing among independent investment alternatives to maximize return subject to current investment fund constraints

5. Machine shop production scheduling to meet product demand deadlines with minimum machine restructure between jobs and given employee mix.

The software partitioning problem has attributes similar to each of these.

In the case of the software partitioning problem, a descriptive statement of the model is as follows:

1. Find a partition that best satisfies alternative evaluation priority functions:

   a. Balance the processing load among the processors
   b. Balance the memory storage utilization
   c. Minimize development costs.

2. Subject to:

   a. Real-time task resource requirements
   b. Predicted performance simulation feedback.

When defining a software task partitioning model, a number of factors must be considered. The model can very quickly get out of hand in terms of size for current optimization techniques. Thus, the model design developed under this contract restricted itself to a static allocation problem that is mathematically stated as a linear goal program problem in Section 3.1. It is static in that it is a generalization of the real-time application tasks to be allocated to a given candidate configuration. In this sense it is not a dynamic real-time allocation algorithm. The static model is very useful in the candidate design evaluation mode, since many numbers are based on predicted task sizing and timing plus anticipated computation iteration frequencies to support given training loads. The static model permits average to worst-case growth analysis in a systematically controlled evaluation environment, which provides the means to ensure a complete design description has been input and independently provides a measure of processor utilization, memory utilization and predicted software development cost.

Even in the static model environment, optimization data base sizing and numerical roundoff problems are encountered for evaluation of a computational system involving much more than three processors, 20 tasks, 40 data blocks, and four memories. Specific sizing is addressed in Section 3.2. For this reason, a heuristic model has been designed. A heuristic model is a means of limiting computations to a logical sequence of iterative improvements via allocation tradeoffs until a certain objective level is either found to be feasible or a bottleneck has been isolated.

This section has discussed partitioning considerations. The resultant algorithm design details are highlighted in Section 3. Implementation considerations are given in Section 4. Section 5 incorporates areas for further research with respect to optimizer techniques and data base selection.

3. MODEL DEVELOPMENT

Software partitionir . development is presented from three different technical viewpoin ais section, including the mathematical definition, the detailed highlights, and a feasibility demonstration synopsis. The mo expressed in generic computational system terms where the major co onents are tasks, data blocks, processors, and memories that are partitioned to service an external baseline load environment. The mathematical model definition delineates all the parameters and the basic relationships that must be satisfied for a valid partition. It also provides a statement of objective functions that permits optimization of the partition when the basic relationships are found to have a feasible solution (i.e., a feasible partition).

The algorithm design highlights are presented here in terms of the systematic procedural step features with cross-references to detailed appendices. Appendix A provides user input information. Output report formats are provided in Appendix B. Appendix C contains the feasibility demonstration that emphasizes the user environment of input formulation, critical intermediate step results, and final output summaries. Detailed computations and design logic are enumerated in Appendix D.

3.1 MATHEMATICAL STATEMENT

This mathematical statement provides mathematical terminology and definitions for alternate evaluation priorities and constraint formulation based on a generic statement of a candidate configuration for which a set of software tasks are to be partitioned. Each mathematical symbol is defined when first introduced. In addition, Appendix C contains a master list of mathematical symbols and related design definitions. A special effort has been made to use a unique symbol for a given entity. It utilizes a combination of symbol definition with a combination of linear programming and goal programming model formulation terminology. Although knowledge of these modeling and solution techniques is helpful, it is not essential to the understanding of the basic expression of the software partitioning problem model.[1] The solution techniques with respect to the software partitioning model are considered in the design highlights of Section 3.2. The model is now stated.

3.1.1 Mathematical Terminology

The mathematical model formulation permits the major decision variables to be enumerated in terms of a baseline software load for a

[1]Ignizio, James P., Goal Programming and Extensions, D. C. Heath and Company, Lexington, Massachusetts, 1976

given real-time interval of length, τ. In the case of the flight trainer, τ might be chosen to represent the maximum time permissible for a complete real-time cycle. The baseline load could represent a stressing training mix of tasks and data relationships that must be performed to support the given trainer facility exercise; for example, a two-on-one, air-to-air, combat maneuvering situation may be selected. For more detailed partitioning loads, τ could be selected to represent a specific segment of the real-time cycle to further analyze and partition parallel versus dependent task/data flow relationships.

The major decision variables (outputs of the algorithm) with respect to software partitioning allocation are defined as follows:

x_{tp} = 1, if task t is assigned to execute on processor p

= 0, otherwise

e_{tp} - number of task t executions on processor p for the evaluation problem time period

y_{tp} - development cost to implement task t on processor p as currently partitioned

s_{mb} = 1, if memory storage m contains block b

= 0, otherwise

h_b - number of memories where block b is stored

a_{mpti} - number of times input block i of task t is input for task t on processor p from memory m.

w_{mpto} - number of times output block o of task t is written or updated by task t on processor p to memory m.

These outputs are determined for a given set of software task and candidate architecture inputs. The basic algorithm control inputs are denoted by:

T - number of tasks to be allocated to processors

P - number of processors

M - number of memories

B - number of distinct storage blocks to be allocated to memories (this includes instruction and data blocks)

Q - number of communication links

β - maximum number of input and/or output blocks per task.

The values of these parameters control the overall algorithm sizing, timing, and looping logic.

The baseline task load may be represented as configuration-independent, processor-dependent, and memory-dependent input parameters. The configuration-independent input parameters are defined as follows for each task, t:

N_t - number of times task t is to be executed during the evaluation interval, τ, for which partitioning is being done

S_t - maximum time limit per task t execution

I_t - number of distinct input blocks for task t

ι_{ti} - global data block index for task t input block i

A_{ti} - percent of information input for task t from block i

O_t - number of distinct output data blocks for task t

o_{to} - global data block index for task t output block o

Ω_{to} - percent of information output from task t to block o.

The processor-dependent task inputs are defined as follows for task t on processor p:

c_{tp} - time for task t execution on processor p

R_{tp} - resource task management coefficient for task t on processor p if time or data enabled task (these tasks require periodic enablement or polling by the processor to which they are assigned)

r_{tp} - resource task management per task t execution on processor p for slaved enabled task (these tasks are enabled by another task)

d_{tp} - the cost coefficient for developing task t to run on processor p independent of allocation

δ_{tp} - the cost coefficient for resource management of task t development on processor p.

Section 4.2 discusses the implementation means for computing these values based on independent task descriptions, processor configuration, and a technology data base. The mathematical model assumes that these values are known.

In addition to the task-to-processor allocation relationships, the storage allocation of blocks to memories operates on a similar concept. A master block list of distinct data and/or instruction blocks is independently defined and then mapped via the candidate configuration and technology memory parameter inputs to supply the following parameters with regard to block b, memory m, processor p, and communication link q:

ℓ_{mb} - length in bits of block b when stored in memory m

L_m - length of memory m in bits

a_{mp} = 1, if access from memory m to processor p exists, i.e., there is at least one access link q for m and p

= 0, if otherwise

α_{mp} - bits/second transfer rate from memory m to processor p based on statistical composite of access links for p and m

W_{mp} = 1, if processor p is permitted to change contents of memory m, i.e., there is at least one write access link q from p to m

= 0, if otherwise

ω_{mp} - bits/second transfer rate from processor p to memory m based on statistical composite of write access links for p and m.

The task relationships to these blocks are defined as part of the real-time constraints in Section 3.1.5.

3.1.2 Processor Utilization and Growth Balance

Given the mathematical terms defined in Section 3.1.1, the processor utilization, U_p, associated with a partition may be expressed as follows (for each processor p=1 to P):

$$U_p = \frac{1}{\tau} \sum_{t=1}^{T} \left[(c_{tp} + r_{tp})\, e_{tp} \right.$$ task computation and resource management time

$$+ \sum_{i=1}^{I_t} A_{ti} \sum_{m=1}^{M} \alpha_{mp}^{-1} \ell_{mti} a_{mpti}$$ task input processing

$$+ \sum_{o=1}^{O_t} \Omega_{to} \sum_{m=1}^{M} \omega_{mp}^{-1} \ell_{mto} w_{mpto}$$ task output processing

$$+ R_{tp} \; x_{tp} \Big] .$$ task resource management

An absolute constraint is that:

$U_p \leq 1$ for p=1 to P.

In other words a processor, p, cannot be more than 100% allocated.

The objective function for processor balance may be written:

$$\text{Minimize} \sum_{i=1}^{P-1} \sum_{j=i+1}^{P} |U_i - U_j| .$$ Minimize differences in processor loads

It should be noted that the presence of absolute values implies a non-linear objective. The processor utilization balance can be mapped (via a ranked ordering of the $U_p \rightarrow U_k'$ such that $U_i' \geq U_j'$) to a linear objective for a given partition.

This objective statement assumes that perfect balance is the ultimate or optimal partition. The candidate design being considered may represent only a portion of a bigger design evaluation problem. In this case, the use of certain processors may be favored, whereas others should not be considered. To handle this more realistic partitioning situation, each processor has two additional parameters, which are user-specified:

L_p - absolute upper limit for processor p's utilization

G_p - goal or target limit for processor p's utilization.

With these additional parameters, the following constraints apply:

$G_p \leq L_p$	Goal must be less or equal to the absolute limit.
$U_p \leq L_p$	Each processor must be below its absolute limit.

The objective for the optimal partition in terms of processor utilization becomes:

$$\text{Minimize} \sum_{i=1}^{P-1} \sum_{j=i+1}^{P} (U_i - G_i) - (U_j - G_j) \quad .$$

This basically states that the processor utilization is in balance with respect to user-specified goals. In the case of a flight trainer software partitioning evaluation, G_p could reflect a percentage that allows for future growth. Thus, $G_p = 0.60$ reflects a 40% growth factor for processor p.

The algorithm as currently designed (Section 3.2) assumes that an initial feasible solution is provided by the candidate design and utilizes a heuristic solution based on the absolute difference between the most heavily loaded processor and the least loaded, taking into account the goal growth reservation to distribute the process load.

3.1.3 Storage Utilization and Growth Balance

Storage utilization, u_m, may be expressed for each memory unit, m=1 to M, as:

$u_m = \frac{1}{L_m} \sum_{b=1}^{B} \ell_{mb} s_{mb}$.	Sum of blocks stored divided by total memory

As with the processor balance formulation, storage utilization cannot exceed the capacity of the device.

$$u_m \leq 1 \text{ for } m=1 \text{ to } M$$

In addition, storage growth balance can be established with a respective goal utilization, g_m, and an absolute limit, δ_m, for each memory as follows:

$$\text{Minimize} \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} (u_i-g_i) - (u_j-g_j)$$

where

$$u_m \leq \delta_m \text{ for } m=1 \text{ to } M.$$

As with the processor utilization, the solution technique defined in Section 3.2 for storage utilization is based on a heuristic driven by the most used and least used memory allocations with respect to input goals.

3.1.4 Development Cost

Software development costs are a function of task complexity and programming support tools available. In particular, the heterogeneous multiprocessor system adds another development cost concern, i.e., coding of a task to perform on more than one processor type. A common program source language significantly reduces duplicated coding efforts. Thus, the development cost for a given software task, t, in the model may be stated as:

$$D_t = \sum_{p=1}^{P} \Big[d_{tp} \; x_{tp} \qquad \text{one-time development}$$

$$+ \; \delta_{tp} \; x_{tp} \qquad \text{resource manager development}$$

$$- \; d_{tp} \; y_{tp} \Big] \qquad \text{duplicate utilization.}$$

where

$$y_{tp} = 0 \text{ for } p=1$$

$$= \max \left\{ \lambda_{ipt} \; x_{ti} \text{ for } i = 1 \text{ to } p-1 \right\} \quad \text{for } p>1$$

where $-\lambda_{ipt}$ = 1, if an identical source language is available on processor i and p ($i \neq p$) for task t

= a technology-specified constant if different languages are to be used ($i \neq p$)

= 0, if i = p.

If the code already exists, then $d_{tp}=0$.

Note that the multiplicative factor for determining y_{tp} can be stated as an equivalent series of linear constraints because of the zero-one variable x_{tp} (task t is either assigned to processor p or it is not). These (p-1) constraints are enumerated as follows for a given task t on processor p (for p > 1).

$$\lambda_{1pt}\ x_{t1} - y_{tp} \leq 0$$

$$\lambda_{2pt}\ x_{t2} - y_{tp} \leq 0$$

$$\vdots$$

$$\lambda_{(p-1)pt}\ x_{t(p-1)} - y_{tp} \leq 0.$$

With this set of constraints, minimizing y_{tp} in the achievement function ensures that y_{tp} will assume the appropriate maximum as defined in the original definition.

The goal objective for software development cost is now stated as:

$$\text{Minimize} \sum_{t=1}^{T} D_t.$$

This is basically a problem of reducing development cost. The design attempted to reduce development cost (Section 3.2) to be less than a user supplied value, $\mathcal{D}$, where $\mathcal{D}$ represents a ballpark estimate for the total software development. The unit used may be man-years or dollars, depending on units established for the technology data base (described in Section 4.2), which will be used to translate the task t instruction mix (Section 4.2) to its one-time development cost (d_{tp}) for processor p.

The common language coefficient, λ_{ipt}, is also a function of the technology-processor-related data (Section 4.2) and the language factor selected for the task.

3.1.5 Real-Time Task Resource Requirements

The major constraint areas interact with the objective priority evaluations to further specify acceptable partitioning attributes. As a minimum, the following constraints apply to basic task resource requirements and processor accountability:

(a) Each task, t, must be assigned to at least one processor. This implies T constraints of the following:

$$\sum_{p=1}^{P} x_{tp} \geq 1 \text{ for } t=1 \text{ to } T.$$

(b) If more than one processor is permitted to perform the same task, a resource management overhead will be allocated to task t processors via the processor utilization objective of Section 3.1.2. However, to ensure that x_{tp} is properly coupled with e_{tp}, the following constraint must be applied:

$$x_{tp} - \frac{e_{tp}}{N_t} \geq 0.$$

In addition, constraints must address task iteration rate and task service times to ensure that real-time task timing requirements are met:

(a) Given that task t must be executed N_t times during the problem time period, τ, the task iteration rate constraint is:

$$\sum_{p=1}^{P} e_{tp} = N_t.$$

(b) If overlap of task t execution is not permitted (i.e., t cannot be executing on more than one processor at a time), the following constraint applies:

$$\sum_{p=1}^{P} (c_{tp} + r_{tp})\, e_{tp} \leq \text{minimum } (\tau, N_t * S_t)$$

where S_t is the maximum time limit for one execution of task t.

Note that if

$$c_{tp} + r_{tp} > S_t$$

then e_{tp} can be automatically assigned a zero value and deleted from consideration.

Task data dependencies must also be satisfied. These constraints include:

(a) All data blocks associated with task input must be available to the processor(s) that are permitted to perform the task. Thus, for input block i_{ti}, the following holds:

$$-x_{tp} + \sum_{m=1}^{M} a_{mp} s_{m i_{ti}} \geq 0$$

for i=1 to I_t, t=1 to T, p=1 to P.

(b) All data blocks associated with task t's output must reside in memory storage m, which can be updated (changed) by any of task t's processor(s) p. If $\bar{x}_{tp}$ satisfies

$$x_{tp} + \bar{x}_{tp} = 1$$

then for a given task output, block $b=o_{to}$, the following holds:

$$x_{tp} + M\bar{x}_{tp} + \sum_{m=1}^{M} w_{mp}\, s_{mb} - h_b \geq 1$$

for t=1 to T, o=1 to O_t, p=1 to P. h_b represents the number of different memories that have duplicate copies of block b; thus, this constraint requires all duplicate blocks to be updated (see next constraint set).

(c) Any duplicate data blocks must be held to a minimum; therefore h_b may be thought of as a penalty to be added as an additional objective function with the following additional constraint:

$h_b \geq 1$ (at least 1 block is in memory)

and

$$\sum_{m=1}^{M} s_{mb} - h_b = 0$$

for $b = 1$ to B.

(d) Input timing must properly account for the number of task t executions on processor p (e_{tp}) for each task input block, i_{ti}, i=1 to I_t:

$$e_{tp} - \sum_{m=1}^{M} a_{mpti} = 0$$

and $a_{mp} \; s_{m i_{ti}} - \frac{a_{mpti}}{N_t} \geq 0$ for m=1 to M

are used to ensure that i_{ti} is available on memory m.

(e) Output timing must account for the number of task t executions on processor p (e_{tp}) for each task output block, o_{to}, o=1 to O_t:

$$e_{tp} - w_{mpto} - N_t (1 - s_{m o_{to}}) \leq 0$$

and

$$\omega_{mp} \; s_{m o_{to}} - \frac{w_{mpto}}{N_t} \geq 0 \text{ for m=1 to M}$$

are used with a corresponding achievement function that minimizes w_{mpto} to ensure that all duplicate blocks of o_{to} are updated.

3.1.6 Performance Simulation Feedback

Sections 3.1.2 through 3.1.5 comprise the fundamental model objectives and constraints that must be set in terms of a valid static allocation of tasks. Performance bottlenecks detected by the simulation mode being developed under separate contract (No. F33615-79-C-0003) will add additional constraints and/or modify coefficients. In particular, the data transfer objective coefficients for given interfaces between a memory and a processor may be readjusted to penalize use of certain processors for a given task and/or memories for certain data block allocations.

A stronger set of timing constraints may be required for dependent software task threads. A task thread, F_k, may be defined as a group of serially dependent tasks with the following notation:

$$F_k = \{ f_{k1}, \ldots, f_{kG_k} \}$$

where f_{kg} indexes one of the T tasks. In general, task f_{kg} must have executed $C_{f_{kg}}$ percent before task F_{kg+1} can be enabled. Thus, the tasks defined as a thread are not permitted to run simultaneously in parallel processors. This constraint may be written for each thread k as follows:

$$\sum_{t \in F_k} \sum_{p=1}^{P} \left(C_{tpk} (c_{tp} + r_{tp}) e_{tp} + R_{tp} x_{tp} \right) \leq \text{minimum } \{\tau , \Upsilon_k\}$$

for k=1 to K, and Υ_k represents feedback timing for thread K. A further assumption is that if task t is an element of a software thread, F_k, then task t may not be an independent task or an element of another task thread. If a task is required in more than one way, it can be defined as a group of different tasks for partitioning purposes.

In general, these threads represent critical system task path flow bottlenecks as determined by the performance simulation of a given partition allocation. The algorithm introduces new or revised constraints until one of the following conditions exists:

(a) Satisfactory solution found
(b) Infeasible condition identified
(c) Maximum feedback iterations performed.

The current solution state is to be saved and/or printed for future evaluation as requested by the user evaluator.

3.2 ALGORITHM DESIGN HIGHLIGHTS

There are many mathematical program techniques, including both linear and nonlinear optimizers and heuristics. The partitioning model requires integer solution values that immediately classify it as a nonlinear global optimization problem even though the model itself consists of linearly expressed objectives and constraints. In addition, two of the three achievement priority functions (i.e., balance the processor load and balance memory storage) are nonlinear in their formulation of minimizing the sums of absolute differences. These nonlinear goals combined with the goal program matrix, which is sized according to the parameters represented in Table 1, would be a challenge to both sizing and timing of commercially available mixed integer linear program models with a single achievement priority.

To determine the viable design alternatives, a study of goal programming was made, including several military goal program applications that have been implemented. Applications included weapon system slice optimization in relation to planning force analysis and a balanced budget allocation model for mixed project/agency funding. Both of these applications interface goal programming models with other analysis tools (such as simulation, input/output analysis, and regression analysis) to provide a set of automated operational evaluation tools. These additional tools provide a means to cross-check and supply detailed model data values that are used to calibrate the goal program model. The calibrated model is then used for selected parametric studies to determine impact on solutions in terms of parametric margins and solution sensitivities. Both of these applications utilize modified versions of the classical textbook[1,2] multiphase goal program computer algorithms. A major drawback to these codes is their susceptibility to numeric roundoff error propagation for problems involving more than 50 to several hundred variables and constraints. In addition to the numerical roundoff errors, the multiphase codes studied do not use dynamic core memory management. This requires the entire matrix and associated bookkeeping variables reside in main memory.

In lieu of funding the development for a mixed integer goal program optimizer for larger problems, an alternative algorithm is the sequential use of a good commercially available linear program optimizer interfaced via a goal program driver that introduces each achievement one at a time. This permits continuous solution problems with up to 16,000 rows to be handled, given adequate dynamic disc storage. Current state-of-the-art integer solutions are restricted to several hundred

---

1 Ignizio, James P., Goal Programming and Extensions, D. C. Heath and Company, Lexington, Massachusetts, 1976

2 Lee, Sang M., Goal Programming for Decision Analysis, Auerbach Publishers, Philadelphia, Pennsylvania, 1972

TABLE 1. BASIC GOAL PROGRAM MATRIX SIZING

CASE	CONTROL PARAMETERS						RESULTANT MATRIX		
	T	P	B	M	I	O	VARIABLES	ROWS	COLUMNS
1	30	2	61	3	3	2	1,330	1,747	4,824
2	30	3	61	4	3	2	2,383	3,009	8,401
3	30	3	120	4	3	3	3,038	3,608	10,224
4	30	3	120	6	3	3	4,358	4,688	13,734
5	60	4	140	6	3	3	10,351	12,271	34,893
Prob 1	5	2	12	3	3	3	264	348	960
Prob 2	7	4	21	6	3	3	1,250	1,642	4,534

Variables = B + P + M + 1 + MB + 3TP + TPM (I + O)

Rows = B + P + M + 1 + 2T + 2TP (1 + I + O) + TPM (I + O)

Columns = Variables + 2 (ROWS)

variables. The sequential use of a linear program optimizer is the approach recommended for further study in addressing a subset of the software partitioning algorithm as designed in this study. The design has remained independent of a specific computer optimizer code.

Even with the sequential mixed integer linear program technique, the sizing of the partitioning problem (given in Table 1) is prone to challenge the best optimizers without some careful matrix selection generation techniques. There are two major areas of concern:

1. The time consumed in determination of an initial feasible solution

2. Excessive iteration thrashing to determine "optimal" integer solutions.

The study of goal programming included a survey of heuristic techniques that can facilitate the search for improved solutions given an initial feasible solution. In practice, application-customed heuristic algorithms have provided an efficient means for handling and reducing the large solution space of alternatives to be searched.[1]

In the case of flight trainer candidate designs, the designers have an implied partition which can be used as the initial solution. The partitioning problem then becomes one of "Does a better solution exist with respect to load balance, memory balance, and development cost?" The incorporation of an initial solution step has been recommended as an implementation step requiring further study for obtaining an expanded evaluation capability. The current algorithm design assumes that an initial solution is supplied and proceeds in a heuristic manner to seek a better solution.

