Parallel Orbit Propagation and The Analysis of Satellite Constellations

by

Scott Thomas Wallace

B.S. Engineering Science,
United States Air Force Academy
(1993)

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1995

Signature of Author_____
Department of Aeronautics and Astronautics
June 1995

Certified by_____
Dr. Ronald J. Proulx
Thesis Supervisor, CSDL

Certified by_____
Dr. Paul J. Cefola
Thesis Supervisor, CSDL
Lecturer, Department of Aeronautics and Astronautics

Accepted by_____
Professor Harold Y. Wachman
Chairman Departmental Graduate Committee

Parallel Orbit Propagation And The Analysis of Satellite Constellations

by

Scott Thomas Wallace

Submitted to the Department of Aeronautics and Astronautics on May 18, 1995 in partial fulfillment of the requirements for the Degree of Master of Science

ABSTRACT

This thesis describes the development of a scalable, portable parallel orbit propagator, with application to the analysis of satellite constellations. The Draper Semianalytic Satellite Theory (DSST) is coupled with the Parallel Virtual Machine (PVM) software package to demonstrate the power of the networked computing paradigm. The PVM/DSST is employed to analyze the stability of the 840 satellite Teledesic constellation, as described in the 1994 FCC filing, under real-world perturbations. Combined with genetic algorithm optimization software, the frozen orbit of a satellite in the presence of arbitrary perturbations is easily determined. This concept is extended to automate constellation design for optimal performance.

Thesis Supervisor: Dr. Ronald J. Proulx
Title: Technical Staff Engineer, The Charles Stark Draper Laboratory, Inc.

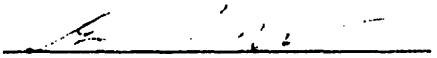
Thesis Supervisor: Dr. Paul J. Cefola
Title: Lecturer, Department of Aeronautics and Astronautics
Program Manager, The Charles Stark Draper Laboratory, Inc.

Acknowledgments

This project represents the effort of many who gave a great deal of time and effort. Many thanks to my advisor, Paul Cefola, for the opportunity to work here and the providing the focus to keep me on schedule. Equal thanks to Ron Proulx for keeping my math straight and continually offering new ideas. Thanks also go to the numerous projects that supported this work at Draper. My experience with the RADARSAT development, under Rick Metzinger, provided the opportunity to be involved in a customer oriented software engineering project. Technical advice from Mr. Herman Rufenacht (SPAR) also contributed to the Draper experience. Dave Carter at Draper Labs offered helpful insight throughout the project. Thanks to Phillips Laboratory for getting all parties involved interested in parallel computing and MIT's Laboratory for Computer Science for parallel computing instruction in addition to CM-5 access. The work of Jason Schott and Andrei Schor in genetic algorithms provided an excellent application for this investigation. The SPI effort offered insight into the software configuration management problem and ideas into doing things better. Thanks especially to Draper Laboratory's DFY95 IR&D Task 615, Parallel Processing and Astrodynamics, that provided direct support for this work.

A special thanks to the ACME Lab and all those who supported it. Stuart Roseman and Kyle McDonald provided extensive technical support in all areas of first rate software development. Thanks to James Beaupre for insight into computer science and parallel computing, as well as providing excellent coffee. To my comrades in arms: those of 13 Bigelow (Kent, Scott, Anthony, Jason) who will all make the world a safer place. Scott Carter must receive a double mention for all the minutes I borrowed. To Matt Lobner for waking me up and keeping me strong. To my comrades not of arms (Chris Stoll, Chris Sabol, Carmen, Kazumi Masuda) many thanks for technical help, cultural enrichment, and many good times. Thanks to wonderful Nimi, my parents, and Clint, who always had faith and overwhelming support and encouragement. Finally, to the Lord, who was always there.

This thesis was prepared at the Charles Stark Draper Laboratory, Inc., with support from Draper Laboratory's DFY95 IR&D Task 615.



Scott T Wallace, Lt. USAF

Contents

List of Figures

List of Tables

1.0 Introduction

1.1 Satellite Constellations

1.1.1 Multiple Satellite Constellations

Satellite constellations, groupings of more than one satellite, require extensive computing facilities and personnel to track, maintain, and control. Some constellations consist of many functionally similar satellites in similar orbits, while other constellations include a wide variety of satellites in various orbits.

Many different requirements mandate that satellites be grouped into constellations. Some of the reasons for grouping satellites into constellations are discussed below.

- Satellite Functionality
- Collision Avoidance / Prediction
- Military Security

Satellite Functionality : Multiple satellites can be used as a single system. The GPS (Global Positioning System) constellation provides position determination over the entire planet and to other satellites in space. The Air Force, in charge of GPS, must carefully maintain the orbits of all these satellites because of their vital role for a number of critical navigation systems. Communication satellites can work together to provide whole Earth coverage. Functionally grouped satellite constellations will increase in number as technology develops and demand for worldwide services grows.

Collision Avoidance / Prediction: All objects in orbit are considered a constellation for the collision prediction and avoidance problem. As the number of satellites in space grows, the possibility of collision greatly increases. According to König-Lopez, over 3,600 launches since 1957 have placed approximately 23,000 objects into Earth orbit. Only 500 of those objects

are currently operational satellites [6]. Non-operational objects consist of dead payloads, rocket stages, and other debris. If objects less than 10 cm in diameter are included, the number of objects rises to 400,000 [6]. Because these small objects are traveling at high velocity, they can cause serious damage to other space objects. Manned missions must be especially aware of orbital debris.

Military Security: Since 1959, the military has kept track of the satellites in space for a variety of reasons [1]. Satellite orbits are used to predict the function of foreign satellites, for example. Cataloging all space objects requires the military to view all objects in space as a heterogeneous constellation. Additionally, many critical military systems depend on satellites. Military communication, surface imaging, and missile launch detection are all critical functions performed by military satellites. As a result, the military must track, maintain and control many heterogeneous homogeneous constellations.

All the constellations described above require flight dynamics processes to manage their satellites. Orbit propagation, defined in section 1.2, is critical to predict the future location of satellites. Orbit determination from raw observations keeps the satellite information current so the accuracy of future predictions does not degrade beyond requirements. Maneuver planning keeps a satellite in the correct orbit. Telemetry uplink and downlink requires satellite positions be accurately predicted into the future so that data transfer can be planned effectively. It is desirable to not reproduce all the work performed for a single satellite when maintaining an entire constellation of satellites. However, the process of scaling-up a system from one satellite to multiple satellites is not a simple challenge.

1.1.2 Global Personal Communication Systems

The recent explosion of interest in developing global personal communications systems (GPCS) presents the newest challenge facing designers of ground based satellite maintenance systems. A system of satellites that allows mobile users on the ground to obtain voice and data communication anywhere, anytime promises to fill the skies in the near

future. Many existing systems were designed to support only one satellite and are not capable of handling a multiple satellite constellation. The Radarsat Flight Dynamics System, for example, provides a variety of functions for the Canadian synthetic aperture radar satellite, RADARSAT. Observation pre-processing, orbit determination, ephemeris generation, ground track generation, burn planning, eclipse entry and exit were among the required capabilities of this system [46]. The system was designed to support one satellite. Because the architecture of the system separated processes into communicating services, it could be redesigned to support multiple satellites on multiple computers. However, it would not make sense to duplicate the single satellite system for each satellite supported.

Figure 1-1 illustrates the size in terms of the number of satellites for several of the currently proposed communication networks.

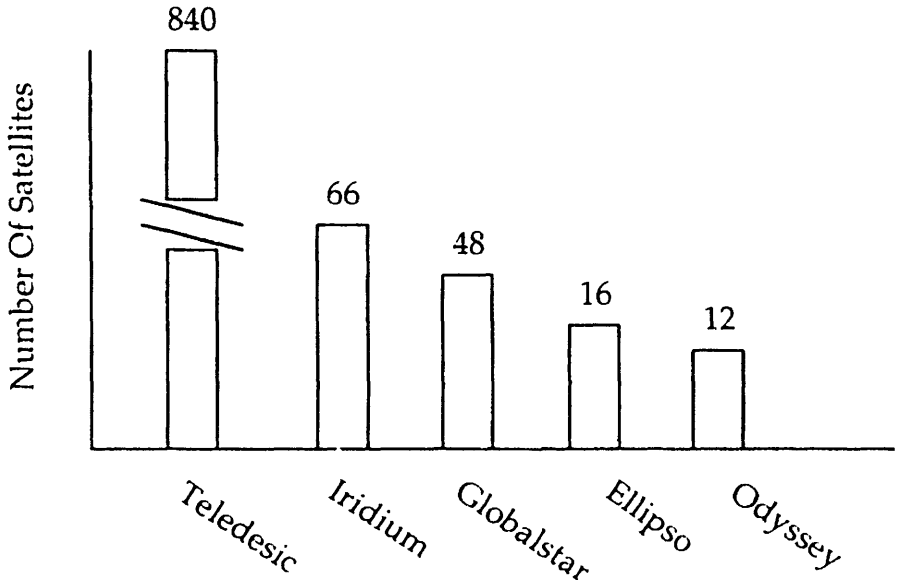


Figure 1-1: Number of Satellites for Each of the Proposed GPCS [47, 48, 66]

These constellations contain more than twice the current number of operating satellites. Each system will require tracking, control and maintenance of each of their satellites.

Satellites provide capability for long distance communication which far exceeds wire and microwave systems in range and coverage [2]. Many previous satellite based communication systems have used a

Geosynchronous orbit [37]. Although the high altitude of a geosynchronous orbit provides a large coverage area and eliminates the atmospheric drag perturbation, geosynchronous orbits also create many problems for communication system design. The high altitude increases the delay time in signal transmission [37]. Figure 1-2 compares the minimum delay times introduced by a signal traveling to and from a geosynchronous satellite. Note this does not include the additional time required for transmission through a ground network.

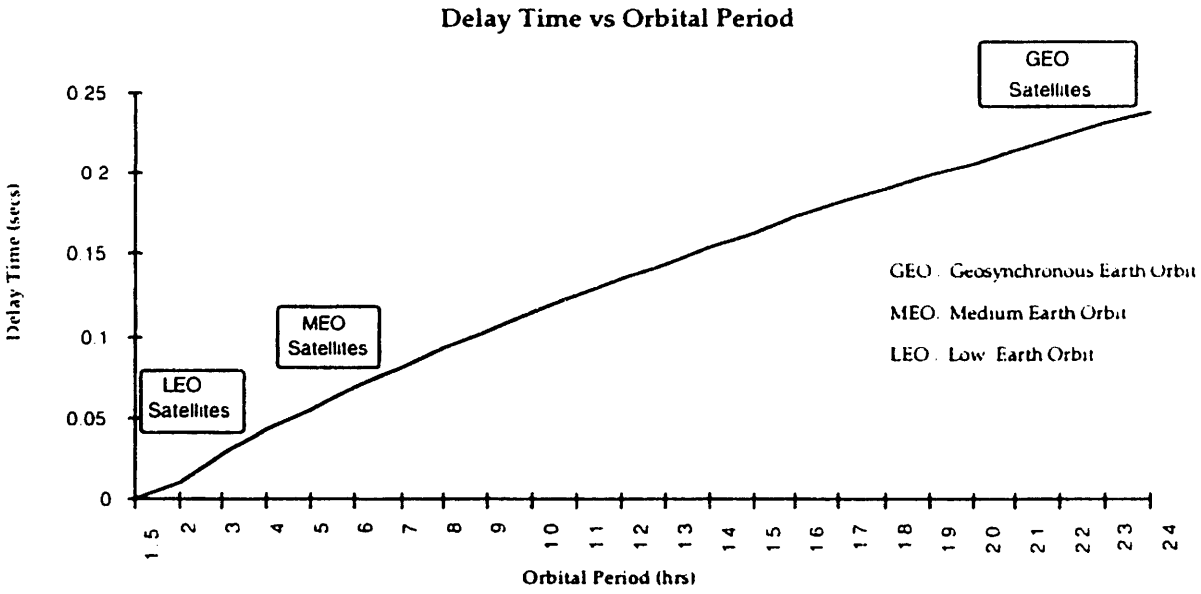


Figure 1-2: Delay Time Vs Orbital Period

For some applications, more than one 'hop' is necessary. A hop represents a signal traveling from the Earth to the satellite and back again [2]. The delay time at geosynchronous altitudes, almost a quarter of one second per hop, degrades voice communication if multiple hops are used [2]. The high altitude of geosynchronous satellites also requires more power and gain in the communications link, both on board the satellite and at the Earth transmitting and receiving station. Because of these disadvantages, most of the proposed personal communication systems are planning on using LEO or MEO orbits for their constellations. However, the lower altitudes have forced designers to use many satellites to achieve world wide availability.

1.1.3 Design and Analysis of Satellite Constellations

The orbit design for satellite-based communications systems can be viewed as a constrained optimization problem. Each system described in Figure 1-1 has a different orbit design. Each designer optimized the constellation within the constraints of the communications system. Some of the systems have similar orbits and vary the number of satellites used. Other systems have chosen different orbits. The weight given to each of the performance parameters along with the system constraints determined the optimal constellation design. A few factors that influence the design of constellation orbits can be seen below.

- System cost
- Area covered
- Launch costs
- Number of satellites in view from the ground
- Percent of coverage above a necessary elevation angle
- System lifetime
- Minimum separation between satellites
- System availability

The large number of parameters that are involved in the GPCS constellation design make the problem very complicated. Table 1-1 groups the five constellations listed in Figure 1-1 according to similarities in their orbit design.

Table 1-1: Orbit types of five GPCS systems

System	Type
Teledesic	LEO
Iridium	LEO
Globalstar	LEO
Ellipso	MEO
Odyssey	MEO

LEO systems choose to minimize power requirements in the communications link but are forced to use many more satellites than their higher altitude competitors. MEO systems favor fewer but more complex satellites. Ellipso proposed a unique solution using high eccentricity orbits to concentrate their coverage in the Northe n hemisphere [48].

1.2 Orbit Propagation

Orbit propagation is the technology of 'computing, from prescribed initial conditions, the value at a specified time of the vehicle state and, optionally, the state partial derivatives' [49]. In many ways, the orbit propagation method used in a ground system is the cornerstone function from which other capabilities are derived. Orbit propagation is required to perform every satellite maintenance function. Orbit determination, for example, depends heavily on the propagation method. Orbit determination uses raw satellite observations to 'estimate the satellite orbit and associated parameters' [49]. The best state is chosen by minimizing the difference between observations and their predicted value; the predicted value is generated by the orbit propagation method.

1.2.1 Influence of Computing Capability on Propagation Methods

In 1975 Wackernagel listed five factors that 'influence the performance of an orbit determination system' [36].

- The completeness of the mathematical theory.
- The models used to approximate the physical world and the statistical nature of the data used for data processing.
- The quality of the data.
- The available computing hardware.
- The number of objects supported.

Because of the heavy use of computers involved with orbit propagation and determination, methods must fit the computation facilities available. When many objects are to be propagated and their states determined, the balance has

been achieved by trading accuracy for computational speed. Historically, computer time was precious and, in some cases, thousands of objects had to be tracked. Propagation methods were reduced in accuracy to meet the minimum required levels while using the limited computer time. Computing availability, the physical models and mathematical theory can be interchanged to provide a propagation system for a multi-satellite constellation. When maximizing accuracy without concern for computational speed, orbit propagation is done using purely numerical techniques, or direct numerical integration of the equations of motion with very high precision physical models. Accuracy of these techniques depends on the force model, integration method, and the time step used. Some situations may be better served by mean elements[1], however. Unfortunately, computing limitations restrict the way orbit propagation is done, especially when large numbers of satellites are involved.

Parallel computing changes the way designers look at computing availability. Using good parallel software design, more computing capability is available by simply adding additional processors to the computer system, without any changes in the software. The concept of infinite computing power becomes more accessible, bringing with it new and better ways of solving problems.

1.2.2 Methods of Orbit Propagation

1.2.2.1 The Equations of Motion

Central to understanding orbit propagation techniques is understanding the equations that describe satellite motion. Work with these equations has been ongoing since 1666, when Newton first discovered the law of gravitation [38]. Starting with the law of gravitation, the equations of relative motion are easily derived after making the two-body assumptions [38]. A full description of this derivation is available in many texts, such as Bate, Mueller and White [38].

---

[1]Mean elements describe the average, or mean, satellite position. Maneuver planning is an example where mean elements can be used more successfully than osculating elements, which describe the true state of the satellite [33].

Adding the perturbative effects to the two-body motion results in equations 1-1.

$$\ddot{\underline{r}} = \frac{-\mu}{r^3}\underline{r} + \underline{q} \qquad (1\text{-}1)$$

where $\underline{r}$ is a three element vector describing the position of a satellite in Cartesian space.

$\ddot{\underline{r}}$ is the second derivative of the position with respect to time

$-\frac{\mu}{r^3}\underline{r}$ is the two body force on the satellite

μ is the gravitational constant Gm, with gravitational constant G and the mass of the central body m.

$\underline{q}$ describes the sum all the perturbative forces on a satellite.

The solution of equation 1-1, without the perturbative force $\underline{q}$, is a conic section [38]. This discussion can be simplified to include only satellites that remain in a finite space about their central body. The escape trajectories, the parabolic and hyperbolic solutions to equation 1-1, will not be considered. Considering only ellipses (circles can be considered special forms of ellipses), the position of a satellite can be described at any instant in time using five constants that describe the shape and orientation of the ellipse, and one variable that describes the position of the satellite in the ellipse. There are many sets of parameters, often referred to as element sets, that will describe the orbit and the position of the satellite at any point in time. The Keplerian orbital elements are the most familiar since they describe the position of the satellite in geometric terms. An additional element set of interest in this thesis is the equinoctial elements. Both sets are described in Appendix A.

