N94-14639

Real-Time Dynamic Simulation of the Cassini Spacecraft Using DARTS

Part II: Parallel/Vectorized Real-Time Implementation

A. Fijany, J.A. Roberts, A. Jain, and G.K. Man
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109

1. Introduction

Part I of this paper presented the requirements for the real-time simulation of Cassini spacecraft, along with some discussion of DARTS algorithm. Here, in Part II we discuss the development and implementation of parallel/vectorized DARTS algorithm and architecture for real-time simulation. Development of the fast algorithms and architecture for real-time hardware-in-the-loop simulation of spacecraft dynamics is motivated by the fact that it represents a *hard real-time problem*, in the sense that the correctness of the simulation depends on both the numerical accuracy and the exact timing of the computation. For a given model fidelity, the computation should be completed within a predefined time period. Further reduction in computation time allows increasing the fidelity of the model (i.e., inclusion of more flexible modes) and the integration routine.

An analysis based on the computational structure of DARTS and the specific dynamic model of the spacecraft is made to determine efficient algorithmic/ architectural techniques for achieving real-time simulation capability. This analysis indicates that a combined parallel/vector algorithmic technique along with a multiple vector processors architecture represents the most efficient and cost-effective approach.

The most important (and the new) issue in this paper is the development of the vectorized algorithms for spacecraft dynamic simulation. Until recently, only the users of vector supercomputers for non-real-time applications were concerned about the vectorization issue. Usually, the vectorization was limited to the use of the automatic vectorizers, provided by the vector supercomputers vendors, using an already developed software code. This represents a suboptimal use of the vector supercomputers computing power since the automatic vectorizers have a very limited capability and are efficient only for low level vectorization. For most problems, a significant speedup can be achieved by developing a new algorithm or restructuring the old algorithm by global vectorization of the computation. However, due to the non-real-time nature of the applications, the fact that the vector supercomputers even in scalar mode (for serial computation) were faster than any other serial processor, the time and effort required for the development of new vectorized algorithms and software codes, the users were, most often, satisfied by the suboptimal performance. As a result, the development of the vectorized algorithms has been studied for very few and mostly regular problems, e.g., matrix-vector operations, direct and indirect (iterative) linear system solution, etc. To our knowledge, the vectorization of multibody dynamics has been only recently studied for a rather simple case of a serial chain of rigid multibody (a robot manipulator) on Cray supercomputers [5,6].

However, this situation is rapidly changing and more effort is being made on the analysis and design of vectorized algorithms and software. This is motivated by the advent of a new generation of low-cost single-board vector processors with computational powers previously offered by the vector supercomputers. These new vector processors provide, for the first time, the opportunity for the design and implementation of high performance embedded computer architecture for real-time applications. However, in order to meet the real-time constraint, efficient vectorized algorithms need to be developed for exploitation of computing power of these new vector processors.

The hardware and software considered in Part II of this paper represent the first generation of such low-cost vector supercomputers. As such, they lacked some important features for efficient vectorization and parallelization. However, since the development of this work, significant improvement has been made on hardware and software of new generations. The approach developed in Part II of this paper and the lessons learned through practical hardware and software implementation, along with these advanced new generations of multiple vector processors, indicate that the real-time simulation capability for more complex systems such as the Space Station is now achievable.

Part II of this paper is organized as follows: Section 2 reviews the different algorithmic and architectural techniques for fast implementation of DARTS, and discusses the features of selected target architecture; Section 3 discusses techniques for global vectorization and efficiency of vector algorithms; Section 4 discusses some implementation issues and several aspects of the implemented algorithms through examples; and finally, Section 5 contains some discussion and concluding remarks.

2. Algorithm and Architecture Selection For Real-Time Simulation

A. AN ANALYSIS OF ALGORITHMIC / ARCHITECTURAL TECHNIQUES FOR FAST IMPLEMENTATION OF DARTS

Generally, there are three algorithmic/architectural techniques that can be used to speed up the computation of a given problem: symbolic manipulation, parallelization, and vectorization. The choice of one or a combination of these techniques depends on: (1) the structure of computational problem, and (2) the availability and cost effectiveness of the required computer architecture.

Symbolic manipulation is a rather straightforward technique that is widely used in multibody dynamics community (see, for example, [12]). Using this technique, a greater computational efficiency can be achieved by eliminating the redundant operations and

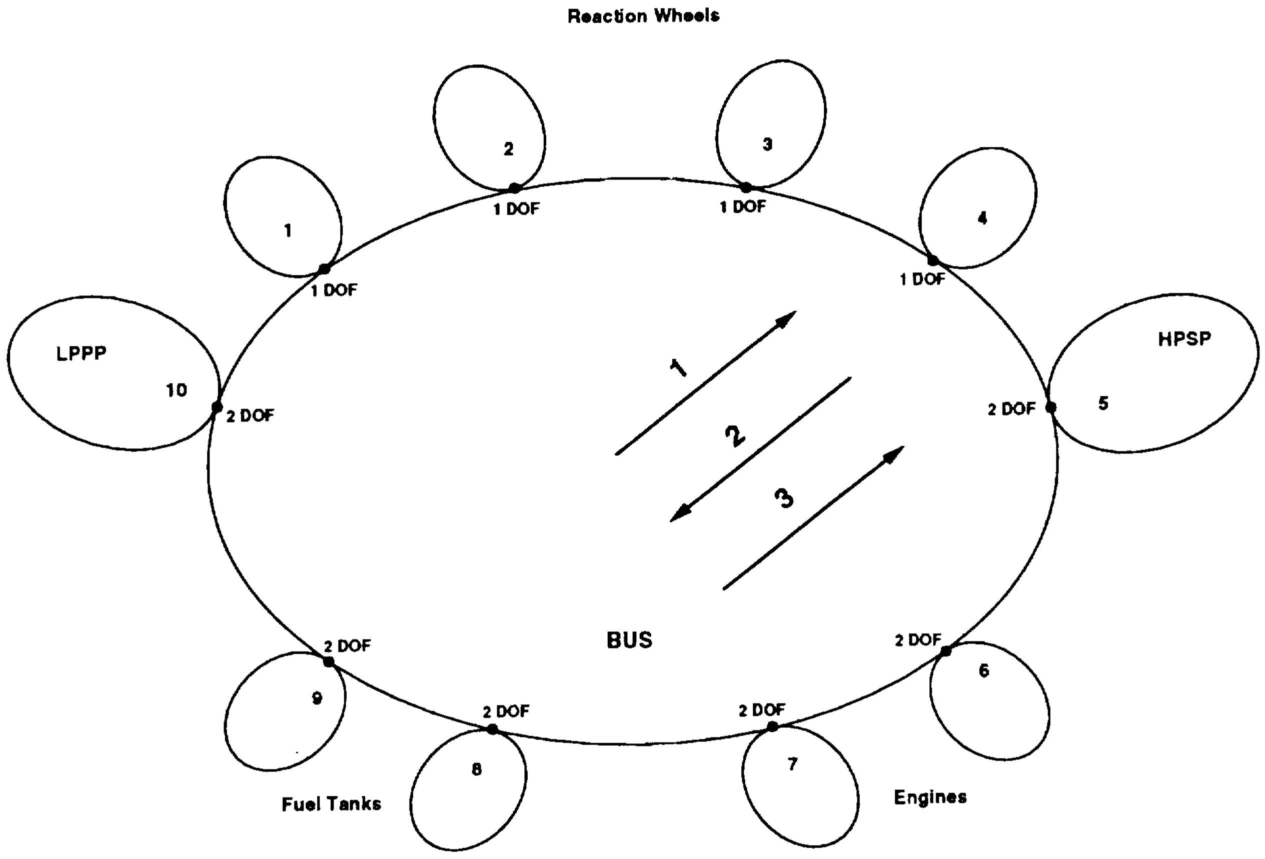


Figure 1. Cassini Spacecraft Star-Topology Dynamics Model and Computational Steps of DARTS

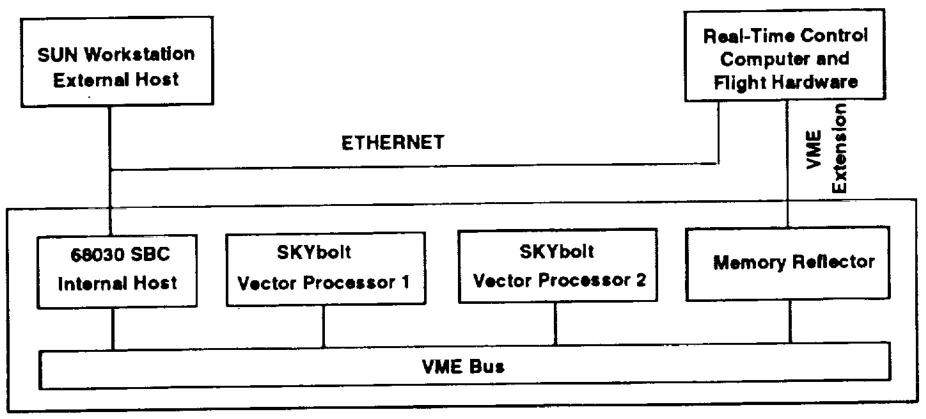

SUN Workstation External Host
Real-Time Control Computer and Flight Hardware
ETHERNET
VME Extension
68030 SBC Internal Host
SKYbolt Vector Processor 1
SKYbolt Vector Processor 2
Memory Reflector
VME Bus

Figure 2. Dedicated Parallel/Vector Computer Architecture for Real-Time Dynamic Simulation of Cassini Spacecraft

generating the symbolic expressions for the equations. However, two issues regarding the application of the symbolic manipulation technique need to be considered. First, the speedup due to the symbolic manipulation should be analyzed in a relative context and not as an absolute one. That is, if the original algorithm has a compact, efficient, and recursive structure–which is the case for DARTS–then the use of symbolic manipulation will not result in a noticeable speed-up. Second, the evaluation of the symbolic expressions is a strictly serial computation. Hence, if symbolic manipulation is used, then it would be difficult to further reduce the computation time by parallelization and/or vectorization. In this case, the only way to reduce the computation time is to use a faster serial processor.

However, both the structure of DARTS and the specific model (star topology and flexible bus) of the Cassini spacecraft make the computation highly suitable for parallelization and/or vectorization. For parallel computation, at first glance it may seem that the computation can be fully parallelized by assigning one processor per body. However, as discussed below, this will lead to a limited speed-up. For vector computation, a large part of the computation can be described in terms of two basic operations: scatter and gather operations, which are highly suitable for vectorization since they involve operations on large matrices and vectors. Furthermore, the size of matrices and vectors increases with both the number of flexible modes and the number of appendages. In order to better assess the suitability of the computation for parallel and/or vector computation and analyze the resulting algorithmic/architectural trade-offs, a more careful study of the structure of the computation is needed. Note that in this study we are only interested in the coarse grain parallelism since it can be exploited by low-cost, commercially available, multiprocessor architecture.

The basic computational steps of the DARTS for the Cassini spacecraft (Figure 1) can be summarized as follows (see [1,3] for a more detailed discussion):

Step I:

1. **Propagate the linear and angular velocity from the bus to appendages.**

   This step is suitable for both parallelization and vectorization. It can be done in parallel for all appendages. It also represents a scatter operation and can be done by performing a single, large matrix-matrix multiplication (see Section 4).

2. **Compute the gyroscopic accelerations and forces of the appendages.**

   This computation is more suitable for parallelization since it can be done in parallel for all appendages. It involves matrix-vector operations with rather small vectors and matrices which makes it less efficient for vectorization (see also Section 4).

Step II:

1. **Propagate Articulated-Body Inertia from appendages to the bus and compute the Articulated-Body Inertia of the bus.**

   The propagation of the Articulated-Body Inertia from appendages to the bus can be performed in parallel for all appendages. But the computation of the Articulated-Body Inertia of the bus remains serial and also involves many-to-one type of interprocessor communication. However, both the propagation of the Articulated-Body Inertia from appendages and the computation of Articulated-Body Inertia of the bus can be described in terms of gather operations, which involve matrix-matrix multiplications with very large matrices and, hence, they are highly efficient for vector computation.

2. **Propagate the residual forces from appendages to the bus and compute the effective residual forces of the bus.**

   Again, the propagation of the residual forces from appendages to the bus can be performed in parallel for all appendages. But the computation of residual force of the bus remains serial and also involves many-to-one type of inter-processor communication. However, both the propagation of the residual forces and the computation of residual force of the bus can be described in terms of gather operations, which involve matrix-vector multiplications of large matrices and vectors and, hence, are highly efficient for vector computation.

Step III:

1. **Compute the acceleration of bus.**

   The computation of acceleration of bus involves the solution of a symmetric, positive definite, linear system which is more suitable for vectorization than for coarse grain parallelization.

2. **Propagate acceleration of bus to appendage.**

   As in Step I.1, this propagation can be performed in parallel but it also involves one type to many types of interprocessor communication. It also represents a scatter operation and can be done by performing a single, large matrix-vector operation.

3. **Compute hinges acceleration.**

   Similar to Step I.2, this computation is more suitable for parallelization since it can be done in parallel for all hinges. It involves matrix-vector operations with rather small vectors and matrices which makes it less efficient for vectorization.