To achieve a well-defined user evaluator interface of partitioning input data, a customed heuristic goal program driver, and solution summary capabilities, the Partitioning Algorithm for Software Systems (PASS) has been designed emphasizing the four major processes denoted in Figure 2:

1. User input interface and processing referenced as PASS1

2. Basic partitioning algorithm referenced as PASS2

3. Augmented partitioning algorithm (PASS3) to handle dynamic performance prediction feedback

[1] Ignizio, James P., "Solving Large Scale Problems: A Venture into a New Dimension," Pennsylvania State University, 1978

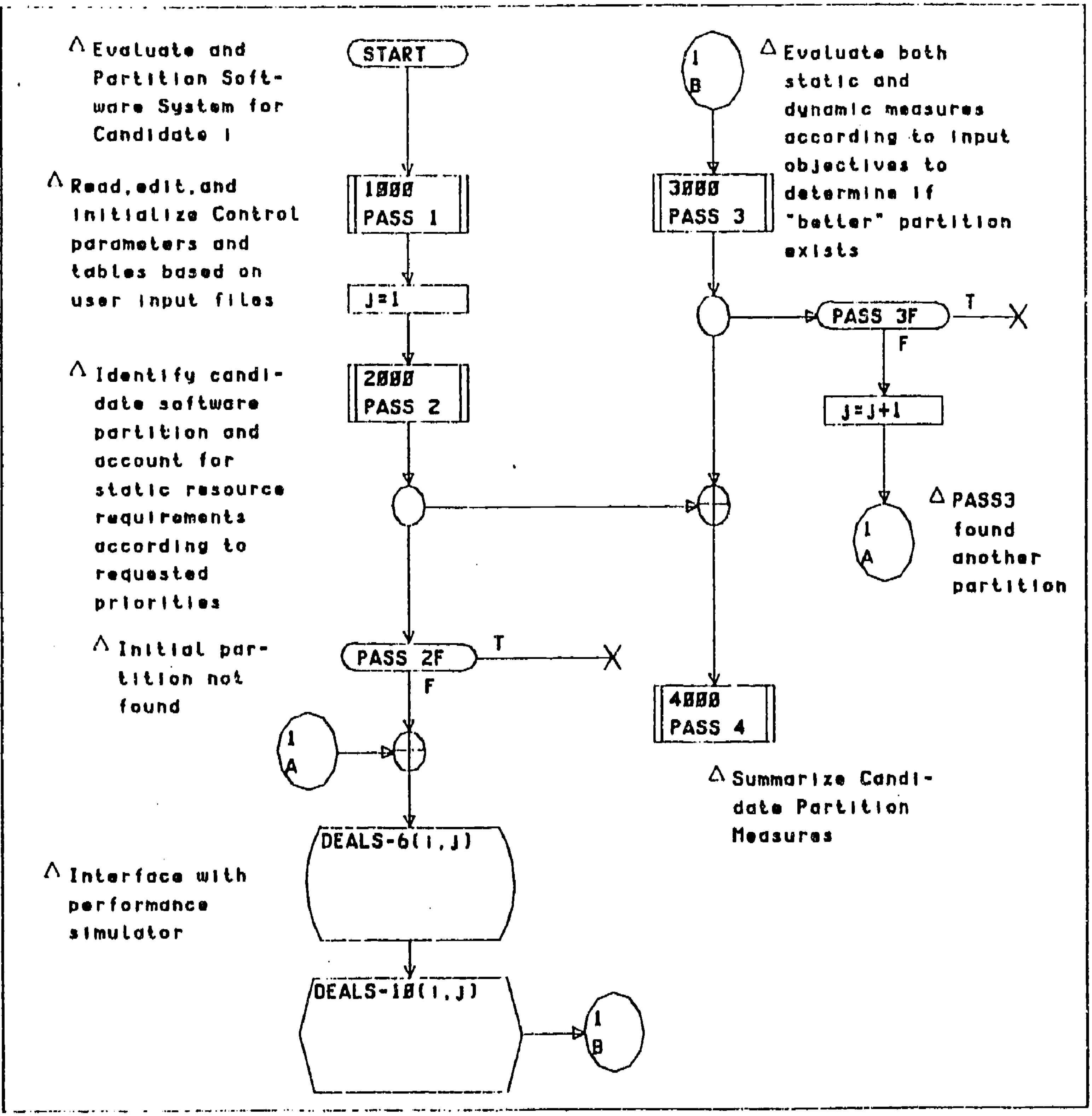


Figure 2. Major partitioning algorithm steps.

4. Solution summary reports (PASS4) of a given partition for candidate design i.

Prior to describing each of these steps, the overall design flow of the steps and their interfaces is presented.

The major external interface (exclusive of an optimizer) with PASS include the evaluation user and a multiprocessor configuration performance predictor simulator. The user interface considerations for actual implementation are expanded in Section 4, with emphasis on incorporating a modular, automated data repository to facilitate input preparation of PASS1 and maintenance of current flight trainer design parameters with respect to given partitions (PASS4). The performance predictor interface is designed to interact with the Computational Performance Predictor Simulator (CPPS) being specified and designed under separate contract. The iterative process of determining a new allocation (PASS3) based on performance prediction feedback is performed until one of the following conditions is reached: (a) satisfactory partition is found, (b) design bottleneck is identified, (c) maximum iterations have been reached.

3.2.1 Input Processing Step PASS1

The mathematical statement of Section 3.1 contains software, hardware, and combined software/hardware parameters. The design efforts of this study have emphasized the separation of any combined parameters into basic hardware and software components with the aid of technology data base tables and computational formulas necessary to generate the given "combined" parameter. Thus, all task/processor and data/memory parameters are derived from independent software and hardware design configuration inputs (see Section 4.2).

The specific inputs are defined in Appendix A. Figure 3 delineates the major design process flow for user input editing and computational sequences to properly set up for the actual partitioning steps that follow. The design demonstration (Appendix C) provides the detailed computations to map the user input into the internal partitioning algorithm control and lookup tables listed in Table 2. Appendix B provides representative report formats for the user input echo, which consists of the reports listed in Table 3.

3.2.2 Basic Partitioning Algorithm (PASS2)

This step provides the basic controls and logic for interfacing with the three user-ordered heuristics to determine whether an improved partitioning solution can be found. As mentioned in the introductory remarks on design in Section 3.2, the basic assumption is that an initial feasible (with respect to real-time constraints) partition is supplied. The resultant basic partitioning algorithm flow is denoted in Figure 4 as

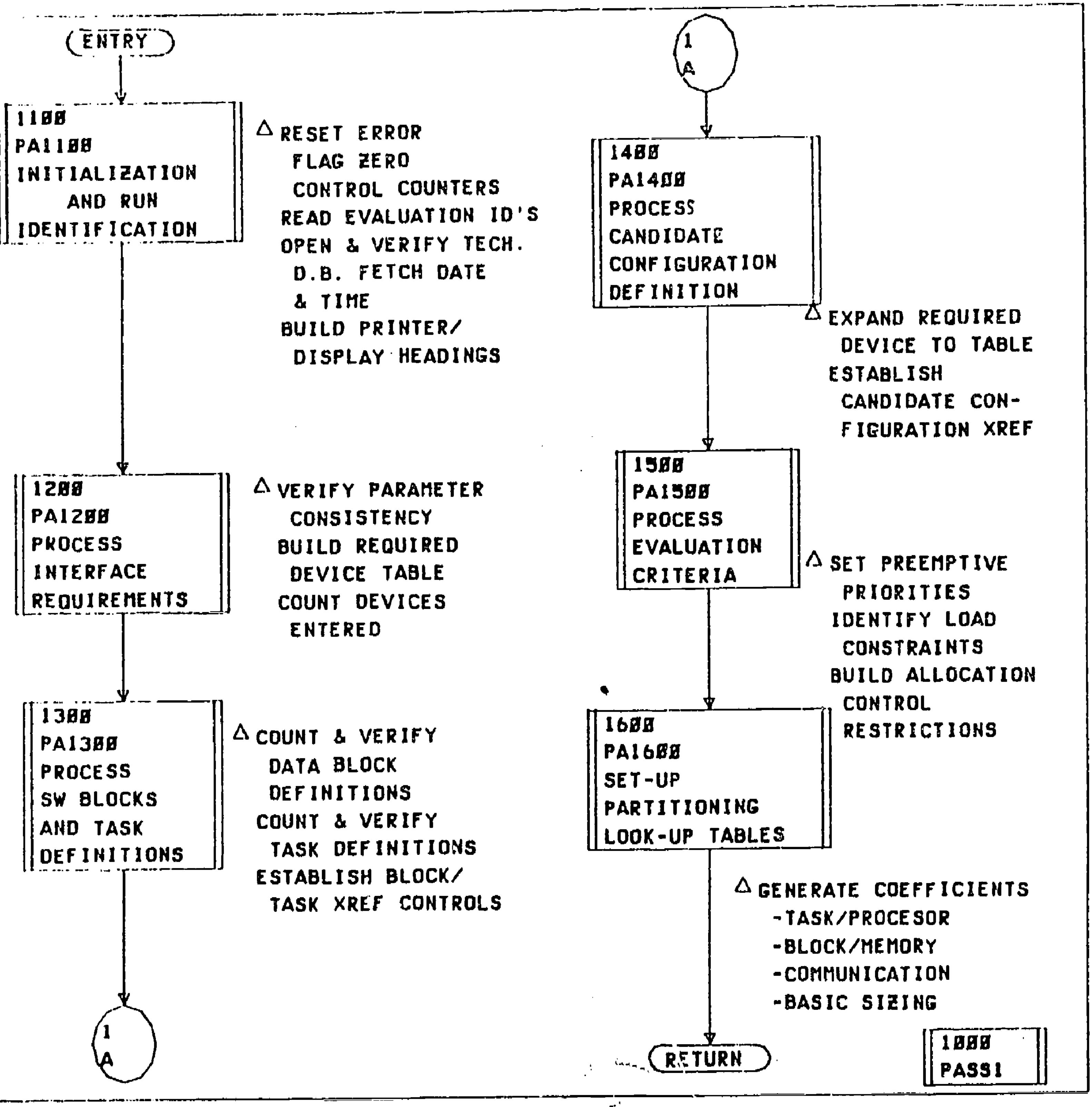


Figure 3. User input process flow.

TABLE 2. INTERNAL PARTITIONING ALGORITHM CONTROL AND LOOK-UP TABLES ESTABLISHED BY PASS 1

GRP	TABLE TITLE
1	Limits, Constants, and Codes
2	Current Problem Sizing Controls
3	Priority Controls
4	Current Processor List
5	Current Memory List
6	Current Communication Link List
7	Current Internal Device List
8	Task/Processor Allocation and Restrictions
9	Memory/Processor Allocation and Restrictions
10	Block/Memory Allocation, Restrictions, and Coefficients
11	Master Block List
12	Master Task List

TABLE 3. USER INPUT ECHO REPORTS THAT ARE SPECIFIED IN APPENDIX B

FORMAT*	REPORT TITLE
1	Standard Run Identification
2	Hardware Component Summary
3	Data Block Summary
4	Task Summary
5	Baseline Load Summary
6	Evaluation Options/Restrictions
7	Evaluation Priorities
8	Basic Partitioning Problem Size

* Format reference to Appendix B

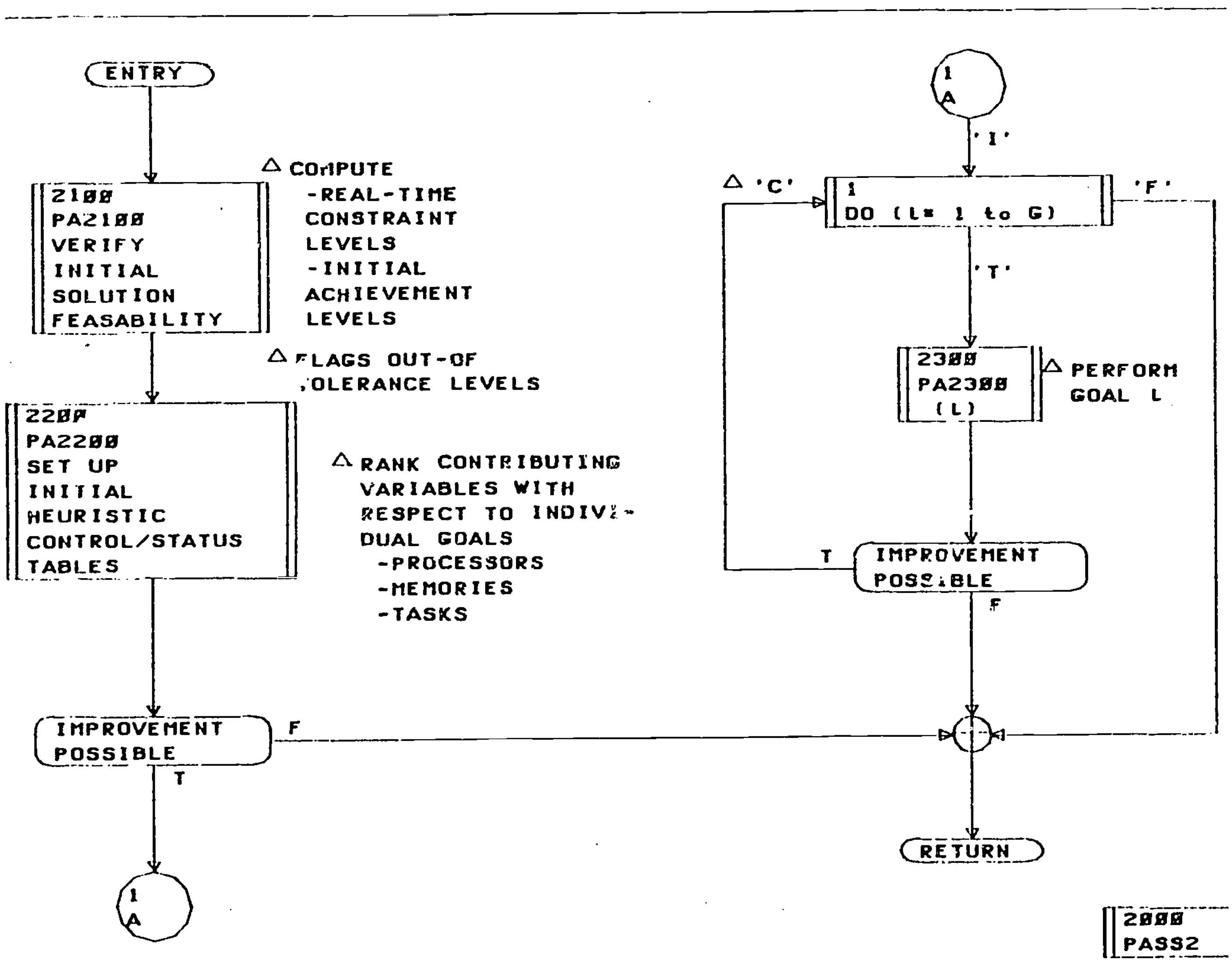


Figure 4. Basic partitioning algorithm control flow.

being comprised of initial solution verification, heuristic control table setup, and user-specified, priority-ordered heuristic executions.

There are three basic heuristic algorithms corresponding to the three objectives or achievement functions: processor utilization (LOADBL), memory utilization (MEMBAL), and development cost (RDCOST). Figure 5 denotes the major selection branch as being a function of the user-specified priority execution order GOAL (g), where g is the current priority level being executed. Prior to invoking the appropriate heuristic, a test is made to determine whether the basic priority goal level has already been achieved. If so, a return is made to proceed to the next priority level.

The major features incorporated in the design permit ranking of the current partition solution variables with respect to impact on the given priority under consideration. The following ranking definitions are utilized for each of the respective heuristics:

1. For the load balance heuristic, processor p's utilization, U_p, is subtracted from its goal, G_p, to define $U'_p = G_p - U_p$. The resultant U' array is then ranked from high to low values (i.e., those below their goal to those above their respective goal in order of difference magnitude). The resultant ranked array is then used to determine whether the load is currently in balance, i.e., ($U'_1 - U'_P$ GTOLPU) with respect to a user-supplied tolerance (GTOLPU) for processor utilization. The object is to offload some of the tasks from the heavily utilized processors to the lighter loaded processors to obtain a better balance, as denoted in Figure 6.

2. For the memory balance heuristic, the allocated memory, u_m, is subtracted from its goal allocation, g_m, to define $u'_m = g_m - u_m$. The resultant array, u'_m, is then ranked (in a similar fashion as processor utilization) to determine whether the current memory allocation is in balance according to the user-supplied goal ($u'_1 - u'_M$ GTOLMU). The objective (Figure 7) is to reallocate some of the blocks from the over-allocated memories to the under-allocated memories to obtain a better balance.

3. The development cost is a minimization problem of individual task development cost. Thus, the tasks are ranked from most expensive to least expensive. The ranked cost array can then be systematically processed (Figure 8) to determine whether a more cost-effective solution is possible (i.e., can this task be implemented on another processor in the candidate configuration of less development expense and still meet real-time constraints?). It should be noted that this priority is only applicable to a heterogeneous set of candidate processors.

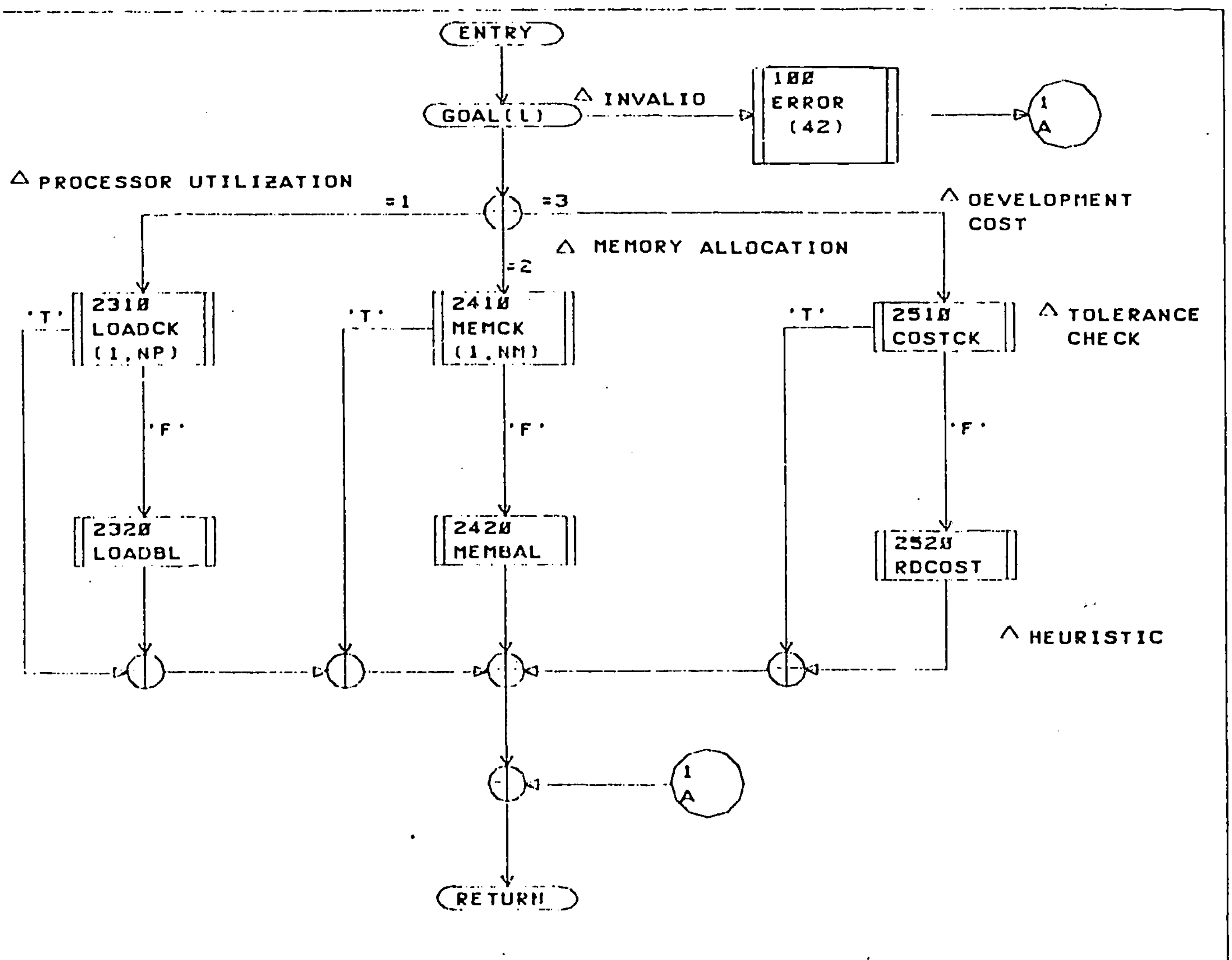
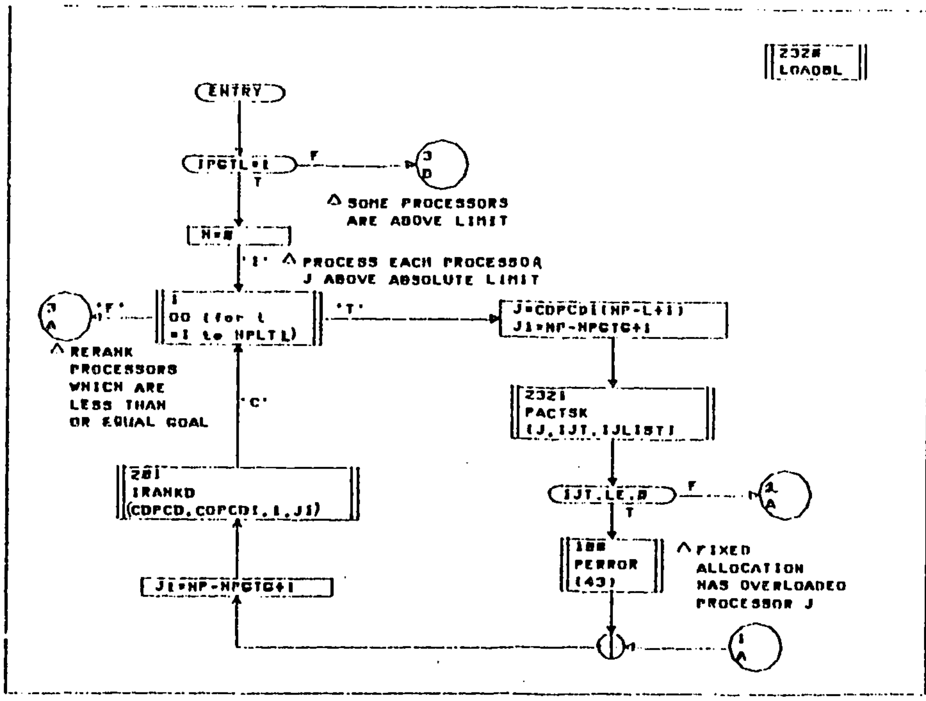


Figure 5. Priority heuristic selection process.