1.2.2.2 The VOP Equations

The variation of parameters (VOP) method of formulating orbital motion can be very helpful, especially when effect of the perturbations is small compared to the unperturbed motion [38]. The VOP equations describe the perturbed motion of a satellite in terms of one of the element sets. Therefore, the physical effects of perturbations on a satellite's orbit are more easily seen than in equation 1-1. The following derivation presents the basic formulation of the VOP equations of motion, as presented by Battin [5, 39].

Separating equation 1-1 into two first order equations gives equations 1-2.

$$\frac{d\underline{r}}{dt} = \underline{v} \qquad \frac{d\underline{v}}{dt} + \frac{\mu}{r^3}\underline{r} = \underline{a_d}(t) + \left[\frac{\partial R}{\partial \underline{r}}\right]^T \tag{1-2}$$

where:

$\underline{r} = \underline{r}(t, \underline{\alpha})$ is the position vector

$\underline{v} = \underline{v}(t, \underline{\alpha})$ is the velocity vector

$\underline{\alpha}$ is a vector containing the six orbital elements $(a, e, i, \omega, \Omega, \tau)$. The quantity τ is the time of pericenter passage.

$\left[\frac{\partial R}{\partial \underline{r}}\right]^T$ is the gradient of the disturbing potential. The disturbing potential contains the conservative perturbations to the two-body motion.

$\underline{a_d}$ is the sum of the non-conservative disturbing accelerations.

By the chain rule of differentiation, the ordinary derivatives of the position and velocity can be transformed into partial derivatives [5].

$$\frac{d\underline{r}}{dt} = \frac{\partial \underline{r}}{\partial t} + \frac{\partial \underline{r}}{\partial \underline{\alpha}}\frac{d\underline{\alpha}}{dt}$$
$$\frac{d\underline{v}}{dt} = \frac{\partial \underline{v}}{\partial t} + \frac{\partial \underline{v}}{\partial \underline{\alpha}}\frac{d\underline{\alpha}}{dt} \tag{1-3}$$

At any instant in time, the disturbed positions and velocities are the same as the two-body position and velocity. The orbital elements of the disturbed and two body satellites are different from the two body orbital elements, however [39]. This is illustrated in Figure 1-3.

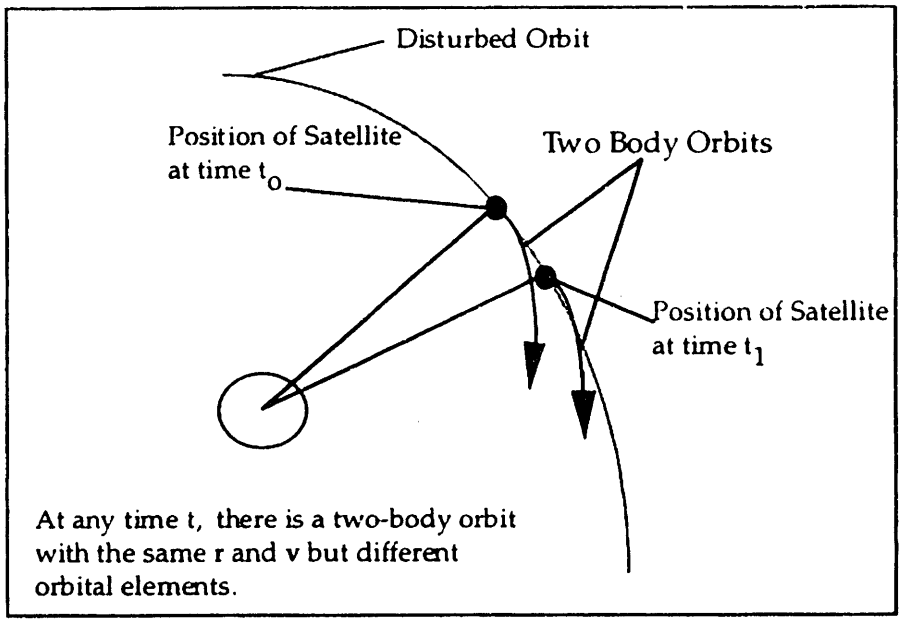


Figure 1-3: The Relationship of the Disturbed and Two-Body Orbits

If the two-body orbit is used in equations 1-3, the second term on the right hand side of both equations will go to zero; the vector $\underline{\alpha}$ does not change with time for two-body motion. Thus, the partial derivative of the two-body position and velocity vector is equal to the ordinary derivative.

As shown in Figure 1-3, at any point in time, velocity vector of the perturbed orbit is the same as the velocity vector of the two body orbit. Therefore, the partial and ordinary derivatives of the position vector are equal for both the disturbed motion as well as the two-body motion. This is shown in equation 1-4a.

$$\frac{d\underline{r}}{dt} = \frac{\partial \underline{r}}{\partial t} \tag{1-4a}$$

The partial derivative of the velocity vector with respect to time represents the two-body acceleration of the satellite [5]. This is shown is equation 1-4b.

$$\frac{\partial \underline{v}}{\partial t} = -\frac{\mu}{r^3}\underline{r} \tag{1-4b}$$

Rearranging equation 1-3 and applying equations 1-4a and 1-4b results in equations 1-5.

$$\begin{aligned} \frac{d\underline{r}}{dt} - \frac{\partial \underline{r}}{\partial t} &= \frac{\partial \underline{r}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = 0 \\ \frac{d\underline{v}}{dt} - \frac{\partial \underline{v}}{\partial t} &= \frac{\partial \underline{v}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = a_d + \left[\frac{\partial R}{\partial \underline{r}}\right]^T \end{aligned} \tag{1-5}$$

Equations 1-5 are the "required six scalar differential equations to be satisfied by the vector of orbital elements [5]." This fact can be seen more clearly in equations 1-6 and 1-7.

$$\frac{\partial \underline{r}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = 0 \tag{1-6}$$

$$\frac{\partial \underline{v}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = \underline{a_d} + \left[\frac{\partial R}{\partial \underline{r}}\right]^T \tag{1-7}$$

1.2.2.3 Lagrange Planetary Equations

The Lagrange Planetary Equations are a form of the VOP equations which include only the conservative perturbations on a satellite. Setting $\underline{a_d} = 0$ in equation 1-7 gives equations 1-8.

$$\frac{\partial \underline{r}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = 0 \tag{1-8a}$$

$$\frac{\partial \underline{v}}{\partial \underline{\alpha}} \frac{d\underline{\alpha}}{dt} = \left[\frac{\partial R}{\partial \underline{r}}\right]^T \tag{1-8b}$$

Equations 1-8 can be put into a more familiar form by first multiplying equation 1-8a by $\left[\frac{\partial \underline{r}}{\partial \underline{\alpha}}\right]^T$ and equation 1-8b by $\left[\frac{\partial \underline{v}}{\partial \underline{\alpha}}\right]^T$, and then subtracting the first from the second. The result is shown in equation 1-9 [5].

$$\underline{\underline{L}}\frac{d\underline{\alpha}}{dt} = \left[\frac{\partial R}{\partial \underline{\alpha}}\right]^T \tag{1-9}$$

where: $$\underline{\underline{L}} = \left[\frac{\partial \underline{r}}{\partial \underline{\alpha}}\right]^T \frac{\partial \underline{v}}{\partial \underline{\alpha}} - \left[\frac{\partial \underline{v}}{\partial \underline{\alpha}}\right]^T \frac{\partial \underline{r}}{\partial \underline{\alpha}}$$

The matrix $\underline{\underline{L}}$ is known as the Lagrange matrix [5]. The Lagrange matrix is a six by six matrix which is skew symmetric matrix and not an explicit function of time [5]. To solve directly for the rate of change of the orbital elements, the inverse of the Lagrange matrix must be determined. This matrix is known as the Poisson matrix, shown in equation 1-10 [39].

$$\frac{d\underline{\alpha}}{dt} = \underline{\underline{P}}^T\left[\frac{\partial R}{\partial \underline{\alpha}}\right] \tag{1-10}$$

where:

$\underline{\underline{P}} = -\underline{\underline{L}}^{-1}$ or $\underline{\underline{P}}^T = \underline{\underline{L}}^{-1}$ as $\underline{\underline{P}}$ and $\underline{\underline{L}}$ are skew symmetric.

$\underline{\underline{P}}$ is known as the Poisson matrix.

Battin describes the derivation of the Keplerian VOP equations from equation 1-9, known as Lagrange's Planetary Equations [5]. More important to this study, however are the VOP equations in equinoctial elements. These equations can be found in Cefola and Broucke [74].

1.2.2.4 General and Special Perturbation Theories

Historically, orbit propagation could account for perturbations using two distinct methods, general and special perturbations [4, 5]. General perturbation methods allow for the prediction of a satellite state to be attained directly, without using numerical integration methods. Satellite states are

represented as a function of time. Special perturbations, on the other hand, calculate the satellite state rates and then step forward in time using a numerical integration method. Both methods have advantages.

General methods use the relatively small difference between Keplerian and non-Keplerian potentials on a satellite. The potential on a satellite can be expressed in Equation 1-11 [4].

$$U = U_{\mu} + R \tag{1-11}$$

where: U is the total potential on a satellite

U_{μ} is the potential from the spherical central body

R is the potential due to the perturbations.

The Keplerian solution for the motion of a satellite only includes the two-body potential U_{μ}. General theories represent the solution to the perturbed motion (the potential U) of a satellite as slowly changing orbital elements. This can be done because of the large difference between U_{μ} and R [4]. This view of orbital perturbations expresses only the conservative forces on a satellite. Non-conservative forces, drag and solar radiation pressure are included in several of the general perturbation theories but many assumptions must be made to include non-conservative forces [75].

When using general perturbation methods the amount of processing required for the calculation of a future state of a satellite is independent of the amount of time between the initial state and the request time. This can be a valuable tool and is the largest single advantage of general perturbation theories. However, because most theories in use truncate at relatively low powers of a small parameter, the accuracy of these techniques is limited. This is especially apparent when compared to purely numerical methods for long time spans. Additionally, developing a general perturbation method requires significant effort to create analytic formulations for each perturbation that is included in the theory.

Special perturbations require less mathematical development in the theory although the accurate determination of the perturbative accelerations can be very complex. The simplicity of the theory, once the perturbative forces are calculated, can be seen more easily in Equation 1-1, the general form of the equation of motion of a satellite in Cartesian space.

A Cowell technique uses the numerical propagation scheme to integrate equation 1-1 forward in time. This method can result in very accurate predictions if the quantity $\underline{q}$ is thoroughly developed. As the time difference between the initial state and the requested value increases, more computer processing is required since each integration-step requires one or more re-calculations of the rates at that point of time. The re-calculation of the rates are very expensive in terms of computer time.

Step sizes in a numerical integration scheme are determined by the frequency content in the rates, the desired accuracy and the method of integration used [7]. This can be easily seen in the mathematics of a numerical integration method. Given the initial condition $\bar{x}_o$, describing the state of a satellite at time t_o , the differential equation

$$\bar{x}' = f(\bar{x},t)$$

$$\text{with the initial conditions } \bar{x} = \bar{x}_o \text{ at time } t = t_o \qquad (1\text{-}12)$$

where: $\bar{x}'$ is the rate of change of $\bar{x}$ at time t

can be solved for a future time, $t_o + \Delta t$. Note the i'th rate is dependent on the entire state $\bar{x}$. After taking n steps of size Δt , the solution to the equation at time $t_o + n\Delta t$ will be found. Of course, using a numerical integration method to solve Equation 1-12 introduces error. The error in each state is described in Equation 1-13 [7].

$$\bar{x}_i(t_o + n\Delta t) - \overline{x_{n_i}} = T + R + \vartheta \tag{1-13}$$

where: $\bar{x}_i(t_o + n\Delta t)$ is the true solution at time $t_o + n\Delta t$

$\overline{x_{n_i}}$ is the numerically integrated state at time $t_o + n\Delta t$

T is the truncation error in the i'th element

R is the round off error in the i'th element

ϑ is the error in the i'th element due to evaluating f_i at $(\bar{x}_n, t)$ rather than $(\bar{x}(t_o + n\Delta t), t)$.

Truncation error is described as the difference between the numerical method used and the infinite series Taylor expansion [7]. A numerical method solution is normally described as a 'nth order method', as the truncation error differs from the numerical solution by the time step to the power of n+1. For example, Runge Kutta Four matches the Taylor series expansion through fourth order.

The round off error is dependent on the number of decimal places used in performing the calculations. It limits the total number of steps that can be taken so that the solution can still be trusted [7]. Modern computers now have very good precision, but this limit still prevents step sizes from becoming exceedingly small.

Finally, the error ϑ describes the error introduced by evaluating f_i using the incorrect value of $\bar{x}$. This error and the truncation error control the upper bound on the step size that can be used, while the lower boundary is controlled by the number steps required to get to a desired time as well as round off error [7]. In order to reduce computation time, the largest time step that can be used without introducing excessive error should be used. By using a higher order method, the upper limit on the step size due to truncation error can be extended, if necessary. A harder look at the error ϑ will show how to lengthen the step size limit imposed by ϑ.

From Kreyszig [7], the error ϑ_i in the i'th element of $\bar{x}$ at the n+1 time step to first order in Δt is described in Equation 1-14.

$$\vartheta_i = \left(f_i\left(\bar{x}(t_n + n\Delta t), t\right) - f_i\left(\overline{x_n}, t\right)\right)\Delta t = \frac{\partial f_i\left(\overline{\overline{x}}, t\right)}{\partial \bar{x}} \eta_n \Delta t$$

(1-14)

where: $\frac{\partial f_i}{\partial \bar{x}}$ is the partial derivative of the i'th state rate with respect to the state vector, the result being a row vector.

$\overline{\overline{x_n}}$ lies between $\bar{x}(t_n + n\Delta t)$ and $\overline{x_n}$ in accordance with the mean value theorem.

$\eta_n = \bar{x}(t_n + n\Delta t)$ - $\overline{x_n}$ or the column vector of errors in $\overline{x_n}$

Equation 1-14 shows that the error contributed by ϑ to each state is approximately the difference between the true and numerically propagated rates multiplied by the time step. In order to lengthen the time step one must try to minimize $\frac{\partial f_i}{\partial \bar{x}}$, as this error is directly dependent on the time step. In terms of Equation 1-1, this value describes the rate at which the state rates change with respect to the states or the frequency content of the right hand side.

1.2.2.5 Semianalytic Techniques

Semianalytic techniques of orbit propagation attempt to take advantage of both the numerical techniques of special perturbation theories and analytic development of general theories. The goal of these methods is to attain the accuracy of numerical techniques and the speed advantages of general methods. Additionally, semianalytic theories also provide mean elements, which are discussed in section 1.2.3.1.

The motivation for developing semianalytic methods is derived from the previous analysis of numerical integration. If the frequency content, $\frac{\partial f_i}{\partial \bar{x}}$ can be reduced, a larger step size can be used in the integration process. This not only increases the speed of the integration process by reducing the number of

steps taken but also reduces the effect of round off error. In a semianalytic theory, the frequency content of the right hand sides are kept to a minimum using the variation of parameters (VOP) equations discussed in section 1.2.2.2 and the generalized method of averaging. Applying averaging to the VOP equations removes the high frequency terms whose secular perturbative effect on a orbit average to zero. This significantly reduces the rate of change of the satellite rates, resulting in smaller $\frac{\partial f_i}{\partial x}$'s and larger step sizes.

1.2.3 Draper Semianalytic Satellite Theory

1.2.3.1 Overall Outline

The software implementing the Draper Semianalytic Satellite Theory (DSST) was developed in the late 1970's and early 1980's. Engineers began to work on it at the Computer Sciences Corporation and continued at Draper Laboratory in Cambridge, MA, where graduate students also contributed to the development [8]. Refinement of the theory and associated software has continued since that time to the present day. The mathematics of the theory is discussed in several reports. The single most complete document has been published by the Naval Postgraduate School [8]. The accuracy of the DSST has been well tested through numerous studies and work with the software [33, 40]. Because the theory is accurate and computationally efficient for long term, high precision predictions, a version of the DSST was used for parallel orbit propagation. The DSST will therefore be discussed in full detail. It is especially important to highlight the difference between mean and osculating elements. The basic flow of the derivation comes from the work of McClain [9]. This derivation will only examine the simplest form of the DSST, where the averaging interval is the period of the satellite. This constraint prevents the inclusion of resonance, which requires more complex averaging intervals.

A generalized form of the Variation of Parameters (VOP) equations is used to start the derivation. As shown in section 1.2.2.2, the VOP equations describe the rate of change of a satellite's in orbital elements. The DSST uses the non-

singular equinoctial element set, thus avoiding problems when propagating near circular, equatorial, or polar orbits. A description of the Keplerian elements, the equinoctial elements, and the relationship between the two can be found in Appendix A.

The VOP equations are expressed in terms of the equinoctial elements. The left hand side of this first equation is then transformed using the near identity transformation and a Taylor's expansion about the mean element rates. The near identity transformation is used to relate the osculating, or actual value of the elements, to the averaged elements through a power series expansion in a small parameter. The mean element rates are then represented as a power series expansion of the same small parameter and functions of the five slowly varying mean elements.

The right hand side of the first equation, the generalized VOP equation, is transformed using a Taylor series expansion. These functions are expanded about the six mean elements. The near identity transformation is used again to express the Taylor series expansion as a power series in the small parameter.