The above analysis clearly suggests that the computation of DARTS for the Cassini spacecraft can be speeded up by both parallelization and vectorization. Furthermore, a combined parallelization/vectorization algorithmic approach can lead to a speed-up greater than that achievable by either parallelization or vectorization alone. This combined algorithmic approach is further motivated by the emergence of low-cost multiprocessor architectures that employ vector processors, such as Intel i860, as the node processor.

There are, however, two issues that need to be considered in applying this combined algorithmic approach, which also can affect the choice of an optimal target architecture for its implementation. The first issue is that a limited speed-up can be achieved by assigning one processor per body since the ratio of fully parallelizable computations over strictly serial computations isn't very large. This is mainly due to the specific model of Cassini spacecraft; that is, rigidity of the appendages and high flexibility of the bus. In fact, most of the fully parallelizable parts of the algorithm involve the computations for the rigid appendages; e.g., Steps I.2 and III.3, which are less intensive than the strictly serial parts that involve the computations for flexible bus, and also the computation of Articulated-Body Inertia (Step II.1) and acceleration (in Step II.1) of the bus.

Another important factor for efficient parallelization of the computation is the processors interconnection. As stated before, the parallelization of DARTS for Cassini spacecraft involves many-to-one and one-to-many types of processors communication. Therefore, without an interconnection structure that can handle fast processors communication of these types, the communication overhead can degrade the achievable speed-up.

The second issue is that there is a trade-off between the degree of parallelization (i.e., number of processors), and the degree of vectorization. To see this, let us consider those steps that are suitable for both parallelization and vectorization (Steps I.1, II.2, III.2, etc.). For example, in Step I.1., with one processor per appendage, the propagation can still be done by performing matrix-vector operations with small matrices. However, if the number of processors is reduced–which also reduces the speedup due to the parallelization–then the propagation for more than one appendage is done by each processor which implies that the size of matrices and hence the speedup due to vectorization increases.

B. THE CHOICE OF TARGET ARCHITECTURE

Based on the above analysis and given the possible options on the commercially available low-cost multiprocessor architectures (at the time of this project) and other constraints on cost, hardware and software development time and effort, we chose a two-vector processors architecture [2]. This choice was based on our conclusion that, in order to speed up the computation, it was more efficient (both from an algorithmic and architectural point of view) to exploit a limited parallelism but attempt to exploit maximum vectorization.

Figure 2 shows the dynamic simulation system [2]. It consists of a SUN workstation and a VME subsystem. The SUN workstation is the host of system, which is used for software development, and is interconnected through ETHERNET to the VME subsystem. The VME subsystem includes a general-purpose single-board computer based on 68030 processor, which is the local host of VME subsystem, two single board vector processors, and a memory reflector board for high speed interface with another VME system, the real-time control computer.

Each vector processor is a SKYbolt VME bus compatible board with an i860 as the vector processor and an i960 as the communication processor. The choice of the SKYbolt over other commercially available i860-based boards was mainly due to a faster main memory [2]. The SKYbolt was the only one that provided a SRAM main memory while the others had a DRAM main memory with a memory bandwidth of half of that of SRAM main memory. As will be discussed in Section 5, our practical implementation showed that the choice of the SRAM main memory resulted in a very decisive factor in meeting the real-time constraint.

The VME compatibility was basically required for the purpose of integration with and the interface to the rest of the spacecraft hardware-in-the-loop simulation hardware. Based on the vendor's specification, the SKYbolt board provided three communication channels through the VME port, VSB port, and AUX (a fast and private I/O) port [13]. Therefore, it was originally assumed that the communication between the two SKYbolt boards would be performed by using the fast AUX port. However, neither AUX port nor VSB port were functional at the time of our implementation. This forced us to use the VME bus as the communication bus between the two SKYbolt boards. However, the VME interface chips on the SKYbolts were not fully functional. This resulted in a significant loss in the communication speed between the two SKYbolts compared to the nominal speed of the VME bus. As a result, our system was highly imbalanced for parallel computation since the processors' computation speed (particularly in full vector mode) was much greater than the bus' communication speed. This implied that only very coarse grain parallelism with minimum communication requirement could be efficiently exploited by the system. Note that, even by using a fully functional AUX channel the system would have still remained imbalanced. This clearly indicates that, without using an extremely fast communication structure, efficient parallel computation with multiple vector processors such as i860 would not be possible (see Section 5).

3. Vectorization Strategy

The SKYbolt can be used as an accelerator, i.e., simply as a fast serial processor, to speed up the serial computation. According to the vendor's claim, in accelerator mode the SKYbolt can provide a speed-up of about 2 over the SUN Sparc II. Our double precision implementation of serial DARTS algorithm on both the SUN Sparc II and the SKYbolt showed a similar speedup (Table I). This result indicated that, in order to meet the real-time constraint, a greater speedup through vectorization of the algorithm was needed.

The i860 has *peak computational power* of 80 and 64 MFLOPS for single- and double-precision computation. It has most of the functional units of vector supercomputers. However, the vector supercomputers, such as the Cray series, in addition to a fast vector processing unit, also have a fast (usually the fastest available) scalar processor for serial computation [8,10]. For i860, both vector and serial parts of the computation are performed by the same units. As a result, the i860 has a poor ratio of speed of serial over vector computation because the speed of serial computation (as can be seen by the above comparison with SUN Sparc II) is much less than that of vector computation. Thus, a higher degree of vectorization, even more than that for vector supercomputers, is required for i860 to achieve a satisfactory *sustained computation power.*

The SKYbolt also provides a software environment almost similar to that offered by the vector supercomputers. However, the i860 and the SKYbolt represent the first generation of such low-cost vector processors and, therefore, lack many important features needed for efficient vector computation (see Section 5). Nevertheless, the strategy for vectorization on the SKYbolt is basically similar to that for other vector processors. In the following, we briefly discuss some of the key issues that have been considered in the design, analysis, and implementation of the vectorized version of a DARTS algorithm.

A. LOCAL AND GLOBAL VECTORIZATION TECHNIQUES

There are two techniques for vectorization of a given computation [9,10]: local and global vectorization. The SKYbolt, like vector supercomputers, provides two tools for local vectorization. The first tool is a library of highly optimized routines for matrix-vector and other operations that can be used by subroutine calls. The second tool is an automatic vectorizer that analyzes the data dependency and then vectorizes the computation of innermost Do-loops (i.e., scalar loops) of the overall computation [7,8-10]. Another widely used technique not in the above computation is chaining of the operations [8-10].

However, if the matrices and vectors are small, then the use of optimized routines does not significantly increase the performance. Also, if a code is already developed and optimized for serial computation, then it may have strong data dependency in which even the most advanced automatic vectorizers cannot vectorize. For most problems a greater speedup can be achieved by recasting the algorithm in a form suitable for vector computation, i.e., by a global vectorization. This is more difficult than the local vectorization and can be only done by the algorithm designer [9,10] as it may require major restructuring of the data and computation of the algorithm. In Section 4, we discuss some examples of such global vectorization. It should be also mentioned that our practical implementation indicated that even efficient use of library routines may require restructuring of the computation. There are not well-defined techniques for global vectorization and it is indeed highly problem dependent [9]. Nevertheless, there are several key issues regarding efficient vectorization that need to be taken into account in the design and analysis of vectorized algorithms. These issues are briefly reviewed here. The reader is referred to [7, 9-11] for a more detailed discussion.

B. THE EFFICIENCY OF VECTOR ALGORITHMS

The speedup of vectorized algorithms, like parallel algorithms, is measured according to the Amdahl's Law.

Let f represent the vectorized fraction of the computation, k the speed of vector operations relative to the speed of scalar operations, and SP the speedup of the vectorized algorithm over serial algorithm. It follows then that

$$SP = \frac{1}{(1-f) + f/k} \tag{1}$$

In order to increase the speedup, f and k should be increased. k is a function of the size of vectors and matrices as well as the type of matrix-vector operation. The computation time, T, of a vector operation is given as:

$$T = \tau + nt \tag{2}$$

where n and t stand for the size of the vectors and the clock time of the vector processor. τ represents the overhead of vector operation due to the loop setup, load and store operations, etc. If n is large enough so $nt >> \tau$ then the computation of vector operation is dominated by nt. That is, k is maximized and the vector processor performs one operation per clock cycle.

There is a vector size below which vector computation becomes less efficient than scalar computation. This size is called breakeven point [7] and is designated as n_B. The value of n_B depends on the type of operation. There is no information on n for the i860. Although originally we suspected n_B to be rather small [3], in practical implementation and for various matrix-vector operations we found n_B to be quite large (several times that for Cray series), which indicates that only operations on very large matrices and vectors can be efficiently implemented on the i860.

As a conclusion, in vectorizing the algorithm an attempt should be made to:

(1) increase the number of matrix-vector operations, and hence increase f; and

(2) increase the size of the vectors and matrices, n, so that $n >> n_B$, and hence increase k.

C. MEMORY BANDWIDTH AND DATA ORGANIZATION

For vector processing, the data movement may sometimes take more time than the computation (see the example in Section 4). Therefore, the second issue in analyzing the performance of vector supercomputers is the data structure. To efficiently use the high speed floating-point units, data should be fed with adequate speed. In the pipelined mode, the i860 can initiate two floating-point operations (one add and one multiply) per clock. This requires fetching four operands and storing two results per clock which indeed requires a very high bandwidth memory. To achieve such a high bandwidth, the vector supercomputers use a hierarchical memory organization. However, in addition to the memory organization, the data structuring is also needed for achieving and maintaining the high bandwidth. For example, while the i860 can perform two floating-point operations per clock cycle, fetching an operand from an arbitrary location in the main memory can take several clock cycles.

The memory organization of vector supercomputers usually includes a set of registers, as a fast and limited size memory; a cache memory, as medium-size medium-speed memory; and a main memory, as a large and slow memory. The i860 has a set of thirty two 32-bit data registers and 8 Kbyte (1 K double-precision) data cache. The selected SKYbolt board provides a 2-Mbyte fast SRAM memory as the main memory. Unlike the register-oriented vector supercomputers, such as Cray series, which utilize a larger size register (in the order of Kbyte), the i860 has a rather small size set of registers. However, it is claimed that the cache memory can be used with the same performance as the registers for vector operations [4].

To minimize the data movement overhead, the following issues need to be considered:

1. Data Contiguity:

   The related data should be located, as much as possible, in the contiguous locations in the cache and main memory. Obviously for vector operation the elements of the vectors (and matrices) should be stored in contiguous locations, i.e., with unit vector stride. The vector instructions that access memory have a known pattern and if the elements of vectors (matrices) are all adjacent, then the maximum speed in data access is achieved by pipelining.

2. Data Locality:

   Given the slow speed of the main memory, the access to the main memory should be minimized. This implies that the intermediate data should be kept in the registers and cache memory. Also, once data is fetched from the main memory and loaded into the cache, all of the operations that require the data should be performed before the data is returned to the main memory, i.e., the vector touch should be minimized. Given the limited size of the cache, this may even require reordering the computation.

As a conclusion on the design of efficient vector algorithms, we would like to quote from [11, p. 47], "*We have shown that the efficiency of a vector- pipeline matrix computation depends upon the vector length, the vector stride, the vector touch, and the data re-use properties of the algorithm. Optimizing with respect to all these attributes is very complicated and something of an art. A good compiler can of course do some of the thinking for us, but do not count on it!*"

Note that in [11] general matrix computations are considered which are much simpler than a rather complex algorithm such as DARTS.

4. Development and Implementation of Vectorized / Parallel DARTS

Based on the analysis of Sections 2 and 3, we first developed a parallel/vectorized version of DARTS [3]. However, the practical implementation of this algorithm resulted in an *interactive vectorization* process. Detailed timing was used to measure the computation time of each subroutine and the overall computation. The data structure and operations of the algorithm were then constantly modified to minimize the computation time. As a result, the final implementation was different from the original algorithm in [3]. Two issues made these modifications necessary. First, the algorithm in [3] was based on general and theoretical assumptions regarding vector processing. Given the fact that this was our first experimentation, many lessons were learned on detailed practical issues through actual implementation. The second, and more important, issue was due to the shortcomings of both hardware and software of the SKYbolt. Some of the necessary routines either were not provided or were not functional. Also, no means was provided to control the cache memory (see also Section 5). As a result, we were forced to develop our own subroutines or to change the computation. Here, we discuss some of the implementation issues. Due to the lack of space, only a few representative examples are given.