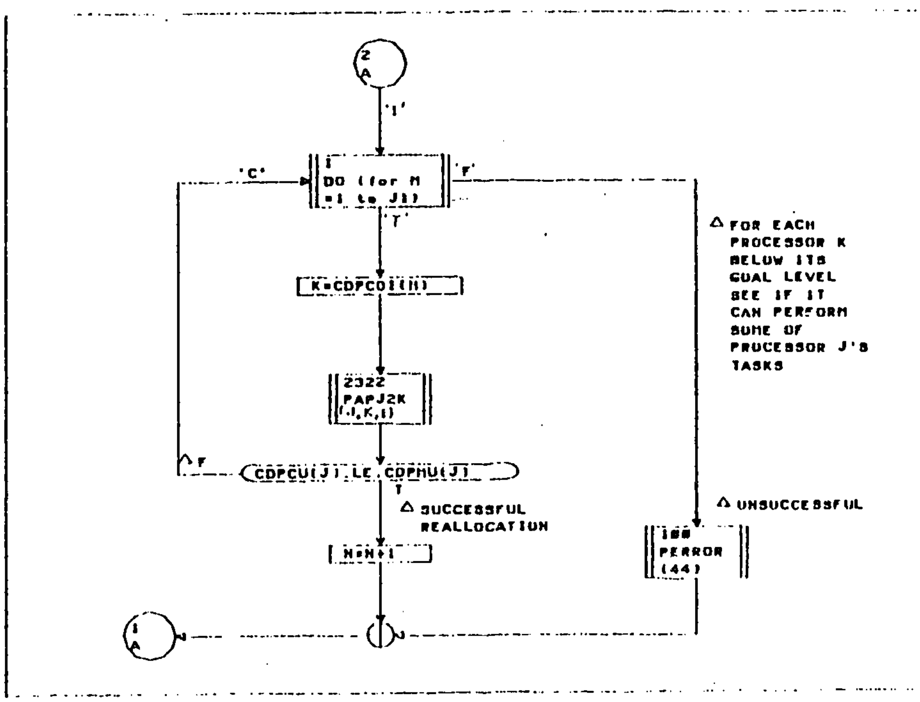


Figure 6. Processor load balance heuristic.

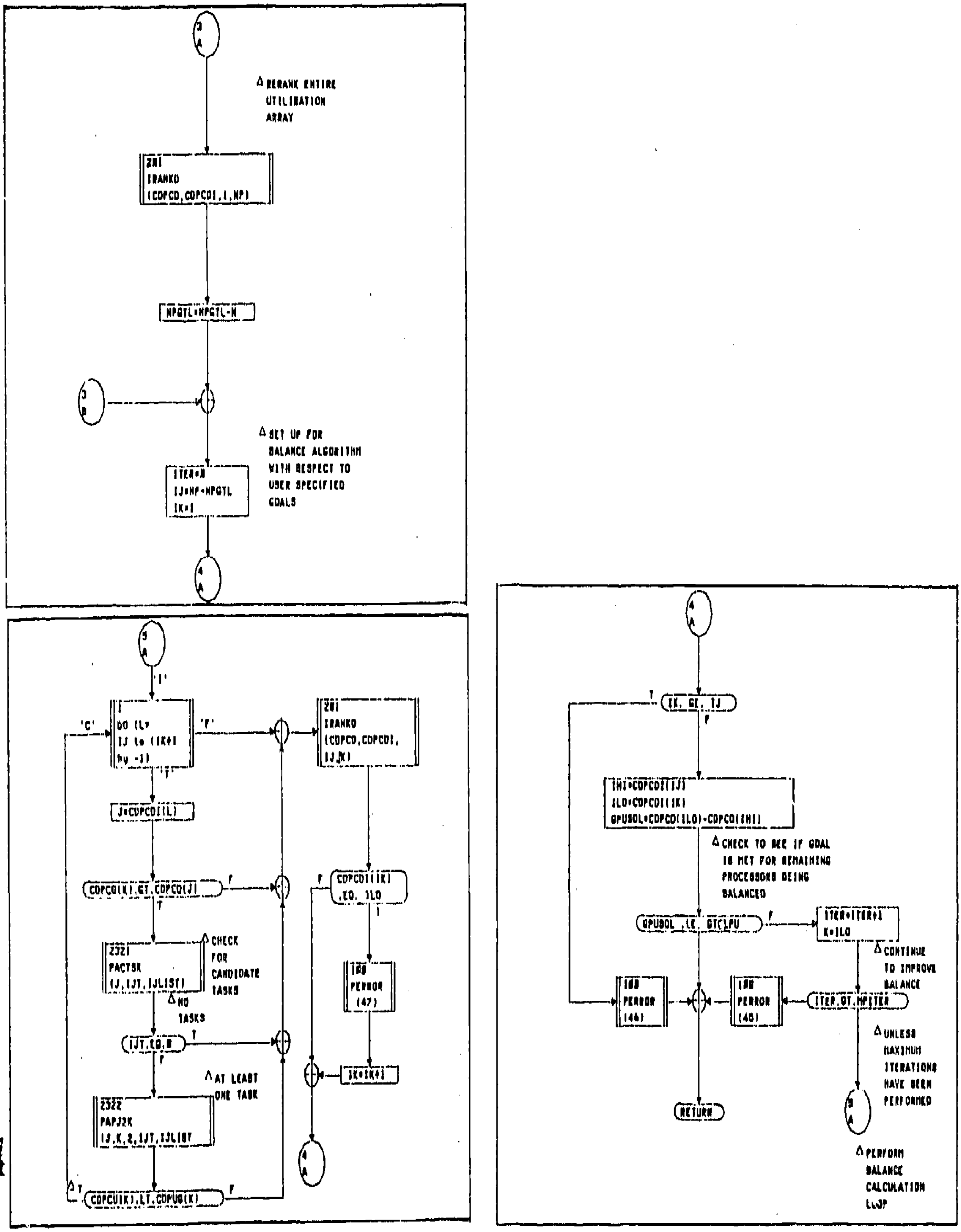
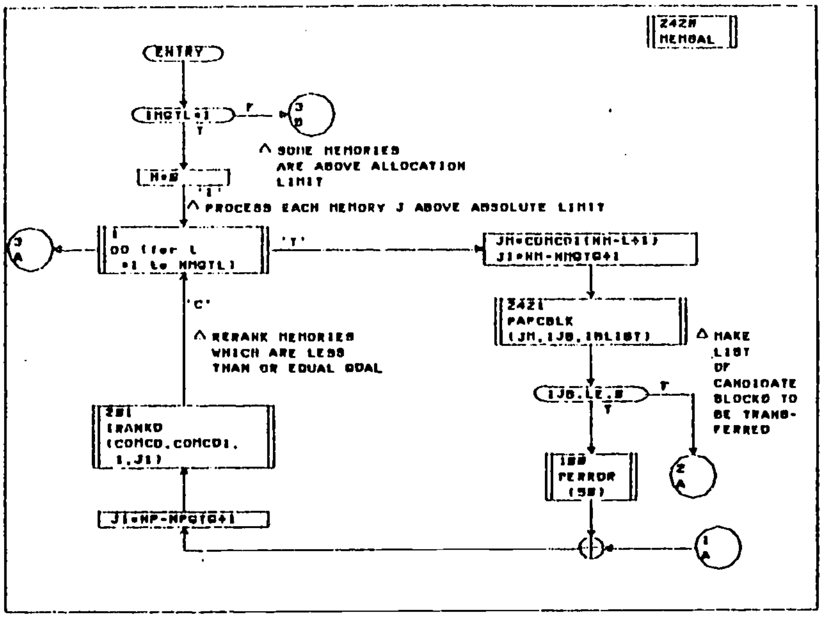


Figure 6. Processor load balance heuristic (Concluded).



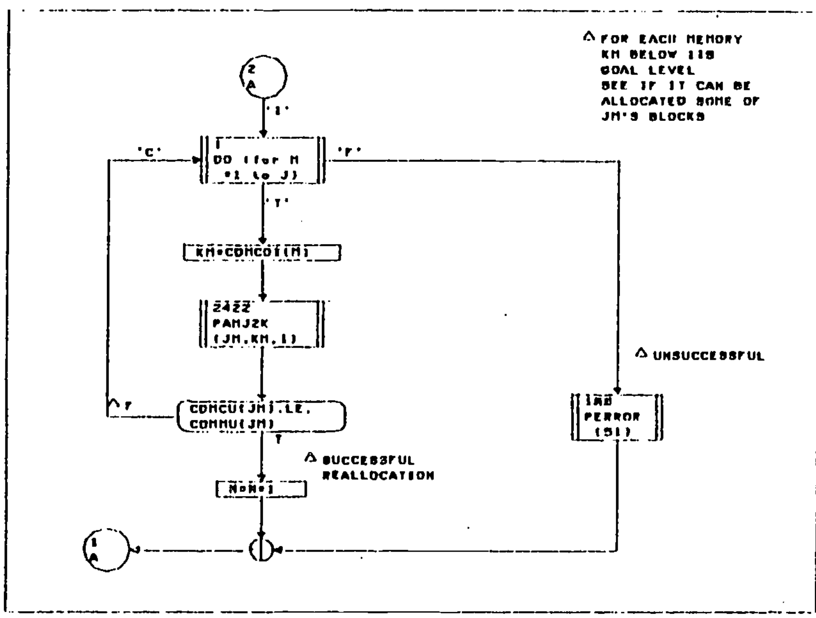
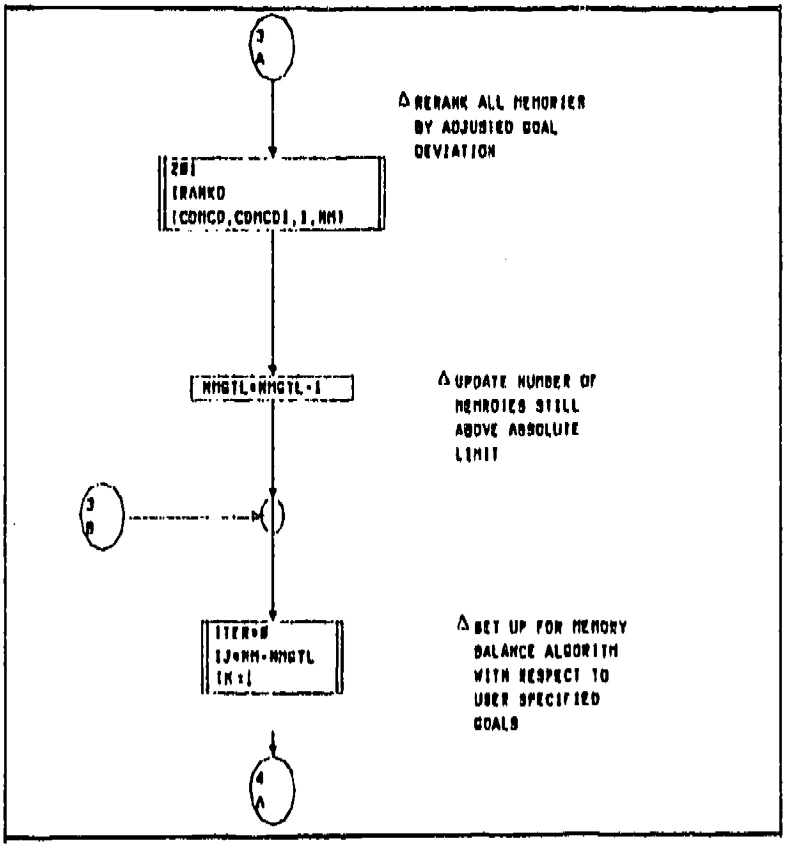
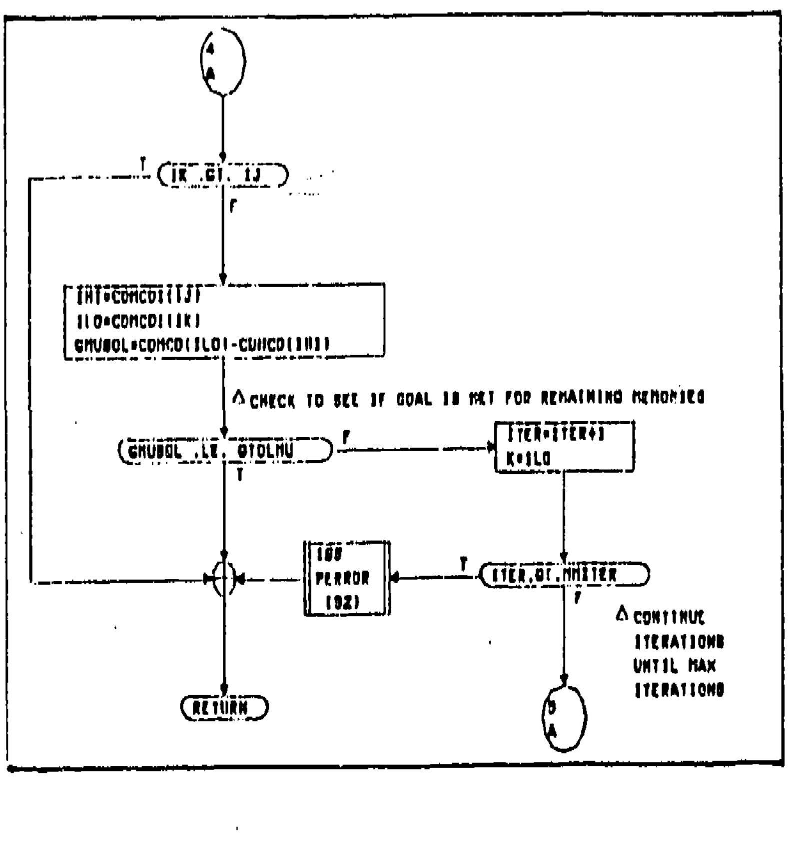


Figure 7. Memory allocation balance heuristic.





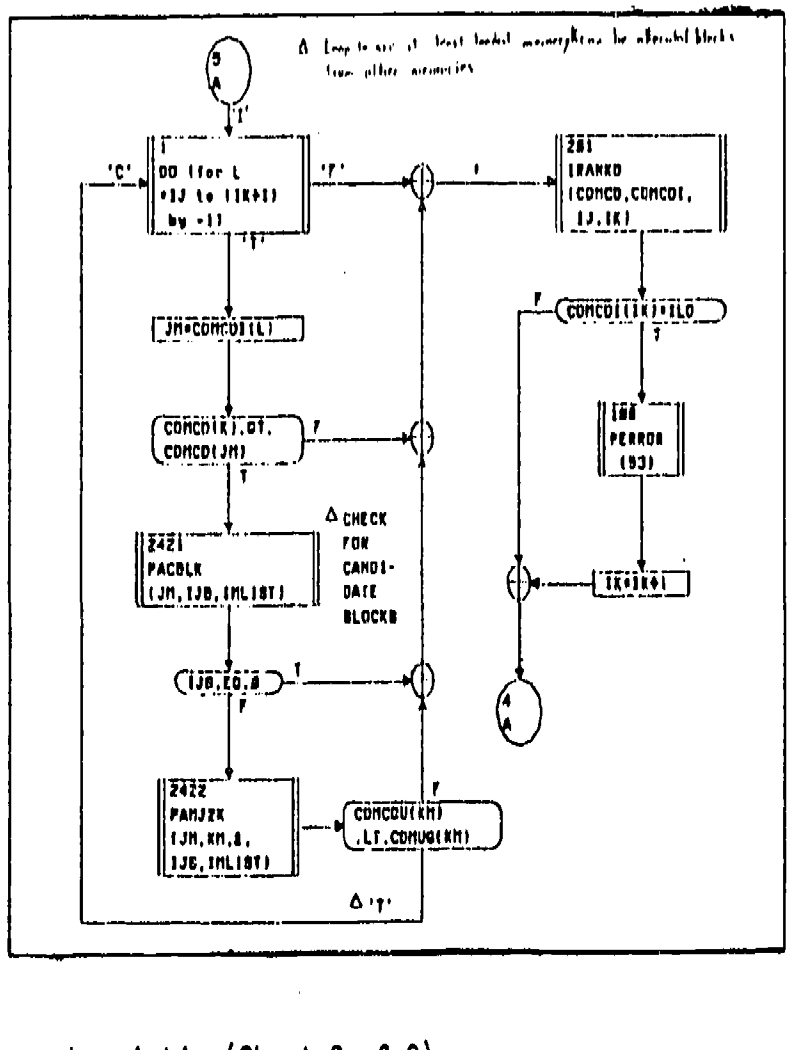


Figure 7. Memory allocation balance heuristic (Sheet 2 of 2)

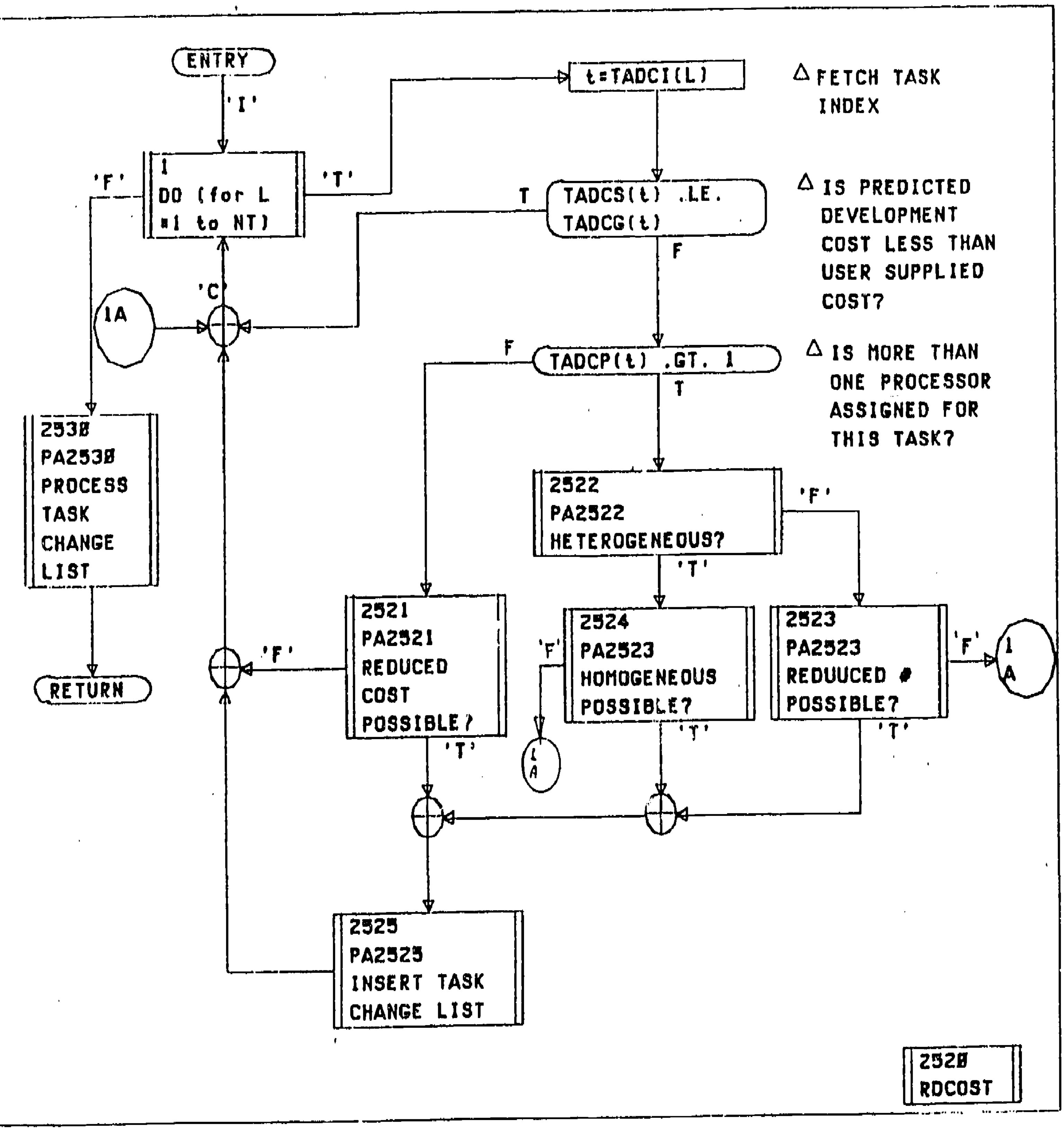


Figure 8. Reduce development cost heuristic.

For each of the heuristics, checks are incorporated to ensure that real-time limitations are not violated by any subsequent new "improved" solutions found by the respective heuristic. Design emphasis was placed on the order for incorporating these checks within the heuristic procedure to avoid excessive calculations when easily determined restrictions would prohibit exploring a given tradeoff. For example, when attempting to reallocate a task to another processor, only those processors that may perform the task are considered. To solve some of the more complex interrelated real-time constraints, a linear program statement might be studied to determine whether effective utilization of an optimizer would be feasible for performing the given tradeoff. The current algorithm incorporates a specific check of constraints as formulated in Sections 3.1.5 and 3.1.6.

The heuristic driver continues at each priority level until it has exhausted its systematic exchange tradeoff search for an improved measurement. The three priority levels are executed in the order as specified by the user evaluation priority inputs of PASS1. The basic computational and logical sequence flows for each of the three priority levels are denoted in Figures 6, 7, and 8, respectively.

3.2.3 Augmented Partitioning Algorithm (PASS3)

This step is an expansion of the PASS2 processes with emphasis on resolving identified performance bottlenecks of the following types:

1. Cycle or thread timing is not sufficient for real-time system response.

2. Specific candidate component (i.e., processor, memory, communication link) utilization is unacceptable.

The basic process decision flow is depicted in Figure 9.

Recognizing that manual user evaluation insight may help expedite the search for an improved partitioning, process PA 3100 facilitates the option that the current allocation can be manually modified. Once any modifications have been processed, the performance data are processed via PA 3200 to readjust coefficients and to set up additional constraint generation controls. The new constraints are then constructed and their basic impact on the current partition is assessed in terms of solution feasibility. Each performance bottleneck is processed individually, in a predetermined order of criticality during this process (PA 3300).