With both sides of the first VOP equation expanded in the small parameter, terms of like powers in the small parameter are equated. These equations are then averaged over the fast variable or one orbital period of the satellite. The equations for the mean element rates are finally determined as functions of the average contributions of the perturbing functions. These rates are of very low frequency; they do not change rapidly with time. This reduces the size of the parameter $\frac{\partial f_i}{\partial \bar{x}}$ so longer step sizes can be used when numerically integrating the mean element rates.

1.2.3.2 Equations of Averaging

The derivation of the simplest form of the DSST begins with a generalization of the VOP equations [9].

$$\frac{da_i}{dt} = \varepsilon F_i(\mathbf{a},\lambda) \qquad i = [1,2,...,5]$$
$$\frac{d\lambda}{dt} = n(a_1) + \varepsilon F_6(\mathbf{a},\lambda) \tag{1-15}$$

where: a is the vector of the five slowly varying orbital elements

λ is the fast variable

n is the mean motion

ε is the small parameter

F_i is the function describing the time rate of change of the i'th element caused by perturbative forces.

All elements in this first expression of the VOP Equations of Motion are osculating elements, representing the true state of the satellite. The quantity ε is called the small parameter because of its relative size. There is always a small constant associated with perturbative forces because their effects are small relative to the motion caused by the two-body forces. The unperturbed equations can be seen if ε is set to zero; the equations become exactly the two-body equations of motion represented in equinoctial elements.

$$\frac{da_i}{dt} = 0 \qquad i = [1,2,...,5] \qquad \frac{d\lambda}{dt} = n(a_1) \tag{1-16}$$

The near identity transformation, equation 1-17, relates mean and osculating elements through the small parameter ε. Equation 1-17 is an important concept for the Generalized Method of Averaging [9]:

$$a_i = \bar{a}_i + \varepsilon\eta_{i,1} + \varepsilon^2\eta_{i,2} + ... \qquad i = [1,2,...,5]$$
$$\lambda = \bar{\lambda} + \varepsilon\eta_{6,1} + \varepsilon^2\eta_{6,2} + ... \tag{1-17}$$

where: $\bar{a}_i$ represents the i'th mean equinoctial element (a mean element designated by the overbar).

η represents 2π periodic functions, dependent on the six mean equinoctial elements.

The near identity transformation states that the osculating elements are dependent on the mean elements plus an infinite series in the small parameter. The small parameter is multiplied by the periodic functions, hereafter referred to as the short periodic functions. It is important to point out the osculating elements represented in equation 1-17 are dependent on the mean elements, including the mean fast variable, and the short periodic functions. The purpose of applying the averaging operator is to remove the fast variable dependence and the short periodic functions from the equations of motion. This alternate set of equations of motion will be called the mean equations of motion.

The next step involves assuming a form for the mean equations of motion. The mean element rates are expressed as an expansion in the small parameter and functions of the slowly varying mean elements.

$$\frac{d\bar{a}_i}{dt} = \varepsilon A_{i,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{i,2}(\bar{\mathbf{a}}) + \ldots \quad i = [1,2,\ldots,5]$$

$$\frac{d\bar{\lambda}_i}{dt} = n(\bar{a}_1) + \varepsilon A_{6,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{6,2}(\bar{\mathbf{a}}) + \ldots \tag{1-18}$$

Equations 1-17 and 1-18 are assumed forms for the osculating and mean elements, respectively. The rest of this derivation will demonstrate how to calculate the short periodic functions $\eta_{i,j}$ and mean functions, $A_{i,j}$

By differentiating the near-identity transformation with respect to time we achieve an equation relating the mean and osculating element rates.

$$\frac{da_i}{dt} = \frac{d\bar{a}_i}{dt} + \varepsilon \sum_{k=1}^{6} \frac{\partial \eta_{i,1}}{\partial \bar{a}_k} \frac{d\bar{a}_k}{dt} + \varepsilon^2 \sum_{k=1}^{6} \frac{\partial \eta_{i,2}}{\partial \bar{a}_k} \frac{d\bar{a}_k}{dt} + \ldots$$

$$\frac{d\lambda}{dt} = \frac{d\bar{\lambda}}{dt} + \varepsilon \sum_{k=1}^{6} \frac{\partial \eta_{6,1}}{\partial \bar{a}_k} \frac{d\bar{a}_k}{dt} + \varepsilon^2 \sum_{k=1}^{6} \frac{\partial \eta_{6,2}}{\partial \bar{a}_k} \frac{d\bar{a}_k}{dt} + \ldots$$

$$i = [1,2,\ldots,5] \qquad (1\text{-}19)$$

where: a_0 takes the place of λ in the summation.

Substituting equation 1-18 into equation 1-19 gives

$$\frac{da_i}{dt} = \varepsilon A_{i,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{i,2}(\bar{\mathbf{a}}) + \ldots$$
$$+ \varepsilon \frac{\partial \eta_{i,1}}{\partial \bar{a}_6} n(\bar{a}_1) + \varepsilon \sum_{k=1}^{6} \frac{\partial \eta_{i,1}}{\partial \bar{a}_k} [\varepsilon A_{6,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{6,2}(\bar{\mathbf{a}}) +] \ldots$$
$$+ \varepsilon^2 \frac{\partial \eta_{i,2}}{\partial \bar{a}_6} n(\bar{a}_1) + \varepsilon^2 \sum_{k=1}^{6} \frac{\partial \eta_{i,2}}{\partial \bar{a}_k} [\varepsilon A_{i,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{i,2}(\bar{\mathbf{a}}) + \ldots] + \ldots$$

$$i = [1,2,\ldots,5]$$

$$\frac{d\lambda}{dt} = [n(\bar{a}_1) + \varepsilon A_{6,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{6,2}(\bar{\mathbf{a}}) + \ldots]$$
$$+ \varepsilon [\frac{\partial \eta_{6,1}}{\partial \bar{a}_k} n(\bar{a}_1) + \sum_{k=1}^{6} \frac{\partial \eta_{6,1}}{\partial \bar{a}_k} [\varepsilon A_{k,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{k,2}(\bar{\mathbf{a}}) + \ldots]$$
$$+ \varepsilon^2 [\frac{\partial \eta_{6,2}}{\partial \bar{a}_k} n(\bar{a}_1) + \sum_{k=1}^{6} \frac{\partial \eta_{6,2}}{\partial \bar{a}_k} [\varepsilon A_{k,1}(\bar{\mathbf{a}}) + \varepsilon^2 A_{k,2}(\bar{\mathbf{a}}) + \ldots] + \ldots \qquad (1\text{-}20)$$

Even though all equations have only been expanded out to second order in ε, terms of up to fourth order are present. The semianalytic theory takes advantage of the fact ε is small. Only in a few cases is it necessary to expand out ε beyond first order [41]. Equations 1-20 can be reduced by combining like powers of ε and ignoring terms of third and higher order.

$$\frac{da_i}{dt} = \varepsilon[A_{i,1}(\bar{\mathbf{a}}) + \frac{\partial \eta_{i,1}}{\partial \bar{a}_6} n(\bar{a}_1)]$$
$$+\varepsilon^2[A_{i,2}(\bar{\mathbf{a}}) + \sum_{k=1}^{6} \frac{\partial \eta_{i,1}}{\partial \bar{a}_k} A_{k,1}(\bar{\mathbf{a}}) + \frac{\partial \eta_{i,2}}{\partial \bar{a}_k} n(\bar{a}_1)]$$

$$i = [1,2,\ldots,5]$$

$$\frac{d\lambda}{dt} = n(\bar{a}_1) + \varepsilon[A_{6,1}(\bar{\mathbf{a}}) + \frac{\partial \eta_{6,1}}{\partial \bar{a}_6} n(\bar{a}_1)]$$
$$+\varepsilon^2[A_{6,2}(\bar{\mathbf{a}}) + \frac{\partial \eta_{6,2}}{\partial \bar{a}_6} n(\bar{a}_1) + \sum_{k=1}^{6} \frac{\partial \eta_{6,1}}{\partial \bar{a}_k} A_{k,1}(\bar{\mathbf{a}})] \qquad (1\text{-}21)$$

The osculating element rates are now represented as functions of the mean elements and short periodic functions. These expansions for the osculating element rates will later be substituted into the left hand side of equations 1-15. The right hand side of the functions in equations 1-15 will now be expanded using a Taylor series expansion about the mean elements. With both sides of equation 1-15 expanded into a power series in the small parameter, like terms can be equated generating equations for the mean functions $A_{i,j}$.

$$F_i(\mathbf{a},\lambda) = F_i(\bar{\mathbf{a}},\bar{\lambda}) + \sum_{k=1}^{6} \Delta a_k \frac{\partial F_i}{\partial \bar{a}_k} + \frac{1}{2}(\sum_{k=1}^{6} \Delta a_k \frac{\partial}{\partial \bar{a}_k})^2 F_i + \ldots \qquad i = [1,2,\ldots,6]$$

(1-22)

where: Δa_k is the difference between the k'th osculating and mean element.

The last term of equation 1-22 can be re-written to make the number of terms more apparent:

$$F_i(\mathbf{a},\lambda) = F_i(\bar{\mathbf{a}},\bar{\lambda}) + \sum_{k=1}^{6} \Delta a_k \frac{\partial F_i}{\partial \bar{a}_k} + \frac{1}{2}\sum_{k=1}^{6}\sum_{l=1}^{6} \Delta a_k \Delta a_l \frac{\partial}{\partial \bar{a}_k}\frac{\partial}{\partial \bar{a}_l}(F_i) + \ldots \qquad i = [1,2,\ldots,6]$$

(1-23)

This expansion can be reduced using the near-identity transformation, equation 1-17. Subtracting the mean elements from both sides of equation 1-17 and ignoring terms of third order and higher in ε gives:

$$\begin{aligned} a_i - \bar{a}_i &= \Delta a_i = \varepsilon\eta_{i,1} + \varepsilon^2\eta_{i,2} \qquad i = [1,2,\ldots,5] \\ \lambda - \bar{\lambda} &= \Delta\lambda_i = \varepsilon\eta_{6,1} + \varepsilon^2\eta_{6,2} \end{aligned} \tag{1-24}$$

Replacing Δa_k in equation 1-23 with equation 1-24 gives:

$$\begin{aligned} F_i(\mathbf{a},\lambda) &= F_i(\bar{\mathbf{a}},\bar{\lambda}) \\ &\quad + \sum_{k=1}^{6}[\varepsilon\eta_{k,1} + \varepsilon^2\eta_{k,2}]\frac{\partial F_i}{\partial \bar{a}_k} + \\ &\quad \frac{1}{2}[\sum_{k=1}^{6}\sum_{l=1}^{6}[\varepsilon\eta_{k,1} + \varepsilon^2\eta_{k,2}][\varepsilon\eta_{l,1} + \varepsilon^2\eta_{l,2}]\frac{\partial}{\partial \bar{a}_k}\frac{\partial}{\partial \bar{a}_l}(F_i)] \\ i &= [1,2,\ldots,6] \end{aligned} \tag{1-25}$$

Again, the equation 1-25 is simplified by combining like terms through second power in ε.

$$\begin{aligned} F_i(\mathbf{a},\lambda) &= F_i(\bar{\mathbf{a}},\bar{\lambda}) \\ &\quad + \varepsilon[\sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i}{\partial \bar{a}_k}] \\ &\quad + \varepsilon^2[\sum_{k=1}^{6}\eta_{k,2}\frac{\partial F_i}{\partial \bar{a}_k} + \sum_{k=1}^{6}\sum_{l=1}^{6}\eta_{k,1}\eta_{l,1}\frac{\partial F_i}{\partial \bar{a}_k}\frac{\partial F_i}{\partial \bar{a}_l}] \\ i &= [1,2,\ldots,6] \end{aligned} \tag{1-26}$$

The mean motion, $n(a_1)$, is the only osculating variable left in equations 1-15 This can also be expanded about $\bar{a}_1$ in a Taylor series expansion.

$$n(a_1) = \bar{n} + (a_1 - \bar{a}_1)\frac{\partial n}{\partial \bar{a}_1} + \frac{1}{2}(a_1 - \bar{a}_1)^2 \frac{\partial^2 n}{\partial \bar{a}_1^2} + \ldots \tag{1-27}$$

Applying the modified form of the near identity transformation, equation 1-24 where k=1, to equation 1-27 gives:

$$n(a_1) = \bar{n} + [\varepsilon\eta_{1,1} + \varepsilon^2\eta_{1,2}]\frac{\partial \bar{n}}{\partial \bar{a}_1} + \frac{1}{2}[\varepsilon\eta_{1,1} + \varepsilon^2\eta_{1,2}]^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1^2} + \ldots \tag{1-28}$$

Again getting rid of the third and higher order terms in the small parameter leaves equation 1-29.

$$n(a_1) = \bar{n} + \varepsilon\eta_{1,1}\frac{\partial \bar{n}}{\partial \bar{a}_1} + \varepsilon^2[\eta_{1,2}\frac{\partial \bar{n}}{\partial \bar{a}_1} + \frac{1}{2}\eta_{1,1}^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1^2}] \tag{1-29}$$

Substituting 1-29, 1-26, and 1-21 into equations 1-15 and then combining like orders of the small parameter results in equations relating the functions of the mean element rates to the 2π periodic functions.

Setting the terms of the first order coefficients equal gives:

$$\begin{aligned} A_{i,1} + \bar{n}\frac{\partial \eta_{i,1}}{\partial \bar{a}_6} &= F_i(\bar{\mathbf{a}}, \bar{\lambda}) \quad i = [1,2,\ldots,5] \\ A_{6,1}(\bar{\mathbf{a}}) + \bar{n}\frac{\partial \eta_{6,1}}{\partial \bar{a}_6} &= \eta_{1,1}\frac{\partial \bar{n}}{\partial \bar{a}_1} + F_6(\bar{\mathbf{a}}, \bar{\lambda}) \end{aligned} \tag{1-30}$$

while the second order terms create equations

$$\begin{aligned} A_{i,2} + \bar{n}\frac{\partial \eta_{i,2}}{\partial \bar{a}_6} + \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{i,1}}{d\bar{a}_k} &= \sum_{k=1}^{6} \eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \quad i = [1,2,\ldots,5] \\ A_{6,2} + n\frac{\partial \eta_{6,2}}{\partial \bar{a}_6} + \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{6,1}}{d\bar{a}_k} &= \sum_{k=1}^{6} \eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} + \eta_{1,2}\frac{\partial \bar{n}}{\partial \bar{a}_1} + \frac{1}{2}\eta_{1,1}^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1^2} \end{aligned} \tag{1-31}$$

The next step is to solve for the $A_{i,j}$ functions. The equations 1-30 and 1-31 provide the expressions to determine the $A_{i,j}$ functions, as everything in the above equations is known, except for the η functions. In the original definitions of the functions η, the only constraint imposed upon them were that they be 2π periodic in the fast variable, λ. Application of the averaging operator to equations 1-30 and 1-31 will develop the averaged equations of motion by removing the 2π periodic functions from those expressions.

1.2.3.3 Averaging

The Generalized Method of Averaging is used to remove the high frequency terms from the equations of motion. The Generalized Method of Averaging removes the variable of interest from the equation through integration. The averaging operator is defined as

$$\langle g(x) \rangle_x = \frac{1}{2\pi} \int_0^{2\pi} g(x)\, dx \qquad (1\text{-}32)$$

where g(x) is the function to be averaged
x is the variable to be averaged over
0-2π is the interval over which the average value is determined

Application of the averaging operator to equation 1-32 removes the variable x from the resulting equation. Similarly, averaging the equations of motion over the fast variable, λ, will remove from the fast variable dependence from the equations of motion. This will result in a set of first order, slowly varying differential equations. The averaged equations of motion can then be numerically integrated with a much larger time step than those that depended on the fast variable.

The averaging operator has many properties which will be useful in the following sections. These properties come from [9].

$$\left\langle X(\bar{a},\bar{\lambda})+Y(\bar{a},\bar{\lambda})\right\rangle_{\lambda}=\left\langle X(\bar{a},\bar{\lambda})\right\rangle_{\lambda}+\left\langle Y(\bar{a},\bar{\lambda})\right\rangle_{\lambda}$$

Superposition Principle

$$\left\langle cX(\bar{a},\bar{\lambda})\right\rangle_{\lambda}=c\left\langle X(\bar{a},\bar{\lambda})\right\rangle_{\lambda}$$

$$\left\langle \frac{\partial}{\partial \bar{a}_k} X(\bar{a},\bar{\lambda})\right\rangle_{\lambda}=\frac{\partial}{\partial \bar{a}_k}\left\langle X(\bar{a},\bar{\lambda})\right\rangle_{\lambda} \quad k=[1,2,\ldots,6] \tag{1-33}$$

Properties of Linear Operators

1.2.3.4 The Averaged Equations of Motion

Applying the averaging operator to the equations 1-30 and 1-31 and using the properties in equations 1-33 gives:

$$\left\langle A_{i,1}\right\rangle+\left\langle \bar{n}\frac{\partial \eta_{i,1}}{d\bar{a}_6}\right\rangle=\left\langle F_i(\bar{\mathbf{a}},\bar{\lambda})\right\rangle \quad i=[1,2,\ldots,6]$$

$$\left\langle A_{6,1}(\bar{\mathbf{a}})\right\rangle+\left\langle \bar{n}\frac{\partial \eta_{6,1}}{\partial \bar{a}_6}\right\rangle=\left\langle \eta_{1,1}\frac{\partial \bar{n}}{\partial \bar{a}_1}\right\rangle+\left\langle F_6(\bar{\mathbf{a}},\bar{\lambda})\right\rangle \tag{1-34}$$

$$\left\langle A_{i,2}\right\rangle+\left\langle \bar{n}\frac{\partial \eta_{i,2}}{\partial \bar{a}_6}\right\rangle+\left\langle \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{i,1}}{d\bar{a}_k}\right\rangle=\left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle \quad i=[1,2,\ldots,6]$$

$$\left\langle A_{6,2}\right\rangle+\left\langle n\frac{\partial \eta_{6,2}}{\partial \bar{a}_6}\right\rangle+\left\langle \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{6,1}}{d\bar{a}_k}\right\rangle=\left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle+\left\langle \eta_{1,2}\frac{\partial \bar{n}}{\partial \bar{a}_1}\right\rangle+\left\langle \frac{1}{2}\eta_{1,1}{}^{2}\frac{\partial^2 \bar{n}}{\partial \bar{a}_1^{\,2}}\right\rangle \tag{1-35}$$

where the averaging in equations 1-34 and 1-35 is done with respect to the fast variable λ.