A. SCATTER OPERATIONS: VELOCITY PROPAGATION

The propagation of velocities is a simple, but representative, example that shows how the topology of the spacecraft allows efficient global vectorization of computation, which follows m and n that stand for the number of bus flexible mode and the number of appendages. Here, the main computation is the evaluation of the deformation variables for all the appendages:

For $i = 1$ to n,

$$\delta_r(i) = \sum_{j=1}^{m} \lambda_{ij}\eta_j = \lambda(i)\eta \tag{3}$$

$$\delta_\ell(i) = \sum_{j=1}^{m} \gamma_{ij}\eta_j = \gamma(i)\eta \tag{4}$$

$$\delta_\omega(i) = \sum_{j=1}^{m} \lambda_{ij}\dot{\eta}_j = \lambda(i)\dot{\eta} \tag{5}$$

$$\delta_v(i) = \sum_{j=1}^{m} \gamma_{ij}\dot{\eta}_j = \gamma(i)\dot{\eta} \tag{6}$$

where $\eta \overset{\Delta}{=} \mathrm{col}\{\eta_j\}$ and $\dot{\eta} \overset{\Delta}{=} \mathrm{col}\{\dot{\eta}\} \varepsilon \Re^{m\times 1}$ are the vectors of modal deformation coordinates of the bus. λ_{ij} and $\gamma_{ij} \varepsilon \Re^{3\times 1}$ are the rotational and translational displacement vectors of the jth mode for the ith appendage, $\lambda(i) \overset{\Delta}{=} \mathrm{row}\{\lambda_{ij}\}$ and $\gamma(i) \overset{\Delta}{=} \mathrm{row}\{\gamma_{ij}\} \varepsilon \Re^{3\times m}$. $\delta_r(i), \delta_\ell(i), \delta_\omega(i)$, and $\delta_v(i) \varepsilon \Re^{3\times 1}$ are the translational and rotational deformation and the linear and angular deformation velocities of appendage i. Due to the star topology of the spacecraft, the propagation for all appendages can be done simultaneously, i.e., the computation in Eqs. (3)-(6) can be done in parallel for all $i = 1$ to n.

For serial computation, the two forms in Eqs. (3)-(6) have the same cost while the second form is more efficient for vector computation. This efficiency for vectorization can be further increased as follows. Define

$$\hat{\eta} = \{\eta \quad \dot{\eta}\} \varepsilon \Re^{m\times 2}; \ \lambda \overset{\Delta}{=} \mathrm{col}\{\lambda(i)\} \text{ and } \gamma \overset{\Delta}{=} \mathrm{col}\{\gamma(i)\} \varepsilon \Re^{3n\times m}; \ \hat{\Pi} \overset{\Delta}{=} \begin{Bmatrix} \lambda \\ \gamma \end{Bmatrix} \varepsilon \Re^{6n\times m};$$

$$\delta_r \overset{\Delta}{=} \mathrm{col}\{\delta_r(i)\}, \delta_\ell = \mathrm{col}\{\delta_\ell(i)\}, \delta_\omega = \mathrm{col}\{\delta_\omega(i)\}, \text{ and } \delta_v = \mathrm{col}\{\delta_v(i)\} \varepsilon \Re^{3n\times 1};$$

$$\delta_{r\ell} = \begin{pmatrix} \delta_r \\ \delta_\ell \end{pmatrix} \text{ and } \delta_{\omega v} = \begin{pmatrix} \delta_\omega \\ \delta_v \end{pmatrix} \varepsilon \Re^{6n\times 1}; \text{ and } \delta = \{\delta_{\omega v} \delta_{r\ell}\} \varepsilon \Re^{6n\times 2}.$$

The computation in Eqs. (3)-(6) can then be performed by a simple matrix-matrix multiplication as:

$$\delta = \hat{\Pi}\hat{\eta} \tag{7}$$

Note that the matrix $\hat{\Pi}$ is constant and can be precomputed. Also, the above computation results in a certain arrangement of the vectors $\delta_r(i), \delta_\ell(i), \delta_\omega(i)$, and $\delta_v(i)$, which affects the rest of the computation and should be taken into account. However, because of the structure of the matrix $\hat{\Pi}$ and the vector $\hat{\eta}$ is not efficient to use, the regular matrix-matrix multiplication routine is based on the vector-dot operation (see below).

B. SCATTER AND GATHER OPERATIONS: FORCE AND ACCELERATION PROPAGATION

Using similar technique as for the velocity propagation, the propagation of force and acceleration can be globally vectorized and represented in terms of large matrix-vector multiplications as [3]:

$$Z(B) = \begin{pmatrix} \Pi^* \\ \phi \end{pmatrix} Z_1^+ + K = (\Pi^*\phi) Z_1^+ + K \tag{8}$$

$$\alpha_1^+ = (\Pi\phi^*)\,\alpha(B) = (\Pi\phi^*)\alpha(B) \tag{9}$$

where $Z(B)$ and $\alpha(B) \varepsilon \Re^{(m+6)\times 1}$ are the vectors of residual force and acceleration of the bus. $Z_1^+(i)$ and $\alpha_1^+(i) \varepsilon \Re^{6\times 1}$ are the residual force and acceleration of appendage i, and $Z_1^+ \overset{\Delta}{=} \mathrm{col}\ \{Z_1^+(i)\}$ and $\alpha_1^+ \overset{\Delta}{=} \mathrm{col}\ \{\alpha_1^+(i)\} \varepsilon \Re^{6n\times 1}$. $\Pi \varepsilon \Re^{6n\times m}$ is an appropriate combination of $\lambda(i)$ and $\gamma(i)$ and can be precomputed. $\phi \varepsilon \Re^{6\times 6n}$ is a sparse matrix that needs to be formed in real time. In Eqs. (8)-(9), $\Pi^*\phi \varepsilon \Re^{(m+6)\times 6n}$ and $\Pi\phi^* \varepsilon \Re^{6n\times(m+6)}$.

The matrix-vector multiplication routine provided by the SKYbolts (and other vector supercomputers) is based on the vector-dot operation. Consider a matrix-vector multiplication as $V = MU$ where M is a $P \times Q$ matrix and let $M^{(i)}$ and $M_{(i)}$ denote the rows and columns of matrix M, respectively. The vector-dot based routine is given:

For $i = 1$ to P,

$$V(i) = M^{(i)} \cdot U \tag{10}$$

However, another possible algorithm for matrix-vector multiplication is based on the SAXPY (scalar-vector multiply plus vector) operation [11]:

For $i = 1$ to Q,

$$V^i = V^{i-1} + M_{(i)} U(i) \tag{11}$$

Both the vector-dot and SAXPY operations are highly suitable for vector computation. The operation in Eq. (10) requires P vector-dot operations on vectors of dimension Q while that in Eq. (11) requires Q SAXPY operations on vectors of dimension P. Based on our discussion in Section 3.B, it then follows that the $P < Q$, the vector-dot based routine, and the $P > Q$, the SAXPY based routine, are more efficient.

For our implementation, the values of n and m were $n = 13$ and $m = 10$. Thus, the vector-dot routine is highly optimal for matrix-vector multiplication in Eq. (8) because it requires 16 vector-dot operations on vectors of dimension 78. However, it is highly inefficient for Eq. (9) as it requires 78 vector-dot operations on vectors of dimension 16. If the SAXPY based routine is used for Eq. (9), then it requires only 16 SAXPY operations on the vectors of dimension 78.

The C language was used for the development of our vectorized code, which implied that the matrices are stored by rows. However, for efficient implementation of the SAXPY based routine, matrices need to be stored and fetched by columns. For Eq. (9), the need for transposing the matrix $\Pi\phi^*$ can be simply eliminated by rewriting it as:

$$(\alpha_1^+)^* = (\alpha(B))^*(\Pi\phi^*)^* = (\alpha(B))^*(\Pi^*\phi) \tag{12}$$

Another advantage of Eq. (12) is that for both Eqs. (8) and (12), only the matrix $\Pi^*\phi$ needs to be formed.

Our SAXPY based routine, though developed in C language, significantly increased the computational efficiency and was used very frequently. Obviously, if this routine is provided by the vendors and developed in assembly language, it can offer an even greater computational efficiency. Note that, for the matrix-matrix multiplication in Eq. (7), we also used a SAXPY based matrix-matrix multiplication routine that is more efficient than the vector-dot based routine.

C. COMPUTATION OF ARTICULATED-BODY INERTIA AND ACCELERATION OF BUS

The computation of the articulated-body inertia, $P(B)\varepsilon\Re^{(m+6)\times(m+6)}$, and acceleration, $\alpha(B)$, of the bus represents the major computation-intensive parts of the vectorized algorithm (over 30% of the total computation time). As stated before, the computation of $P(B)$ represents a gather operation and was globally vectorized in a similar fashion as the computation of $Z(B)$. However, significant reduction (more than 50%) in computation time was achieved by several changes in the data structure and the type of operations to find the most optimal way for computation of $P(B)$. In the final form, the symmetry of $P(B)$ was exploited and only the diagonal and lower triangular parts of $P(B)$ were computed. $\alpha(B)$ is computed as the solution of the system.

$$P(B)\alpha(B) = \varepsilon(B) \tag{13}$$

We first used a Cholesky-based routine provided by the SKYbolt's library for the solution of the symmetric, positive definite, system in Eq. (13). Later, we developed a routine based on the LDL^* decomposition [11] that did not require square-root operation. Although our routine was developed in C and was not vectorized, it was significantly faster than the SKYbolt's routine. However, the main motivation for and the advantage of this routine was that, given the way the matrix $P(B)$ was computed, it could easily be used for solution of Eq. (13) without any need for data movement. Again, if this routine is developed by the vendor in assembly language and in fully vectorized form, it can offer an even greater computational efficiency.

D. DATA MOVEMENT MINIMIZATION

Major improvement in the efficiency of the vectorized algorithm was achieved by minimizing the data movement overhead through modification of the data structure and operations of the algorithm. Here, a few examples are discussed.

1. Matrix-Matrix Multiplication

The computation of vectorized DARTS involves many matrix-matrix multiplications as $A = BC$ for both small and very large matrices. A vector-dot based matrix-matrix multiplication routine requires that first the matrix C be transposed. However, if matrix C is symmetric, then it does not need to be transposed. The SKYbolts provided two matrix-matrix multiplication routines for the general (nonsymmetric matrix) case and the special case (symmetric matrix). However, even for the general case, whenever possible we eliminated the need to transpose the matrix C by either forming C^* (if it could be precomputed) or directly computing C^*.

Another frequently used operation was chained matrix-matrix multiplication, as $A = BCB^*$, for both small and very large matrices and with C being a symmetric matrix. For example, this type of matrix-matrix multiplication occurs in projection of mass matrices of the appendages onto the bus frame or in computation of $P(B)$. This operation can be performed without any need for matrix transposition by simply rewriting it as $A = B(BC)^*$. This simple modification resulted in a significant reduction of the data movement overhead particularly for computation of $P(B)$ wherein the matrices involved in the computation were very large.

2. Vector Touch Minimization

One of the widely used operations in the vector processing is a GAXPY (matrix-vector multiply plus vector) operation [11] as $V_1 = MV_2 + V_3$ wherein M is a matrix and V_1, V_2, and V_3 are vectors. In addition to the computational efficiency, a GAXPY routine reduces the vector touch since the vector $V_4 = MV_2$ does not need to be explicitly computed, stored, and reloaded. However, the SKYbolt's library did not provide such a routine and we had to develop our own routine. Several other routines were also developed for other operations with the purpose of minimizing the vector touch.

3. Data Structure Modification

Our major effort in reducing the data movement overhead was based on modifying the data structure of the algorithm to find the most optimal form. Here, we give a simple example that underlines the importance of the data movement overhead minimization. The computation times of evaluating angular, a_{wi}, and linear, a_{vi}, gyroscopic acceleration of appendages for $i = 1$ to n, were measured as 137 μs and 121 μs. For the rest of the computation, it was then required to merge the vectors a_{wi} and a_{vi} and form a vector $a_i = \begin{Bmatrix} a_{wi} \\ a_{vi} \end{Bmatrix}$. However, it took 143 μs to form the vectors a_i for $i = 1$ to n which was greater than the computation time for either a_{wi} or a_{vi}. The algorithm was then modified to directly compute and form the vectors a_i without any data movement. This simple example clearly shows that for vector processing the data movement time can be even greater than the computation time.

E. GLOBAL VECTORIZATION OF SMALL MATRIX-VECTOR OPERATION LOOPS

A rather significant part of the DARTS algorithm, which seemed to be unvectorizable, involved many Do-loops with small vectors and matrices. An example of such frequently occurring Do-loops is:

For $i = 1$ to n,

$$V_{1i} = M_i V_{2i} + V_{3i} \tag{14}$$

where V_{1i}, V_{2i}, and V_{3i} are 3×1 vectors and M_i is 3×3 matrix. Due to the small dimension of vectors and matrix, it is more efficient to use scalar (serial) routines for such Do-loops. However, we developed a technique for global vectorization of such Do-loops.

Algorithm	Computation Time (in ms)	Speedup
Serial DARTS on SUN SPARC II	24.39 ms	-- (Reference Time)
Serial DARTS on 1 SKY Bolts	12.37 ms	2 Faster Hardware
Vectorized DARTS on one SKY Bolt	7.29 ms	1.7 Vectorization
Parallel/Vectorized DARTS on two SKY Bolts	4.82 ms	1.5 Parallelization & Vectorization

Table I. Comparison of different algorithms/architecture computational efficiency

To see this, let us define

$$V_1 \triangleq \text{ col } \{V_{1i}\}, V_2 \triangleq \text{ col } \{V_{2i}\}, V_3 \triangleq \text{ col } \{V_{3i}\} \varepsilon \Re^{3n \times 1}, \text{ and } M \triangleq \text{ diag } \{M_i\} \varepsilon \Re^{3n \times 3n}$$

The above loop can then be replaced by a single matrix-vector multiplication:

$$V_1 = M V_2 + V_3 \tag{15}$$

Of course, due to the sparse structure of matrix M, it is highly inefficient to compute Eq. (15) by performing a general matrix-vector multiplication. However, the matrix M is a banded matrix with the nonzero elements only on its five leading diagonals. The computation in Eq. (15) can be efficiently done by performing the matrix-vector multiplication by diagonals. To see this, let M^j, $j = -2$ to 2, denote the diagonals of matrix M, where M^0 is the main diagonal.