If a cycle or thread is the bottleneck, then the respective resource management and data communication links are examined to determine the major bottleneck within the thread or cycle. Penalty coefficient

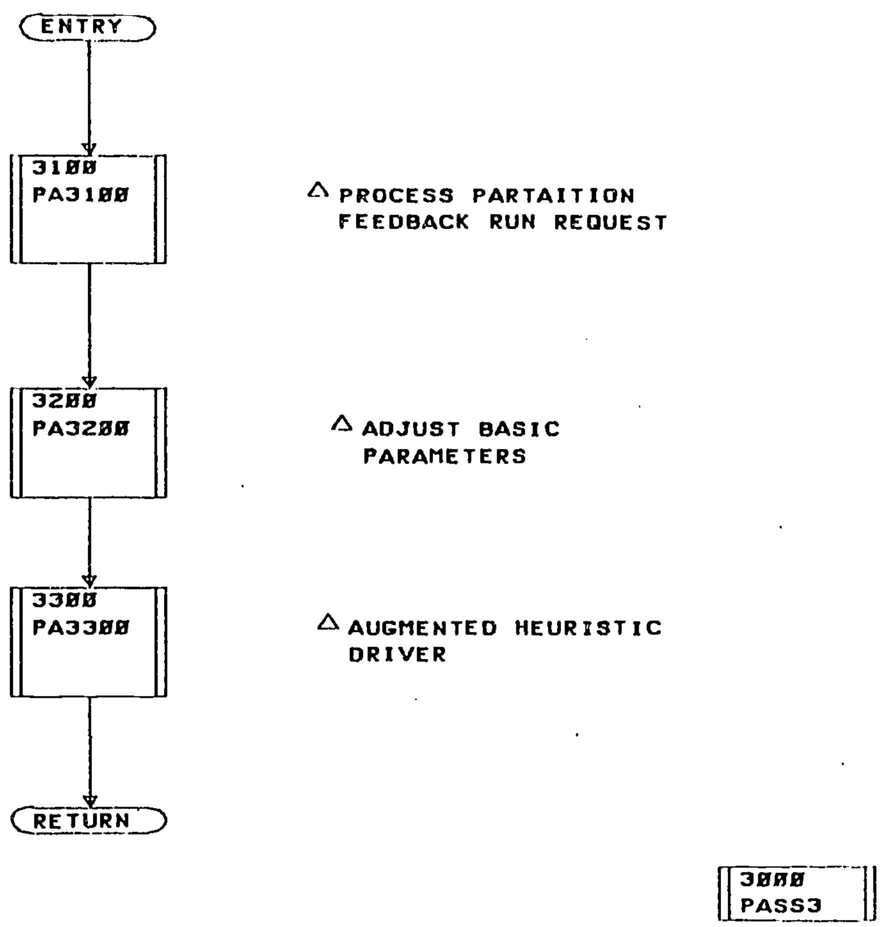


Figure 9. Augmented partitioning algorithm flow.

adjustments are made to the processor utilization equation. An alternate partition is sought that satisfies the end-to-end time requirement of the given cycle or thread under these more stringent constraints.

If a component is above its allotted utilization, a check as to processor or memory balance bottleneck is made. If it is a processor, the processor heuristic is used to offload the offending processor. If it is a memory problem, an attempt is made to find a faster access memory or add a duplicate block if shared memory access is the bottleneck.

As the processing of bottlenecks is performed, the augmented heuristic driver invokes PASS2 partitioning modules interspersed with additional checks for maintaining the appropriate thread and/or cycle constraints. If a new partition is found to be acceptable, it is saved for feedback to the performance simulation and further manual analysis. If not, the problems are identified for user evaluation. Appendix D contains the detailed design flows necessary to fully enumerate the algorithm. Additional changes are anticipated as the details of the performance simulator design are enumerated under Contract No. F33615-79-C-0003.

3.2.4 Solution Summary Reports (PASS4)

The report generation features of PASS4 are designed to provide printed summaries of a partition found by either PASS2 or PASS3 for a given candidate configuration. The specific formats chosen present the partition solution from five complementary, but different, aspects, including (a) partitioning priority level measurements, (b) task allocations, (c) data block allocations, (d) processor allocations, (e) memory allocations.

Figure 10 reflects a modular design flow based on user requests for any of the reports for a given partition j of candidate configuration i. This particular report generation capability should be implemented for access from batch job control, special user codes, as well as interactive displays to obtain maximum evaluation flexibility to automatically recall and/or print alternative partition solutions for a given candidate.

Specific output report formats are presented in Appendix B. The design demonstration, Appendix C, has sample output reports for user reference.

3.3 FEASIBILITY DEMONSTRATION

In deriving a meaningful, yet simple, sample problem, specific preliminary design material was obtained from Williams AFB with regard to an ongoing expanded design for the Advanced Simulation for Pilot Training multiple processor visual computational support subsystem. The

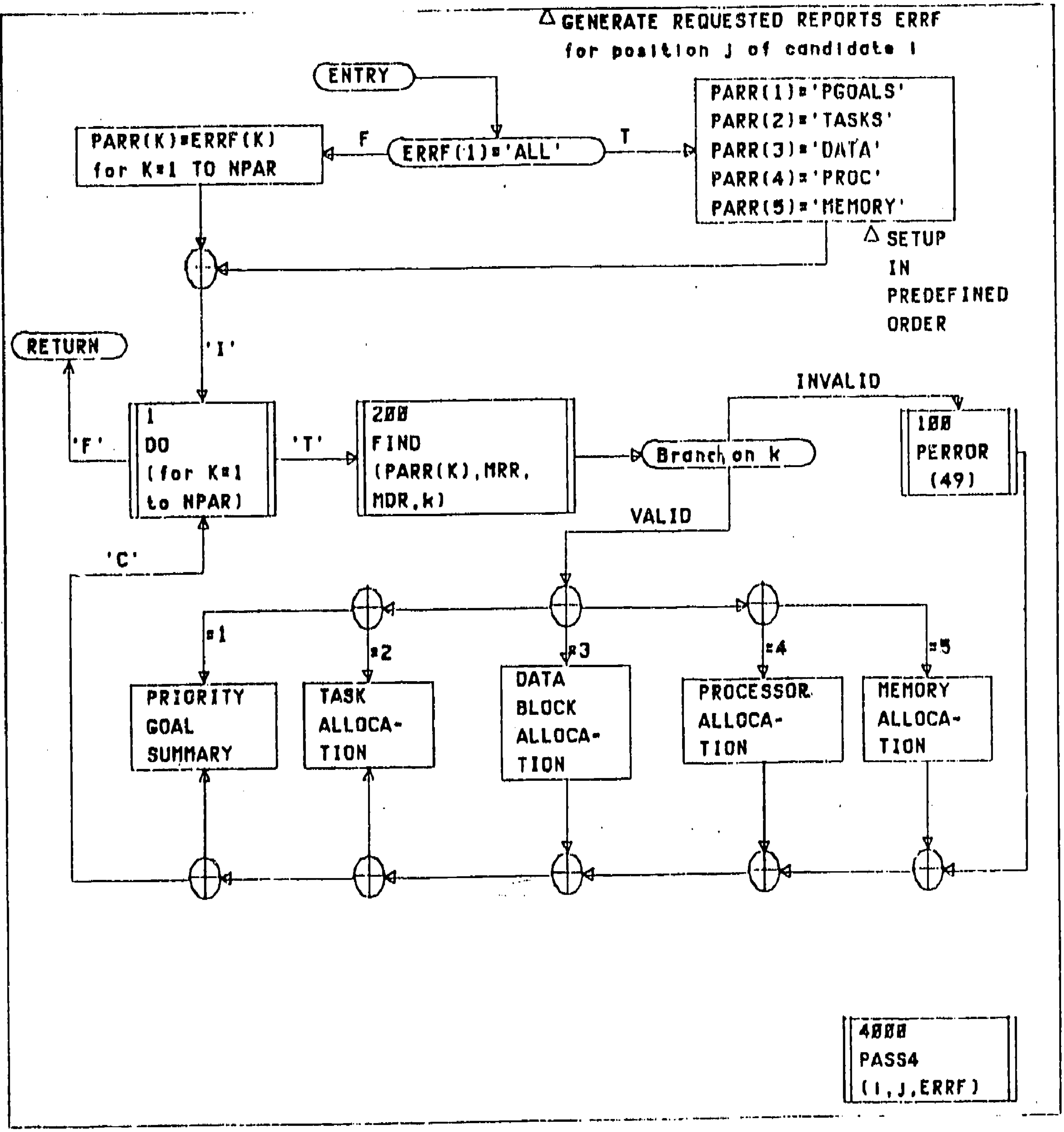


Figure 10. Report generator design.

preliminary design material provided a realistic source of the format for ongoing trainer computational design input. It also included a mix of general-purpose and special-purpose processors. The information in this memorandum provided a good base for generating a sample problem; however, the resulting sample problem required simplification of the configuration described to permit a flexible, yet easy-to-follow, manual demonstration problem to be obtained.

The design factors in the original problem were very restrictive as to Central Processing Unit (CPU) task assignments and thus left very little room for alternative partitioning. This reinforces the fact that, in software design, tasks tend to be defined in terms of the selected hardware configuration features to meet computational needs, as opposed to specifying application computations and then matching tasks to the hardware selection. For the partitioning algorithm to be applicable to alternative allocations and partitions, the major feasibility issue concerns design language and means for inputting the problem definition from which the partitioning model is to operate. These issues are discussed in Section 4.

For demonstration purposes, overview inputs, restrictive inputs, and detailed inputs have been incorporated to illustrate various aspects and paths of the partitioning process and to point out the tradeoffs in utilization of detailed inputs versus general estimates. The complete algorithm feasibility demonstration is included as Appendix C to this report. The basic order is the sample problem definition, user input sheets, user input echo summary, basic partitioning priority calculations, sample performance feedback contingencies, and solution summary outputs.

Figures 11 through 13 illustrate the major partitioning components as extracted and simplified from a set of Williams AFB ASPT preliminary design notes for the visual subsystem. The overall processor configuration is denoted in Figure 11. The memory and external communications are illustrated in Figure 12 to include both private and shared memory devices. It also includes processor-to-processor direct data transfer. Figure 13 denotes the simplified task flow used for demonstrating the input and output steps of the algorithm. The tasks of Figure 13 may be further divided into more detailed tasks for demonstrating and testing specific features of the partitioning algorithm, once an automated version of the algorithm is implemented.

The sample demonstration (delineated in Appendix C) permits the definition of potential automated implementation processes for handling real-world partitioning problems. The examples demonstrate the feasibility of an automated tool. Section 4 provides recommended implementation steps for verifying and validating the partitioning tool. These steps will require that the basic algorithm be automated to properly evaluate and demonstrate its performance characteristics for more realistic partitioning problems that tend to be of larger size than the

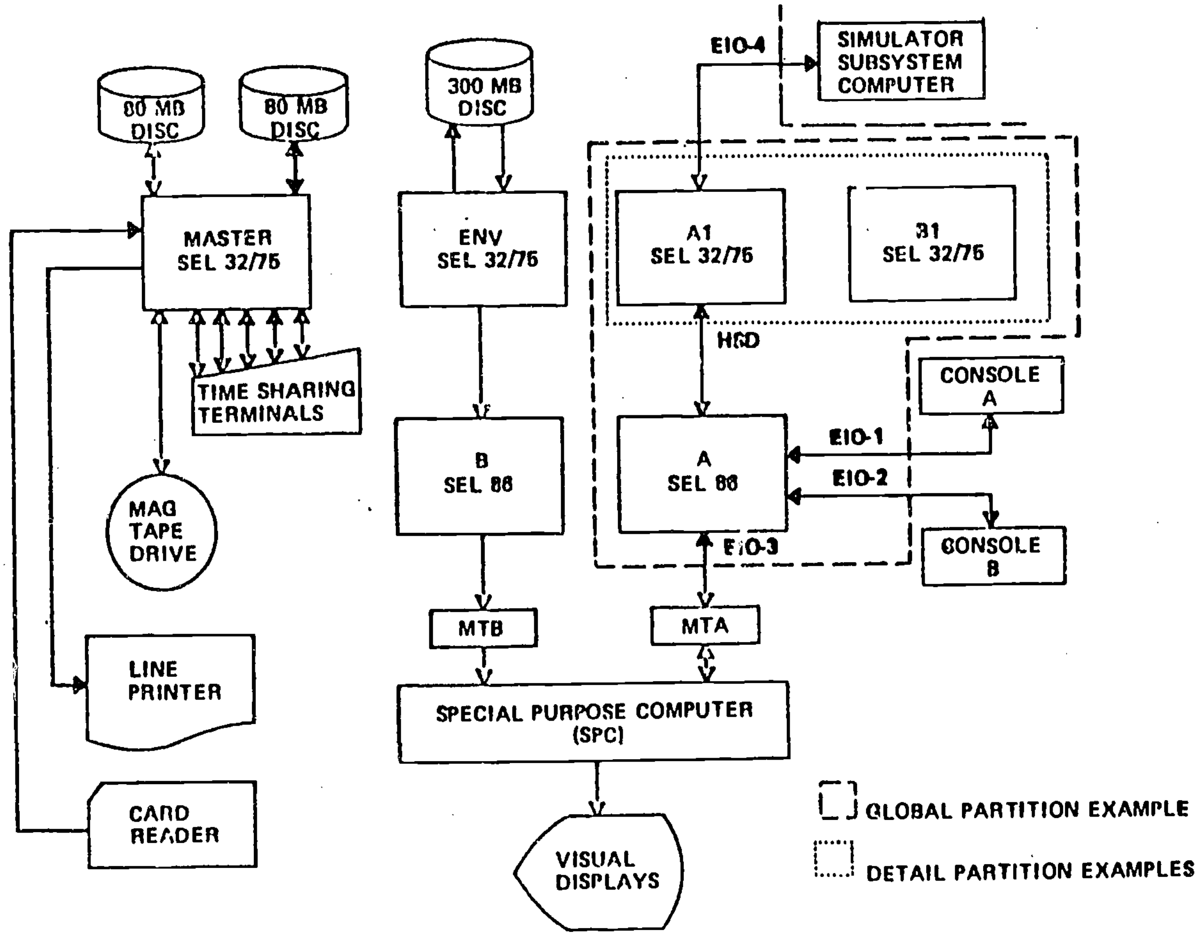


Figure 11. Sample problem configuration.

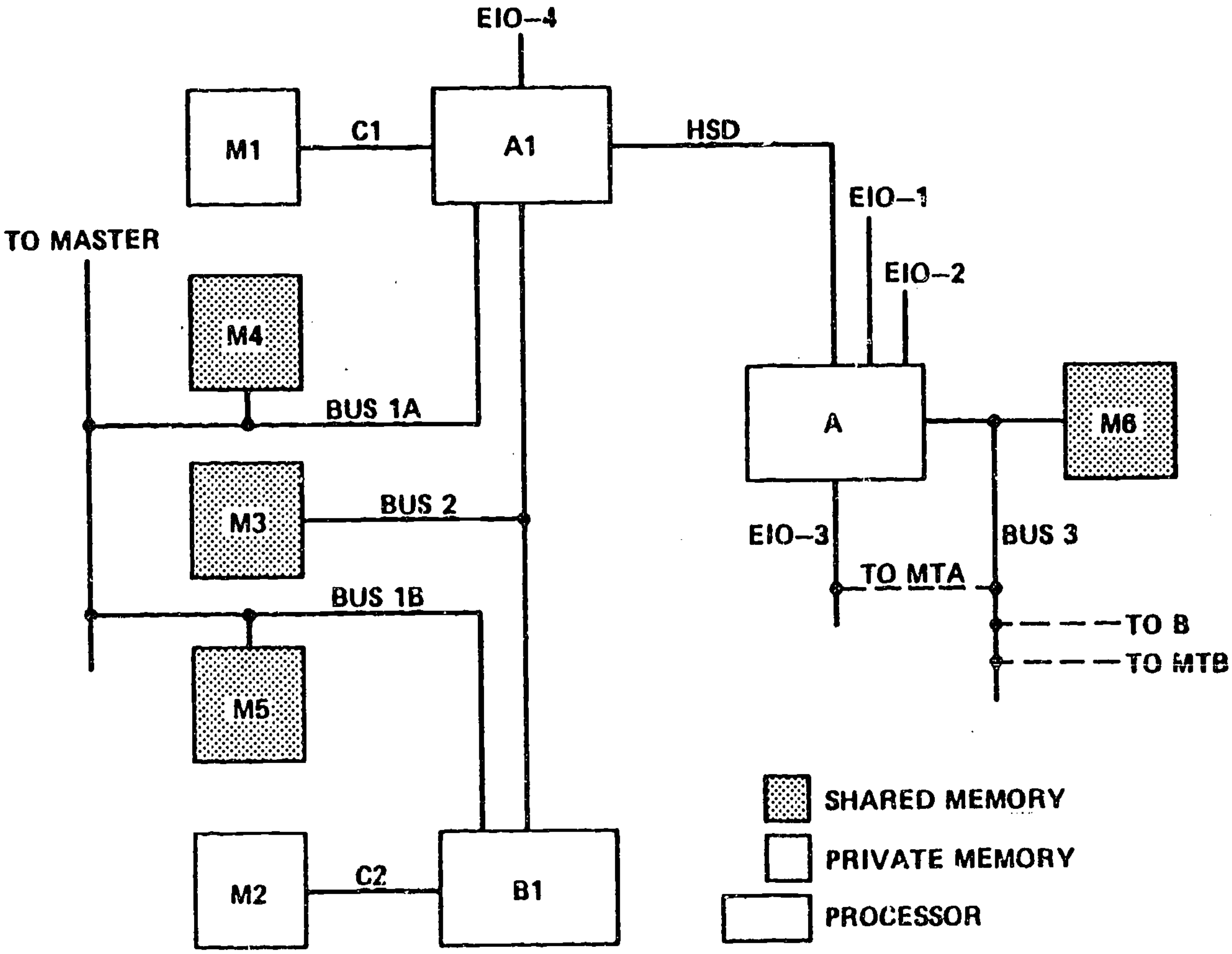


Figure 12. Sample configuration memory processor communications.

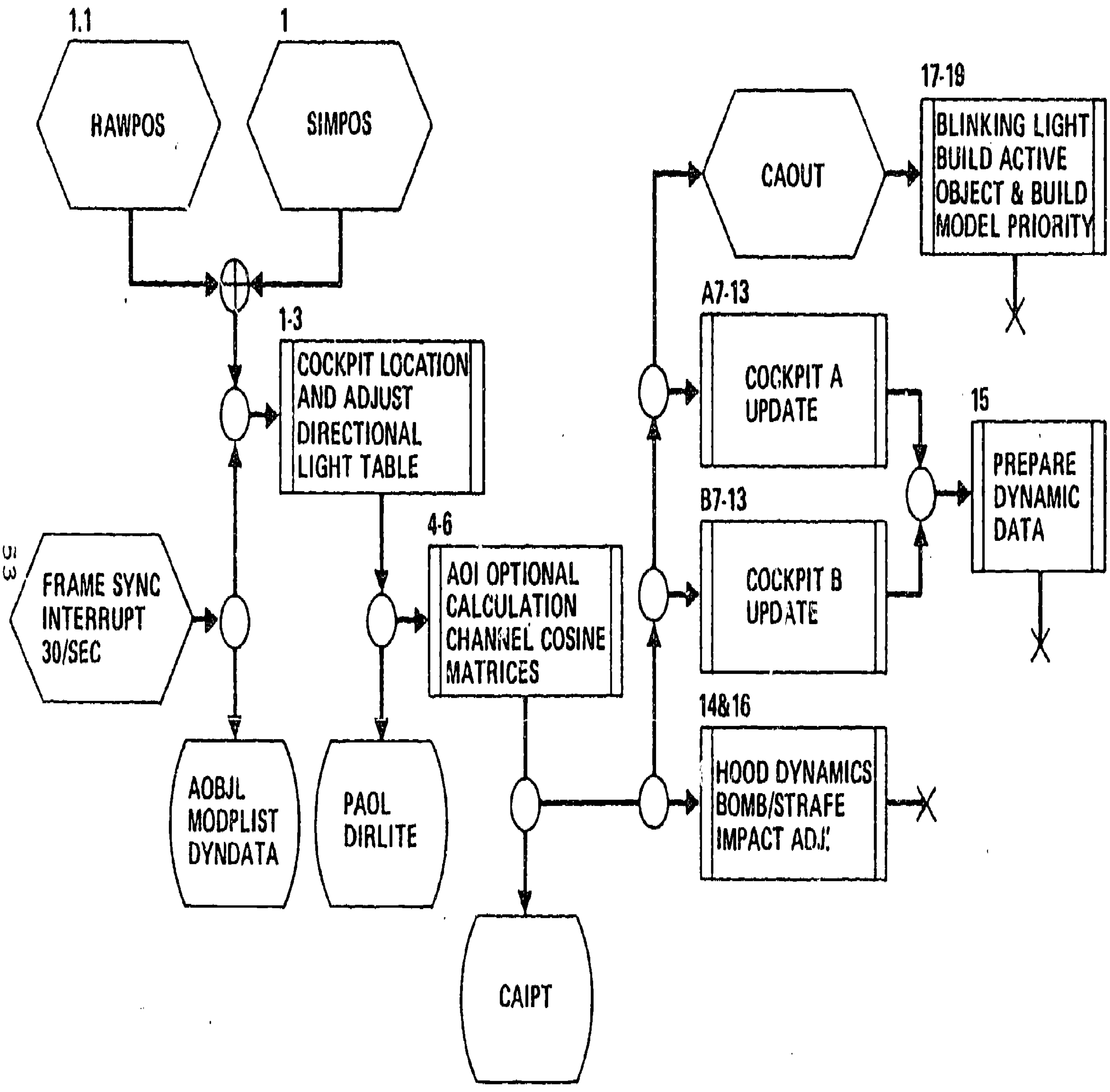


Figure 13. Application flow.

manual demonstration examples. The manual examples will permit the basic logic to be verified for a controlled, small-scale application prior to "cranking out" large-scale partitioning problems. This will permit an initial level of confidence to be established in the automated version.

4. MODEL IMPLEMENTATION CONSIDERATIONS

To successfully implement the software partitioning algorithm, an up-to-date technology data base for the flight training simulator computational devices is essential. This section delineates the data collection process and decision steps recommended for potential automation and quality control of the algorithm defined in Section 3. This section has been organized to go from an overview of the candidate design evaluation environment into a detailed evaluation support data base repository description, followed by computer selection criteria and the recommended implementation schedule for automation of the software partitioning evaluation algorithm.

4.1 FLIGHT TRAINING SIMULATOR EVALUATION ENVIRONMENT

Typically, the development of flight training simulator candidate designs for the Air Force are contracted out by the Simulation System Program Office (ASD-SD24). The computational subsystem design development is monitored and evaluated by the Deputy of Engineering Simulation (ASD-EN). In some cases, the flight trainer development is directly contracted by a specific system office (such as in the case of the F-16 trainer). Currently, the contracted organization has the primary responsibility for establishing both hardware and software requirements of the computational system, subject to certain Air Force guidelines and training capability objectives. The candidate design evolves through an iterative refinement of documentation and algorithm enumeration analysis, which typically progresses from system specification functional flows followed by the detailed enumeration of the candidate design. Each of these levels has narrative descriptions interspersed with a variety of technical charts, drawings, tables, flow diagrams, interface definitions, etc.; however, as denoted in Figure 14, the volume of documentation for a training simulator quickly becomes unwieldy unless documentation traceability and content standards are adhered to and enforced via constructive reviews, which are geared to detecting and correcting errors early in the development phase.