The original definition of the short periodic function, $\eta_{i,j}$, requires it to be 2π periodic in λ. When averaged these functions are identically zero.

$$\left\langle \frac{\partial \eta_{i,1}}{d\bar{a}_6} \right\rangle = 0$$

(1-36)

Combining equation 1-36 with equations 1-34 and 1-35 and solving for the functions A leaves equations 1-37.

$$A_{i,1} = \left\langle F_i(\bar{\mathbf{a}}, \bar{\lambda}) \right\rangle \quad i = [1,2,\ldots,5]$$

$$A_{6,1}(\bar{\mathbf{a}}) = \left\langle \eta_{1,1} \frac{\partial \bar{n}}{\partial \bar{a}_1} \right\rangle + \left\langle F_6(\bar{\mathbf{a}}, \bar{\lambda}) \right\rangle \tag{1-37}$$

$$A_{i,2} = \left\langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_i(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle - \left\langle \sum_{k=1}^{6} A_{k,1} \frac{\partial \eta_{i,1}}{d\bar{a}_k} \right\rangle \quad i = [1,2,\ldots 5]$$

$$A_{6,2} = \left\langle \sum_{k=1}^{5} \eta_{k,1} \frac{\partial F_6(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle + \left\langle \eta_{1,2} \frac{\partial \bar{n}}{\partial \bar{a}_1} \right\rangle + \left\langle \frac{1}{2} \eta_{1,1}{}^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1^2} \right\rangle - \left\langle \sum_{k=1}^{6} A_{k,1} \frac{\partial \eta_{6,1}}{d\bar{a}_k} \right\rangle \tag{1-38}$$

Equations 1-37 and 1-38 can be further reduced by noting that the $A_{i,j}$ functions are not dependent on the fast variable, λ. Applying the properties described by equations 1-33 gives:

$$\left\langle \sum_{k=1}^{6} A_{k,1} \frac{\partial \eta_{i,1}}{d\bar{a}_k} \right\rangle = \sum_{k=1}^{6} A_{k,1} \left\langle \frac{\partial \eta_{i,1}}{d\bar{a}_k} \right\rangle = \sum_{k=1}^{6} A_{k,1} \frac{\partial \langle \eta_{i,1} \rangle}{d\bar{a}_k} = 0 \quad i = [1,2,\ldots,6] \tag{1-39}$$

$$\left\langle \eta_{1,1} \frac{\partial \bar{n}}{d\bar{a}_k} \right\rangle = \frac{\partial \bar{n}}{d\bar{a}_k} \left\langle \eta_{1,1} \right\rangle = 0 \tag{1-40}$$

Note that the short periodic functions in equations 1-39 and 1-40 are not multiplied by another function of the fast variable. The properties of equations 1-33 apply only if the function removed from the operator is considered a constant by the averaging operator.

The simplifications described in equations 1-39 and 1-40 allows equations 1-37 and 1-38 to completely specify the $A_{i,j}$ functions in terms of the averaged force contribution and an expansion of the mean motion.

$$A_{i,1} = \left\langle F_i(\bar{\mathbf{a}}, \bar{\lambda}) \right\rangle \quad i = [1,2,\ldots 6] \tag{1-41}$$

$$A_{i,2} = \left\langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_i(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle \quad i = [1,2,\ldots,5] \tag{1-42}$$

$$A_{6,2} = \left\langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_6(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle + \left\langle \frac{1}{2} \eta_{1,1}{}^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1{}^2} \right\rangle \tag{1-43}$$

Replacing equation 1-18 by the functions described by equations 1-41 through 1-43 completes the development of the averaged equations.

$$\frac{d\bar{a}_i}{dt} = \varepsilon \left\langle F_i(\bar{\mathbf{a}}, \bar{\lambda}) \right\rangle + \varepsilon^2 \left\langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_i(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle \quad \mathrm{i} = [1,2\ldots,5] \tag{1-44}$$

$$\frac{d\bar{\lambda}_i}{dt} = n(\bar{a}_1) + \varepsilon \left\langle F_i(\bar{\mathbf{a}}, \bar{\lambda}) \right\rangle + \varepsilon^2 \left\langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_6(\bar{\mathbf{a}}, \bar{\lambda})}{d\bar{a}_k} \right\rangle + \varepsilon^2 \left\langle \frac{1}{2} \eta_{1,1}{}^2 \frac{\partial^2 \bar{n}}{\partial \bar{a}_1{}^2} \right\rangle \tag{1-45}$$

One more thing is interesting to note about the averaged equations of motion. The second and higher order terms are not independent of the short periodic functions. The second order contributions $A_{i,2}$ depend on the first order short periodic functions.

1.2.3.5 Short Periodic Functions

With the averaged equations of motion explained, the next step is to develop equations for the short periodic functions. While the DSST numerically integrates the averaged equations of motion, analytical expressions are developed for the short periodic variations. These expressions are expanded in a Fourier Series and then integrating analytically. Like the previous derivation, this development follows the work cited in reference [9].

Subtracting equations 1-37 and 1-38 from equations 1-30 and 1-31 leaves equations 1-46 and 1-47.

$$n\frac{d\eta_{i,1}}{d\lambda} = F_i(\bar{\mathbf{a}},\bar{\lambda}) - \left\langle F_i(\bar{\mathbf{a}},\bar{\lambda})\right\rangle \quad i=[1,2,\ldots,5]$$

$$\bar{n}\frac{\partial\eta_{6,1}}{\partial\bar{a}_6} = \eta_{1,1}\frac{\partial\bar{n}}{\partial\bar{a}_1} - \left\langle \eta_{1,1}\frac{\partial\bar{n}}{\partial\bar{a}_1}\right\rangle + F_6(\bar{\mathbf{a}},\bar{\lambda}) - \left\langle F_6(\bar{\mathbf{a}},\bar{\lambda})\right\rangle \tag{1-46}$$

$$\bar{n}\frac{\partial\eta_{i,2}}{\partial\bar{a}_6} = \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle - \left[\sum_{k=1}^{6}A_{k,1}\frac{\partial\eta_{i,1}}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}A_{k,1}\frac{\partial\eta_{i,1}}{d\bar{a}_k}\right\rangle\right]$$
$$i=[1,2,\ldots,5]$$

$$n\frac{\partial\eta_{6,2}}{\partial\bar{a}_6} = \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle + \eta_{1,2}\frac{\partial\bar{n}}{\partial\bar{a}_1} - \left\langle \eta_{1,2}\frac{\partial\bar{n}}{\partial\bar{a}_1}\right\rangle$$
$$+\frac{1}{2}\eta_{1,1}^{\,2}\frac{\partial^2\bar{n}}{\partial\bar{a}_1^{\,2}} - \left\langle \frac{1}{2}\eta_{1,1}^{\,2}\frac{\partial^2\bar{n}}{\partial\bar{a}_1^{\,2}}\right\rangle - \left[\sum_{k=1}^{6}A_{k,1}\frac{\partial\eta_{6,1}}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}A_{k,1}\frac{\partial\eta_{6,1}}{d\bar{a}_k}\right\rangle\right] \tag{1-47}$$

Now, the equations 1-46 and 1-47 can be used to solve for the functions $\eta_{i,j}$.

$$\eta_{i,1} = \frac{1}{n}\int [F_i(\bar{\mathbf{a}},\bar{\lambda}) - \langle F_i(\bar{\mathbf{a}},\bar{\lambda})\rangle]\, d\bar{\lambda} \quad i = [1,2,\ldots,5]$$

$$\eta_{6,1} = \frac{1}{n}\int [\eta_{1,1}\frac{\partial \bar{n}}{\partial \bar{a}_1} - \left\langle \eta_{1,1}\frac{\partial \bar{n}}{\partial \bar{a}_1}\right\rangle + F_6(\bar{\mathbf{a}},\bar{\lambda}) - \langle F_6(\bar{\mathbf{a}},\bar{\lambda})\rangle]\, \partial\bar{\lambda} \tag{1-48}$$

$$\eta_{i,2} = \frac{1}{n}\int [\sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_i(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle + - \left[\sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{i,1}}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{i,1}}{d\bar{a}_k}\right\rangle\right]]\, d\bar{\lambda}$$

$$i = [1,2,\ldots,5]$$

$$\eta_{6,2} = \frac{1}{n}\int [\sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6}\eta_{k,1}\frac{\partial F_6(\bar{\mathbf{a}},\bar{\lambda})}{d\bar{a}_k}\right\rangle + \eta_{1,2}\frac{\partial \bar{n}}{\partial \bar{a}_1} - \left\langle \eta_{1,2}\frac{\partial \bar{n}}{\partial \bar{a}_1}\right\rangle + \frac{1}{2}\eta_{1,1}^{\,2}\frac{\partial^2 \bar{n}}{\partial \bar{a}_1^{\,2}} - \left\langle \frac{1}{2}\eta_{1,1}^{\,2}\frac{\partial^2 \bar{n}}{\partial \bar{a}_1^{\,2}}\right\rangle - \left[\sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{6,1}}{d\bar{a}_k} - \left\langle \sum_{k=1}^{6} A_{k,1}\frac{\partial \eta_{6,1}}{d\bar{a}_k}\right\rangle\right]]\, d\bar{\lambda} \tag{1-49}$$

With the short periodic equations of motion determined, they can be integrated to solve for the osculating elements at any time.

1.2.3.6 Interpolation

Before calculating the short periodic contributions to the equations of motion, it is valuable to mention how the implemented versions of the DSST actually calculate the mean elements at each request time. Because the short periodic functions are removed from the mean elements, long integrator step sizes can be used to calculate the mean elements. Typical step sizes used are a day or more [32]. Rather than numerically integrating to each request time an interpolator is used whenever possible. Long step sizes are used to propagate ahead of the next request time. The interpolator then generates a polynomial which describes the elements over the request interval. The mean elements can then be calculated for any time within the propagated time span by a simple polynomial evaluation.

An interpolator is also used in evaluating the short periodic functions. Once the mean elements are evaluated at the request time, a check is done to see if the request time is within the short periodic coefficient interpolators. If the interpolators do not exist, an interval containing the request time is divided up into equal size steps. The short periodic coefficients are evaluated at each step and a coefficient interpolator is set up for the interval.

In the current software, the mean element and short periodic interpolators are aligned to the same times. This is not a requirement, however.

The flow of calculations in the DSST is depicted in figure 1-4.

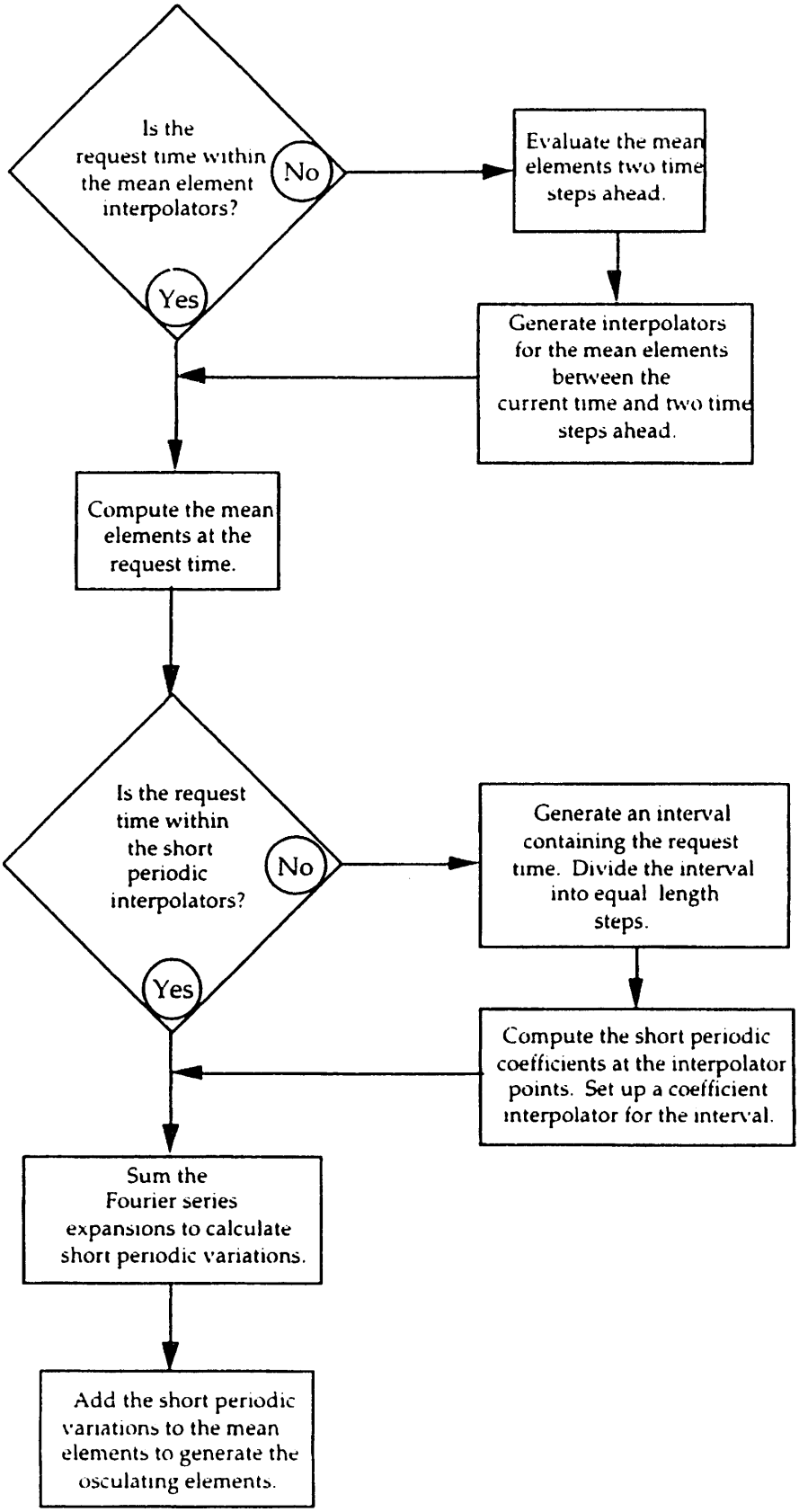


Figure 1-4: Flow of the DSST

1.2.4 Orbital Perturbations

For artificial satellites placed in orbit about the Earth, the acceleration of perturbations in comparison to that of the spherical body is relatively small. However, because the goal is to analyze satellite orbits over a significant period of time, perturbations will be important as they can cause large changes in the location of a satellite over a long time span. A sun synchronous orbit, for example, uses the Earth's equatorial bulge to rotate the satellites longitude of ascending node through 360° per year, thus keeping the orientation of the satellites orbital plane constant with respect to the Sun [3]. For communication systems composed of a constellation of satellites, the effects of perturbations impact the constellation design as well as the satellite design. The orbits in a constellation must be designed to maintain the required orbital parameters within mission constraints.

The mathematics of the various perturbations is discussed in a variety of places, therefore a full development will not be done here. Some references that can be used for more information on orbital perturbations include Battin [5] , Fonte [11], Jablonski [33], and Sabol [50]. This short discussion on orbital perturbations will examine how the perturbations effect the orbital elements.

1.2.4.1 Secular, Long Periodic and Short Periodic Effects

Orbital perturbations are classified with respect to how they change each of the elements over time. A secular change appears as a monotonically decreasing or increasing change on the orbital element. A short periodic effect appears as a periodic variation in the orbital element. A long periodic effect is similar to a short periodic effect, but has a much longer period.