The computation in Eq. (15) can then be done:

Set $V_1^{-3} = V_3$

For $j = -2$ to 2,

$$V_1^j = M^j \odot V_2 + V_1^{j-1} \tag{16}$$

where $\odot$ indicates element-by-element multiplication of two vectors. The element-by-element vector multiplication plus vector operation in Eq. (16) is highly efficient for vector computation and it is also provided by the SKYbolt's library. The efficiency of this technique for global vectorization results from the fact that it involves five such vector operations on large vectors. However, we did not implement this technique since the routine provided by the SKYbolt's library was not functional for double precision. Furthermore, its efficient implementation requires the minimization of the data movement overhead which may occur in forming the diagonals of matrix M. This requires a direct control of cache memory, which was not possible on the SKYbolt. Nevertheless, for future applications, this technique is very promising as it allows the seemingly strictly serial Do-loops to be vectorized.

5. Discussion and Conclusion

Table I shows a comparison of the different implementations of DARTS for a 13-body and 10-flexible modes model of Cassini spacecraft. As stated before, the speedup of the vectorized algorithm increases with the increase in the number of flexible modes and/or the number of bodies. For a 13-body and 20-flexible modes model, the vectorized algorithm achieves a speedup of 2 over serial DARTS on one SKYbolt. We did not discuss the algorithm's parallelization in detail. Suffice it to mention that, despite using several strategies to overlap the computation and communication as much as possible, the communication overhead from the slow VME bus remained a major bottleneck, which explains the rather poor speedup of parallelization.

Here, we would like to summarize some of the shortcomings of the SKYbolt and to discuss some desired features.

A. DOUBLE-PRECISION PERFORMANCE

The i860 is claimed to be a 64-bit vector processor [4]. However, it has a 128-bit wide data path, which means that only two double-precision operands can be simultaneously loaded from or stored to the cache memory. This significantly reduces the speed of the processor for those vector operations that involve three operands. We implemented our vectorized algorithm with double precision. Although, we did not try the single-precision implementation of the algorithm, given the high vectorization degree of our algorithm, a much greater speedup can be expected for single-precision implementation.

B. LIMITED CACHE MEMORY AND LACK OF CACHE MANAGEMENT

The SKYbolt did not provide a means for managing the cache memory. Thus, we could not further reduce overhead caused by the data movement between the cache and main memory by explicitly defining the physical location of data in the cache memory and, hence, increasing the data re-use. As a result, most of the computation was performed on the data located in the main memory which, in addition to increasing the overhead, significantly reduced the computation speed of both scalar and vector operations. Given this extensive use of the main memory, a DRAM main memory with a slower speed over an SRAM main memory, would have increased the overhead by a factor of 2.

For double precision, the size of i860 cache is 1 K. However, the vectorized algorithm involved the operations on matrices larger than the size of the cache. For example, the matrix $\Pi^* \phi$ in Eqs. (8) and (12) is of dimension 16×78 and the computation of $P(B)$ involves even bigger matrices. An efficient technique for handling such cases is the segmentation of the computation [10]. However, this requires a direct control of the cache memory which, as stated before, was not possible.

C. SKYBOLT'S LIBRARY

As stated before, the SKYbolt's library did not provide some of the useful routines that were frequently used in our implementation. Also, some of the routines provided were not functional either at all or for double precision.

Despite all the above shortcomings, the SKYbolt was highly cost effective and allowed us to meet our goal (see Table I) with a relatively short development time. As stated before, the SKYbolt represents the first generation of low-cost vector processors. The new generations not only provide a drastic reduction in the cost over performance ratio, but also significant improvements in both hardware and software. The size of cache memory in the new versions of i860 has increased by a factor of 2. Single board multiple i860 processor-based architectures [13] are now offered that present a much more balanced system for parallel computation since the communication between processors can be performed on board and via a fast interconnection network. The library routines are also improved. In particular, based on our suggestions to the vendors, new routines including some of the routines developed by us, e.g., the SAXPY-based matrix-vector and matrix-matrix multiplication routines and LDL^* routine for linear system solution,were added to the library.

The results of our work, along with the significant improvements in both the price and performance of these architectures, clearly suggest that the parallel/vector algorithms and architectures present a highly efficient and low-cost approach for achieving real-time simulation capability for even more complex and computationally demanding multibody systems, such as the Space Station. In particular, it should be mentioned that the Space Station has a star topology that allows the application of a similar global vectorization strategy as for Cassini spacecraft. Also, due to the flexibility of Space Station appendages, not only the computation for appendages can be vectorized but also, based on our analysis in Section 2, more vector processors can be used to increase the speedup to the parallelization.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and

Space Administration. The authors gratefully acknowledge the assistance of R. Graves, K. Pendergast, and J. Hernandez in implementing the hardware and software discussed in Part II of this paper.

References

1. A. Jain and G. Rodriguez,"Recursive Flexible Multibody System Dynamics using Spatial Operators," *J. Guidance, Control, and Dynamics*, Nov. 1992. In Press.

2. A. Fijany and J.A. Roberts,"Dedicated Computer Architecture for Real-Time Dynamic Simulation of CRAF/Cassini Spacecraft: Requirements and Specifications," JPL Engineering Memorandum EM 343-91-236 (internal document), Jet Propulsion Laboratory, Pasadena, California, March 1991.

3. A. Fijany,"Parallel/Vectorized Algorithms for Real-Time Dynamic Simulation of CRAF/Cassini Spacecraft," JPL Engineering Memorandum EM 343-91-1230 (internal document), Jet Propulsion Laboratory, Pasadena, California, June 1991.

4. L. Kohn and N. Margulis,"The i860 64-Bit Supercomputing Microprocessor," *Proc. Supercomputing 89*, pp. 450-546, 1989.

5. H. Cheng and K.C. Gupta,"Vectorization of Robot Dynamics on a Pipelined Vector Processor," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 96-101, April 1991.

6. S. McMillan, D.E. Orin, and P. Sadayappan,"Real-Time Robot Dynamic Simulation on a Vector/Parallel Supercomputer," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1836-1841, April 1991.

7. B.L. Buzbee,"A Strategy for Vectorization," *Parallel Computing*, Vol. 3, pp. 187-192, 1986.

8. K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Inc., New York, New York, 1984.

9. K. Hwang,"Advanced Parallel Processing with Supercomputer Architectures," *Proc. of IEEE*, Vol. 75(10), pp. 1348-1379, 1987.

10. K. Hwang and S-P Su,"Vector Computer Architecture and Processing Techniques," *Advances in Computers*, Vol. 20, pp. 115-197, 1981.

11. G.H. Golub and C.F. Van Loan, *Matrix Computation*, second edition, Johns Hopkins University Press, Baltimore, Maryland, 1989.

12. D.E. Rosental and M.A. Sherman,"High Performance Multibody Simulation via Sym-

bolic Equation Manipulation and Kane's Method," *J. Astro. Sciences*, Vol. 34(3), pp. 223-239, 1986.

13. *SKY Computer Product Summary*, Sky Computers Inc,. Chelmsford, Massachusetts, April 1992.

USE OF HARDWARE-IN-THE-LOOP SIMULATION FOR SPACECRAFT MISSION PLANNING/PREPARATION/SUPPORT

L. SLAFER

FIFTH ANNUAL
NASA/NSF/DOD WORKSHOP ON
AEROSPACE COMPUTATIONAL CONTROL

SANTA BARBARA, AUGUST 1992

HUGHES

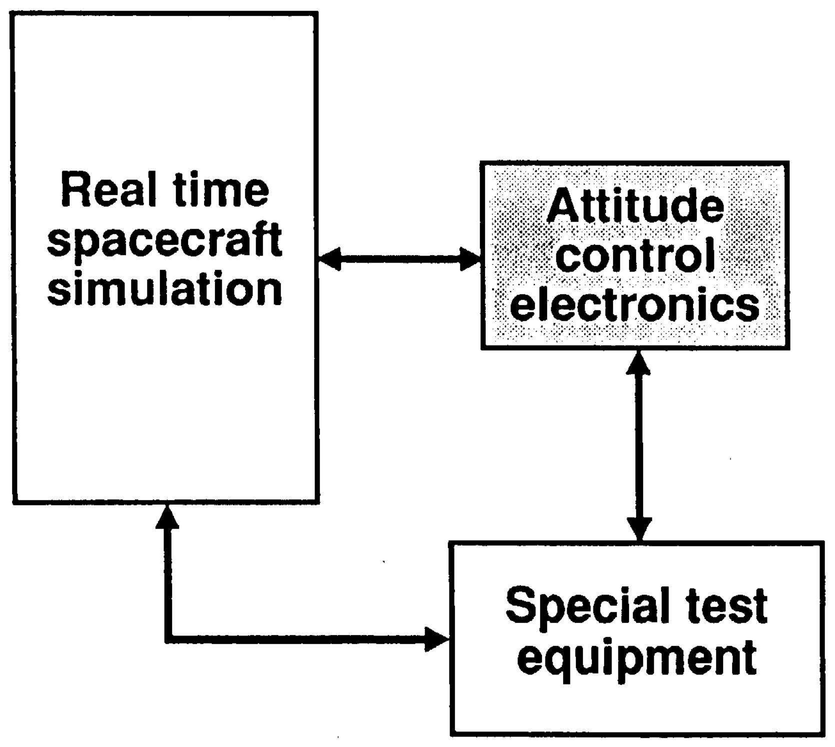
Overview

HUGHES

- 'Mixed simulation' testing at Hughes SCG
- 'Ground station MST' concept
- Ground station MST implementation - 'the whole is greater than the sum of the parts'
  - System 100
  - Dynamic satellite simulator
  - Ground station H/W and S/W
- Ground station MST current/future applications
- HS 601 mission support system facility
- Concluding remarks

Mixed Simulation Testing Is Key to ACS Development/Validation

HUGHES



- MST combines breadboard and/or flight control electronics with detailed spacecraft, sensor, actuator simulations
- MST is real time, simulated on-orbit environment for

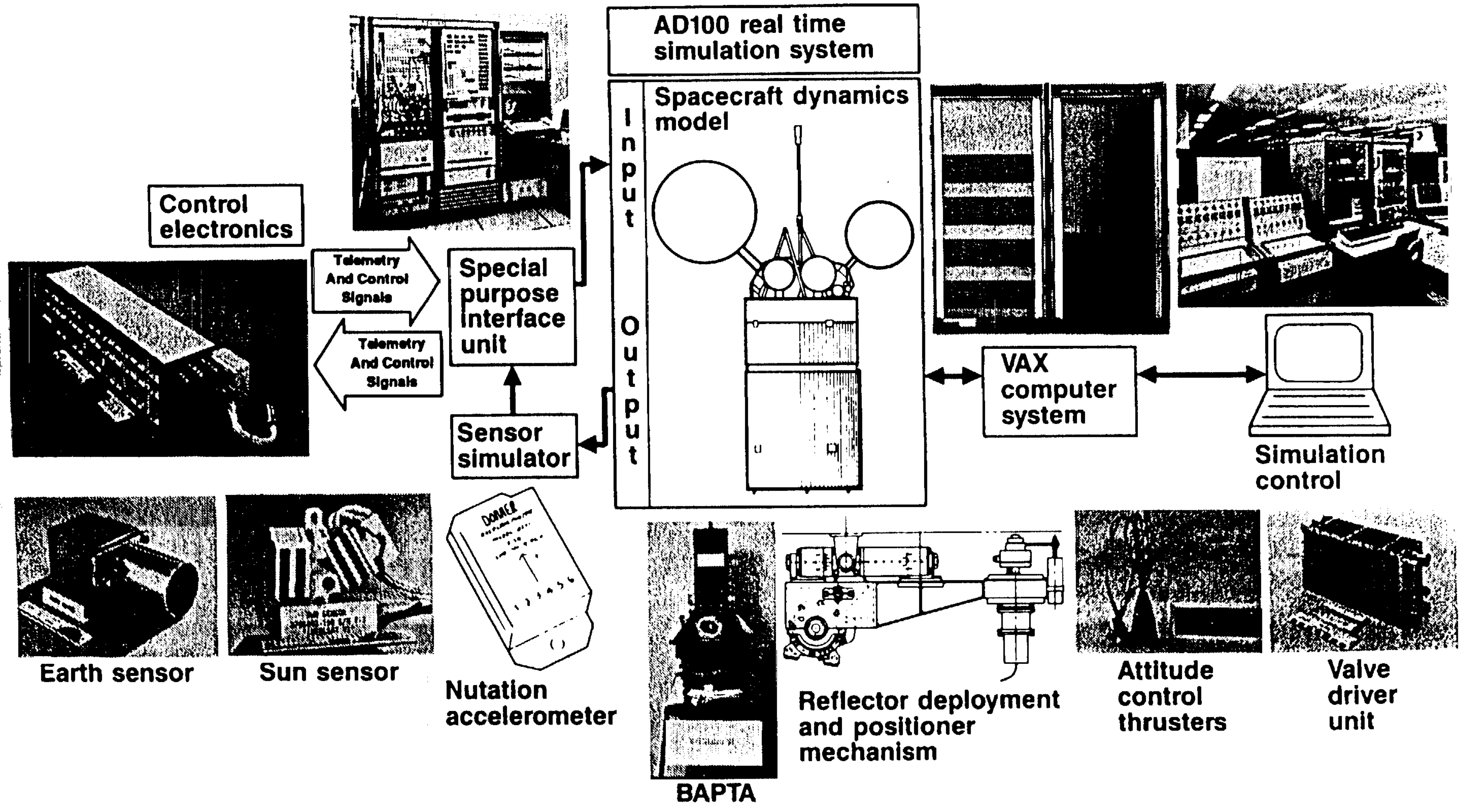
  ACS design and performance validation

  Flight S/W development and qual testing

  ACE qual/acceptance testing

Spacecraft Control System MST Creates Simulated On-orbit Environment

HUGHES


AD100 real time simulation system
Spacecraft dynamics model
Input
Output
Control electronics
Telemetry And Control Signals
Telemetry And Control Signals
Special purpose interface unit
Sensor simulator
VAX computer system
Simulation control
Earth sensor
Sun sensor
Nutation accelerometer
BAPTA
Reflector deployment and positioner mechanism
Attitude control thrusters
Valve driver unit