This effort has specifically addressed the software partitioning aspects of candidate design evaluation. The three major outputs of the partitioning algorithm are measures of the processing load balance, memory utilization, and estimated development implementation cost based on given timing and sizing input requirements of the respective tasks and data load for a given candidate configuration. For effective use of the software partitioning algorithm, the underlying mathematical model of Section 3.1 must be understood in terms of the processor utilization, memory utilization, and development cost formulations, which are the primary outputs.

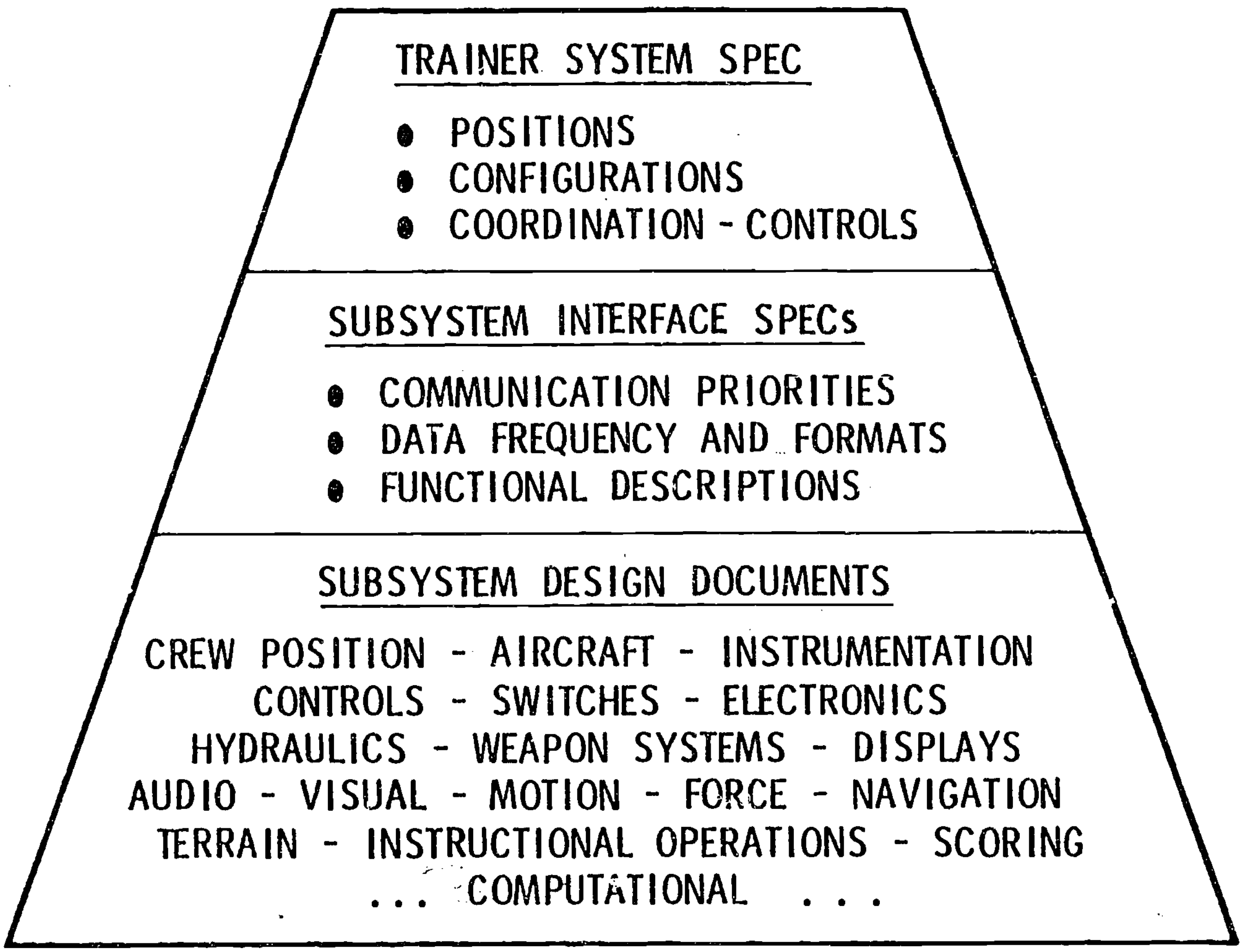


Figure 14. Hierarchy of flight trainer documents, which relates to candidate design evaluation, can quickly become unwieldy if content and traceability standards are not adhered to or enforced. The simulator computational subsystem interfaces with and coordinates a large number of the trainer simulator subsystems.

To obtain reliable outputs, a consistent, systematic procedure needs to be established with appropriate configuration management and quality assurance provisions and controls. The major implementation consideration for such a procedure is the establishment of a consistent data repository for pertinent flight trainer computational design data. No central repository for Air Force flight trainer computational designs currently exists, although various organizations (such as ASD-EN) do have their own evaluation data repositories.

During the course of this contract, it was learned that the Naval Training Equipment Center (NTEC) in Orlando, Florida, does have a repository of all documentation associated with Navy training devices to include the computational subsystem. NTEC recently modified the required Data Item Descriptions related to the computational subsystem to be an integral part of training device development in conjunction with a proposed Appendix A to the trainer specification, MIL-STD-1644, entitled "Trainer Software Design, Control, Production Testing and Acceptance Procedures and Requirements." This proposed specification incorporates the top-down structured design approach with minimum standards that are required of each milestone document and its associated review content, error detection/correction actions, and milestone completeness determination. The procedures are in basic agreement with the development cycle presented in Section 2.1. This set of documents permits a consistent repository to be established and maintained for current reference and analysis input for new development considerations. Unfortunately, it is still primarily a manual information storage and retrieval system when it comes to accessing data pertinent to software partitioning.

The factors identified in Section 3.1 that influence optimal software allocation (such as: data block, task, processor, and memory descriptions) remain the same regardless of the system assumptions or presentation format. Indeed, these factors (Table 4) must typically be extracted from more than one document to obtain the complete set of input and constraint parameters defined in the mathematical statement of Section 3.1. To assist in the review of documents with respect to software partitioning of the computational subsystem, the supporting data base parameters have been segmented into five major areas with respect to flight trainer simulator:

1. Trainer Computational Interface Requirements
2. Baseline Application Components
3. Candidate Hardware Configuration Components
4. Technology Data Base
5. Evaluation Criteria/Constraints and Partitioning Load.

Figure 15 reflects the interactive nature of these data base areas with respect to technology capabilities and the development cycle up through the completion of the design but prior to actual implementation and testing. The upper area relates to milestone documents of the training

TABLE 4. DEVELOPMENT DOCUMENTS AND THEIR RELATIONSHIP TO THE PARTITIONING ALGORITHM FOR SOFTWARE SYSTEMS

DOCUMENT(S)	INPUT AREA
Computational Subsystem Interface Specification	External Device Interfaces Required Components Functional I/O Map Communication Rules and Priorities Baseline Load(s)
Software Design and Data Base Specifications	Data Block Descriptions Task Descriptions Task Threads Baseline Load(s) Tasking
Hardware Configuration Design Specifications	Processors Memories Interfaces (Internal and External) Communication Rules

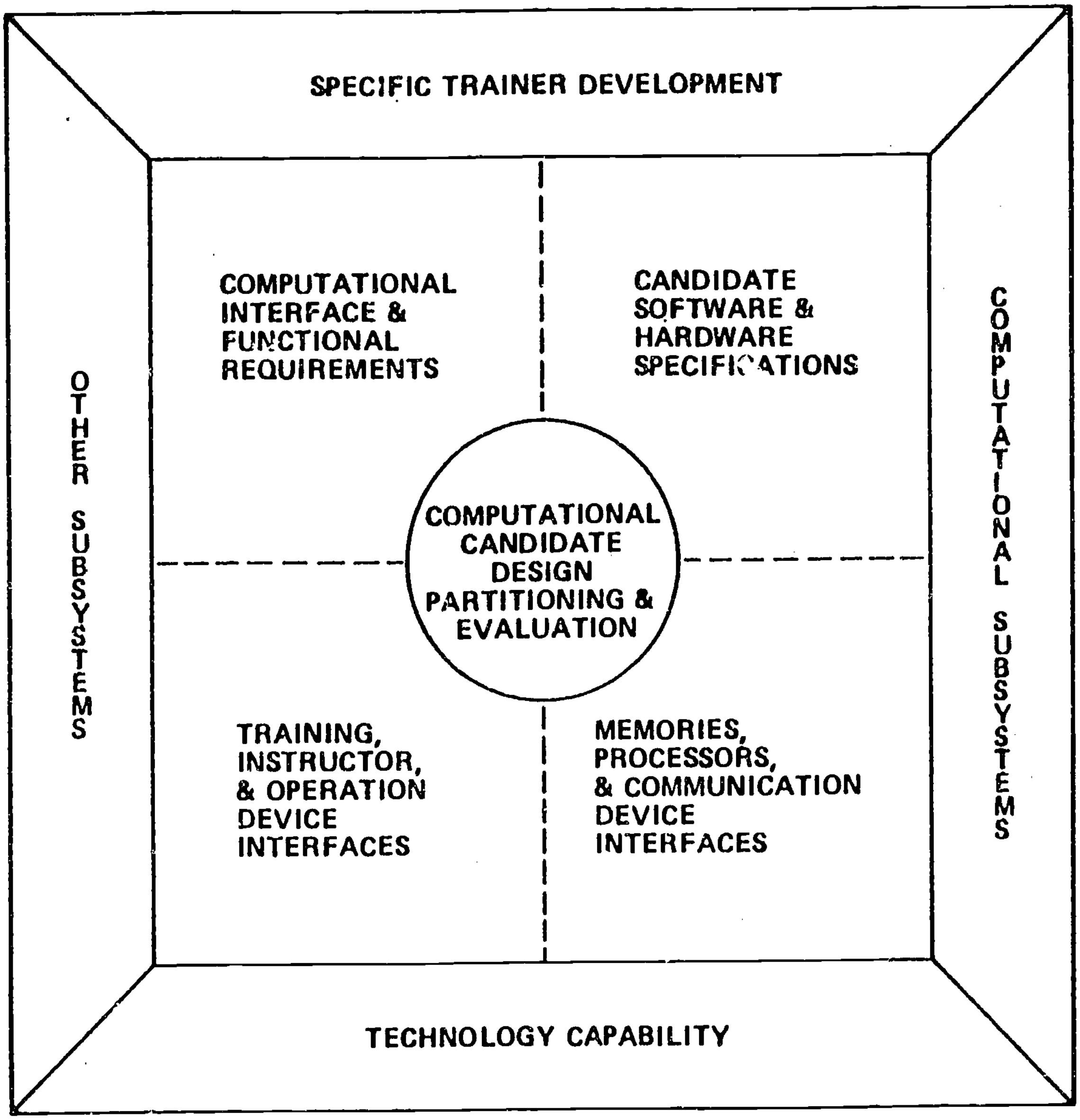


Figure 15. Computational design evaluation must relate a specific design in terms of current technology capabilities for both external communications and internal computational subsystem details.

computational interface requirements, software design, and hardware design respectively. The lower half represents the technology data base, which permits an abbreviated means for entering the design details on which the partitioning algorithm is to operate. The left half relates the devices to be serviced by the computational subsystem, and the right half reflects the internal computational subsystem structure organization and devices.

Although the data are extracted from independent sources, it requires interactive coordination and configuration controls to ensure that accurate, up-to-date, best estimates are utilized for the evaluation at hand. The evaluation criteria and constraint inputs facilitate configuration controls, parametric analysis, and partitioning flexibility with respect to prohibited and/or preassigned allocations in addition to initial allocations. The details of this segmented data base are now described in terms of implementation considerations.

4.2 DATA BASE MANAGEMENT

Two major recommendations are being made to facilitate orderly consolidation of the storage and retrieval for each of the five data base areas that provide the driving source of information for the partitioning algorithm and candidate design evaluation process. These recommendations are as follows:

1. The addition of a standard set of candidate design specification tables that address the software and hardware designs as independent sets of parametric measures.

2. The establishment of a design evaluation data base repository utilizing an interactive file management system under the configuration control of ASD/ENETC.

This subsection supplies key factors that should be evaluated and modified as necessary to facilitate an orderly transition to an automated algorithm implementation as presented in Section 4.4. Proper utilization will require a training indoctrination as to the potential benefits to both the flight trainer developer and evaluator communities. Before the recommended input forms are described, several master data structures are delineated that have a direct influence on validity of data entries and provide the key to independent software and hardware design characterization.

4.2.1 Master Data Structures

These master structures include (a) data block characterization, (b) memory characterization, (c) task characterization, and (d) processor characterization.

Combinations of these structures are incorporated into the recommended forms for each of the five data base input areas presented in Appendix A.

4.2.1.1 Data Block Characterization - Data characteristics such as source, volume, frequency, content, and destination are the real-time drivers of the computational subsystem from both external device and internal task communications, command, and control. Table 5 denotes attributes required by the software partitioning algorithm for each data block that is acted upon or created by the computational subsystems being partitioned. Note that these attributes do not tie the data block to a specific storage device. Only external system blocks are identified as being related to a given type of peripheral interface; for example, a cockpit control setting input buffer block has a definite source device that must be monitored at a predetermined sample rate. On the other hand, the data to be computed by one task and used by a sequentially dependent task are described in terms of minimum storage device requirements for their storage and retrieval utilization. These master block definitions are then referenced by the block identification when referenced in the task descriptions (Section 4.2.1.2) or in evaluation allocation restrictions (Section 4.2.3).

4.2.1.2 Memory Characterization - A wide variety of memories may be incorporated into a candidate design configuration for a flight trainer. For purposes of partitioning, memories are categorized (as denoted in Table 6) to include read-only memory (ROM), writable control stores (WCS), main random access memory (RAM), rotating random access mem ory (RRAM), and sequential memories (SM). Within each of these categories are additional retrieval and storage characteristics for data representations of addressable units. These representations permit the generic data block parameters of Section 4.2.1.1 to be matched with appropriate memory devices in the candidate configuration for which partitioning is being performed.

4.2.1.3 Task Characterization - Specification of task attributes, which are independent of the processing hardware, poses a very challenging problem area for incorporating the traditional hardware-dependent design customs and notations that have evolved not only in flight training simulator design but computational system designs in general. At this point in software design history, several emerging philosophies for design standards seem to be contradictory concerning the level of specification and the documentation language used to convey the detailed software algorithms to be implemented. At one extreme is the use of English-like structured pseudo code, which is favored for its features of being easy to follow and comprehend. On the other hand, there is an emphasis for precise, unambiguous mathematically enumerated representations that provide the specific computations but, if not annotated with English descriptions, they become very hard to follow, except for persons who are very familiar with the specifics of the algorithm. Most designs are generally a mixture of these two approaches,

TABLE 5. DATA BLOCK CHARACTERIZATION

ATTRIBUTE	VALUES	UNIT/MEANING
Identifier	6-Character Mnemonic	Provides a unique identifier for cross-reference and labeling purposes
Level	1 Character	
	= 'S'	System Interface
	= 'G'	Global (used by more than one task)
	= 'L'	Local to one task but must be saved
	= 'T'	Temporary scratch area for a given task
Discipline	4-Character Code	Provides basic I/O requirement for determining suitable memory device allocation
	= 'FIFO'	Queue
	= 'LIFO'	Stack
	= 'SEQ'	Sequential
	= 'RAN'	Random
	= 'ROR'	Ready-Only Random
	= "ROS'	Ready-Only Sequential
	= 'CBUF'	Circular Buffer
Sizing		
• Maximum Records	Positive Integer	Records
• Bits/Character	Positive Integer	Bits
• Characters/Word	Positive Integer	Bytes
• Average Words/Record	Positive Integer	Words
• Maximum Words/Record	Positive Integer	Words
• Minimum Words/Record	Positive Integer	Words

66

TABLE 6. MEMORY DEVICE CHARACTERIZATION

ATTRIBUTE	VALUES	UNIT/MEANING
Identifier	10-Character Mnemonic	Provides a unique identification for each memory device in the technology data base for which the following attributes define
Type	4 Characters	
	= 'ROM'	Read Only Memory
	= 'RAMM'	Random Access Main Memory
	= 'RRAM'	Rotating Random Access Memory
	= 'SM'	Sequential Memory
	= 'WCS'	Writable Control Store
Size in Bits		
• Minimum	Positive Integer	Bits
• Maximum	Positive Integer	Bits
• Increments	Positive Integer	Bits
Number of Different Addressable Units	Positive Integer	
For Each Addressable Unit		
• Level	4-Character Code	
	= 'BIT'	Bit Addressable
	= '6BB'	6-Bit Byte Addressable
	= '8BB'	8-Bit Byte Addressable
	= 'WORD'	Word Addressable
• Bits/Unit Level	Positive Integer	Exclusive of Parity or Error Deletion Correction Bits
• Read Access Time	Real	Nanoseconds
• Read Cycle Time Unit	Real	Nanoseconds
• Maximum Sequential Units Transferred for Single Read	Positive Integer	Same as Unit Level
• Write Access Time	Real	Nanoseconds

TABLE 6. MEMORY DEVICE CHARACTERIZATION (Sheet 2 of 2)

ATTRIBUTE	VALUES	UNIT/MEANING
• Write Cycle Time/Unit	Real	Nanoseconds
• Maximum Sequential Units for Single Write Access	Positive Integer	Same as Unit Level
• Error Detection/ Correction	6-Character Code = 'PARITY' = 'SECDED'	 Parity Bit Single Bit Error Correction Double Bit Error Detection
Number of Sup- pliers for Each Supplier	Positive Integer	
• Identifier	10 Characters	Unique Identifier
• MTBF	Real	Hours - Mean Time Between Failures
• MTTR	Real	Hours - Mean Time to Repair
• MSPM	Real	Hours - Rescheduled Preventive Maintenance
• MTPM	Real	Hours - Mean Time for Preven- tive Maintenance

68

which facilitates the overall functional flow, high-level presentation and permits a traceability structure for enumeration of detailed design computations and decision logic.

The remaining problem area of design specification relates to the specific notation. Certain aspects of flight trainer computational algorithms have become well-defined, i.e., aircraft flight kinematics. These algorithms are generally used for making benchmarks on new candidate processors. Thus, for well-established algorithms, a master set of simulation task benchmarks can be established for each candidate processor being considered. New algorithms require a more fundamental breakout of the instruction mix to ascertain timing and sizing elements. In summary, a master set of software task attributes are presented in Table 7. The establishment of a master instruction mix, task I/O descriptors, and task enablement features is recommended as one of the steps (Section 4.4) toward algorithm implementation. Related to this master instruction mix is the development language for task code generation. Recent trends in simulator coding have incorporated FORTRAN code for the scientific mathematical application models, but there is still a strong dependence on the assembly level code for expressing real-time executive and I/O handler modules to meet the real-time timing requirements. The selection of a task design instruction mix notation should be coordinated with the simulation high-order language efforts and processor instruction architectures.

One way to obtain this information would be the use of a graphical task flow representation, which included a standard design notation to indicate the instruction sequences, loops, and relationships with I/O. A flow notation, such as TBE's Input/Output Relationships and Timing Diagrams, can be automatically traversed with the instruction mix and I/O features being identified and reformatted for use with the partitioning algorithm. This would require that a standard flight trainer computational design language and flow representations be established, thus providing a standardized way for documenting the detailed task computational designs.

An important note is made here regarding the traditional means of expressing task sizing and timing in terms of adds, multiplies, branches, etc. The instruction mix need not be at the machine level. Instead, it should reflect a set of simulation macros, such as single variable linear table interpolation, and trigonometric functions. Each of these, in turn, is characterized for each candidate processor as to timing and sizing. If the simulation macro has been implemented in firmware or as part of a mathematical package, the sizing is reduced in terms of the main instruction storage for the task.

4.2.1.4 Processor Characterization - Processor technology is constantly expanding in terms of operating system and instruction set capabilities. Table 8 lists processor attributes that pertain directly to the software partitioning algorithm. The operating system features

TABLE 7. TASK CHARACTERIZATION

ATTRIBUTE	VALUES	UNIT/MEAINING
Identifier	6-Character Mnemonic	Provides a unique identifier for cross-reference and labeling purposes
Source Language	10-Character Code	Must match entry in the master source language list maintained for current processor technology
Instruction Mix for Each Instruction Type:		
• Instruction Identifier	10-Character Code	Must match entry in master simulator instruction mix identifiers
• Sizing Count	Positive Integer	Number of times this instruction appears in code
• Execution Count		Number of instruction interactions considering looping conditions for average and worst-case logic
Average	Positive Integer	
Worst Case	Positive Integer	
Data Retrieval for Each Task Input		
• Block Identifier	6 Characters	See Table 5
• When	6-Character Code	
	= 'START'	All records read at first of task before main processing
	= 'ALONG'	Records processed one at a time
• Average Input	Positive Integer	Records
• Minimum Input	Non-Negative Integer	Records
• Maximum Input	Positive Integer	Records

70

TABLE 7. TASK CHARACTERIZATION (Sheet 2 of 2)

ATTRIBUTE	VALUES	UNIT/MEANING
Data Storage for Each Task Output:		
• Block Level	1 Character	See Table 5
• Block Identifier	6 Characters	See Table 5
• When	6-Character Code	
	= 'ALONG'	Records are output via individual processing
	= 'END'	Records are output just prior to task exit
• Average Output	Positive Integer	Records
• Minimum Output	Non-Negative Integer	Records
• Maximum Output	Positive Integer	Records
Enablement		
• Type	4-Character Code	
	= 'TIME' = 'DATA' = 'SLVD' = 'TAD'	Time Enabled Data Enabled Slaved to Master Task Time and/or Data Enabled
• Frequency 1	Real	Iterations/Second for Time Enablement
• Frequency 2	Real	Iterations/Second for Data Enablement
• Frequency 3	Real	Iterations/Second for Slaved

TABLE 8. PROCESSOR CHARACTERIZATION

ATTRIBUTE	VALUES	UNIT/MEANING
Identifier	10 Characters	Unique identifier for processor with the following attributes
Operating System		
• Multitasking		
▲ Levels	Integer .GE.1	
▲ Number of Priority Levels	Integer .GE.0 .LE. Levels	These many levels are serviced in a priority fashion. The remaining levels are serviced in a circular time-shared fashion.
• Enablements	Integer	Enablements/Second
▲ Maximum Time Enablement Frequency		
▲ Resource Management per Time Enablement	F1Ø.9.GE.Ø	Microseconds
▲ Maximum Data Enablement Frequency	Integer	Enablements/Second
▲ Resource Management per Data Enablement	F1Ø.9.GE.Ø	Microsecond
▲ Maximum Slaved Enablement Frequency	Integer	Enablements/Second
▲ Resource Management per Slaved Enablement	F1Ø.9.GE.Ø	Microseconds

72

TABLE 8. PROCESSOR CHARACTERIZATION (Sheet 2 of 3)

ATTRIBUTE	VALUES	UNIT/MEANING
• For Each Task Level L		
▲ Maximum Number of Task Level L	Integer .GE.1	
▲ Task Service Scheme for Level L	Code	
	= 'P'	Priority
	= 'C'	Circular
	= 'F'	First-in, First Out
• Level Resource Management	F10.9 .GE.0	Microseconds
Simulation Instruction Set Measurements for Each Benchmark Instruction 1		
• Sizing Measurements		
▲ Number of Code Memories Involved		
The Memory Type for Each Code Memory m (the first memory is the user task code -- any other memories are predefined for this processor)	4-Character Code	Must agree with master memory types defined in Group 4
▲ Length of Code in Memory m	Integer .GE.1	Number of basic units used to describe memory m (see Group 4)

TABLE 8. PROCESSOR CHARACTERIZATION (Sheet 3 of 3)

ATTRIBUTE	VALUES	UNIT/MEANING
• Timing Measurements for Each Code Memory m and k=1,2		k=1 Implies Average k=2 Implies Worst Case
▲ Number of Scratch Data Store Waits	Integer .GE.Ø	
▲ Number of Scratch Data Store Waits	Integer .GE.Ø	
▲ Computational Total for All Memories	Integer .GE.Ø	Cycles
• Application Development Measurements Using Language L of the Master Language List		
▲ One Item Development Charge	Integer	Man-hours
▲ Change per Application Instruction of this Type	Integer	Man-hours

74

applicable to software partitioning relate to multitasking disciplines, limits, and resource management services. The instruction set is characterized in terms of the master simulation instruction set as described in Section 4.2.1.3, along with attributes for user memory I/O versus preprogrammed resources plus development cost estimates.