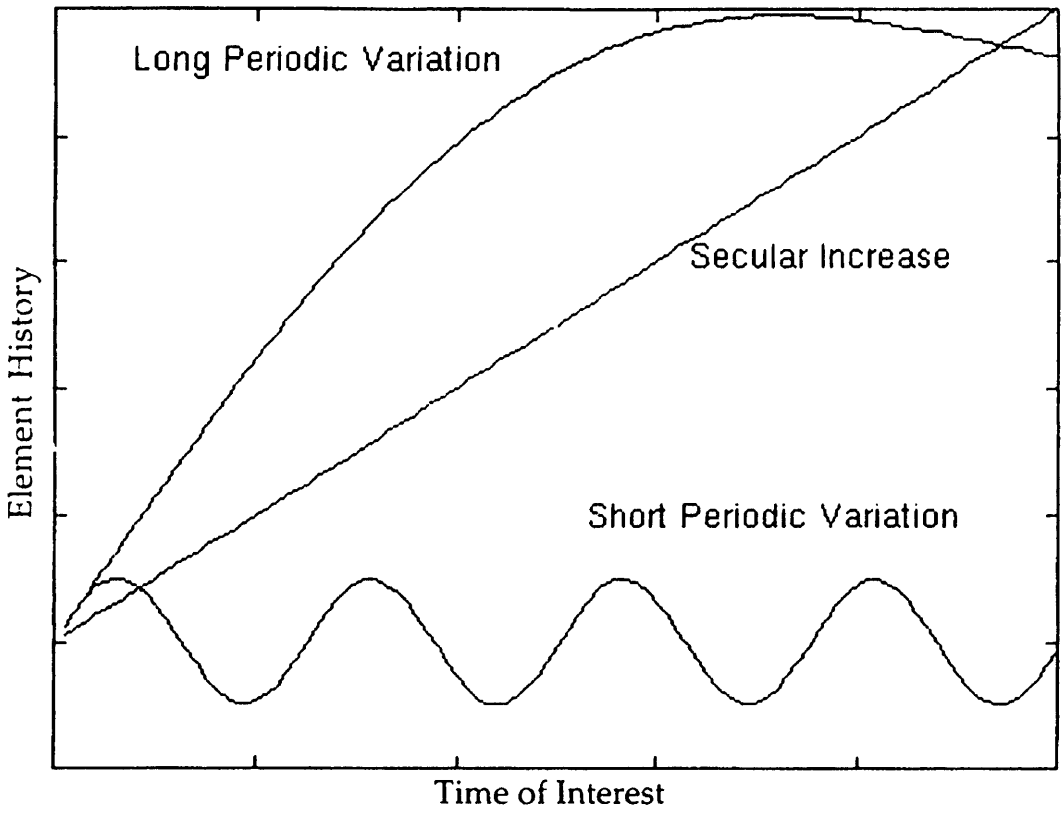


Figure 1-5: Short Periodic, Long Periodic, and Secular Variations

As can be easily seen in Figure 1-5, the time of interest controls the difference between the short periodic variations and long periodic variations. If a much shorter time interval was used, a short periodic variation would look like a long periodic variation. For this analysis, the important length of time to be considered will be the lifetime of the satellite system. Generally, communication system satellites have a lifetime on the order of five to ten years. There are exceptions to this rule, however. Orbcomm is a satellite system composed of 24 satellites that will target the US for low data rate communication. This satellite only plans on a two year lifetime for each spacecraft [12]. Table 1-2 lists the expected lifetimes of each of the satellite systems mentioned in Figure 1-1 [48].

Table 1-2: Lifetimes per satellite for the proposed communication systems [37]

System	Lifetime In Years
Teledesic	10
Iridium	5
Globalstar	7.5
Ellipso	5
Odyssey	12

It is helpful to distinguish between secular, short periodic and long periodic effects as these categories help the orbit designer understand the impact perturbations will have on each of the elements. Some secular effects and long periodic effects must be compensated for by thrusting maneuvers or may require changes in the nominal orbit that remove the undesirable perturbative effects. Other effects are essential to the orbit design, as in the sun-synchronous orbit. Short periodic variations can also cause variations greater than the tolerance allowed in an orbit design.

1.2.4.2 Effects Considered

In general, there are four major perturbations considered in performing orbit analysis for artificial satellites about the earth. These perturbations are:

- Geopotential
- Drag
- Third Body
- Solar Radiation Pressure

Of the above perturbations, drag and solar radiation pressure are non-conservative. Non-conservative perturbations change the energy of a satellite. The geopotential perturbations are divided into the zonal, sectoral, and tesseral harmonics. Each type of geopotential perturbation effects a satellite differently. As previously mentioned, there is always a small parameter associated with each of these orbital perturbations. This parameter helps describe the relative magnitude of each of the perturbations. Table 1-3 lists the force per unit mass of several of the perturbations [19].

Table 1-3: Force per Unit Mass (meters/sec^2)[19]

	ALTITUDE (KM)			
PERTURBATION	150	750	1500	36164
Geopotential				
Zonals				
J2	30e-3	20e-3	14e-3	160e-7
J3	.09e-3	.06e-3	.04e-3	.08e-7
J4	.07e-3	.04e-3	.02e-3	.01e-7
Tesserals				
J2,2	.09e-3	.07e-3	.04e-3	.5e-7
Drag Area/Mass= 0.0212 (m sq/ kg)	3e-3	1e-7	NA	NA
Third Body (Lunar Solar Attraction)[2]	1e-6	1e-6	1e-6	7e-6
Solar Radiation Pressure[3]	1e-7	1e-7	1e-7	1e-7

1.2.4.3 Effects of Orbit Perturbations on Satellites

It is difficult to generalize the effects of most orbital perturbations on all satellites because of their sensitivity to the satellites orbit. However, it is possible to generate a table of the type of effects orbital perturbations will have on the classical orbit elements. Table 1-4 describes the effects of perturbations on the orbital elements.

[2]Based on the Vanguard I Satellite. Reference Blitzer [19]
[3]Based on the Vanguard I Satellite. This is not the direct attraction but the effective disturbing force. Reference Blitzer [19]

Table 1-4: Effect of Perturbations [19, 20]

	Semi-Major Axis	Eccentricity	Inclination	Longitude of Node	Argument of. Perigee	Mean Anomaly
Geopotential						
Even Zonals	Periodic	Periodic	Periodic	Secular	Secular	Periodic
All Zonals	Periodic	Periodic	Periodic	Periodic	Periodic	Periodic
Tesserals	Periodic	Periodic	Periodic	Periodic	Periodic	Periodic
Drag	Sec Decrease	Sec Decrease	Periodic	Periodic	Periodic	Periodic
Solar / Lunar	Periodic	Periodic	Periodic	Sec / Periodic	Sec / Periodic	Periodic
Solar Radiation Pressure	Sec / Periodic	Sec / Periodic	Sec / Periodic	Sec / Periodic	Sec / Periodic	Sec / Periodic

Because the semi-analytic theory is important to this project, the next section gives an example of including a perturbation in the semianalytic theory. This example includes just the J2 perturbation effect on the mean elements. A further expansion of this mathematical development would demonstrate that a recursive method to include arbitrary degree and order of spherical harmonics can be developed.

1.2.4.4 Decomposition of J2 into its Average Contribution

One of the largest perturbations on any artificial satellite is caused by the oblate Earth. This effect is apparent when examining the zonal harmonic contributions to a satellite's orbit. The second harmonic, which describes the magnitude of the bulge around the Earth's equator, is the largest zonal effect on LEO satellite motion, two orders of magnitude larger than any other harmonic. This perturbation is extremely important when examining the orbit of satellite.

From [10] the central body potential U acting at some distance r from the center of mass of the attracting body can be described as:

$$U(r,\phi,\psi)=\frac{\mu}{r}+\frac{\mu}{r}\sum_{n=2}^{N}\sum_{m=0}^{M}\left(\frac{R}{r}\right)P_{nm}(\sin\phi)(C_{nm}\cos m\psi+S_{nm}\sin m\psi) \qquad (1\text{-}50)$$

where r is the radial distance from the center of mass of the central body to the satellite

ϕ is the geocentric latitude

ψ is the geographic longitude

μ is the central body gravitational constant

R is the central body mean equatorial radius

P_{nm} is the associated Legendre function of order m and degree n

C_{nm}, S_{nm} are the geopotential coefficients

M is the maximum order of geopotential field ($M < N$)

N is the maximum degree of geopotential field

The first term is the attraction caused by the Earth if it were perfectly spherical. This force is the largest single force acting on a satellite's motion. The rest of the potential will be referred to as the Disturbing Potential, as it disturbs the motion of the satellite from its Keplerian orbit.

This analysis will only consider the Disturbing Potential of an axially symmetric Earth expanded to second degree (N=2, M=0). The Disturbing Potential then becomes:

$$U(r,\phi)=\frac{\mu}{r}\left(\frac{R}{r}\right)^{2}C_{2,0}P_{2,0}(\sin\phi) \qquad (1\text{-}51)$$

Equation 1-51 can be put into a more familiar form by specifying J2=-$C_{2,0}$. Equation 1-51 then becomes:

$$U(r,\phi)=-\frac{\mu}{r}J_{2}\left(\frac{R}{r}\right)^{2}P_{2,0}(\sin\phi) \qquad (1\text{-}52)$$

The next step involves applying the averaging operator to equation 1-52. In order to do this some other definitions and expansions must be made. From

[10] the function $\sin\phi$ can be put into equinoctial elements by the transformation:

$$\sin\phi = \alpha\cos L + \beta\sin L$$
$$\alpha = \frac{-2p}{1+p^2+q^2} \quad \beta = \frac{2q}{1+p^2+q^2} \tag{1-53}$$

where p and q are the equinoctial elements describing the orientation of an elliptical orbit

L is the true longitude

Inserting Equation 1-53 into equation 1-52 gives:

$$U(r,L) = -\frac{\mu}{r} J_2 \left(\frac{R}{r}\right)^2 P_{2,0}(\alpha\cos L + \beta\sin L) \tag{1-54}$$

The Modified Addition Formula [10] can then be used to expand the associated Legendre Polynomial.

$$P_{2,0}(\alpha\cos L + \beta\sin L) = \frac{1}{2}\left[\begin{array}{c} \frac{3}{2}(\alpha^2-\beta^2)\cos 2L + 3\alpha\beta\sin 2L \\ +\frac{3}{2}(\alpha^2+\beta^2)-1 \end{array}\right] \tag{1-55}$$

Substituting equation 1-55 into 1-54 elaborates the J_2 potential function in terms of the equinoctial elements.

$$U(r,L) = -\frac{1}{2}\frac{\mu}{r} J_2 \left(\frac{R}{r}\right)^2 \left[\begin{array}{c} \frac{3}{2}(\alpha^2-\beta^2)\cos 2L + 3\alpha\beta\sin 2L \\ +\frac{3}{2}(\alpha^2-\beta^2)-1 \end{array}\right] \tag{1-56}$$

With the potential function expressed completely in terms of the equinoctial elements, the averaging operator can then be applied. The averaged form of equation 1-56 results in equation 1-57:

$$\overline{U} = -\frac{1}{2}\frac{\mu}{a^3}R^2 J_2 \left[\begin{array}{c} \frac{3}{2}(\alpha^2-\beta^2)\frac{1}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^3 \cos 2L\, d\lambda + \frac{3\alpha\beta}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^3 \sin 2L\, d\lambda \\ +\frac{1}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^3 \frac{3}{2}(\alpha^2+\beta^2)d\lambda - \frac{1}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^3 d\lambda \end{array} \right] \quad (1\text{-}57)$$

The next step involves evaluating the integrals in equation 1-57. These integrals are elaborated in great detail in Cefola and Broucke, 1975 [10]. A special function, known as the Hansen coefficient, is the critical factor in the solution of the above integrals. For the zonal harmonics, the critical integrals are seen in equation 1-58.

$$\begin{aligned} \frac{1}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^{n+1} cos(mL)d\lambda &= x^{2n-1}R_n^m C_m(k,h) \\ \frac{1}{2\pi}\int_0^{2\pi}\left(\frac{a}{r}\right)^{n+1} \sin(mL)d\lambda &= x^{2n-1}B_n^m S_m(k,h) \end{aligned} \quad (1\text{-}58)$$

where: $x = (1-h^2-k^2)^{-\frac{1}{2}}$ where k and h are equinoctial elements.

$B_{n+1}^m = \dfrac{n!P_n^m(x)}{(n+m)!x^n e^m}$

$P_n^m(x)$ is the associated Legendre Polynomial.

$C_m(k,h) = \mathrm{Re}(k+jh)^m$ Note that these are different from the

$S_m(k,h) = \mathrm{Im}(k+jh)^m$ C_{nm},S_{nm} defined in equation 1-45.

After using the Hansen coefficients to solve the integrals in equation 1-57 and further manipulation and simplification, the averaged potential for J2 can be evaluated in terms of equinoctial elements.

$$\overline{U} = -\frac{1}{2}\frac{\mu}{a^3}R^2 J_2 \left[\begin{array}{c} \frac{3}{2}(\alpha^2-\beta^2)x^3 B_2^2 C_2(k,h) + (3\alpha\beta)x^3 B_2^2 S_2(k,h) \\ + \frac{3}{2}(\alpha^2+\beta^2)x^3 - x^3 \end{array} \right] \qquad (1\text{-}59)$$

Because we are interested in mean elements, the mean equinoctial VOP equations must be derived. These are listed in Danielson [8]. The mean VOP equations require the partial derivatives of the mean potential function with respect to the equinoctial elements. Lagrange's form of the VOP equations can be used because the zonal harmonics are a conservative perturbation. Finally, the J2 contribution to the averaged equations of motion is derived. This has been done analytically for the J2 disturbing potential and can be found in Danielson, Neta and Early, 1994 [8]. This completes the development of the averaged contribution of the J2 disturbing potential. It is obvious here that calculating the perturbative contribution to the potential functions in the VOP format is not a trivial process.

1.3 Parallel Computing

Livingston and Stout listed several motivations for parallel computing in the Supercomputing 92 conference [51].

- Many problems are inherently parallel, so parallel models fit these problems well.
  - Physical processes: fluid flow, planetary orbit, nuclear reactions and plant growth
  - Social processes: wolf packs, assembly lines, ant colonies
  - Sensing / Learning / Intelligence: vision, artificial reality

- Parallel computers are the only way achieve specific computational goals within a given amount of time.

- Parallel computers can be the cheapest way to provide the necessary computational ability.

- Parallel computers can provide fault tolerance.

1.3.1 Previous availability

The concept of parallel problem solving is not new to engineering. Many people often work together to solve the same problem. However, in order that more than one person can work on one problem together, it takes someone in charge directing the work. The same is true for computers. For more than one computer to work together on a problem, an additional process is required to hand out the work to the available processors. Of course this also means computers must be able to accept messages and communicate results with another processes. The extra work involved in setting up a distribution process and communicating with other processors has previously been very difficult and computationally expensive.

In 1980 Jeffrey Shaver investigated the application of parallel algorithms to the orbit determination process [14]. This thesis references Shaver's document as a way to compare how the past fifteen years of development have changed an engineers perspective on parallel computing. Of special interest is the change in the availability of parallel hardware and software in a typical engineering facility.

The target architecture for the study completed by Shaver was a SIMD[4] machine. He was not, however, able to implement his algorithms on a SIMD machine due to many reasons. Computer time on such a machine was very expensive and software was not standard, so his target platform could not use the same software as his development platform. At that time, parallel computing was only accessible to those with a great deal of knowledge in computer science and parallel computing, working to solve enormous computation problems that were not possible on a serial machine.

[4]SIMD parallel computers will be discussed in the next chapter.

1.3.2 *Current Status*

More currently, a report on high performance computing by Horst Simon in December 1993 notes that all manufacturers of High Performance Computers have abandoned the SIMD architecture except for Masspar. He also points out that SIMD machines are very good in raw performance but can be very slow if the algorithms used are not 'completely parallel' [15]. SIMD use required implementation of algorithms of the complexity of that developed by Shaver. Such algorithms and machines would produce very fast execution times. However, by developing software for very specific hardware, such developments would not be cost effective for commercial or government applications interested in COTS (Commercial Off The Shelf) hardware and software.

Many manufacturers continue to make very specialized machines, supercomputers, capable of enormous computing power. Almost all have gone to multi-processor systems. Some of the current manufacturers include Thinking Machines, Cray, IBM, Kendall Square Research, and Paragon. Although these machines far surpass the machines of just fifteen years ago, they are still very expensive and used for scientific and computing research. Parallel computing, however, has not been contained to such a small community. Workstations, computers typically found in most laboratories and universities making extensive use of computers, are now being offered with multi processing capability. These machines are not expensive; they are actually being purchased because the capability they provide is cheaper than comparable processing power available on separate machines. SUN corporation offers the following workstations at the prices shown in Table 1-5.

Table 1-5: Workstation cost comparison[5] [59]

Workstation Description	Cost
SPARC 20/50 (Single Processor)	$12,695
SPARC 20/502 (Two Processors)	$14,195

[5]The computers come with a standard set of peripherals. Both computers listed here came with the same options except for the additional processor.

The availability of parallel computing does not stop there, however. Software, like the system used for this thesis, can turn several, single processor machines into a virtual multi-processor platform. These machines are readily available to most engineers developing computationally intensive applications. The software to allow the communication can be purchased at a reasonable price or even found as public domain, available at no charge. Additional software, however, must still be designed to take advantage of a multiprocessing system.

1.3.3 Current use of Parallel Computing

With low cost parallel computing available to a wide variety of users without requiring special training, parallel computing is quickly gaining popularity. At Draper Laboratory, much work is now being done in developing applications to run in a parallel environment [16]. Because the cost of an entirely new software development effort is so high, many older applications are being upgraded to work on newer systems rather than starting from the beginning. Flight dynamics systems, such as the type developed for RADARSAT, are adding functionality to their systems by using legacy software [46, 60]. Rather than adding more functionality to a single piece of software, old software is used 'as is'. New software must only be developed to interface between the applications [60]. In addition to making use of tested legacy software, such a system lends itself to a parallel computing environment; different processes can execute independently on different processors.

1.4 Thesis Overview

This document describes the development of a parallel version of the DSST, using the Parallel Virtual Machine (PVM) software package to support message handling between computers and processors. The parallel DSST (PVM/DSST) is then integrated with an optimization algorithm to help automate the orbit design process. Finally, both the propagator and the optimization tool are applied to the analysis and modification of a proposed 840 satellite constellation.

Chapter two is an overview of parallel processing, presenting enough information to show how the design of the parallel orbit propagator was chosen, and what other options were available at this time. Chapter three goes on to describe the design of this orbit propagator based on the requirements for this software development and what methods were employed to ensure the software met the goals of project. Also presented are the speedup gains achieved using the parallel propagator and what could be expected with more machines. Chapter four discusses an application of the propagator to a proposed satellite constellation as well as its integration with an optimization algorithm. Chapter five discusses the conclusions and opportunities for future work in this area.