MST Activities at Hughes SCG

HUGHES

MST at Hughes has evolved over 25 yrs to become essential element of spacecraft development

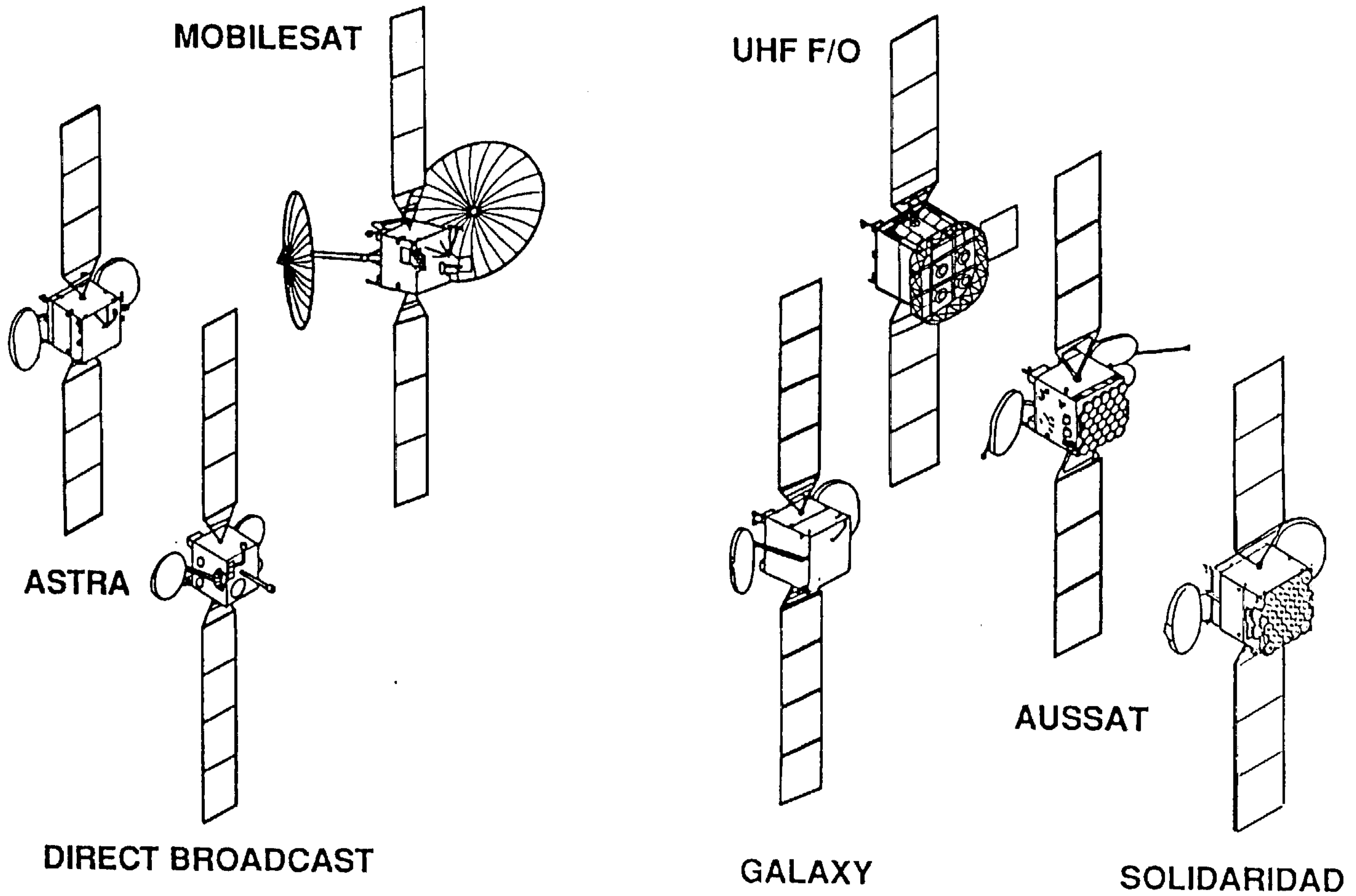
MST used in many areas

- ACS performance predictions/verification
- Developing insight into on-orbit response characteristics
- Hardware and software design verification/validation
- Unit and spacecraft testing
- Mission planning
- On-orbit anomaly investigations

In many instances, MST has been key tool in understanding and solving problems on ground and in-orbit

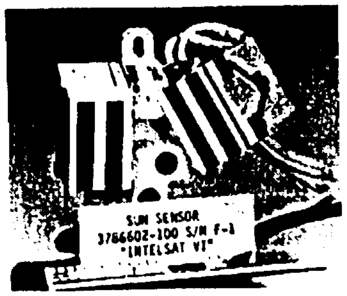
HS601 FAMILY OF SPACECRAFT REPRESENTS VERY COMPLEX NEW SYSTEM FOR HUGHES

HUGHES

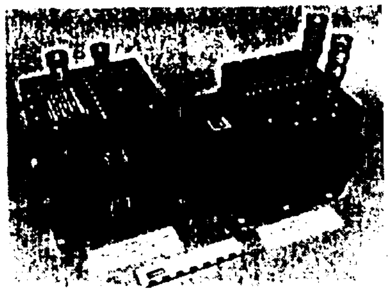


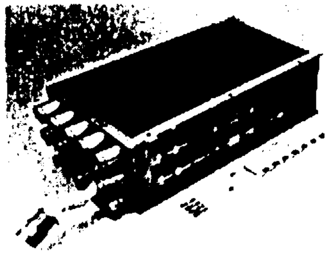
HS 601 Attitude Control Subsystem Components

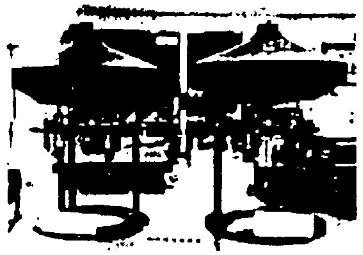
HUGHES

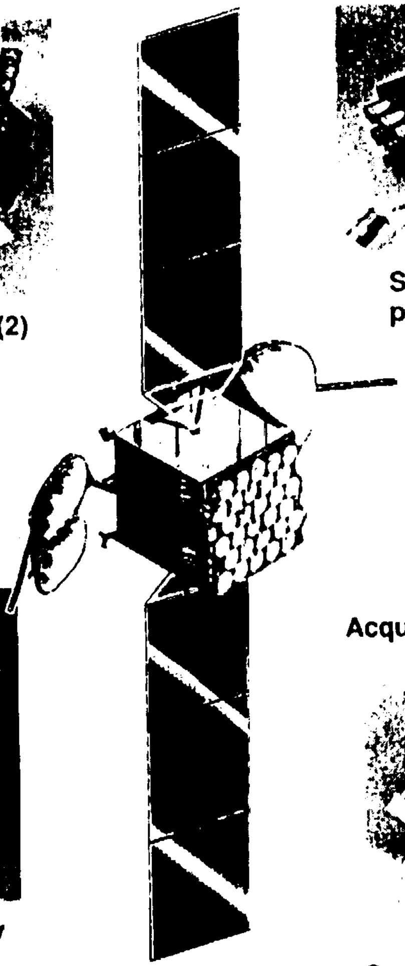


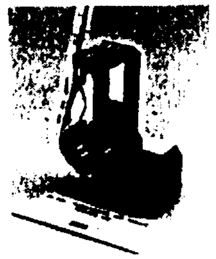
Transfer orbit sun sensors (2)

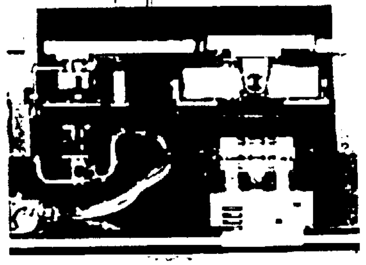

Inertial reference unit (2)

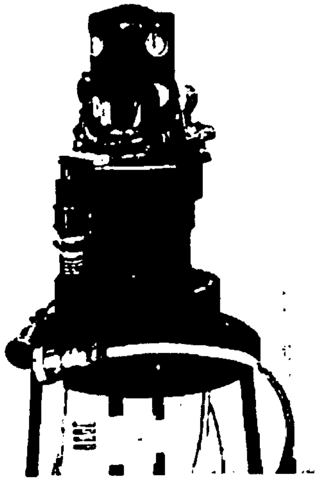

Spacecraft control processor (2)

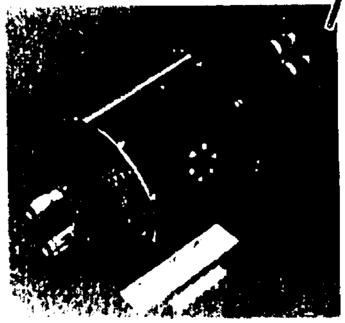

Momentum wheel and wheel platform

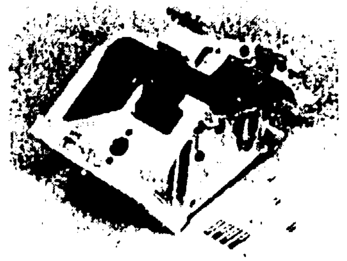



Acquisition sun sensors (2)


Reflector positioner and deployment activator


Solar wing drive (2)


Earth sensor assembly


Omni deployment actuator

Attitude control thrusters

M921561-106

HUGHES PRODUCTS FOR OUR SATELLITE CUSTOMERS

HUGHES

•*REAL-TIME GROUND STATION SYSTEM*

- RECEIVE/PROCESS/DISPLAY SPACECRAFT TELEMETRY
- FORMAT COMMANDS TO THE SPACECRAFT
- PROVIDE OPERATOR INTERFACE TO SPACECRAFT
- SOFTWARE TO MONITOR HEALTH AND STATUS OF SPACECRAFT (DATA PLOTTING, CONFIGURATION BLOCK DIAGRAMS, LONG-TERM TREND ANALYSIS)
- AUTOMATIC HEALTH/STATUS MONITORING (ALARMS TO OPERATORS)

•*DYNAMIC SATELLITE SIMULATOR*

- REAL-TIME COMPUTER SIMULATION OF SATELLITE BUS AND PAYLOAD
- FUNCTIONAL REPRESENTATION OF ALL ELEMENTS
- FORMATS/TRANSMITS TELEMETRY FROM SIMULATOR
- RECEIVES REAL-TIME COMMANDS
- PROVIDES TLM AND CMD INTERFACE TO GROUND STATION COMPUTER SYSTEM
- OPERATOR TRAINING AID

Ground Station MST Heritage

HUGHES

Ground station MST developed with recognition by ACS, systems engineering, and mission operations senior staff that method of validating critical HS 601 operating procedures and mission software did not exist ⇒

- Significantly increased system and operational complexity (over HS 376) required high fidelity hardware-in-the-loop simulator
- Multiple HS 601 programs with different system designs required low cost techniques to perform operations design and validation
- High fidelity simulator essential as training/rehearsal tool

Experience with first program (Aussat) led to utilization on other HS 601 programs (UHF, Galaxy, MSAT...)