4.2.2 Suggested Input Forms

The forms, as designed, may be used directly by a data keying operator to produce keypunched cards or entry directly onto a file via an interactive data entry terminal. Specific physical file formats are not specified since they will be a function of selected computer file image capabilities described in Section 4.3. Because of the volume of input sheets, they are presented in Appendix A for each of the data base files.

During the design of the input forms, emphasis was placed on consolidation and cross-reference techniques that facilitate an organized straightforward user input interface. The software partitioning algorithm requires an assortment of specific data to fully define trainer system interfaces plus computational hardware and software design details that must be accurate if a good partition allocation is to be obtained. The separation of forms is based on the five major input areas, and it is recommended that these areas be standardized for presenting the respective interface requirements, software task/data design relationships, candidate hardware design configuration, technology capabilities, and evaluation priorities, including the candidate initial design allocation as a starting point for partitioning optimization.

4.3 TARGET COMPUTER AND SOURCE LANGUAGE SELECTION

The selection of the computer system for the partitioning algorithm should consider, as a minimum, the following features, which must be incorporated to facilitate automatic implementation of the partitioning algorithm and its potential expansions:

1. Data base management system
2. Structured program language
3. Modified linear mixed integer program optimizer
4. Computational speed and accuracy.

Each of these features is described in more detail in the following paragraphs.

4.3.1 Data Base Management System

The interrelated, yet separate, data files (described earlier in this section) of the recommended flight trainer automated repository are best implemented under a standard data base management system that permits creation, update maintenance, and configuration management of all data and program files. It is recommended that system data file management utilities be available to the user in several different modes, including batch job control, interactive terminal commands, and user program code directives to permit a flexible, yet controlled, data access environment. Direct record access capability is an essential feature for implementation of the software task and block description plus the technology data base files.

The amount of data is a function of the flight training simulator computational candidate designs to be evaluated. Table 9 provides an abbreviated summary of sizing relationships for each record type group contained in the respective files required for the partitioning algorithm. The data base management should include memory management of code and data required for execution. Internal tables utilized by the algorithm are sized in Table 10. The algorithm code is estimated to be 10,000 lines of structured FORTRAN exclusive of potential data manager and optimizer extensions.

4.3.2 Structured Program Language

Evaluation code (code used to facilitate manual analysis) is a very useful tool if it can be maintained under configuration control and permit expansion to more detailed models when necessary for a given evaluation analysis. Structured source code facilitates modularity and, thus, permits model expansion. Several source languages are included here as candidates for the partitioning algorithm implementation, including FORTRAN 77, JOVIAL, and ADA. These languages were selected based on current DOD-approved languages and language development activities. Pros and cons for each are now presented.

The widespread recognition of FORTRAN for scientific and mathematical programming makes it the preferred language of the three languages considered. The newest ANSI FORTRAN 77 standards incorporate character manipulation, which is independent of machine architecture. Its use of structured logic includes both true and false process definitions without the use of extraneous "GO TO's." File manipulation capabilities have also been expanded to include file status checks and standardization of certain types of data storage/retrieval mechanisms that have previously required vendor-peculiar FORTRAN extensions. Some problems may be encountered with new compilers being released to meet the new FORTRAN standards, but these compilers should evolve rather quickly to support most of the ANSI 77 features. This will result in code that is more easily transported from one machine to another. This is an important aspect, since the partitioning algorithm does not require a dedicated computer system, and as such, it is envisioned as being a useful tool for flight training simulator developers and maintenance reconfiguration analysts, as well as for Air Force evaluators. Each of these specialists generally has his own in-house computer system tailored for specific analysis needs.

TABLE 9. EXTERNAL FILE SIZING REQUIREMENTS

FILE			RECORD		
IOPT	TITLE	TYPE	GROUP	TITLE	SIZE
1	Trainer Computational Interface Requirements	Sequential 80-Column Card or Keyboard	1	File ID	20 Characters
			2	System Interface Device	1 to 3 Cards per Device
			3	System Data Block (or Buffer)	1 Card per Block
2	Baseline Application Components	Sequential 80-Column Card or Keyboard Entries	1	Software Job/Task ID	20 Characters
			2	Data Block Definitions	1 Card per Block
			3	Task Definition	1 Header Card 1 Card per Instruction Mix Definition 1 Card per Task Block Reference
			4	Baseline Load Definition	1 Load Header Card per Load 1 Card/Task/Load

TABLE 9. EXTERNAL FILE SIZING REQUIREMENTS (Sheet 2 of 3)

FILE			RECORD		
IOPT	TITLE	TYPE	GROUP	TITLE	SIZE
3	User Evaluator Inputs	Sequential 80-Column Card or Keyboard Entries	1	Run File Identifiers	2 Cards
			2	Global Evaluation Factors	3 Cards
			3	Specific Evaluation Factors	1 Card/Factor
			4	Partitioning Assignment Constraints	1 Card/Constraint
			5	Selective Coefficients	Coefficient Selection
4	Candidate Configuration Definition	Sequential 80-Column Card or Keyboard Entries	1	File Identifiers	1 Card
			2	Candidate Device Definition	1 to 3 Cards per Device
5	Technology Data Base	Random Access Hierarch-ical File Data Base Structure	1	File Identifer	20 Characters
			2	Master Technology Lists • Component Categories • Devices/Component	 10 Char/Category 10 Char/Device

TABLE 9. EXTERNAL FILE SIZING REQUIREMENTS (Sheet 3 of 3)

FILE			RECORD		
IOPT	TITLE	TYPE	GROUP	TITLE	SIZE
5	Technology Data Base (Concluded)			• Instructions • Block Disciplines • Block Types • Languages	10 Char/Instr 4 Char/Disp 1 Char/Type 10 Char/Lang
			n+2	Component n's Specific Device Definitions and Attributes	See IOPT-5 GRP n+2

TABLE 10. INTERNAL ALGORITHM TABLE SIZING REQUIREMENTS

TABLE IPT NO.	TABLE TITLE	WORDS (60-bit words)
1	Limits, Constants, and Code	20
2	Current Problem Sizing Controls	9
3	Priority Controls	28
4	Current Processor List	P*(13+i)
5	Current Memory List	11*M
6	Current Communication Link List	(3+3*QND)*9
7	Current External Device List	(4+DB)*d
8	Task/Processor Allocation and Restrictions	9*T*P
9	Memory/Processor Communications Allocation and Restrictions	(4+4e)*M*P
10	Memory/Block Allocation and Restrictions	5*M*B
11	Master Block List	(11+M+2T)*B
12	Master Task List	(16+5i+6*B+e)*T
13	Scratch and Local Parameters	To be Defined

80

JOVIAL is mentioned because of its recognition by the Air Force as a standard language for embedded computer systems development. A major drawback is its limited I/O capabilities, which is a major factor with regard to the partitioning algorithm's large data base handling requirements.

ADA is also mentioned since it is the DOD language being developed with source language standardization as a major goal to support software development of new military computational subsystems. The on-going compiler developments are limited to experimental compilers and compiler design efforts. Therefore, at this time it is not a feasible candidate for actual algorithm development and testing. It will be 2 to 3 years before it is available in an operational development setting. Further implementation/expansion should monitor and consider ADA since its features will permit more configuration control as well as the structured expression of concurrent process control flows, I/O, and computations with concise data base definition.

In conclusion, FORTRAN is the recommended language for implementation of the partitioning algorithm.

4.3.3 Modified Linear Mixed Integer Program Optimizer

The partitioning algorithm has the potential for future interfaces with a modified linear program mixed integer program optimizer. The current algorithm design is based on a heuristic algorithm driver that assumes that an initial feasible partition exists with respect to the basic real-time processing requirements of data availability, task timing, and less than 100% processor/memory allocation. From this initial feasible solution, it seeks to determine and make improvements on the initial partition with respect to three goals: (a) processor load balance within given growth allotments, (b) memory utilization within growth tolerances, and (c) minimization of development costs. Although heuristics do not guarantee an optimal solution, it is anticipated that the complexity of priorities and data constants will change frequently, which makes the finding of the true optimal a meaningless exercise. However, optimizers can be employed to help find an initial feasible solution and to find optimal subset solutions under the control of the heuristic decision tree. In the case of the partitioning algorithm, the initial feasible solution poses the largest problem in terms of sizing and numeric accuracy techniques that are required. Table 1 summarizes the optimizer sizing as a function of the size of candidate designs to be evaluated.

4.3.4 Computational Speed and Accuracy

Although the partitioning algorithm is not as demanding as real-time simulation or control codes, it is important that it be able to support quick-turnaround evaluation runs to expedite the given evaluation case. The complexities of the processor utilization calculations in terms of task computations, resource management, and I/O are iterated with respect to potential processor tradeoffs for load balance calculations that involve a variety of attributes. Since the basic computations are subject to mathematical model expansions and changes, floating point capabilities are recommended to permit new equations to be introduced, as required, without the burden of fixed-point scaling.

Units have been selected to keep related variable numeric order of magnitudes within computational limits of most scientific machines. These units should be periodically examined as technology advancements are made. For example, many current real-time flight trainer application cycles are based on 1-sec intervals with subcycles or subframes measured in terms of milliseconds. As timing improvements are made, these may take on smaller increments of time for application cycling, hence the need for their periodic reappraisal. Another factor is machine cycle time, which is currently measured in nanoseconds; thus, certain calculations involving memory I/O must be accumulated separately to obtain totals that can then be used to determine any appreciable I/O timing for tasks that handle large volumes of data in addition to computational processing. Typically, 32-bit floating point can represent six significant digits. Thus, if a basic unit is assumed to be 1 sec, the nanosecond effectively is disregarded unless accumulated separately. However, if either double precision (64 bit) or 60-bit single precision is used, there is no problem. An alternative is for task memory I/O, resource management, and individual instruction timing computations to be accumulated for total task time in microseconds, and then task times may be added separately for a given application cycle time in terms of current task/cycle relationships. Thus, there is the need for floating point, with a minimum of 32-bit words sufficing for most operations, and either segmented units or double precision variables to account for application subtask timing computations.

The use of preemptive priorities rather than weighted priorities permits processor loading, memory allocation, and development costs to remain in their standard units without any input scaling and output rescaling. However, in each priority level, numbers for a given task or data block should be summed separately from totals being used for total memory or total processor utilization to avoid underflow accumulation problems.

4.4 RECOMMENDED IMPLEMENTATION SCHEDULE

The major tasks and their hierarchical relationships are depicted in Figure 16. Each of these tasks is briefly described in this section with cross-references to appropriate report sections for related details. Although some parallel task sequences are depicted, there are some interdependencies, as denoted in Figure 16. These interdependencies are basically handled at major detailed reviews, which are recommended to be held quarterly to assess the implementation progress, to

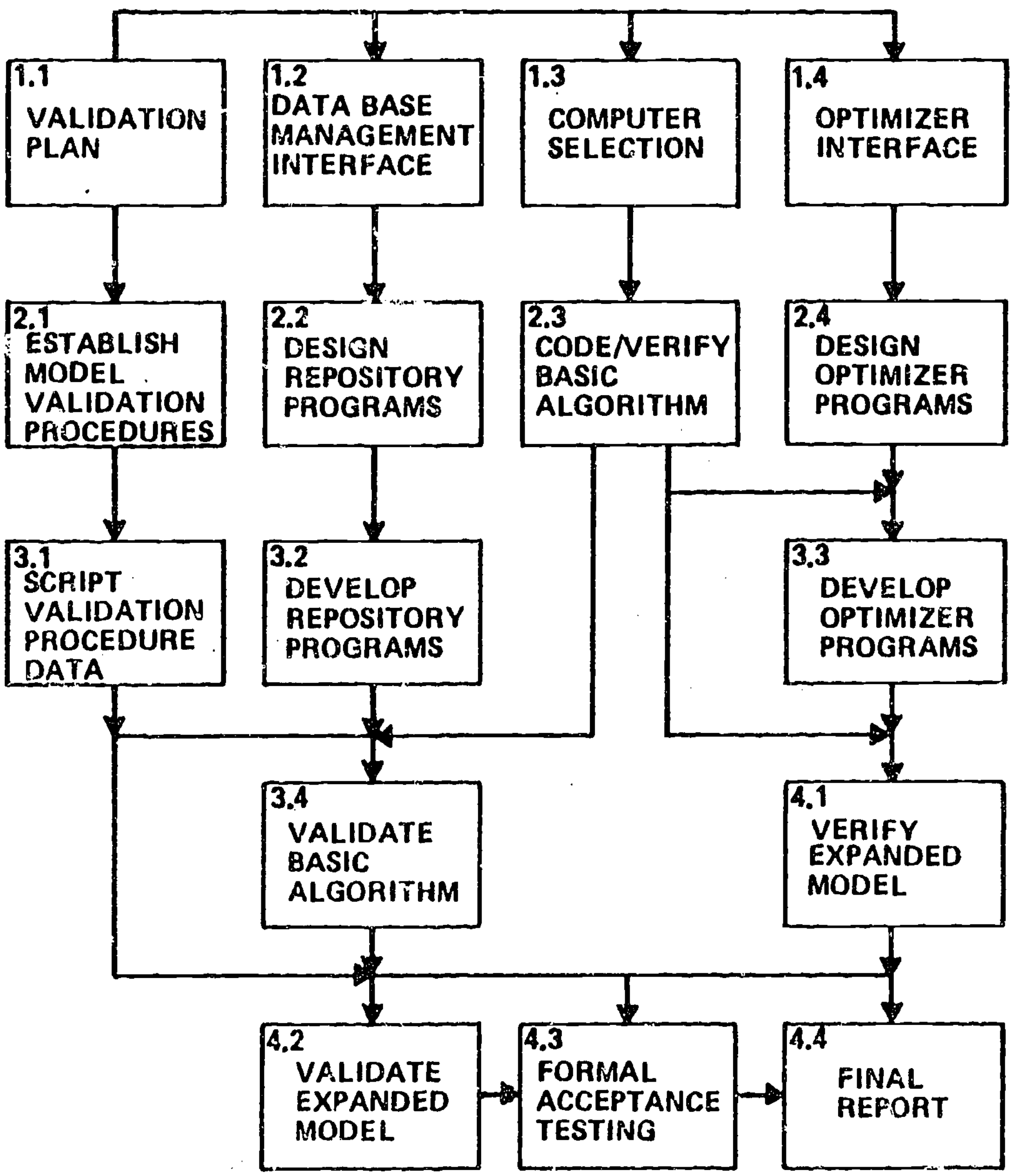


Figure 16. Algorithm implementation tasks.

ensure that interface definitions are adhered to, and to establish more detailed interfaces as the appropriate operational consideration details become known.

Figure 17 groups the tasks into four major implementation phases over a 2.5-year period. There is an overlap between Phase III and Phase IV, with the major emphasis of Phase III placed on basic (as currently designed) algorithm validation and with Phase IV emphasis on an expanded validated model incorporating an optimizer for selected aspects of the partitioning algorithm. The implementation tasks are now described by phase. To make a complete task statement, there is some redundancy with earlier report sections. Cross-references are made to avoid excessive redundancy.

4.4.1 Model Validation Plan and Selected Computer Interfaces

Although the candidate computer selection aspects have been described (Section 4.3), the specific computer implementation must be further delineated to obtain a practical partitioning allocation and evaluation tool for flight trainer simulator computational candidate design. Existing evaluation computer facilities should be reviewed for current formats and data collection procedures in addition to the current computer capabilities to contribute basic inputs to the Phase I tasks, which are now briefly described.

4.4.1.1 Validation Plan - The sample problems manually demonstrated under this contract have verified the feasibility of the partitioning algorithm design. However, they do not constitute a model calibration case from which a confidence level of model validity may be derived. As evidenced in the mathematical statement of the partitioning problem (Section 3.1), there are many interrelated variables and factors that drive the partitioning process, necessitating some parametric automation techniques to fully analyze the automated design validity and stability for real-world data. The validation plan will permit controlled algorithm implementation testing to determine its validity with respect to known partitioning situations of selected flight training simulator computatational designs. By addressing evaluation partitioning problems to be handled prior to algorithm coding, the evaluation community is essentially establishing the foundation for the algorithm acceptance test with respect to its role as an evaluation tool.

As a minimum, the validation plan should identify the flight trainer system(s) to be used as the algorithm implementation baseline. It should also extrapolate intended sizing of the algorithm application in terms of the number of each data base item described in Section 4.2 (i.e., number of tasks, blocks, processors, memories, etc.). A set of test cases should be drafted in an outline format as to specific algorithm features to be incorporated and tested for both the basic model and the expanded model.

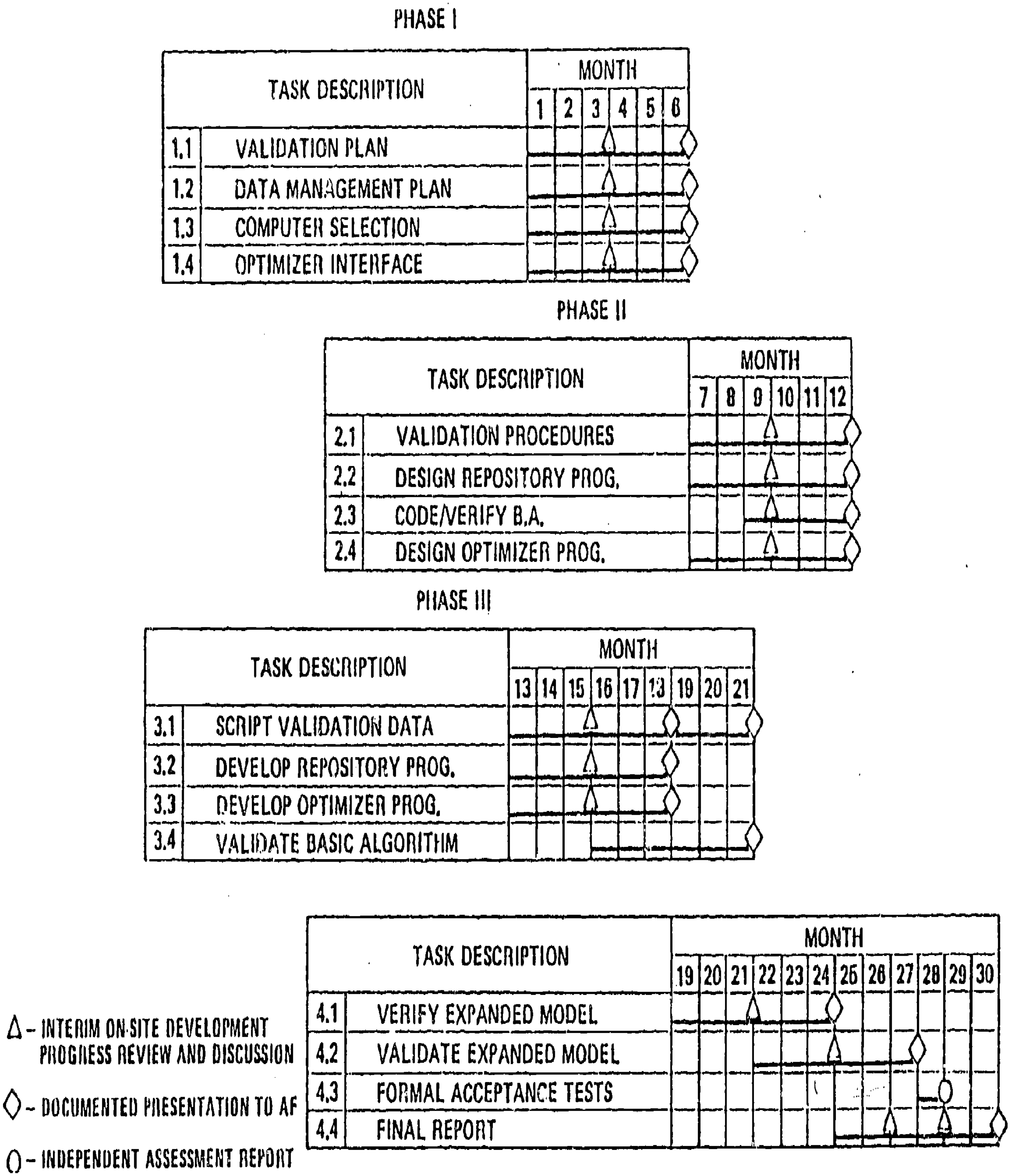


Figure 17. Projected time relationship of tasks.

81

4.4.1.2 Data Base Interface - The specific flight trainer computational design repository format and data base management utilities should be delineated by this task. This includes finalization of the user interface formats (such as those contained in Appendix A) and the format by which the partitioning algorithm may retrieve its inputs and store its outputs with respect to the repository and the interactiv and/or batch user.

This task incorporates the data collection, storage, and retrieval mechanisms, plus quality assurance steps necessary for algorithm implementation and usage. The repository data management should incorporate responsible agencies for each input area and make maximum use of pre-editing and file management utilities of the selected computer system. The results of this task should be compiled in the form of a users' manual for the flight trainer design repository and specifically address the partitioning algorithm interfaces. These interfaces include the master design simulation language instruction set and guidelines for processor, memory, task, and data baseline descriptions (covered in Section 4.2) that will streamline the orderly preparation of inputs and permit gradual controlled growth into a fully tested and implemented repository system for multiple evaluations.

4.4.1.3 Computer Selection - Computer candidate selection has been discussed in Section 4.3. This task ties Phase I activities together to determine the specific coding standards and interfaces to be employed for algorithm implementation for a given computer facility.