The appendices supplement the thesis in a few specific areas. Appendix A describes the Keplerian and equinoctial element sets. Appendix B lists the important software developed in conjunction with this work. Appendix C describes the input data files used with the PVM/DSST. Finally, Appendix D describes how to execute the software from within Draper Laboratory.

2.0 Parallel Computing

Effective software design requires an understanding of the target computing environment. Therefore, a study of parallel computing was necessary before designing and implementing the parallel orbit propagator. This chapter presents parallel computing concepts and the approaches that were available to the author at the time this project was initiated. One the most helpful sources for current information were the news groups available on the Internet. The two groups most often examined were *comp.parallel.pvm* and *comp.parallel.mpi*.

2.1 Parallel Computing Concepts

Parallel computing introduces new concepts that software designers must be aware of when developing applications. Without understanding these concepts and how they effect performance, applications may not achieve the desired speed-up.

2.1.1 Definitions

The terminology in this technical area is evolving over time so it is important to define several terms before continuing on in this chapter.

basic block — "A sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end [16]. "

bandwidth — Maximum rate of communication between processors. Normally expressed in MB/sec [51].

cache coherency — Information in a cache is correct and consistent.

computer	At least one processor, memory, and the hardware needed to operate the processor. A computer can have many processors. The terms computer and host are synonymous.
processor	A specific chip that has a defined instruction set. This term is currently well defined, although the single chip is now performing more than one instruction at a time with techniques such as pipelining and very long instruction words [17]. The distinction between multi-processors and single processors will become more vague as single processors continue to perform more operations simultaneously.
process	For most programmers, a process is best understood as an executing program. A process, or job, is well defined in a UNIX1 operating system. A process can have more than one thread operating at the same time on more than one processor, using multi-threading techniques.
thread	An set of instructions that are executed in sequential order. A thread is also known as a lightweight process as threads do not have the overhead associated with a processes.
network	A system of connections and routers that allow computers to communicate.
target environment	Network, processors, routers, operating system, and programming language that make up the parallel environment where a program is executed.

2.1.2 *Measuring Performance of Parallel Algorithms*

Measuring performance of a parallel algorithm is critical to demonstrate the usefulness of working in a parallel environment. Without demonstrating performance, it is impossible to quantify the gain achieved in moving from a

[1] UNIX is a trademark of Bell Laboratories.

serial to a parallel environment. Additionally it is important to demonstrate effective use of the resources so that speed increases do not require excessive amounts of additional hardware. Finally, it is important to indicate how parallel algorithms scale to more processors. Algorithms may be designed for a limited number of machines so that additional speed increases can not be achieved by adding more processors.

Two measures describing the effectiveness of a parallel algorithm are speedup and efficiency [18]. Speedup of an algorithm is described as [18]:

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \tag{2-1}$$

where: p is the number of processors

n describes the problem size

T_p is the execution time of the parallel algorithm on p processors

T^* is the time of execution of the best serial algorithm

S_p is the speedup of the algorithm

Efficiency of an algorithm is defined in [18]. It is used to describe how effectively all processors are being used.

$$E_p(n) = \frac{S_p(n)}{p} = \frac{T^*(n)}{pT_p(n)} \tag{2-2}$$

where: E_p is the efficiency of the algorithm on p processors

2.1.3 *Granularity and Communication Costs*

Granularity describes the amount of computation in a program segment that executes serially [72]. A very small grain size has more potential for parallelism but requires more communication and scheduling overhead [72].

Therefore, when decomposing a problem for parallel applications, it is essential to ensure the granularity of the decomposition matches the target environment. If a fine-grain decomposition of a problem is performed so that the program is divided into many small pieces for execution, communication between many processors will be more frequent. If the problem exhibits coarse-grained parallelism, more computation will be performed on each processor before communication occurs. Parallel computing environments exist to solve both types of problems. Many designers of parallel computers have designed sophisticated networks and used relatively inexpensive, comparatively slow processors. Others have used simple networks with very capable individual processors. The tradeoff between fine grain problem decomposition and communication is depicted in Figure 2-1.

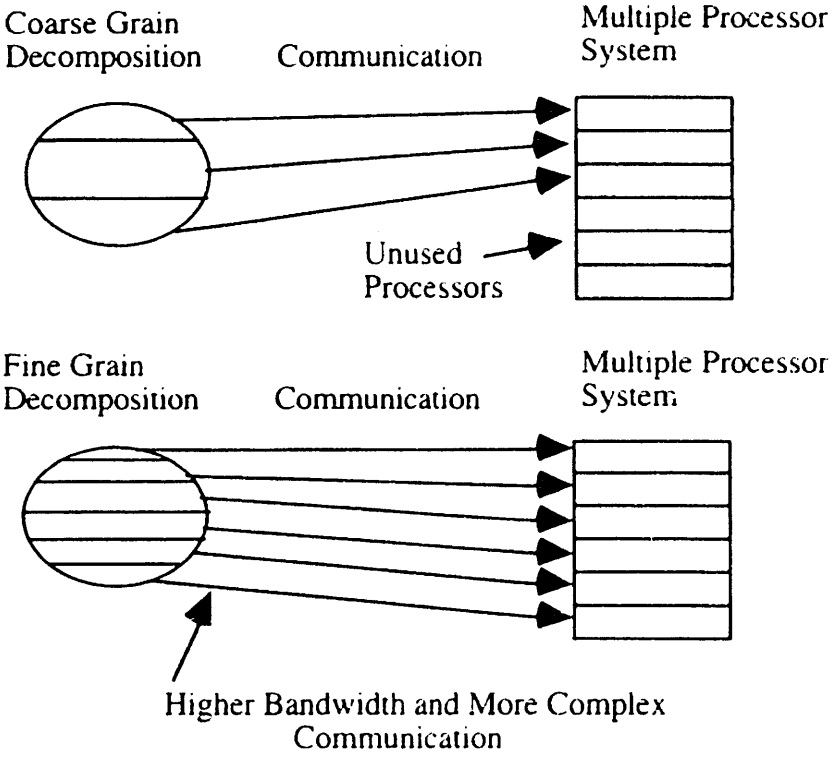


Figure 2-1: Fine and Coarse Grained Parallelism

Applying equations 2-1 and 2-2 to the previous discussion on problem decomposition quantifies the advantages and disadvantages of coarse and finely grained parallelism. A coarsely grained parallel algorithm limits

speedup. This is demonstrated more clearly in Amdahl's Law, equation 2-3 [18].

$$S_p(n) \le \frac{1}{f + (1-f)/p} \le \frac{1}{f} \qquad (2\text{-}3)$$

where: f is the fraction of the problem that is inherently sequential

1- f is the fraction of the problem that is fully parallelizable

A division of the problem in half, so that $f = \frac{1}{2}$, limits the theoretical speedup to two. If communication and setup costs are added, even if they are minimal, the speedup will be reduced to below that amount. The efficiency of a coarse grained algorithm is relatively high, however, as a low ratio of communication and setup time to work means that the processors will be busy most of the time, so that efficiency will approach one.

A fine grained decomposition allows speedup to be increased until all available processors are being used at the same time. However, many, small jobs will also increase the amount of communication required, as seen in Figure 2-1. If the network does not efficiently handle the communication, T_p will contain larger communication overhead, increasing pT_p, and decreasing efficiency. Therefore, a decomposition that does not match the target environment, will increase speedup but will reduce efficiency. Far too much decomposition on a slow network could even translate into longer execution times, or reduced speedup. The tradeoffs between fine grain parallelism and coarse grained parallelism makes it difficult to efficiently match a single parallel model to a wide variety of target environments.

In a real world situation, p is limited. However, it is always desirable for a parallel program to be 'scalable'. A scalable algorithm remains efficient as the number of processors available increases. A coarse grained decomposition could limit the maximum number of processors used. A poorly designed fine grained decomposition can reduce efficiency with a large number of processors. It is very difficult to predict how many processors will be available

to a user in the future, so the software will be useful for a longer time if it is scalable to an infinite number of processors. Figure 2-2 depicts how a typical parallel algorithm scales to more processors. The concept of linear speedup, or an efficiency equal to one, is also displayed on Figure 2-2.

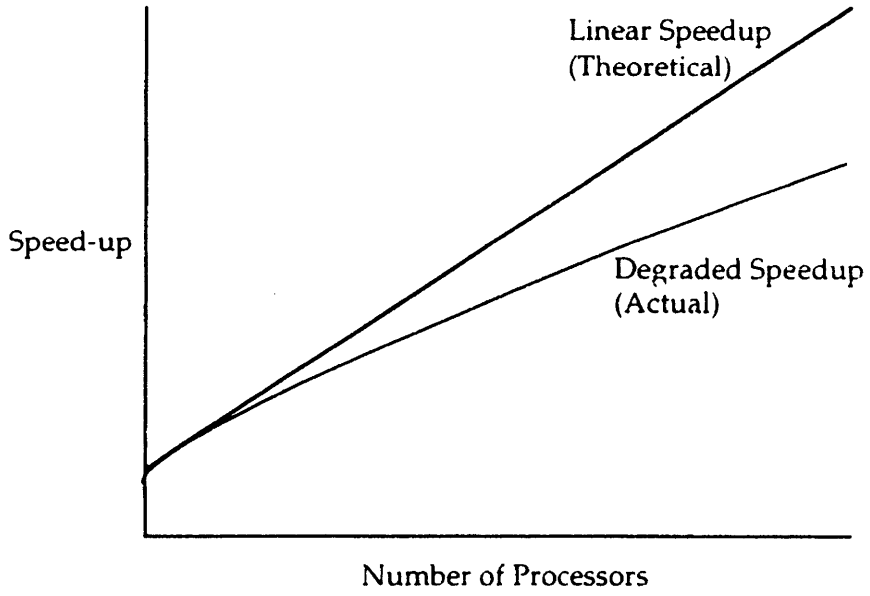


Figure 2-2: Speedup Vs Number of Processors [15]

2.1.4 *Levels of Abstraction*

Computers can be viewed from many different levels of abstraction. The highest level is seen by the programmer through high level programming languages. The programmer may have varying degrees of control based on the programming model (Sec 2.3.1). Below the high-level software is the compiler. The compiler interfaces the programming language to the operating system, changing high level commands into machine specific instructions. The operating system is responsible for directing the computers work, accessing data from a disk, and managing memory resources. The hardware, the actual pieces that make up the computer and how they are interconnected, make up the last level. Because parallel computers can be very complicated, describing the environment from these perspectives makes the entire system easier to understand. Figure 2-3 depicts the computing levels.

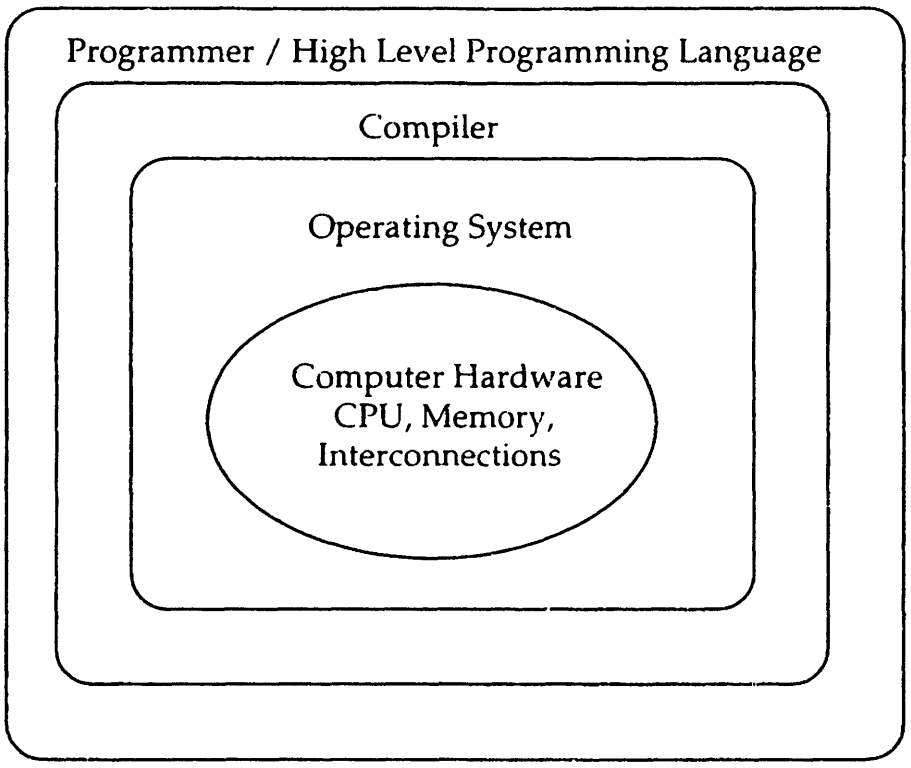


Figure 2-3: Hierarchy of the Levels of Abstraction

Viewing computers from the four levels also emphasizes the importance of the interfaces. High performance can only be attained if there is efficient communication between each of the levels of abstraction. This concept reemphasizes the necessity for a software designer to understand the target environment.

Sections 2.2 and 2.3 describe parallel computers from the bottom up, omitting the operating system and compiler levels. These sections provide the understanding which will be necessary and required for developing effective parallel applications.

2.2 Parallel Hardware

Parallel hardware, at the lowest level, starts on the computer's CPU. At the highest level, parallel computing involves multiple computers working together. This section will describe parallelism in computers starting with parallelism on the CPU. Much of this discussion references High Performance Computing by Kevin Dowd [17]. This text provides an excellent overview of parallel computing concepts and ideas.

2.2.1 Computer Memory/ Basic Computer Architecture

Memory is not one homogeneous area of a computer. Memory is divided into many layers, so that instructions and data can move as fast as possible from a storage area into the processing area. Access time to the memory closest to the processor is the fastest; the access time to the memory farthest from the processor is the slowest. Figure 2-4 depicts the memory structure of a basic computer. Not all computers fit this model exactly, especially as manufacturers continue to tune their computers to achieve the best performance.

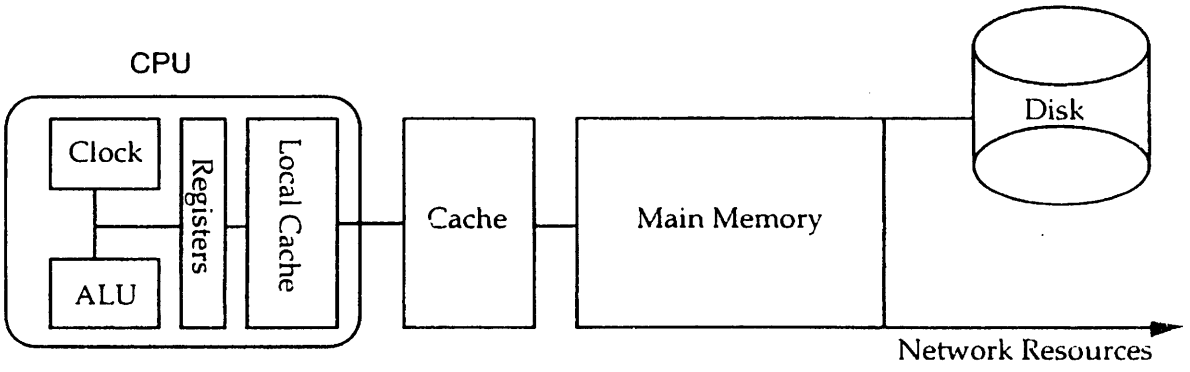


Figure 2-4: Computer Memory Hierarchy [42, 52]

Each of the units shown above is described in Table 2-1.

Table 2-1: Description of Computer Components

Unit	Description
Main Memory	The area most commonly referred to as memory. All executing programs must reside here unless the computer is swapping to disk. Normally made up of dynamic RAM (DRAM) for cost reasons.
Cache	A smaller, fast memory unit that is normally off the CPU. Often made of static RAM (SRAM).
Local Cache	An intermediate memory unit generally located on the CPU.
Registers	Memory that loads information directly into the ALU.
Clock	The device that controls the rate at which all operations happen.
Arithmetic Logical Unit (ALU)	Unit that actually performs operations on the data.

As described earlier, each memory unit is increasing in capacity but decreasing in speed from left to right in Figure 2-4. This is important as main memory access speeds are slower than the clock rate. If main memory was connected directly to the CPU, calculations would be limited by memory access speed [52]. Using a hierarchical structure allows a small amount of very high speed memory to keep the CPU busy. Instructions and data can then be loaded from memory into the cache, the cache into the local cache, the local cache into the registers, and the registers into the ALU.

2.2.2 *Parallel Computing on the Chip*

At the lowest level, parallel computing can take place on the CPU. Parallelism at this level can be achieved in many different ways. Multiple functional units can be added to the CPU to perform more than one instruction at the same time [52]. Functional units can be designed to perform specialized tasks, such as floating point operators. Multiple floating point units can be used at the same time, if more than one operation can be performed at the same time while insuring all calculations maintain coherency. This requires work by the compiler, to identify computer instructions that can be executed in parallel without corrupting other data [17]. An aware programmer can help the compiler by writing code which supports parallel instruction execution. A programmer supporting

parallelism on the CPU in the code design is an example of the relationship between levels described in Figure 2-3.

Pipelining of instructions is a form of CPU level parallelism. Pipelining involves decomposing instructions into the stages that are involved in executing an instruction [17]. Stages can be executed one right after another, so that more than one instruction is being executed at the same time. An example pipeline from Dowd [17] assumes all commands are decomposed into five stages, as shown in Table 2-2. The number of stages is actually dependent on the computer type.