Ground Station Mixed Simulation Testing for HS 601

HUGHES

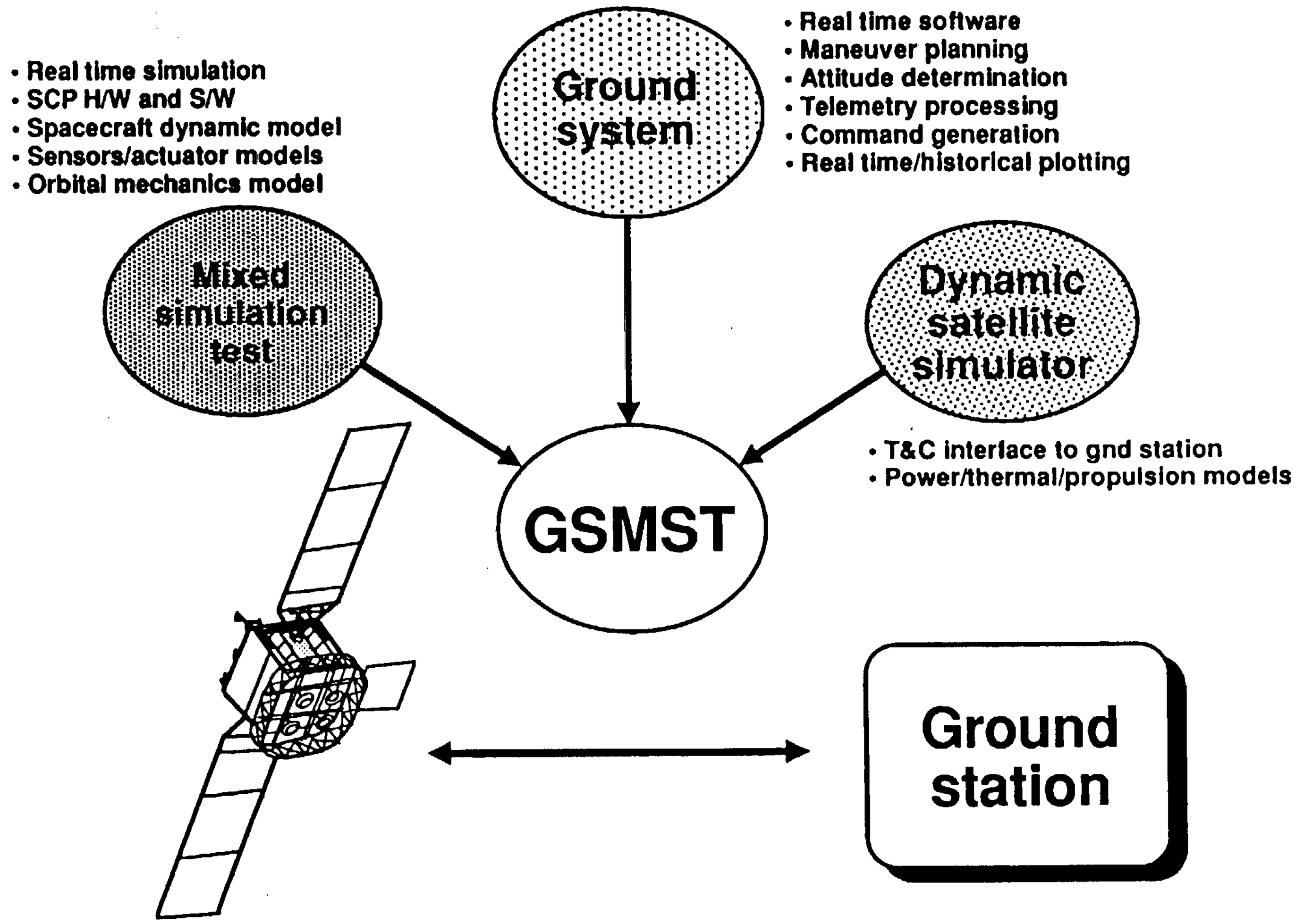
Ground station MST integrates

- ACS MST (with breadboard or flight control electronics)
- Core of spacecraft dynamic satellite simulator (DSS)
- Ground station real time software (tlm and cmd)
- Ground station command generation S/W
- Ground station telemetry processing S/W
- Ground station attitude determination and mission planning software

To create complete, high fidelity, real time simulated on-orbit environment to

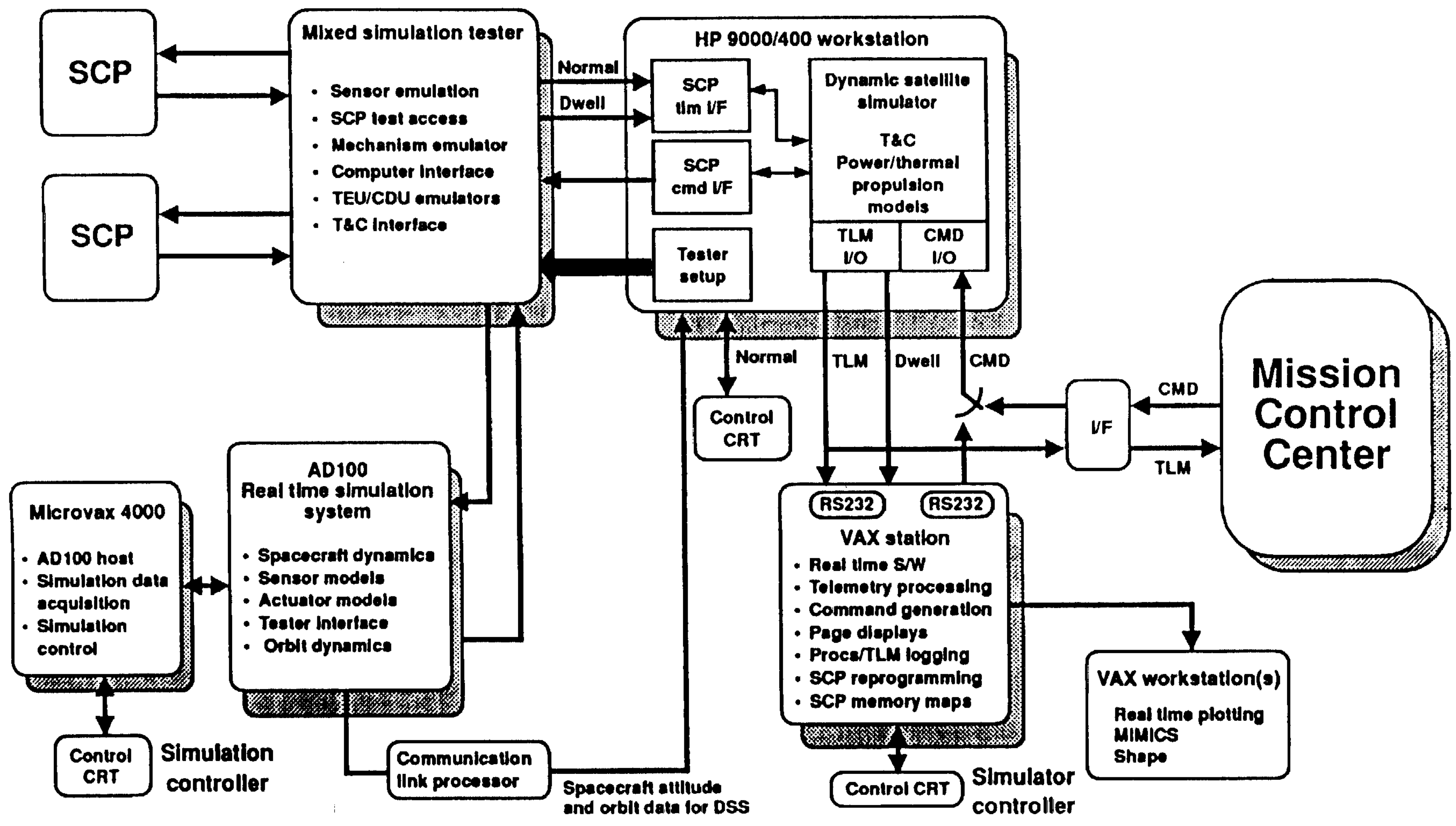
'Fly the spacecraft on the ground'

GSMST Integrates 3 Existing Elements


• Real time simulation
• SCP H/W and S/W
• Spacecraft dynamic model
• Sensors/actuator models
• Orbital mechanics model
Mixed simulation test
Ground system
• Real time software
• Maneuver planning
• Attitude determination
• Telemetry processing
• Command generation
• Real time/historical plotting
Dynamic satellite simulator
• T&C interface to gnd station
• Power/thermal/propulsion models
GSMST
Ground station

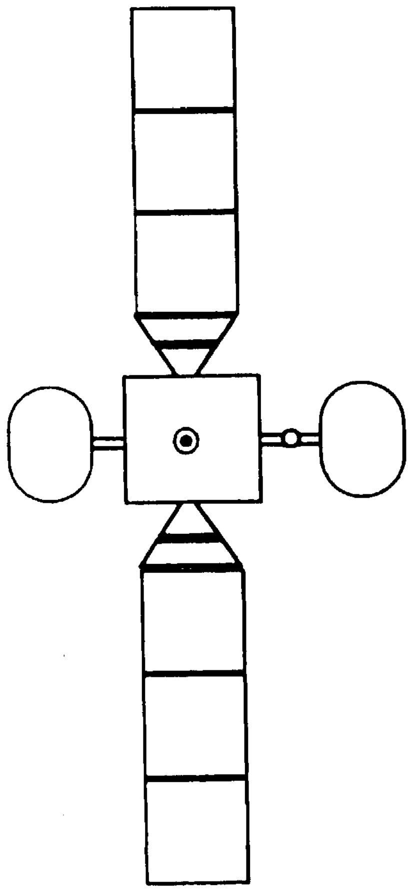
GSMST Configuration

HUGHES



AD100 Simulation Features

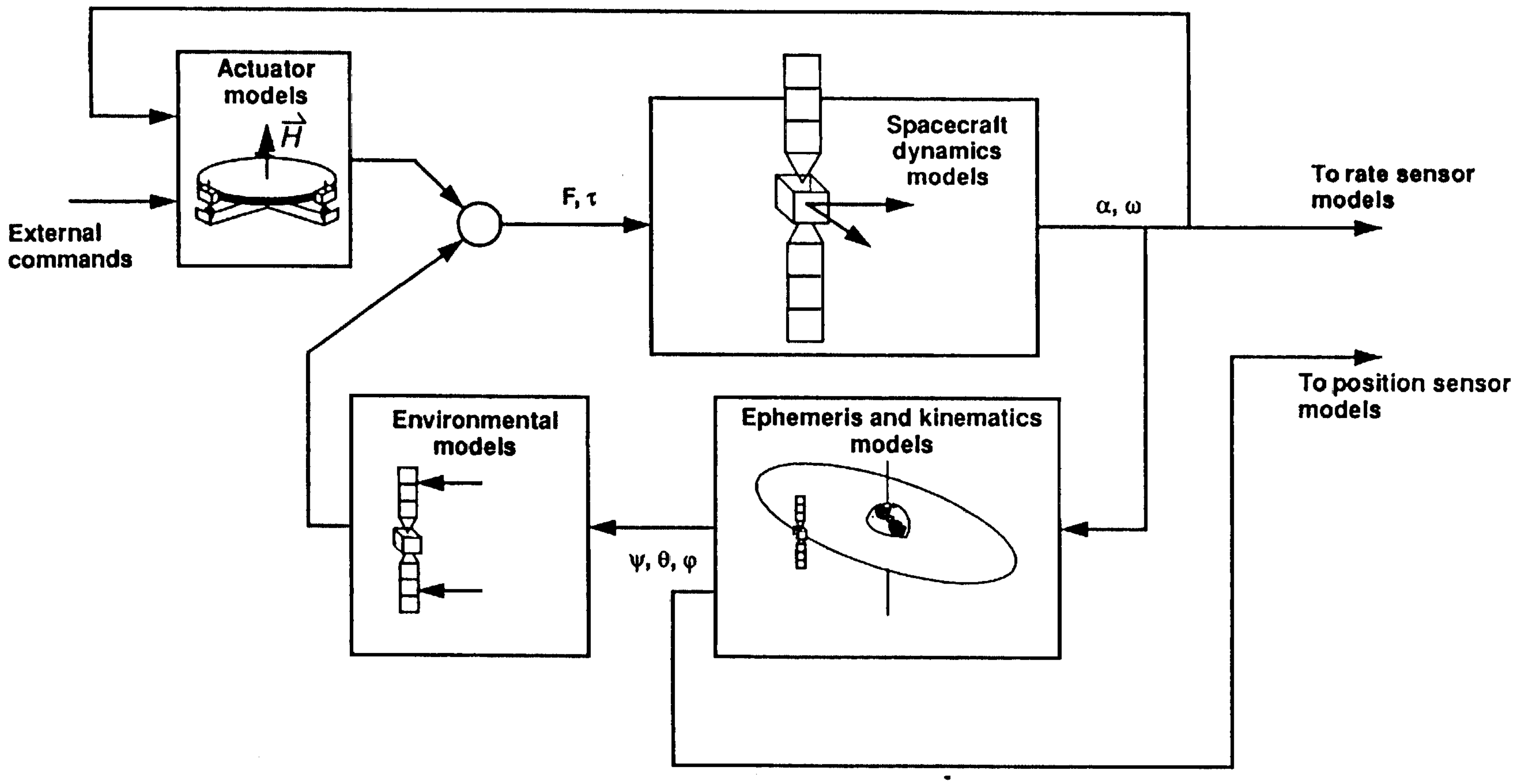
HUGHES



- Rigid dynamics with 16 DOF
- Flexible modeling with 12 modes/wing, 4 modes/reflector
- Large angle kinematics modeling
- Selectable orbits (low earth to geosync) and ephemeris models
- Selectable mass property and appendage configurations
- Solar and magnetic environmental torque models
- On-orbit maneuver fuel slosh modeling
- Eclipse thermal shock modeling
- High fidelity actuator and sensor modeling
- Selectable anomaly/failure mode modeling
- I/O interface to command and telemetry subsystem