4.4.1.4 Optimizer Interface - This task permits the long-range interface goals to be defined for potential optimization steps in the heuristically driven partitioning algorithm. This is a major area for further study and, as such, is recognized in Section 5.3.

4.4.2 Automated Algorithm Verification and User Design Foundation

Phase II permits the initial automation of the basic algorithm and delineates additional programs that will aid in the bookkeeping and increase computational confidence levels of an expanded partitioning algorithm. Each of the tasks is now defined.

4.4.2.1 Establish Model Validation Procedures - This task expands and enumerates the test cases outlined in the test plan of Phase I. The nature of the basic partitioning algorithm is to seek and, if possible, find an improved partition of tasks. Thus, the test procedures must include the means for reconfiguring the subject flight trainer for which a supposedly "better" partition has been found. In addition, related performance measurements of the newly partitioned configuration must be specified as to what and how they are to be collected and evaluated to access the predicted partition improvements of the partitioning algorithm. To assist in this step, the multiple processor simulator being designed under separate contract may be used to provide a quick look at the dynamic aspects of the new partition prior to making a reconfiguration decision. All of these considerations must be placed into a timeline for algorithm validation testing to account for permissible reconfiguration in the partitioning restriction. For example, if special-purpose tasks may only reside on special-purpose processors, they should be declared as such in the partitioning algorithm evaluation options. Thus, realistic, measurable validation test procedures are the goal of this task.

4.4.2.2 Design Repository Programs - The users' manual of Phase I will undoubtedly require specific repository storage/retrieval programs to be designed to augment the system-supplied data base capabilities to support the flight trainer evaluators "input jargon" and to efficiently handle the input and subsequent updates to each of the various files to ensure consistency and completeness of any given repository transaction. The results of this task constitute the detailed design of each and all repository programs to be implemented in Phase III.

4.4.2.3 Code/Verify Basic Algorithm - This task is the most straightforward of all of the tasks and simply entails the coding, debugging, and verifying of the basic algorithm as designed and demonstrated as part of this subject contract. This provides the working baseline for all future expansion in both model repository and optimizer interfaces. The results of this task provide a source code listing, verification test case execution outputs, and documented interpretation.

4.4.2.4 Design Optimizer Programs - The emphasis of this task is to be placed on upgrading and complementing an existing mathematical optimizer package selected in Phase I with respect to computational and logic needs peculiar to the partitioning application. This task requires extensive knowledge and experience with mathematical optimization codes and their numerical stability in terms of accuracy, scaling, iteration, and masking techniques that can judiciously expedite the solution space search for initial feasible solutions. The task also requires knowledge and experience with optimal subproblem solutions as called by the heuristic driver of the basic algorithm. The results of this task will comprise the detailed design of programs to be implemented to support the optimizer interface.

4.4.3 Basic Model Validation and Expanded Program Interface Development

This critical phase permits the large-scale, real-world data assessment of the basic algorithm to be made. The first part of Phase III is associated with specific data collection, scripting, and support program coding. The latter part of this phase incorporates efforts of the first part for basic algorithm validation testing. In addition, the optimizer programs are developed in preparation for the Phase IV expanded model. Each of the Phase III tasks is now described.

4.4.3.1 Script Validation Data - Validation input data must be collected and prepared utilizing the validation input procedures for each test case for basic algorithm and expanded algorithm validation cases. A test case can not proceed until its basic inputs have been properly prepared.

4.4.3.2 Develop Repository Programs - The programs designed in task 2.2 of Phase II are coded, debugged, and verified by means of validation input procedures to assist in the input processing of task 3.1.

4.4.3.3 Develop Optimizer Programs - This task codes and debugs the programs designed in task 2.4 of Phase II in preparation for expanded algorithm verification and validation of Phase IV.

4.4.3.4 Validate Basic Algorithm - Each validation test case is made in the order prescribed in the test procedures. If any problems are encountered, their impact on the test plan and case procedures must be fully evaluated to determine what action, if any, is necessary to continue the test program. All test execution reports should be included as appendices to the test summary report. It is anticipated that certain validation tests can be run prior to complete implementation of the repository to exercise the fundamental paths of the algorithm.

4.4.4 Expanded Model Verification, Validation, and Formal Acceptance Testing

Phase IV paves the final path to the realization of the partitioning algorithm as part of the standard flight trainer simulator computational design evaluation and/or design guide tool. The full repository and added optimizer capabilities developed in the first three phases are now integrated and tested to provide a controlled user interface for multiple evaluation situations. The tasks are now defined.

4.4.4.1 Verify Expanded Model - This task consists of selected basic algorithm test cases to verify that these cases are still properly handled in the expanded model. In addition, new path verification tests are incorporated by the designer to verify that new capabilities are working as designed.

4.4.4.2 Validate Expanded Model - This task performs the extensive testing as defined in the validation procedures for the extended model. As with basic algorithm validation, if any problems are encountered, their impact on the test program must be evaluated and it must be determined whether any action is necessary for continuance of the test program. All execution results should be included as appendices to the test summary documentation.

4.4.4.3 Formal Acceptance Test - The complexity of the partitioning algorithm and its potential evaluation decision-making impact necessitates the need for formal Government acceptance tests. These tests should be scripted and performed by an independent organization to fully assess the delivered capability with respect to completeness of documentation, configuration, quality, and purpose. The major developer is involved as a consultant to explain or expand documents and to respond to any questions concerning the delivered operational package. It is anticipated that Government flight trainer system evaluators will be responsible for scripting and conducting these independent test procedures since the test will serve as a training task that emphasizes the intended operational user environment of the algorithm.

4.4.4.4 Final Report - The emphasis of this task is to be placed on finalizing documentation of the automated algorithm capabilities, findings, and conclusions. This documentation should be accompanied with the final user, test, and program maintenance documentation for specific program implementation details.

5. CONCLUDING REMARKS

Software partitioning is a complex, design development/evaluation, decision-making process with many tradeoffs to be analyzed for selecting a good candidate flight training simulator computational design for a particular operational trainer implementation or upgrade. This section briefly summarizes the details presented in Sections 2 through 4 in terms of the study findings, related work, and areas for further study.

5.1 FINDINGS

Candidate software designs expressed independently of candidate hardware are the basic key design feature that permits software partitioning flexibility. This is not the traditional design approach currently in use for system design. This project has defined the types of design data that will permit independent assessment of baseline software tasks for alternative multiple-processor configurations. The key data areas are the establishment of a standard design language and an automated repository for the given application design data.

The partitioning algorithm has been designed as a general partitioning algorithm for software systems, and it is the data collection process (Section 4.2) that will make this algorithm unique for a given application implementation. In this way, it is seen as a useful tool for the evaluation of a wide variety of computational subsystem designs since it is not constrained to current configuration, technology, or application.

5.2 RELATED WORK

The results of this effort are closely coordinated with Contract No. F33615-79-C-0003 for the AFHRL Advanced Multiple Processor Configuration Study. The multiple-processor study is concerned with features and techniques for assessing the predicted performance of given alternative candidate designs. The partitioning algorithm is looking at task/data allocation from a static analysis point of view to ensure that real-time computational requirements are met with a balanced load. The number of entities that must be considered requires that parametric analysis in terms of average or worst-case numbers be used in the partitioning process. The dynamic environment of the flight trainer computational task allocation requires the addition of network, queuing, and simulation (batch mode) tools to predict and assess the performance of a given allocation partition with respect to representative scenario loads and resource management rules. The multiple-processor configuration contract is incorporating and expanding the conceptual repository to include the dynamic performance design aspects that are pertinent to alternative computational candidate design evaluations for operational flight trainers. The results of this related effort are to be published in the final report scheduled to be distributed on or about 31 Oct 80.

5.3 AREAS FOR FURTHER STUDY

Advancements in systems development and training features are sources of continuous change for flight trainer systems. A "good" system today may be obsolete in 5 years or less if it does not possess modular design capabilities. This is particularly true of the computational system, which must act as a coordinator, interface, and decision-maker to assist the human operators and commanders to better perform their jobs. As new/upgraded flight trainer systems are required, the basic design models plus new/modified modules may very likely require reallocation of new processor, communication, and memory technologies. Two major areas of study have been isolated as the key to potential realization of a truly automated software partitioning algorithm:

1. The employment and expansion of mathematical, mixed integer, program optimizer techniques for large-scale partitioning with multiple objectives

2. The development of a master flight training simulator computational subsystem design repository.

These two areas have been incorporated as major tasks associated with automation of the partitioning algorithm described in Section 4.4.

In conclusion, automated software partitioning is feasible. It will require further study, design, and test steps that are directly related to computer facility selection for its implementation. The major training simulator candidate design impact would be toward standardization and separation of the software design representation and data from processor hardware configuration representations and data. The results of the standardization would permit a consistent flight trainer computational design automated repository to be established and used in both new design and current design evaluation tradeoffs in the areas of software partitioning and predicted performance of multiple-processor configurations. The use of an optimizer will permit certain tradeoffs to be automatically made and determined in a more straightforward manner, permitting more time for manual evaluation comparisons and decisions.

APPENDIX A.

USER INPUTS

A.1 WORKSHEETS

EVALUATION RUN IDENTIFICATION

PARTITION NUMBER

FILE	IDENTIFIER
COMPUTATIONAL INTERFACE REQUIREMENTS	
BASELINE APPLICATION COMPONENTS	
CANDIDATE CONFIGURATION COMPONENTS	
BASELINE PARTITIONING LOAD	
TECHNOLOGY DATA BASE	

EVALUATION RUN IDENTIFIER

GLOBAL EVALUATION FACTORS

PRIORITY	ORDER	GOAL	UPPER LIMIT	COEFFICIENT LEVELS	MAXIMUM ITERATIONS
PROCESSOR UTILIZATION (% BUSY)				AVERAGE WORST CASE	
MEMORY UTILIZATION (% OCCUPIED)				AVERAGE WORST CASE	
DEVELOPMENT COST (PERSON YEARS)				AVERAGE WORST CASE	

COMPUTATIONAL SUBSYSTEM INTERFACE REQUIREMENTS FILE IDENTIFIER

REQUIRED COMPONENTS

COMPONENT TYPE	COMPONENT DEVICE	UNIQUE SYSTEM IDENTIFIER	OPTION 1, 4, 7, ...	OPTION 2, 5, 8, ...	OPTION 3, 6, 9, ...	CONTINUE
PU	PROCESSOR DEVICE		1 - ACTIVE LEVELS 4 - LANGUAGE 2	2 - OPERATING SYSTEM 5 - LANGUAGE 3	3 - LANGUAGE 1 6 - LANGUAGE 4	CONTINUE IF MORE LANGUAGES ARE SPECIFIED
MM	MEMORY DEVICE		1 - SIZING UNIT	2 SIZE		
CL	COM DEVICE		INTERFACING DEVICE TYPE	INTERFACING COMPONENT	PRIORITY	CONTINUE IF MORE INTERFACING DEVICES

92

TECHNOLOGY DATA BASE IDENTIFIER ______

PROCESSOR: ______ BYTE = ___ BITS WORD = ___ BITS

OPERATING SYSTEM: ______ CYCLE TIME: ______ μS/WORD ACCESS TIME ______ μS/WORD

MULTI TASK LEVELS:	___			**LANGUAGES**
PRIORITY LEVELS:	___	**SIZING**	**UNITS**	1 ______ 5 ______
ADDRESSABLE MEMORY:	______	K M	BYTES WORDS	2 ______ 6 ______
OPERATING SYS. MEMORY:	______	K M	BYTES WORDS	3 ______ 7 ______
SUPPORT LIBRARY MEMORY:	______	K M	BYTES WORDS	4 ______ 8 ______

MULTI TASKING FEATURES AND RESOURCES

LEVEL i to j	MAX TASKS	SERVICED P – PRIORITY C – CIRCULAR F – FIFO	RESIDENT (R) NON-RESIDENT (NR) BATCH (B)	ENABLEMENT RESOURCE MANAGEMENT: CIRCLE TYPE(S) TIME SLAVE DATA	EFFICIENCY TIME SECOND	OVERHEAD/ ENABLEMENT
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______
______-______	______	___	R NR B	T S D	______	______

·CHNOLOG· DATA BASE IDENTIFIER

	·STRUCTION BENCHMARK		ON PROCESSOR	
			USING · PERATING SYSTEM	
BASIC MEASU· ·MENTS	· ·IC (BYTES) · ·E CODE	·OR	TIMING (CYCLES) FETCHES · COMPU·ATIONAL	DEVELOPMENT (MAN YEARS) ·NE TIME · PER OCCURR·NCE
AVE· ·GE				
W· · CASE				
LANGUAG· · ACTORS:				

[illegible] DATA BASE ID: __________________

MEMORY DEVICE DEFINITION

IDENTIFIER ________

TYPE: ROM [illegible] [illegible]AM SM WC[illegible]

ADDRESSABLE UNITS

	UNIT SIZE		READ (FETCH)			WRITE (STORE)		
	DATA BITS	[illegible]ROR BITS	ACCESS (NANO SEC)	CYCLE (NANO SEC)	MAX UNITS READ	ACCESS (NANO SEC)	CYCLE (NANO SEC)	MAX UNITS WRITTEN

TECHNOLOGY DATA BASE IDENTIFIER

COMMUNICATION LINK DEVICES

DEVICE IDENTIFIER	BASIC TRANSFER UNIT MAX UNITS MAX RATE (UNITS/SEC)	INTERFACING DEVICE ATTRIBUTES					
		TECH TYPE	DEVICE IDENTIFIER	TRANSMIT UNIT	TRANSMIT RATE	RECEIVE UNIT	RECEIVE RATE

CANDIDATE CONFIGURATION IDENTIFIER

PROPOSED DEVI[illegible] TO COMPL[illegible] REQUIRE[illegible]
LIST COMMUNICATIO[illegible] LAST
REPEAT REQUIRED COM[illegible]NES IF ADD[illegible] DEVICE[illegible] [illegible]FACE[illegible]

COMPON[illegible] TYPE	COMPON[illegible] [illegible]	CANDIDATE IDENTIFIER	OPT[illegible]ON [illegible]	[illegible]	OPTION [illegible]	CONTINUE
PU	PROCESSOR DEVICE		1 - ACTIVE LEVEL[illegible] 4 - LANGUAGE 2	[illegible] SYSTEM [illegible]GUAGE 3	3 - LANGUAGE 1 6 - LANGUAGE 4	CONTINUE IF MORE LANGUAGES ARE SPECIFIED
MM	MEMORY DEVICE		1 - SIZING UNIT	2 - SI[illegible]		
CL	COM DEVICE		INTERFACING DEVICE TYPE	[illegible] COMPONENT	PRIORITY	CONTINUE IF MORE INTERFACING DEVICES

BASELINE SOFTWARE APPLICATION IDENTIFICATION

DATA BLOCK DEFINITIONS

ID	LEVEL	DISCIPLINE	SYSTEM INTERFACE DEVICE	MAXIMUM RECORDS	MINIMUM BITS/BYTE	MINIMUM BYTES/WORD	RECORD SIZE IN [illegible] MINIMUM	RECORD SIZE IN [illegible] AVERAGE	RECORD SIZE IN [illegible] MAXIMUM
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						
			-						

SOFTWARE APPLICATION IDENTIFICATIO

TASK DE...ON

TA...

EMPLACEMENT TYPE:

MASTER TASK: (IF SLAVED)

MAXIMUM TIME:

...S NO

LANGUAGE:

FREQUENCY:

INSTRUCTION MIX				...K I/O PER EXECUTION					
...TION	SIZING COUNT	AVERAGE EXECUTIONS	MAXIMUM EXECUTIONS	DATA BLOCK	INPUT* UPDATE OUTPUT	ST... AL... EN...	RECORDS PROCESSED MIN	AVE	MAX
					IUO				
					IUC				
					IIK				
					I JO	SA			
					IUO	SA			
					IUO	S...			
					IUO	SA			
					IUO	SA			
					IUO	SA			
					IUO	S...			
					IUO				
					IU...	S...			
					IUO	S...F			
					IUO	...AF			
					IUO	...AF			
					IUO	...AF			

* CIRCLE ONE PER ENTRY

BASELINE PARTITIONING LOAD IDENTIFIER ________

PARTITIONING TOTAL TIME FRAME ________ SECONDS

INDEPENDENT TASK LOADS*

TASK	TIME/SLAVED RATE		DATA ENABLED RATES		TIME LIMIT PER EXECUTION	
	AVERAGE	MAXIMUM	AVERAGE	MAXIMUM	AVERAGE	MAXIMUM

*USED TO OVERRIDE DEFAULT TASK DEFINITION LOADING. IF TASK IS ENTERED AS ZERO VALUE FOR GIVEN PARAMETERS MEANS TASK IS TO BE DROPPED FOR THIS PARTICULAR EVALUATION

EVALUATION RUN IDENTIFICATION ____________________

SPECIFIC EVALUATION FACTORS

PRIORITY (CIRCLE ONE)			COMPONENT IDENTIFIER	GOAL*	UPPER* LIMIT	CIRCLE* COEFFICIENT LEVLE	
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W
PB	MB	TC	__________	__________	__________	A	W

* IF BLANK DEFAULT IS TO GLOBAL EVALUATION LEVEL

EVALUATION RUN IDENTIFICATION

PARTITIONING ASSIGNMENT CONSTRAINTS

ASSIGNMENT TYPE F - FIXED I - INITIAL P - PROHIBITED	COMPONENT ASSIGNMENT D - DATA T - TASK	APPLICATION COMPONENT IDENTIFIER	CANDIDATE CONFIGURATION COMPONENT IDENTIFIER	VALUE IF APP.

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
1	SYSTEMS REQUIREMENTS FILE ID	RSRID	20 Characters	Used to uniquely identify system being specified
2	SYSTEM INTERFACE REQUIRED DEVICE			
	For each component			
	.Identifier	RICID	10 characters	Components Identifier to map into candidate configuration component (IOPT-4)
	.Technology type	RICTT	2 character code	Must match an entry in the technology category codes as defined in the technology data base IOPT-5 group 2
	.Specific Device Identification	RICTD	10 characters	Must match to device in technology data base IOPT-5 group 2 based upon category type

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Required options for Interfaces where k=I to number of options for Technology type RICTT(i)	RICOP (k)	Dependent upon RIC	Options must be specified to match format in techno-logy data base IOPT-5, group.
3	SYSTEM DATA BLOCK/ BUFFER			
	For each data block			
	.Block identifier	RSDBI		
	.System device iden-tifier to which assigned	RSDBD		
	.Discipline	RSDAT (I)	4 character code	Same as DEALS-IOPT5 Group 2 master data discipline codes
	.Maximum Records	RSDAT (2)	Positive Integer	
	.Record Length -Bits/Character	RSDAT (3)	Positive Integer	BITS
	-Characters/Word	RSDAT (4)	Positive Integer	Characters

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Minimum Words	RSDAT (5)	Positive Integer	words
	.Maximum Words	RSDAT (6)	Positive Integer	words
	.Average Words	RSDAT (7)	Positive Integer	words
4	SYSTEM FUNCTION REQUIREMENTS			
	For each function			
	.Function Identifier	RSFID	10 characters	
	.Execution frequency & timing	RSFFQ		TBD
	.Number of system interfaces serviced by function	RSFIS	Positive Integer	
	.Identifier for each system interface J serviced	RSFII (J)	6 characters	Must match entry in RICID (Group 2)

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	For each function to function interface			
	.Interface identifier	RSMFI	10 characters	Must match system block of group 2
	.Source function identifier	RSMFS	10 characters	Must match entry in RSFID(i)
	.Destination function identifier	RSMFD	10 characters	Must match entry in RSFID(i)
	.Communication Frequency	RSMFC	Positive Integer	Records/second

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
1	SOFTWARE JOB/TASK FILE ID	SWID	characters	20 characters maximum that identify software definition file used for run
2	GLOBAL DATA BLOCK DEFINITIONS			
	For each data block			
	.Identifier	SDBID	10 character mnemonic	SBID(i).NE.SDBID(j) for i .NE. j
	.Type			
	.Description block attribute j	SDBAT(j)		
	-Discipline	SDBAT(1)	4 char. code	same as DEALS IPT-9 group 1 master data discipline codes
	-Maximum Records	SDBAT(2)	positive integer	
	-Record Length			
	.Bits/character	SDBAT(3)	positive integer	bits

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Characters/Word	SDBAT(4)	positive integer	characters
	.Minimum words	SDBAT(5)	positive integer	words/record
	.Maximum words	SDBAT(6)	positive integer	words/record
	.Average words	SDBAT(7)	positive integer	words/record
3	TASK DEFINITIONS			
	For each task			
	.Task identifier	STTSI	6 characters	
	.Source Language	STSOL	Character Code	Must match master Language List maintained in Technology Data Base DEALS IOPT-5 group 2
	.Instruction mix parameter k for each instruction type j	STIMX (j,k)		

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Identifier	STIMX (J,1)	1Ø character code	Must match master instruction list maintained in technology data base DEALS IOPT-5 group 2
	-Count for sizing	STIMX (J,2)	positive integer	
	-Execution counts for timing			
	Average	STIMX (J,3)	positive integer	
	Worst case	STIMX (J,4)	positive integer	

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Overlay Flag	STOLF	Character Code	
			O	Task resides on an overlay and can be stored on disc when not executing
			N	Task does not reside on an overlay
	.Data storage/ retrieval for each block referenced by task			
	-Block identifier	STDBI		

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Task data storage/ retrieval attribute k with respect to block records	STDBR (j,k)		
	.When requested	STDBR (j,1)	1 character code	
			="S"	all records processed prior to the bulk of processing
			="A"	records processed individually during execution
			="E"	records processed after the bulk of task processing
	.How requested	STDBR (2)	1 character code	
			="I"	input to task
			="U"	update by task(I/O)
			="O"	task output

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Minimum processed	ST[illegible]	Positive Integer	records/task execution
	.Maximum processed	ST[illegible]	Positive Integer	records/task execution
	.Average processed	ST[illegible]	Positive Integer	records/task execution
	-Task execution parameters			
	.Enablement type	STXET	4 character code	
			='TIME	time enabled
			='DATA'	data enabled
			='TAD'	time and data enabled
			='SLVD'	slaved

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Maximum Time Limit Per Execution	STMTL	Positive Real	Seconds
4	BASELINE PARTITIONING LOAD			
	For each representative baseline load supply			
	.Benchmark identifier	SLBMI	20 characters	
	.Partitioning total time period duration	SLBMT	Positive Real	

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.For each idependent task			
	-Identifier	SLITI	6 characters	Task identifier
	-Frequency for time enabled or slaved	SLITF (*)		
	.Average	SLITF (1)		
	.Worst Case	SLITF (2)		
	-Data Arrival rate for Data enabled	SLITD (*)		
	.Average	SLITD (1)	Non-negative Real	records per second
	.Worst Case	SLITD (2)	Non-negative	records per second
	-Maximum execute time			
	.Average	SLITT (1)	Real	milliseconds
	.Worst Case	SLITT (2)	Real	milliseconds