Table 2-2: Example Stages of an Instruction

Stage of Instruction	Stage Description
1 Instruction Fetch	Fetching an instruction from memory
2 Instruction Decode	Decode or recognize the instruction
3 Operand Fetch	Fetch the necessary operands
4 Execute	Perform the instruction
5 Writeback	Place the results back into memory

In a pipeline, instruction 1 is fetched into the beginning of the pipeline at time 0. At time 1, instruction 1 is decoded in the next stage of the pipeline while instruction 2 is fetched into the first stage. This is repeated until five instructions fill the five stage pipeline. All instructions move through the pipeline in lockstep [17]. If one of the stages of an instruction takes longer than just one step to complete, the rest of the pipeline is stalled. The processor must be very careful how it feeds the pipeline in order to achieve optimal performance [17].

There are more ways of exploiting parallelism on the CPU, especially as computer designers seek to make faster computers. The ones presented here are some of the most common and are used in the majority of modern day computers. The next section moves away from the CPU to exploiting parallelism among multiple CPUs.

2.2.3 *Multiprocessor Memory Use*

The use of memory by multiple CPU computers (multiprocessors) defines their structure and is a common way to categorize multiprocessor environments. As shown in Figure 2-4, the term memory is most accurately portrayed as a series of layers. Multiprocessors can be categorized by the layer of memory the CPUs share for communication. In theory, multiprocessing machines could be placed into a continuum, from those that communicate at the cache level to those that communicate across the network or through the disk. In practice, two types of systems are commonly defined to describe different types of multiprocessors: those that share main memory (shared memory) and those that have their own main memory (distributed memory). As the technology continues to develop, machines are communicating through multiple layers, trying to reduce communication time.

2.2.3.1 Shared Memory

A shared memory system contains one large memory bank for the processors that intend to work together [17]. Shared memory systems have tremendous advantages in terms of speed of communication. By keeping all processors connected to the same system of memory, processes can quickly communicate by placing information in an area where another process knows to look. Figure 2-5 demonstrates how a shared memory system exchanges information.

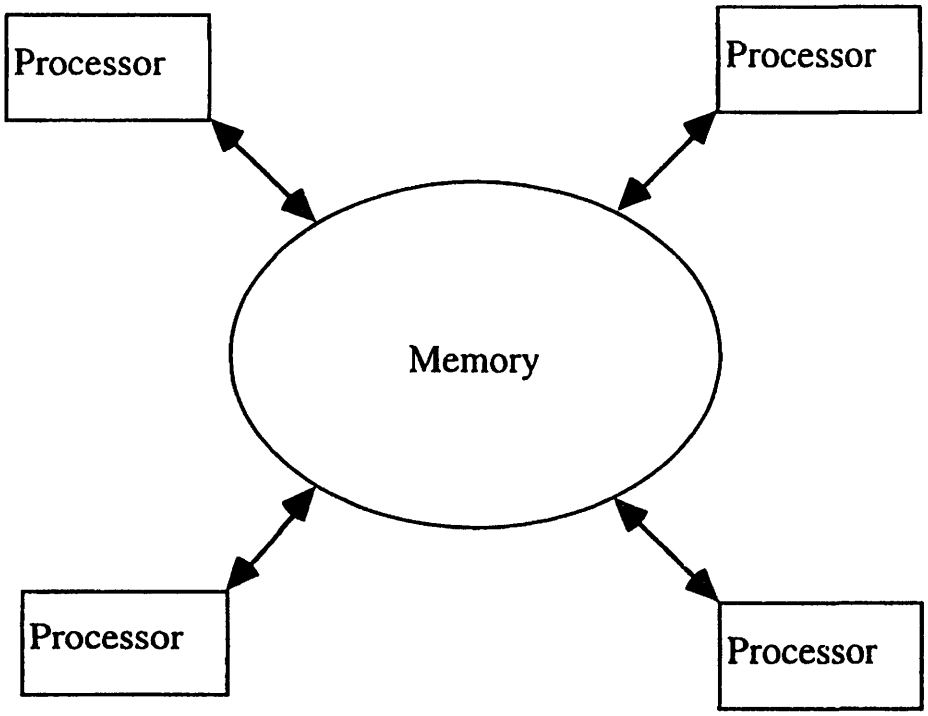


Figure 2-5: Conceptual Illustration of a Shared Memory Parallel Computer

A shared memory system must be careful in reading and writing to memory. If the first processor updates a specific memory location in shared memory, followed by a read by the second processor, the second processor will retrieve the new value, even if it was expecting the old one. Therefore, if a processor writes to a data location it knows the other processors might look at, it must indicate to the others that it has written there. There are different protocols for maintaining data coherency. Additionally, a shared memory system cannot be easily expanded. One cannot just simply add another CPU to a shared memory environment, due to the complexity involved with more than one processor using the same physical memory.

2.2.3.2 Distributed Memory

Distributed memory multiprocessors allow each computer to have its own, private memory resources. Of course, the computer must communicate with the other computers in the group in order to exchange information between processors. This is done by sending messages over a network. Such systems

allow for tremendous flexibility in the design of an application. There is no chance that any computer will infringe on another's memory. Distributed memory systems do particularly well for applications requiring a large amount of computation for each basic block, or coarse grained parallelism. Figure 2-6 conceptually illustrates a distributed memory system.

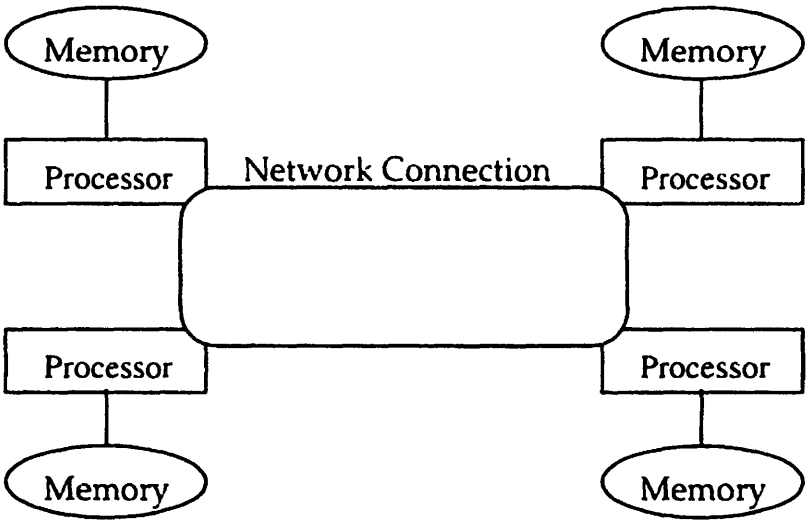


Figure 2-6: Conceptual Illustration of a Distributed Memory Parallel Computer

Because computers may be physically separated and connected by a low bandwidth data connection, messages can require excessive time to transfer between processors. This problem is becoming less significant as communication links increase in bandwidth. As bandwidth increases, parallel computing becomes more feasible over a network of distributed machines.

At Draper Laboratory, most machines are connected by ethernet connections. Some higher bandwidth connections, such as a Fiber Distributed Data Interconnect (FDDI) ring, are also used in less frequent cases. Figure 2-7 illustrates the relative capacities of various communication networks and when these technologies became available [13].

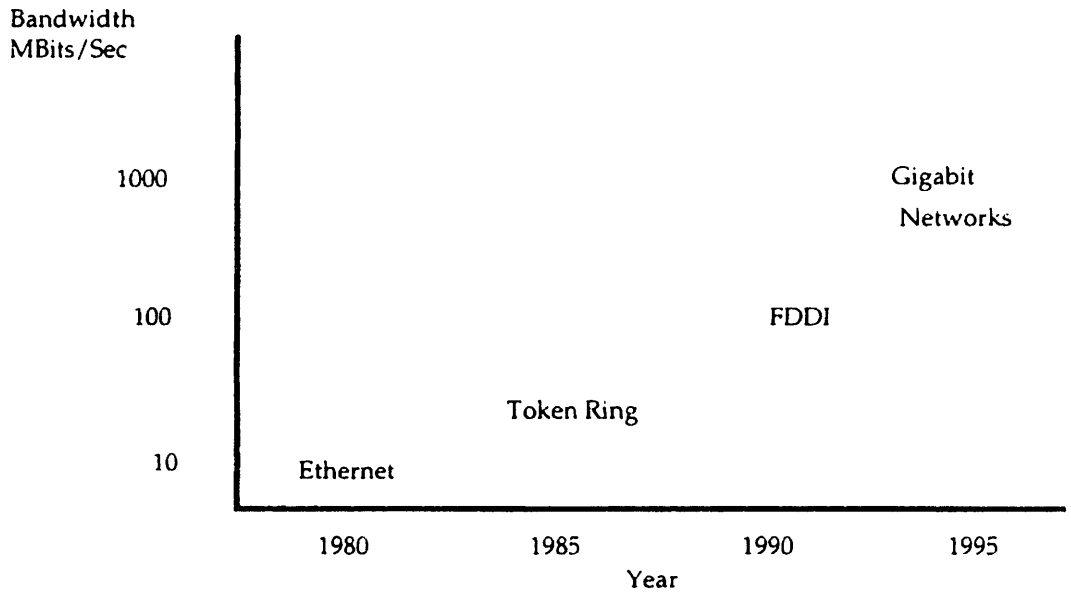


Figure 2-7: Network Capacities [13]

Figure 2-7 demonstrates the future will continue to promise higher bandwidth connections between computers.

2.2.4 *Network Design*

A network allows multiple processors to communicate. Network design is a very complex subject and greatly influences performance of multiprocessors. Bertsekas lists several factors which are important to network performance [18].

Table 2-3: Performance Metric Definitions for Network Topologies [18]

METRIC	DESCRIPTION
Diameter	The maximum distance between processors. Distance is the minimum number of links that must be traversed. A link is a connection between two processors.
Connectivity	The number of independent paths between nodes.
Flexibility	The ability to emulate other topologies.
Communication Delay in Standard Tasks	The number of steps it takes to send the required information through the topology of interest.

The diameter is one of the most common metrics for classifying networks. A small diameter means fast communication as messages will not be relayed through many other nodes before reaching their destination node. The ideal network, in terms of diameter, would directly connect each processor to every other processor [18]. This concept does not scale well, however. To connect each processor to every other requires $\frac{N*(N-1)}{2}$ connections or a very complex bus [17]. For four processors this networking scheme works well, requiring six total connections. For 512 processors the total number of connections increases to 130,816 connections, which is too many connections for cost and complexity reasons. This type of network is known as a complete graph [18]. The network with the worst diameter is a linear array [18]. All the processors in this array are connected in a line so that each processor can only communicate with its nearest neighbors. The same 512 processor machine would require only 511 connections on a linear array. However, the diameter increases to 511. A six node linear array and complete graph are depicted in Figure 2-8.

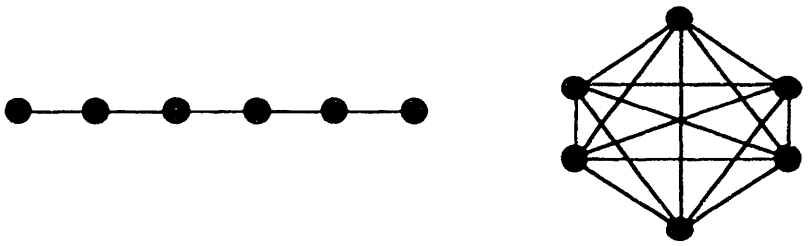


Figure 2-8: A Six Node Linear Array and Complete Graph

A common topology for networking multiple processors is a hypercube. Bertsekas describes the hypercube as "the set of points in d-dimensional spaces with each coordinate equal to zero or one [18]." Additionally, a hypercube is connected between "every two points differing in a single coordinate [18]." It easier to picture a hypercube if a bit address is assigned to each node or processor. The nodes that differ in exactly one bit are connected. A 3-d hypercube is shown in Figure 2-9.

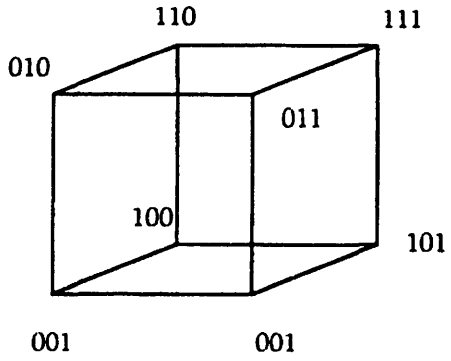


Figure 2-9: A 3-d Hypercube

There are many more topologies for connecting a network of processors. Table 2-4 lists several networks with p processors. The diameters, number of connections, and general advantages and disadvantages are also listed.

Table 2-4: Network Topologies [17, 18, 71]

Topology	Diameter	Number of Connections	Advantages	Disadvantages	Example Machine
Linear Array	p-1	p-1	Simple to construct.	Long latency associated with large diameters.	N/A
Ring	(p)/2	p	Up to twice as fast than the linear array for some operations.	Diameter increasing linearly with the number of processors reduces scalability.	KSR-1
Binary Balanced Tree	$2 * k$ where k is the number of levels and $2^k \leq p < 2^{k+1} - 1$	p-1	Low number of connections. Faster than linear array for some operations	Low connectivity.	CM-5 (Actually uses a variant known as the 'Fat Tree' [71])
d dimension, mesh, edges not wrapped	$\sum_{i=1}^{d} (n_i - 1)$ where n_i is the number of processors along the i^{th} dimension	Depends on dimension.	Works well for problems tied to physical geometry.	Can expand to a large number of connections.	CM-2 Illiac IV
Hypercube of dimension d and $p = 2^d$	d or $\log_2 p$	$\frac{d * p}{2}$	Scales well to a large number of processors. Very flexible topology.	Can expand to a large number of connections.	CM-2 nCUBE
Complete Graph	1	$\frac{p * (p-1)}{2}$	Minimum diameter topology.	Many connections required.	N/A

2.2.5 *Flynn's Taxonomy*

In addition to network topologies, parallel computers can be categorized by their ability to use instruction and data parallelism [51]. Flynn's taxonomy assigns a four character designator to every parallel computer based on the computers capabilities. Table 2-5 describes Flynn's taxonomy [51].

Table 2-5: Flynn's Taxonomy

$$\begin{Bmatrix} S \\ M \end{Bmatrix} I \begin{Bmatrix} S \\ M \end{Bmatrix} D$$

where:

SI Single Instruction	All processors are working in 'lockstep,' sharing one global clock and executing the same instruction at the same time.
MI Multiple Instruction	Processors are processing independently, with their own clock.
SD Single Data	All processors have the same data available at the same time.
MD Multiple Data	Processors may be using different data sets at the same time.

Two different types of parallel computers, according to Flynn's chart, will be examined in the next two sections, SIMD and MIMD. The other two schemes, SISD and MISD, are rarely used for parallel computing designed to increase performance [51].

2.2.5.1 SIMD

SIMD computers are composed of many distributed memory processors. The processors execute commands in 'lockstep', all sharing the same clock [17]. This type of processing is known as synchronous execution [18]. A SIMD computer uses distributed memory. A simple loop is an example where such a machine would be very useful. Consider the following section of FORTRAN:

```
DO I=1,N
  Y(I) = Z(I)*2
  X(I) = Y(I)**4
ENDDO
```

If N was very large, this simple loop could require a significant amount of time. If a SIMD machine had N processors, the entire the loop calculation would be performed in one iteration on N machines, rather than N iterations

on one machine. One could imagine each computer doing the same calculation according to a global, shared clock [18]. At time $t1$, each processor would multiply Z(I) times two. At time $t2$, Y(I) would be raised to the fourth power. (Of course, each instruction is broken down into many smaller instructions by the compiler. These smaller instructions are actually the ones that are synchronized). Here I is not only the index of each array but also the processor number. If the computer had only N/2 processors, it would take the computer two times through the loop, plus the overhead. Obviously, these type of calculations would run very fast on a SIMD machine.

2.2.5.2 MIMD

MIMD systems differ from SIMD machines as each processor has its own clock. Each processor in a MIMD computer operates independently. There is no requirement of synchronization between processors; however, such synchronization can be imposed on the system if desired. MIMD computers can use either shared memory, distributed memory, or a combination of the two.

Each processor in a MIMD machine is generally more powerful than that of a SIMD machine. With a MIMD computer, a programmer can send an entire section of work to be performed to an awaiting processor, which can then perform the work at its own pace. Even the work performed on each processor can be completely different. However, the same loop described above can also be implemented with a lesser degree of synchronization. Both loop steps can be performed on each of the processors and the results sent back to a central location, for example. Some processors may finish the two calculations earlier than others, so they will just be waiting until the last processor gets done before they begin the next job. Obviously, it is not desirable to have processors waiting for one another, so optimal implementation on a MIMD machine may require asynchronous algorithms.

A distributed network of processors is a type of MIMD parallel processing computer. It would make little sense to impose an entirely synchronous process on such a system because of the difference in machine speeds, the

differing workload on each of the machines, and the high price of communication (in terms of time).

2.3 Programming in a Parallel Environment

Section 2.2 discussed the hardware inside a parallel computer. This section describes how the programmer interfaces with the hardware through the programming environment. The compiler and operating system levels of a parallel computer, mentioned in section 2.1.4, will not be discussed. The programmer should assume the operating system and compiler have been designed to achieve some performance out of the parallel computer. If the programmer gives all control of the parallelism to the compiler and the operating system, optimal performance cannot be guaranteed.