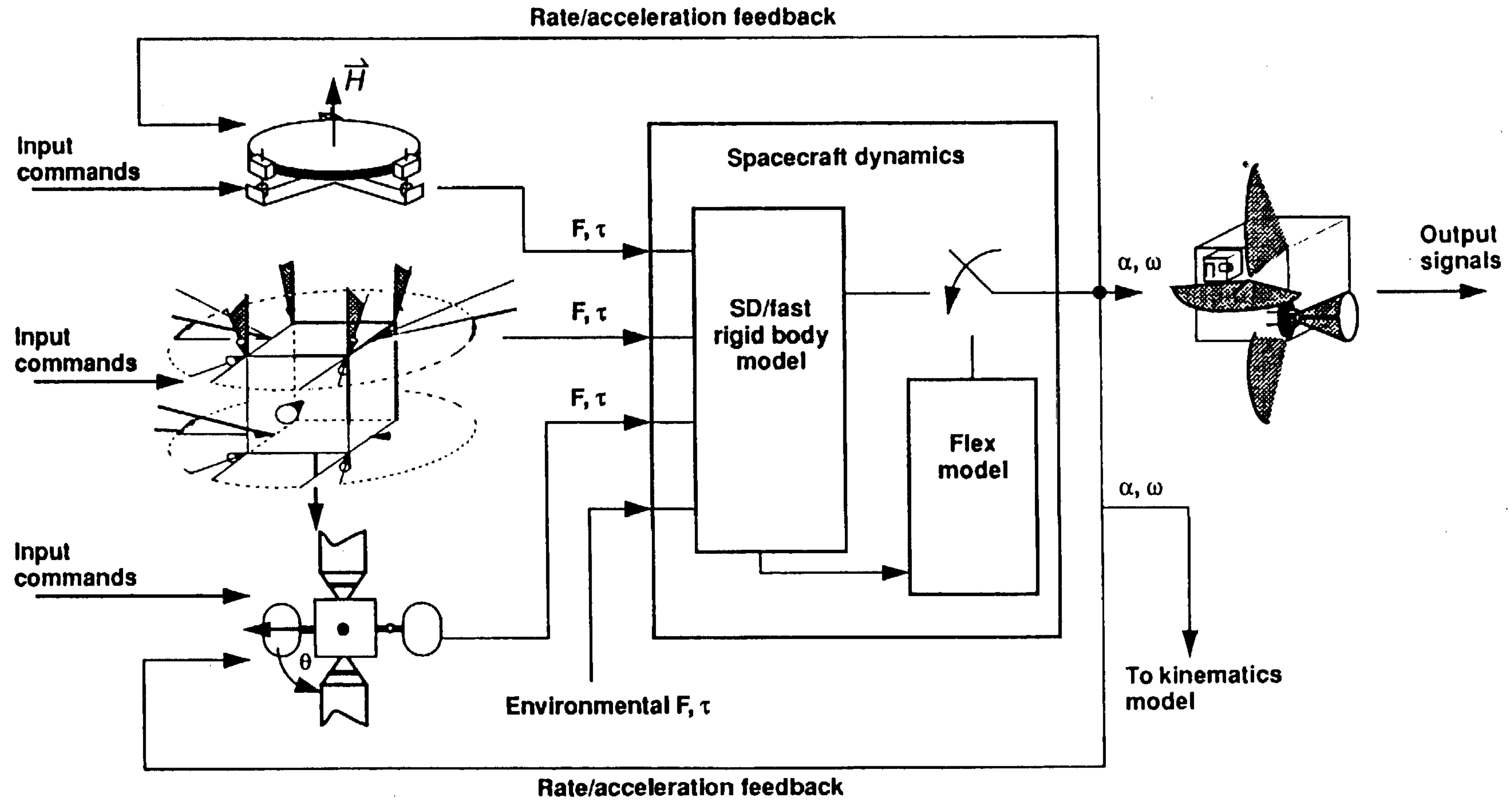
HS 601 AD100 Model Elements

HUGHES



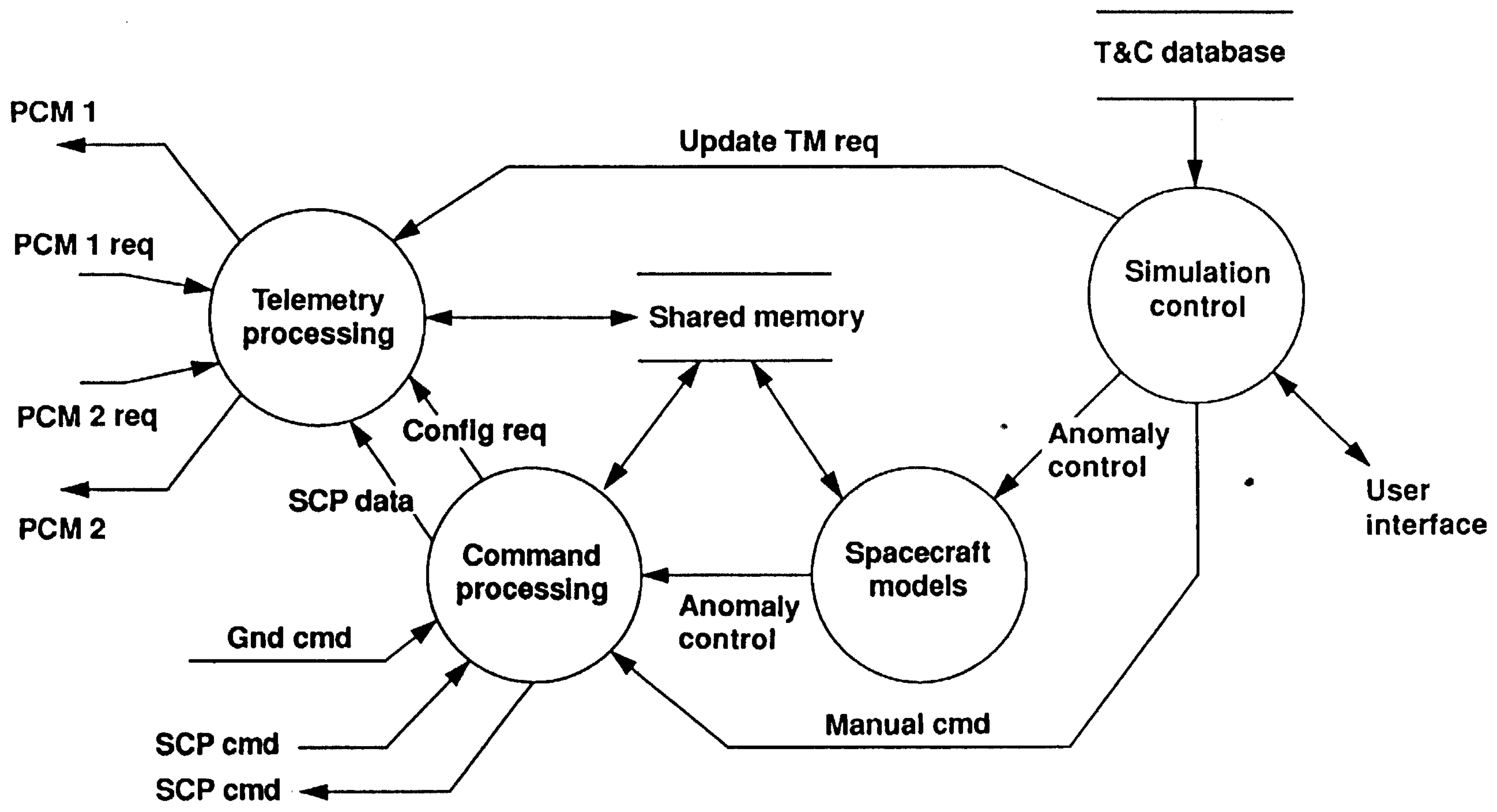
Dynamics Modeling

HUGHES



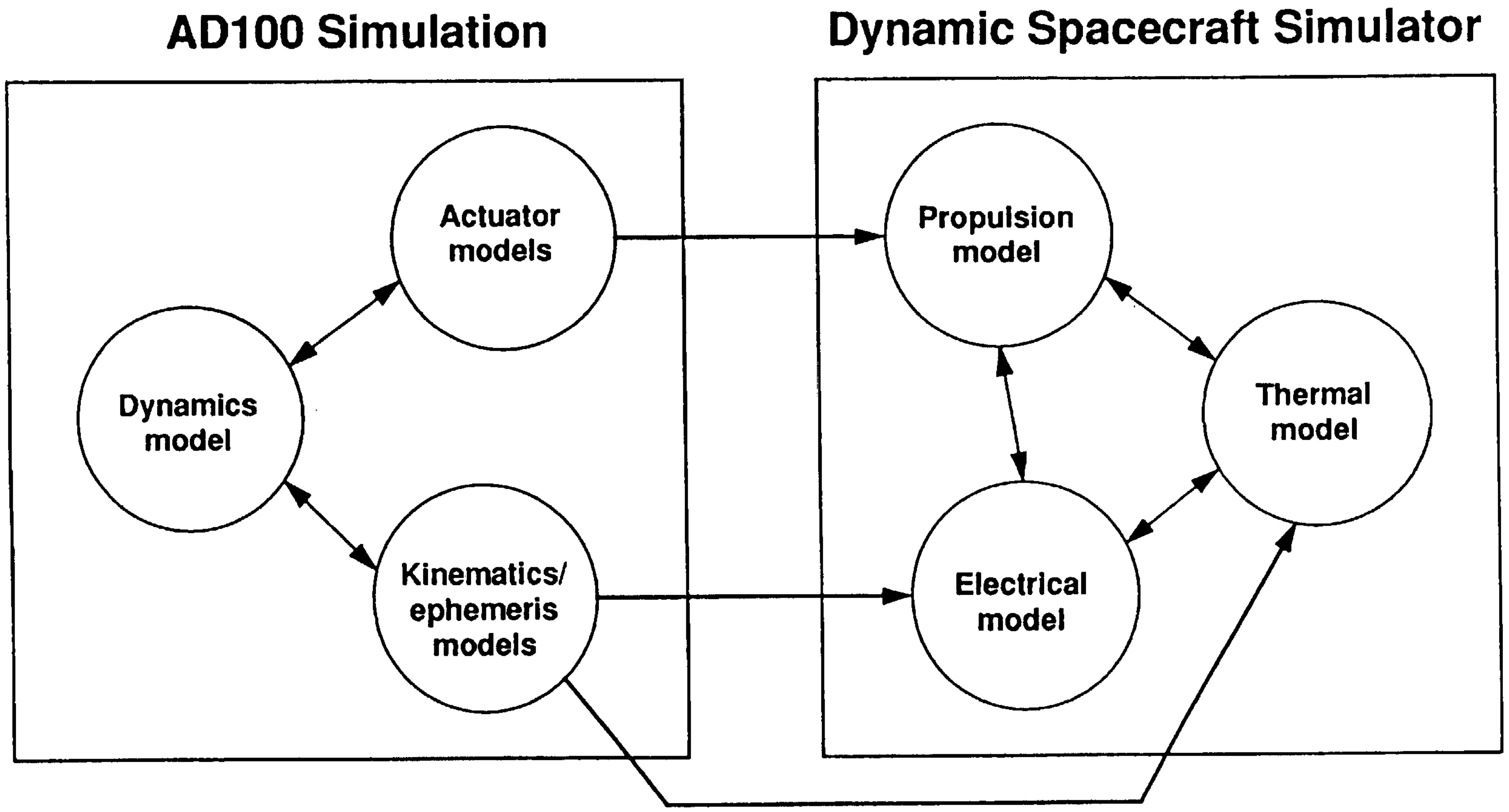
Dynamic Spacecraft Simulator

HUGHES



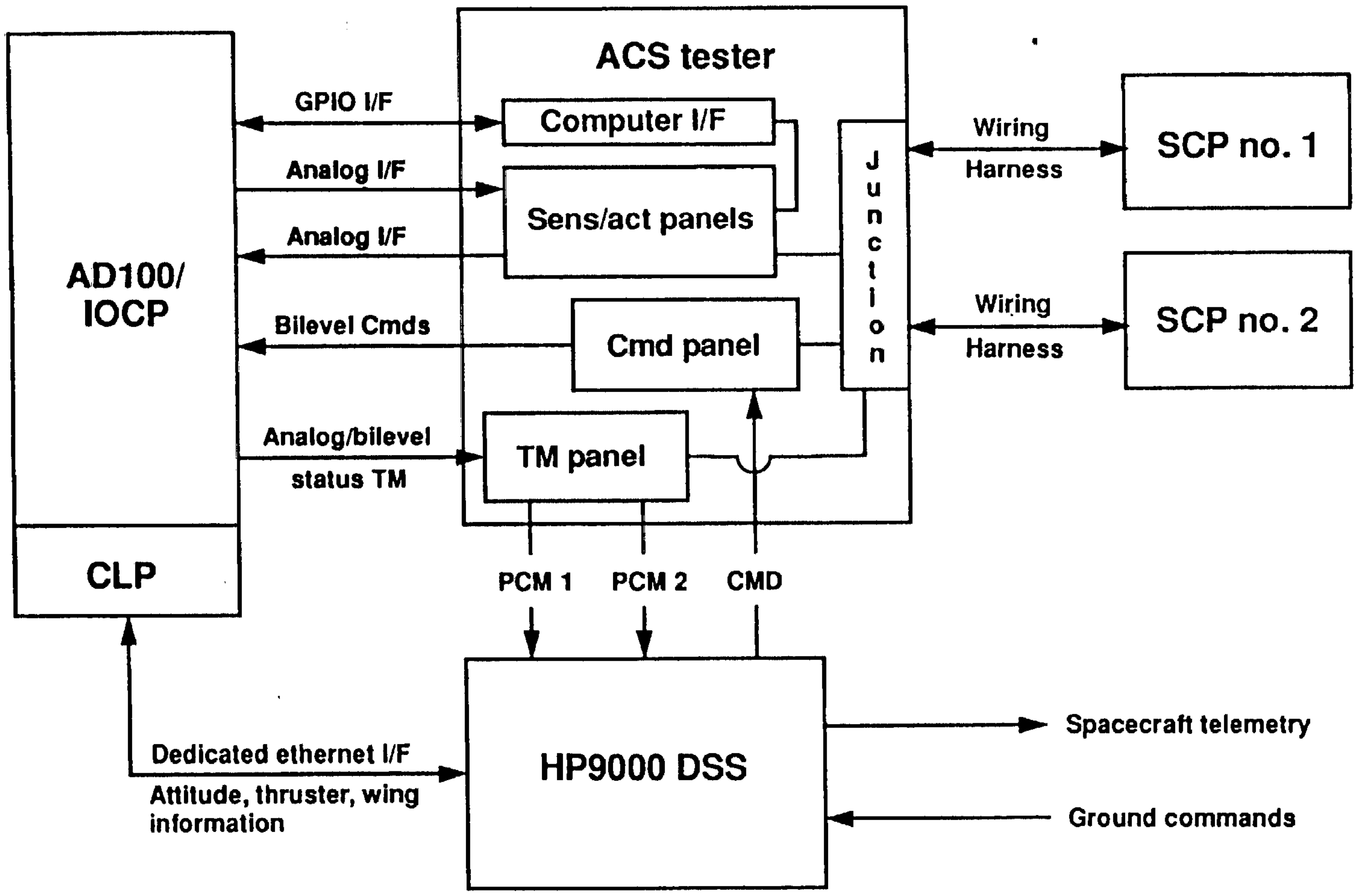
AD100 Simulation - DSS Information Flow

HUGHES



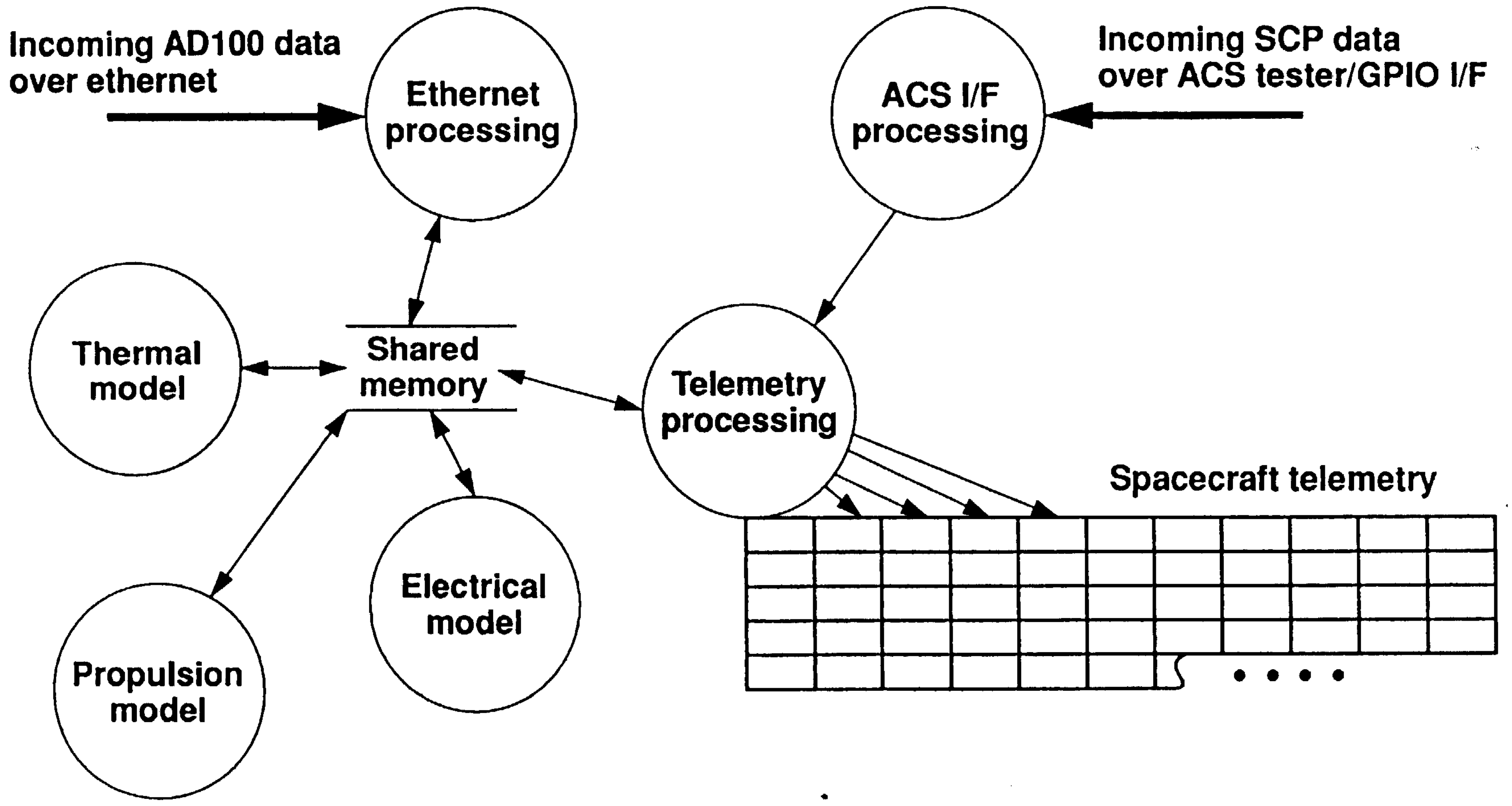
AD100 - DSS Interfaces

HUGHES



Building Spacecraft TM Frame

HUGHES



HS 601 GSMST Provides Extensive Range of Capabilities

HUGHES

Mission planning/preparation
- Development/validation of operational procedures
- Development/checkout of on-orbit test procedures
- Verification of mission sequence of events

Groundstation software
- Groundstation software development/validation with actual flight hardware and software
- Validation of telemetry and command data bases/processing
- Develop/validate groundstation attitude determination/maneuver software
- Design/test of command procedure software

Training
- High fidelity environment for Hughes mission rehearsals (sim lab or HMCC)
- Mission rehearsals from customer groundstation
- Customer operator training

On-orbit support
- Real time mission support for pretesting of procedure mods
- On-orbit anomaly investigations
- On-orbit RAM patch development/checkout/preuse validation
- Provides a mission oriented environment for additional flight software testing (system stress testing)

Other Hughes customers consider this capability essential to their mission operations

Current and Future GSMST Applications