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
7	EVALUATION RUN IDENTIFIERS			
	.Evaluation run title	ETITL	characters	20 characters maximum for run identification
	.Computation subsystem interface requirements identifier used to label output and fetch appropriate system requirements file data	ESRID	characters	20 characters maximum, if blank assume value from file otherwise perform equality check
	.Baseline Software Application task definitions identifier used to label output and fetch appropriate software task/job definition file data	ESWID	characters	20 characters maximum if blank assume value from file otherwise perform equality check
	.Candidate architecture identifier used to label output and fetch appropriate candidate architecture file data	ECCID	characters	20 characters maximum, if blank assume value from file otherwise perform equality check

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Technology identifier used to label output and fetch appropriate technology data	ETDID	characters	20 character maximum, if blank assume value from file otherwise perform equality check
	.Benchmark Load ID	EPBMI	20 characters	Must match one of the Benchmark Load Identifiers in IOPT-2 Group 5
	.Partition ID	EPPID	Integer	
			0	Initial
			1,2,...,MPS	Specific partition from database
			MPS+1	Use highest number partition in data base

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
2	GLOBAL EVAULATION FACTORS			
	.For each predefined priority level i=1 to 3			
	-Objective level to be assigned if zero objective i is not applicable	EFPLD(i)	Positive Integer=Ø,1,2,3	
		EFPLD (1)		level for processor utilization
		EFPLD (2)		level for memory allocation
		EFPLD (3)		level for development cost

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Development cost goal	EFDCG	F10.2	Man Hours
	.Development cost upper limit	EFDCU	F10.2 EFDCU.GE. EFDCG	Man Hours
	.Basic unspecified goal percentage for memory utilization	EFMUB	0.LT.EFMUB.LT. 1.0	%Memory utilization that will provide desired storage growth balance
	.Memory utilization upper limit	EFMUU	EFMUB.LE.EFMUU .LE.1	%Utilization
	.Basic unspecified goal percentage for processor utilization	EFPUB	0.LT.EFPUB .LT. 1.0	%Processor utilization that will provide desired utilization balance
	.Processor utilization upper limit	EFPUU	EFPUU .GE.EFPUB	%Utilization
	.Processed default coefficient level	EFDCL (1)	character code ='A' ='W'	Average Worst case
	.Memory default coefficient level	EFDCL (2)	character code ='A' ='W'	Average Worst Case
	.Task default coefficient level	EFDCL (L3)	character code ='A' ='W'	Average Worst Case

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
3	SPECIFIC EVALUATION FACTORS			
	Selective goal i attributes			
	-Goal type identifier	EFCUS(1,i)	2 character code	
			='PB'	Processor utilization balance
			='MC'	Memory utilization balance
			='TC'	Task development cost
	-Component Identifier	EFCUS(2,i)	1Ø characters	Must match memory of processor component identifier in IOPT-4
	-Selective goal to be utilized for component i	EFCUS(3,i)	Ø.LT.EFCUS(3,i) .LT. 1.Ø for EFCUS(1,i)= 'PU' or 'MM'	%Utilization desired for processor and memory components
			EFCUS(3,i) .GT. Ø for EFCUS (1,i)='TC'	Manyears

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Selective upper limit goal to be utilized for component i	EFCUS(4,i)	Same as EFCUS(3,i)	
	-Selective coefficient level	EFCUS(5,i)	1 character code	
			='A'	Average
			='W'	Worst case

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
4	PARTITIONING ASSIGNMENT CONSTRAINTS			
	For each constraint			
	.Partition Map	EPMAP(*)		
	-Type restriction	EPMAP(1)	character code	
			F	Fixed allocation
			I	INitial allocation
			P	Prohibited allocation
	-Task or data Assignment Flag	EPMAP(2)	T	Task
			B	Block
	-Task or data block identifier	EPMAP(3)	6 characters	Must match baseline software task identifier if EPMAP(2) ='T'
			10 characters	Must match baseline software data block identifier if EPMAP(3) ='D'
	-Component identifier	EPMAP(5)	10 characters	Must match a candidate component (IOPT-4 Group 2) Identifier or required component (IOPT-1 group 2)

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
5	SELECTIVE COEFFICIENTS			
	-Selective Coefficient Levels may override specific technology component attribute	ESTPR (*)		
	.Component Identifier	ESTPR(1)	10 characters	Must match component identifier in IOPT-4
	.Attribute index	ESTPR(2)		
	.Coefficient Level for Attribute	ESTPR(3)	character code/ same as for EUTPR	
	.Coefficient Level selections			
	-Software load attribute level to be ESLPR used in coefficient determinations	ESLPR	Character code/ A W	 Average numbers to be applied Worst case number to be applied

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Basic unspecified technology attribute level to be used in coefficient determinations	EUTPR	Character code/ A W	 Average numbers to be applied Worst case numbers to be applied

124

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
1	CANDIDATE CONFIGURATION DEFINITION			
	Candidate identifier	CCID	characters	Maximum of 20 characters which are used to label and identify candidate architecture defined
2	CANDIDATE DEVICE DEFINITION			
	For each component device			
	.Identifier	CCCID	10 characters	Candidate Component Identifier
	.Technology type	CCCTT	2 character code	Must match an entry in the technology category codes as defined in the technology data base IOPT-5 group 2
	.Specific device identification	CCCSD	10 characters	Must match to device in technology data base IOPT-5 group 2 based upon category type

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Selected option k for component	CCCOP(k)		Supplied options are dependent upon specific component and correlate with appropriate type/ identifier options in techology data base IOPT-5 group 2

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
1	TECHNOLOGY FILE IDENTIFIER	TDBID	20 Characters	
2	MASTER TECHNOLOGY LISTS MASTER CATEGORY LIST			
	Number of current categories	TCNC	Positive Integer=14	
	For each category i=1 to TCNC			
	.Category identifier	TCCI(i)	2 character code	Tentative list includes:
	The first three are the primary areas	TCCI(1)	PU	=processor unit
		TCCI(2)	CL	=communication line (voice/data)
	required for software paritioning.	TCCI(3)	MM	=memory
	The others represent sources or destina-	TCCI(4)	CP	=cockpit instrumentation panels
	tion for processing I/O and only require	TCCI(5)	CC	=cockpit controls/switches
	as a minimum respec-	TCCI(6)	KB	=keyboard/teletype
	tive I/O transfer	TCCI(7)	DP	=display
	rates, blocking, and	TCCI(8)	MB	=motion base
	protocol interface.	TCCI(9)	GE	="g" equipment
	Category 15 labeled black box is a catch all category. Other	TCCI(10)	IC	=instructor/operator control switches

ISSUE 1 DATE 27-NOV-79 ID DEALS SEC IDPT-5 PAGE 1

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	categories can be added. The particular design evaluation environment must be considered as to what categories and level of data needs to be collected.	TCCI(11) TCCI(12) TCCI(13) TCCI(14)	CM PR CR BB	=TV Camera/Model board =Printer =Card reader =Black box
	MASTER DEVICE LIST FOR CATEGORY i			
	.Number of devices of category i currently in technology data base	TCND(i)	Non negative integer	
	.Device list (for category i) of device identifiers j=1 to TCND(i)	TCDL(i,j)	10 characters	Each entry must be unique within the list for given category i
	MASTER INSTRUCTION LIST			
	.Number of benchmark instructions	TCNI	positive integer	
	.Instruction list for instruction i=1 to TCNI	TCIL(i)	10 character code	TCIL(i) not equal TCIL(j) for i not equal j

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	MASTER BLOCK DISCIPLINES			
	.Number of disciplines	TCNBD	positive integer=7	
	.List of discipline keys for i=1 to TCNBD in alphanumeric order	TCBDL(i)	4 character codes	
		TCBDL(1)	='CBUF'	circular buffer
		TCBDL(2)	='FIFO'	queue
		TCBDL(3)	='LIFO'	stack
		TCBDL(4)	='RAN'	random I/O
		TCBDL(5)	='ROR'	read only random
		TCBDL(6)	='ROS'	read only sequential
		TCBDL(7)	='SEQ'	sequential I/O
	MASTER BLOCK TYPES			
	.Number of types	TCNBT	positive integer=5	
	.List of block type keys for i=1 to TCNBT in alphanumeric order	TCBTL(i)	1 character code	
		TCBTL(1)	='G'	global
		TCBTL(2)	='I'	instructions
		TCBTL(3)	='L'	local
		TCBTL(4)	='S'	system
		TCBTL(5)	='T'	temporary or scratch

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	MASTER DEVELOPMENT SOURCE LANGUAGES			
	.Number of languages	TCNL	Positive Integer	
	.List of language keys for i=1 to TCNL in alpha-numerical order	TCLL (i)	10 characters	
3	PROCESSOR ATTRIBUTES FOR EACH PROCESSOR P			
	Operating System Features			
	.Multitasking			
	-Levels	TPOSM(p,1)	Integer .GE. 1	Number of distinct task execution levels

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Number of priority service levels	TPOSM(p,2)	Integer .GE. Ø .LE.TPOSM(p,1)	The remaining TPOSM (p,1)-TPOSM(p,2) Levels are assumed to be service in a circular fashions
	.Enablements			
	-Maximum Time enablement frequency	TPOSM(p,3)	Integer	Enablements/second
	-Resource management per time enablement	TPOSM(p,4)	F1Ø.9 .GE. Ø	Seconds accurate to Nano seconds
	-Maximum data enablement frequency	TPOSM(p,5)	Integer	Enablements/second
	-Resource management per data enablement	TPOSM(p,6)	F1Ø.9 .GE. Ø	Seconds accurate to Nano seconds
	-Maximum slaved enablement frequency	TPOSM(p,7)	Integer	Enablements/second
	-Resource management per slaved enablement	TPOSM(p,8)	F1Ø.9 .GE. Ø	Seconds accurate to Nano seconds

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	-Resource management overhead per second per task	TPOSM(p,9)	F10.9 .GE. 0	Seconds accurate to Nano seconds
	.For each task level L=1 to TPOSM(p,2)			
	-Maximum number of tasks level L	TPOLT (p,L,1)	Integer .GE. 1	
	-Task service scheme for level L	TPOLS (p,L,2)	Code	
			='P'	Priority
			='C'	Circular
			='F'	First in first out
	.Level resource management	TPOLM (p,L,3)	F10.9 .GE.0	Seconds accurate to Nano seconds
	.List of compatible user memories			
	Simulation instruction set measurements for each benchmark instruction i			
	.Sizing measurements			
	-Number of code memories involved	TPSCM(p,i)		

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	The memory type for each code memory m (the first memory is the user task code--any other memories are predefined for this processor)	TPSMT (p,i,m)	4 character code	Must agree with master memory types defined in Group 4
	-Length of code in memory m	TPSMT (p,i,m,3)	Integer .GE.1	Number of basis units used to desirbe memory m (see Group 4)
	.Timing Measurements for each code memory m and k=1,2			k=1 implies average k=2 implies worst case
	-Number of instruction of scratch data fetch waits	TPTM (p,i,m, 1,k)	Integer .GE. Ø	
	-Number of scratch data store waits	TPTM (p,i,m, 2,k)	Integer .GE. Ø	
	-Computational total for all memories	TPTT (p,i,k)	Integer .GE. Ø	Cycles

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Application development measurements using language L of the master language list			
	-One time development charge	TPDC (p,i,1,L)	Integer	Manhours
	-Change per application instruction of this type	TPDC (p,i,2,L)	Integer	Manhours
4	COMMUNICATION LINE ATTRIBUTES			To be defined in Multiple processor communication analysis task
5	MEMORY ATTRIBUTES FOR MEMORY DEVICE M			
	Type	TMTP(m)	4 characters	
			='ROM'	Read only memory
			='RAMM'	Random access main memory
			='RRAM'	Rotating random access memory
			='SM'	Sequential memory

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
			='WCS'	Writable Control Store
	Number of different addressable units	TMNAU	Positive Integer	
	Size in bits			
	.Min	TMSZ(m,1)	positive integer	bits
	.Max	TMSZ(m,2)	positive integer	bits
	.Increments	TMSZ(m,3)	positive integer	bits
	For each addessable unit u=1 to TMNAU			
	.Level	TMAUP (m,u,1)	4 character code	
			='BIT'	bit addressable
			='6BB'	six bit byte addressable
			='8BB'	eight bit byte addressable
			='WORD'	word addressable
			='HWRD'	half word addressable
			='DBWD'	doubleword addressable

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	.Bits/unit level	TMAUP (m,u,2)	positive integer	exclusive of parity or error deletion correction bits
	.Read access time	TMAUP (m,u,3)	real	Seconds accurate to nano-seconds
	.Read cycletime per unit	TMAUP (m,u,4)	real	Seconds accurate to nano-seconds
	.Maximum sequential units transferred for single read	TMAUP (m,u,5)	positive integer	same as unit level
	.Write acces time	TMAUP (m,u,6)	real	Seconds accurate to nano-seconds
	.Write cycletime/ unit	TMAUP (m,u,7)	real	Seconds accurate to nano-seconds
	.Max sequential units for single write access	TMAUP (m,u,8)	positive integer	same as unit level
	.Error detection/ correction	TMAUP (m,u,9)	6 character code	
			='PARITY'	parity bit
			='SECDED'	single bit error correction double bit error detection

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
	Number of Suppliers for each supplier	TPMNS(m)	positive integer	
	.Identifier	TPMSP (m,s,1)	10 characters	unique identifier
	.MTBF	(TPMSP (m,s,2)	real	hours-mean time between failures
	.MTTR	TPMSP (m,s,3)	real	hours-mean time to repair
	.MSPM	TPMSP (m,s,4)	real	hours-rescheduled preventative maintenance
	.MTPM	TPMSP (m,s,5)	real	hours-mean time for preventative maintenance
6	COCKPIT INSTRUMENTATION PANEL ATTRIBUTES			
7	COCKPIT CONTROLS/SWITCHES			
8	KEYBOARD/TELETYPE ATTRIBUTES			
9	DISPLAY ATTRIBUTES			

DEALS

GRP	PARAMETER NAME	MNEMONIC	VALUES	UNITS/VALUE MEANING
10	MOTION BASE ATTRIBUTES			
11	"G" EQUIPMENT ATTRIBUTES			
12	INSTRUCTOR/OPERATOR CONTROL/SWITCH ATTRIBUTES			
13	TV CAMERA/MODEL BOARD ATTRIBUTES			
14	PRINTER ATTRIBUTES			
15	CARD READER ATTRIBUTES			
16	GENERIC BLACKBOX ATTRIBUTES			

APPENDIX B.

REPORT FORMATS

B.1 USER ECHO REPORTS

MM/DD/YY HH:MM:SS EVALUATION ________ CANDIDATE ________ PASS PAGE ___

SYSTEM ________ TECHNOLOGY ________ SOFTWARE ________

FORMAT 1. STANDARD RUN IDENTIFICATION

HARDWARE COMPONENT SUMMARY

CATEGORY/DEVICE		ID	REQUIRED
XX	XXXXXXXXXX	XXXXXXXXXX	XXX
XX	XXXXXXXXXX	XXXXXXXXXX	XXX
XX	XXXXXXXXXX	XXXXXXXXXX	XXX

*** TOTAL NUMBER OF COMPONENTS = 9999

FORMAT 2. HARDWARE COMPONENT SUMMARY

DATA BLOCK SUMMARY AND EXTERNAL SOURCE/DESTINATION

BLOCK	IDENTIFIER	LEVEL -FLAG	DISCIPLINE -FLAG	MAXIMUM RECORDS	BITS/ BYTE	BYTES/ WORD	WORDS-PER-RECORD AVE	MIN	MAX	EXTERNAL COMPONENT BASIC	SPEC	FREQUENCY
999	XXXXXXXXXX	X-XX	XXXX-XX	999999	9	99	9999	9999	9999	XXXXXX	999	999
999	XXXXXXXXXX	X-XX	XXXX-XX	999999	9	99	9999	9999	9999	XXXXXX	999	999

FORMAT 3. DATA BLOCK SUMMARY

TASK SUMMARY

TASK	IDENTIFIER	LANGUAGE	INPUT BLOCKS	OUTPUT BLOCKS	ENABLEMENT DISCIPLINE	FREQ 1	FREQ 2	FREQ 3
99	XXXXXX	XXXXXXXXXX	XXXXXXXXXX XXXXXXXXXX	XXXXXXXXXX	XXXX	9999	9999	9999
99	XXXXXX	XXXXXXXXXX	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	XXXX	9999	9999	9999

FORMAT 4. TASK SUMMARY

BASELINE LOAD SUMMARY

BASELINE LOAD XXXXXXXXXXXXXXXXXXXX
PARTITIONING TOTAL TIMEFRAME 9999999999

TASK ID	TIME/SLAVED RATE ENABLEMENTS/SECOND		DATA ENABLED RATE ENABLEMENTS/SECOND		TIME LIMIT PER EXECUTION MILLISECONDS	
	AVERAGE	MAXIMUM	AVERAGE	MINIMUM	AVERAGE	MAXIMUM
XXXXXX	9999999999	9999999999	9999999999	9999999999	9.99999999E±99	9.99999999E±99
XXXXXX	9999999999	9999999999	9999999999	9999999999	9.99999999E±99	9.99999999E±99
XXXXXX	9999999999	9999999999	9999999999	9999999999	9.99999999E±99	9.99999999E±99

FORMAT 5. BASELINE LOAD SUMMARY

EVALUATION ASSIGNMENT CONSTRAINTS

ASSIGNMENT TYPE*	COMPONENT ASSIGNED**	APPLICATION COMPONENT IDENTIFIER		CONFIGURATION COMPONENT IDENTIFIER	VALUE WHEN APPLICABLE
XXXXXXXXXX	XXXX	XXXXXXXXXX	ON	XXXXXXXXXX	XXXXXXXXXX
XXXXXXXXXX	XXXX	XXXXXXXXXX	ON	XXXXXXXXXX	XXXXXXXXXX
XXXXXXXXXX	XXXX	XXXXXXXXXX	ON	XXXXXXXXXX	XXXXXXXXXX

* {FIXED / PROHIBITED / INITIAL}

** {DATA / TASK}

FORMAT 6. EVALUATION ASSIGNMENT CONSTRAINTS

EVALUATION PRIORITIES

LEVEL	PRIORITY IDENTIFIER/ UNITS	GOAL/ TOLERANCE	COEFFICIENT LEVEL & MAX ITER	COMPONENT	GOAL	TOLERANCE PERCENT
	------------------MAJOR PRIORITIES----------------------			------------PRIORITY COMPONENTS------------		
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	C 9999	XXXXXXXXXX	99999.99	99.999
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	C 9999	XXXXXXXXXX	99999.99	99.999
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	C 9999	XXXXXXXXXX	99999.99	99.999
				XXXXXXXXXX	99999.99	99.999
				XXXXXXXXXX	99999.99	99.999
				XXXXXXXXXX	99999.99	99.999

FORMAT 7. EVALUATION PRIORITIES

BASIC PARTITIONING PROBLEM SIZE

999 TASKS . 99 PROCESSORS

999 DATA BLOCKS 99 MEMORIES

9 PRIORITIES SELECTED

FORMAT 8. BASIC PARTITIONING PROBLEM SIZE

PRIORITY GOAL SUMMARY

LEVEL	PRIORITY IDENTIFIER/ UNITS	GOAL/ TOLERANCE	CURRENT ACHIEVEMENT/ LEV/FLAG	COMPONENT	GOAL	TOLERANCE PERCENT	CURRENT ACHIEVEMENT LEVEL	FLAG
-------------------MAJOR PRIORITIES--------------------				--------------PRIORITY COMPONENTS----------------				
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	99999.99 FF	XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	99999.99 FF	XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF
9	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXX	99999.99 99.999	99999.99 FF	XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF
				XXXXXXXXXX	99999.99	99.999	9999.99	FF

FORMAT 101. PRIORITY GOAL SUMMARY

TASK ALLOCATION

TASK	PROCESSOR	EXECUTIONS	TOTAL TIME	FLAG	TASK I/O BLOCK	MEMORY	INPUT	OUTPUT
XXXXXX	XXXXXXXXXX	999/999	9.99999	FF	XXXXXX	XXXXXXXXXX	9.999999	9.999999
					XXXXXX	XXXXXXXXXX	9.999999	9.999999
					XXXXXX	XXXXXXXXXX	9.999999	9.999999
	XXXXXXXXXX	999/999	9.99999	FF	XXXXXX	XXXXXXXXXX	9.999999	9.999999
					XXXXXX	XXXXXXXXXX	9.999999	9.999999
					XXXXXX	XXXXXXXXXX	9.999999	9.999999
XXXXXX	XXXXXXXXXX	999/999	9.99999	FF	XXXXXX	XXXXXXXXXX	9.999999	9.999999
					XXXXXX	XXXXXXXXXX	9.999999	9.999999

FORMAT 102. TASK ALLOCATION

DATA BLOCK ALLOCATION

BLOCK	MEMORY	LENGTH	PERCENT	PROCESSOR	STORES	FETCHES	TOTAL	FLAG
XXXXXX	XXXXXXXXXX	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF
				XXXXXXXXXX	999999	999999	9999999	FF
				XXXXXXXXXX	999999	999999	9999999	FF
	XXXXXXXXXX	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF
XXXXXX	XXXXXXXXXX	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF

FORMAT 103. DATA BLOCK ALLOCATION

PROCESSOR ALLOCATION

PROCESSOR	TASK	EXECUTIONS	COMPUTATIONAL TIME	COMPUTATIONAL PERCENT	INPUT/OUTPUT TIME	INPUT/OUTPUT PERCENT	RESOURCE MGMT TIME	RESOURCE MGMT PERCENT	FLAG
			--------------- PROCESSOR UTILIZATION					----------------	
XXXXXXXXXX	XXXXXXXXXX	999	9.9999	99.99	9.9999	99.99	9.9999	99.99	FF
	XXXXXXXXXX	999	9.9999	99.99	9.9999	99.99	9.9999	99.99	FF
	XXXXXXXXXX	999	9.9999	99.99	9.9999	99.99	9.9999	99.99	FF
	TOTAL	99999	99.9999	99.99	9.9999	99.99	9.9999	99.99	FF

FORMAT 104. PROCESSOR ALLOCATION

MEMORY ALLOCATION

MEMORY	BLOCK	LENGTH	PERCENT	PROCESSOR	STORES	FETCHES	TOTAL	FLAG
XXXXXXXXXX	XXXXXXXXXX	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF
	XXXXXXXXXX	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF
				XXXXXXXXXX	999999	999999	9999999	FF
	**TOTAL	999999	99.99	XXXXXXXXXX	999999	999999	9999999	FF
				XXXXXXXXXX	999999	999999	9999999	FF
				**TOT PROC	9999999	9999999	99999999	FF

FORMAT 105. MEMORY ALLOCATION