HUGHES

GSMST has become a 'best seller'

It is considered essential by Hughes management for mission planning/preparation/validation

On-going applications
- Aussat B-1 (first HS 601 built for Australia; controlled from Sydney groundstation)
- UHF Follow-on (Navy satellites which use Air Force CSOC facility)
- Galaxy
- MSAT
- DBS

Because of importance of this system, Group authorized funds to develop dedicated facility solely for support of mission activities

HS 601 'Mission Support System'

HUGHES

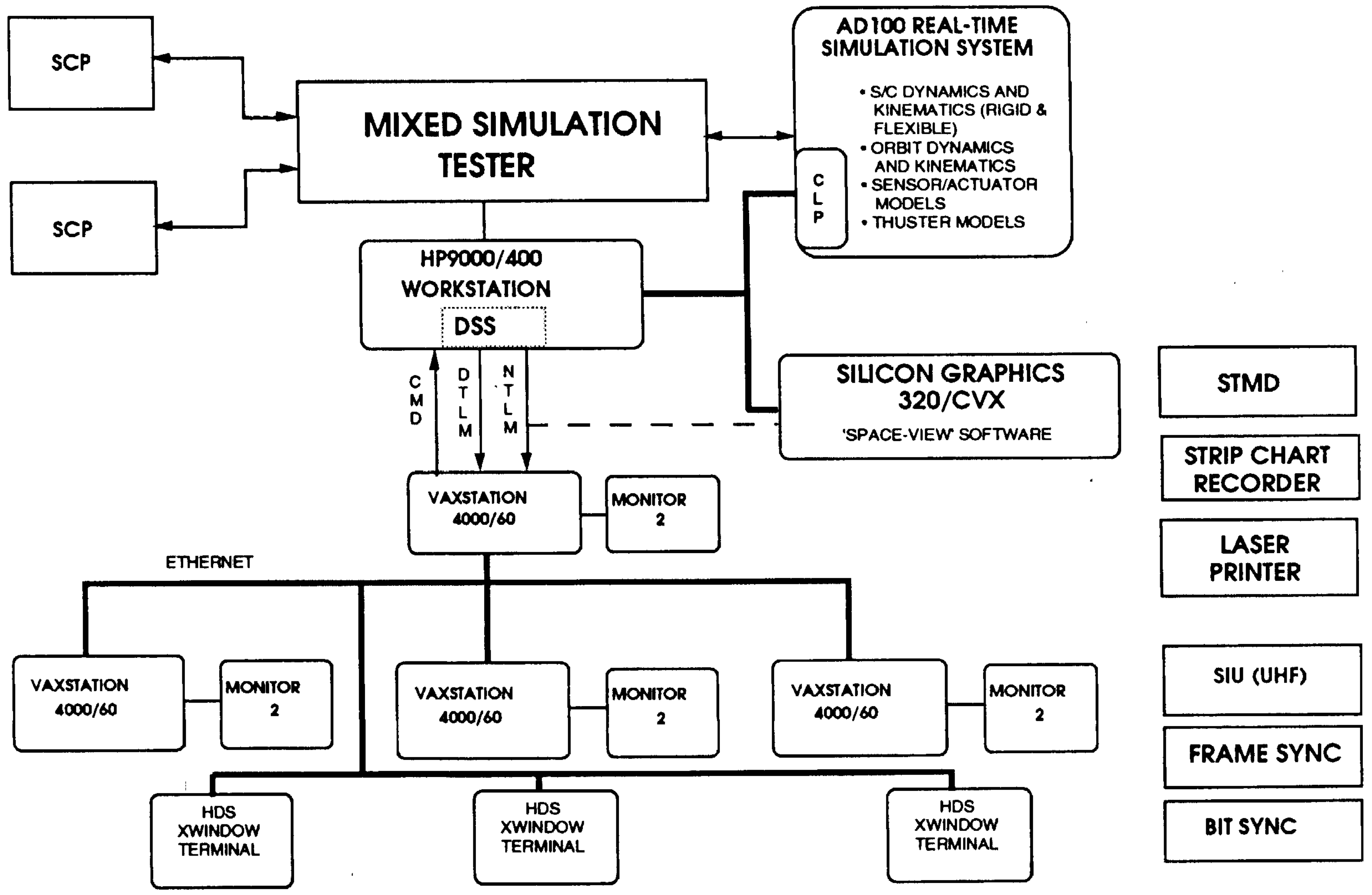
GSMST system dedicated to support HS 601 spacecraft mission operations

Located within control and dynamics simulation lab (now in building E4)

Provides capability for

- Local, internal rehearsals or procedure/software checkout
- Communications link to Hughes mission control center (in building S67) for full rehearsals
- Remote operation from customer groundstation (Australia, Colorado Springs already demonstrated)
- Receipt and processing of mission/on-orbit telemetry data for analysis

HS601 MISSION SUPPORT GSMST SYSTEM CONFIGURATION



Concluding Remarks

HUGHES

- Hardware in loop MST essential to development of our satellite attitude control subsystems
- Extending MST to include rest of spacecraft and ground segment elements was natural addition to the MST capability
- GSMST proven itself to be essential and cost effective method of supporting mission operations
- Future use of system will provide high-fidelity training aid for our customers