2.3.1 Levels of Programmer Control

Different programming models allow different degrees of execution control. More programmer control allows the engineer to specify which processors execute which pieces of software, how communication will take place, and when, in the course of program execution, each machine executes an instruction. This can be advantageous, especially when tuning software for minimum communication time and maximum performance. This type of model also requires much more detail out of the programmer.

On the other hand, some models let the compiler divide up the work among the available processors. Programming within these models requires much less work. The algorithms used by the computer are not specified by the programmer. At the same time, the programmer loses the ability to tune algorithms for a particular application. The models that remove flexibility from the programmer limit the performance that can be attained from a parallel computing environment for a particular application.

The next three sections discuss three different programming models. Figure 2-10 displays these models as a continuum from minimum to maximum control. Sections 2.3.2 through 2.3.4 will also be addressed in this order.

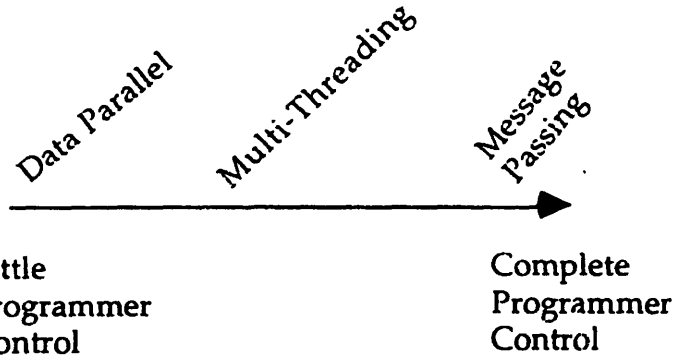


Figure 2-10: Continuum of User Control in Parallel Programming Models

2.3.2 *Data Parallel Model*

The data parallel programming model requires a data parallel language and data parallel compiler, as it is the compiler that distributes the work among the available processors. One of the most popular data parallel languages is FORTRAN 90 or HPF FORTRAN. These versions of FORTRAN are just now being used on a variety of machines. Compilers for these languages are still fairly expensive, as they are just being released. CM-FORTRAN is the data parallel language available on the Thinking Machines supercomputers and is very similar to FORTRAN 90.

An example CM-FORTRAN statement that takes advantage of multiple processors can be seen below.

A=c*B+D

where: A is a matrix size nxn
B is a matrix size nxn
c is a scalar
D is a matrix size nxn

Figure 2-11: Data Parallel Example

This statement is executed using an algorithm for parallel matrix multiplication and matrix addition. The programmer did not know what algorithm was being used, specify how many processors to use, or which processors would do certain calculations. Obviously , this method of parallel computing makes programming simpler. Some new functionality is also added to a data parallel language to take advantage of the multi-processor environment. Use of these features may require rework of serial algorithms.

Although simpler to use, data parallel languages also have disadvantages at the current time. These disadvantages include:

- Compilers must be purchased for each class of multiprocessing environment
- Software must be modified for each compiler (non-portable)
- Inability to effectively use old (legacy) software without significant modifications
- Lack of user control

2.3.3 *Mult.-Threading Models*

Multi-threading requires programmers to develop their own algorithms for parallel execution. Programmers must create and destroy threads to perform specific calculations. However, the programmer cannot specify which processor will execute a specific thread.

Multi-threading has gained popularity, especially on symmetric multi-processing, shared memory platforms [21]. The term multi-threading

indicates multiple threads of control in one process. It is easiest to understand multi-threading using the UNIX notion of process to describe what most programmers are used to as a single application occurring sequentially, or a single thread of control in every process. Multi-threading allows asynchronous process control within the same UNIX process [22]. Using multiple threads of control, a process can be doing more than one thing at the same time. The main advantage of multi-threading over message passing is that threads require less overhead than a UNIX process, thus switching between threads is requires less time than between UNIX processes.

An example operation that can make effective use of multi-threaded process control is disk I/O [22]. If just one thread of control is allowed, a request for information from a disk will require a program to wait until the operating system can access the data. If multiple threads are used, several I/O accesses can be performed at the same time. If multiple processors are present, different threads can execute on different processors, although all threads are only seen by one process. A process could have many requests for I/O, each having a separate thread of control [22]

Multi-threading is very similar to message passing in that a separate thread performs its work and returns its result so another thread can use it. However, the information that needs to be passed between threads is global to all the threads. Synchronization is slightly more difficult when developing a multi-threaded application as compared to message passing.

2.3.4 *Message Passing*

Most message passing environments allow complete programmer control as to which processor is performing which calculations. At the same time, this requires that the programmer specify all the control information, which can often be a complicated and cumbersome task.

A message passing program which performs the program fragment described in Figure 2-11 is shown in Figure 2-12. The part titled master would be the controlling program. This program has all the data, sends the data out to the

'slaves' numbered 1 through N, and then puts together their return. This is definitely not the best way to accomplish this task, especially as it assumes N=nxn processors are available and must send as much extra information as the information that is actually being used. Note that each slave must receive the indices as well as the numbers to be multiplied. The slaves will finish their work in a random order, thus returning the values to the master in a random order. To make sure the values get placed in the correct location, the slaves must receive their indices just to send them back with the answer.

```
Master                                  Slave Number (n*(i-1)+j)
      multicast(c)                            receive(c)
      do i=1,n
        do j=1,n
          send((n*(i-1)+j),B(i,j))            receive(b)
          send((n*(i-1)+j),D(i,j))            receive(d)
          send((n*(i-1)+j),(n*(i-1)))         receive(index1)
          send((n*(i-1)+j),j))                receive(index2)
        end do
      end do
                                              a=b*c+d
      do i=1,n
        do j=1,n
          receive(index1)                     send(master,index1)
          receive(index2)                     send(master,index2)
          receive(a(index1,index2))           send(master,a)
        end do
      end do
```

Figure 2-12: Message Passing Example

As can be seen, this program is written in standard FORTRAN 77 that must be linked with a message passing library. The commands from the message passing library are:

- multicast(value) Send value to all slaves
- send(slave,value) Send value to slave
- receive(value) Receive value from another process

These commands are described here in a very generic way, but almost every message passing library contains these simple commands (some may not have a multicast command).

A comparison between Figure 2-11 and 2-12 shows the disadvantages of a message passing environment. As all parameters in a message passing environment must be specified, the problem becomes much more complex. However, there are some advantages of working in a message passing environment.

- The algorithm used for dividing up the work can be specified by the programmer
- A more standard language, such as FORTRAN or C, can be used
- Legacy code can be more easily incorporated

In the construction of the flight dynamics system for RADARSAT, Draper Laboratory chose a message passing approach to combine the functionality of legacy software [60]. Although the software was designed for one computer, using the message passing approach allowed legacy software to remain essentially unchanged. The new system was developed with much less effort than if the legacy software was combined into one program that incorporated the capability of the individual functions.

2.4 Specific Approaches Considered for IPC (Interprocess Communication)

Sections 2.2 and 2.3 described parallel programming hardware and programming environments. This background will be used to examine the options that were available for developing the parallel orbit propagator.

2.4.1 Availability

The previous section on parallel hardware, section 2.1, described how hardware is built to support communication between processors. The software paradigms describe different methods of developing software to perform the interprocess communication. With this level of understanding, several different methods of communication that were readily available to the author can be compared.

Table 2-6 lists all the different software packages considered together with a short description of the software. This list represents the software available to the author when development decisions were made. A more thorough list, containing descriptions of 70+ parallel software environments has been compiled by Louis Turcotte [24].

Table 2-6: Parallel Software Models Considered

METHOD	DESCRIPTION
Data Parallel	
CM-FORTRAN	Data parallel language available on the CM-5. Programming style very similar to FORTRAN 90.
FORTRAN 90	Latest release of FORTRAN with data parallel constructs. Would need a new compiler for each machine to be developed on.
Message Passing	
CMMD	Message passing library on the CM-5.
PVM	Creates a virtual machine of several UNIX platforms. Portable to a variety of platforms. Available via anonymous ftp.
MPI	Message passing standard. Requires individual vendors to develop MPI libraries for their systems.
Multi-Threading	
SOLARIS 2.3	Available at Draper Laboratory on a SPARCstation 20-514. Libraries are written to be included in C programs.
POSIX Threads	Attempt at a standard for multi-threading applications.
Shared Memory	
Network Linda	Similar to PVM but uses a virtual shared memory concept for communication. Users must purchase software.

It should not be assumed the above are all options to solving the same problem. Two of the above items, MPI and POSIX Threads, are standards rather than specific systems. Because every vendor making a multi-processing system provides a different method for interprocess communication it is very difficult to design applications that will run on more than one platform. For each multi-processing platform a developer would have to change their application to interface with a particular system.

Standards describe interfaces for developing message passing and multi-threaded programs and leave it up to parallel environment developers to implement the interfaces between the standards and the underlying communication system. This concept can be seen more clearly in Figure 2-13.

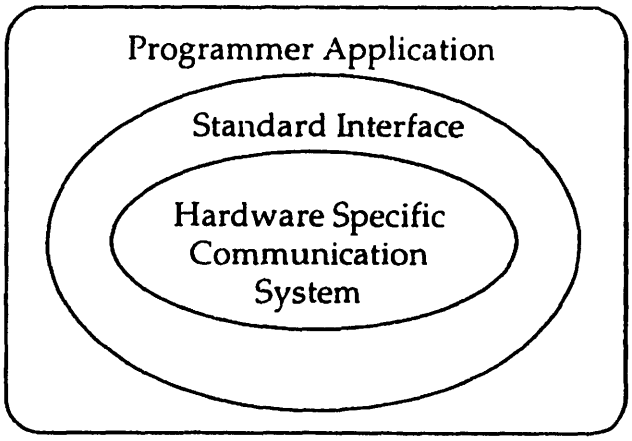


Figure 2-13: Levels of Interfaces to Communication Systems

In Figure 2-13, the programmer's application is not affected by the hardware specific communication system. The standard interface will have the same shape on the outside, despite the shape of the hardware system, so the application will 'fit' onto a variety of hardware systems. The hope is that by making standards, users will be more likely to develop parallel applications as they will be able to run them on a variety of platforms.

Except for the standards, each implementation described above was developed by different people, requires different hardware, and does different jobs. The developer creating parallel applications will be provided with a different set of functionality and develop different software depending on the choice made.

The above systems will be explained in further detail. As mentioned earlier, two attempts at this standardization include POSIX Threads and MPI, the Message Passing Interface. Information on MPI can be found in the book Using MPI [22] or in the MPI newsgroup comp.parallel.mpi. The POSIX Threads standard is a part of the IEEE standard for portable computing [23].

2.4.2 *FORTRAN 90 / HPF*

FORTRAN 90 is the latest release of the FORTRAN programming language. It is a super-set of the widely used FORTRAN 77 standard and also contains many functions that have been added in vendor specific versions of FORTRAN 77. High Performance FORTRAN is similar to FORTRAN 90 but is geared specifically toward parallel computing. Both languages contain constructs for using parallel processing as shown in the program fragment, figure 2-11. A FORTRAN 90 compiler must be purchased for each development platform a programmer wishes to use. Parallel applications developed in these languages, if compiled and run on a multi-processor system, could be made to take advantage of that system. CM FORTRAN is an example of such a system. It takes advantage of the processing power available on the CM-5 by separating the work onto the available processors. If a section of the code cannot be broken down to run on all available processors, it runs serially on one computer.

2.4.3 *CMMD*

CMMD is the message passing library on the CM-5, a parallel supercomputer at MIT's Laboratory for Computer Science (LCS). Although recently Thinking Machines filed for bankruptcy, Thinking Machines corporation was formerly one of the developers of leading edge, high performance, parallel computers. A Connection Machine 5E (CM-5) with 128 SuperSPARC processors was available to the author over the course of the research project as a part of project SCOUT, a nationally funded super-computing project. The CM-5 has many models available for parallel computing, including their own versions of the data parallel languages, CM-FORTRAN and C*. The message passing library on the Connection Machine is known as CMMD. This library allowed for communication between processes as described in Figure 2-6. Table 2-7 describes some of the commands available within the CMMD message passing library, highlighting some of the atypical functionality present that can be useful to a programmer.

Table 2-7: Sample CMMD Functions [31]

FUNCTION	DESCRIPTION
CMMD_send_block	Send information to a specific processor.
CMMD_swap	Swap information between two processors.
CMMD_sync_with_nodes	Global synchronization between all processors.
CMMD_scan_double	Perform a scan on specified information i.e. add up all the values on each of the nodes.
CMMD_open_send_channel	Open a virtual channel between two nodes. Future sends of the same size to the same processor can be done with less overhead.
CMMD_write_channel	Write the information to an open virtual channel.

CMMD provides many 'standard' message passing capabilities but also has many extras, especially those dealing with global operations and reducing communication overhead.

2.4.4 PVM

PVM (Parallel Virtual Machine) is a package of library routines and two executable programs that make a network of UNIX workstations into a single parallel virtual machine [13]. The two executable programs pvmd and pvm are described below:

pvm — The console program used to configure the virtual machine, show the status of the virtual machine and tasks, and aide with debugging.

pvmd — The daemon that controls the communication between hosts. Only one daemon runs on a host even if the host has multiple processors, in which case PVM uses the native message passing scheme developed for that particular multi-processor.

PVM allows users to develop applications in FORTRAN 77 or C and link with libraries that provide message passing capabilities similar to those available on the CM-5 using the CMMD libraries. One of the main advantages of PVM is that it is available via anonymous ftp, thus free of charge. Portability is also a strength of PVM, because the 'virtual machine' can be made up of a group of heterogeneous computers. Table 2-8 lists the platforms on which the current version of PVM, 3.3.7, can be used [30].

Table 2-8: Platforms For Which PVM 3.3.7 is Available [30]

AFX8	Alliant FX/8
ALPHA	DEC Alpha/OSF-1
ALPHAMP	DEC Alpha multiprocessor/OSF >= 3.0
BAL	Sequent Balance
BFLY	BBN Butterfly TC2000
BSD386	80[34]86 running BSDI, 386BSD, NetBSD, FreeBSD
CM2	Thinking Machines CM-2 Sun front
CM5	Thinking Machines CM-5
CNVX	Convex using IEEE floating-point
CNVXN	Convex using native f.p.
CRAY	Cray
CRAY2	Cray-2
CRAYSMP	Cray S-MP
CSPP	Convex Exemplar SPP
DGAV	Data General Aviion
E88K	Encore 88000
HP300	HP 9000 68000 cpu
HPPA	HP 9000 PA-Risc
I860	Intel RX Hypercube
IPSC2	Intel IPSC/2
KSR1	Kendall Square
LINUX	80[34]86 running Linux
MASPAR	Maspar/Dec Mips front-end
MIPS	Mips
NEXT	NeXT
PGON	Intel Paragon
PMAX	DEC/Mips arch (3100, 5000, etc.)
POWER4	IBM Power-4
RS6K	IBM/RS6000
RT	IBM/RT
SCO	80[34]86 running SCO Unix
SGI	Silicon Graphics IRIS
SGI5	Silicon Graphics IRIS OS >= 5.0
SGIMP	Silicon Graphics IRIS multiprocessor with OS >= 5.0
SGI64	Silicon Graphics IRIS OS >= 6.0
SGIMP64	Silicon Graphics IRIS multiprocessor with OS >= 6.0
SUN3	Sun 3
SUN4	Sun 4, 4c, sparc, etc.
SUN4SOL2	Sun 4 running Solaris
SUNMP	Sun 4 multiprocessor
SX3	NEC SX-3
SYMM	Sequent Symmetry
TITN	Stardent Titan
UVAX	DEC/Microvax
UXPM	Fujitsu running UXP/M
VCM2	Thinking Machines CM-2 Vax front

Because it has gained such widespread use, help and discussion about PVM can found in the newsgroup comp.parallel.pvm. A full description of PVM, where to get it and how to develop PVM applications can be found in the book PVM [13].

2.4.5 *MPI*

MPI, the Message Passing Interface, is a library of message passing routines. This library defines the programming environment. The implementations of the MPI library are left up to the vendors of multi-processing machines and the designers of software for parallel computing over a network of workstations. MPI promotes the development of complex parallel software by standardizing the interface to the programmer [21].

The MPI library was designed with the implementor as well as the programmer in mind [21]. The MPI standard includes many complex functions useful to the programmer. The library has also attempted to allow specialized parallel environments to achieve the highest level of performance. The goal of MPI was to achieve efficiency without sacrificing portability or functionality. [21].

Developing parallel software for an MPI environment has potential to be portable to a variety of platforms well into the future. There is no guarantee that this standard will become widely used, however.

2.4.6 *SOLARIS Threads*

A SPARC multiprocessor platform was recently purchased during the author's time at Draper Laboratory. The workstation purchased was a SPARC 20-514, having four processors using shared memory [26]. A threads library for multi-threaded application development was provided with the operating system, SOLARIS 2.3 [22]. To the author's knowledge, no other method for parallel program development came with the multi-processing platform. Although a multi-threaded application differs from a message passing system,

as described in the previous section, some of the same functionality is available within both systems.

The threads library available could only be used with the C programming language. No FORTRAN 77 library was readily available. POSIX threads were not available for this machine when the research was initiated.

2.4.7 *LINDA*

Network Linda creates a virtual bulletin board, known as tuple space [17]. Processors that have work to do post it on the bulletin board while processes that are ready to do work pull a tuple off the bulletin board and work on the tuple. The processor then posts the results back on the board.

Linda runs on a network of workstations. It's primary advantage is its ease of use. There are only six commands associated with Linda and the specifics of the sending the messages are removed from the application developer [25].

A full examination of Linda was not performed as a part of this review, as Network Linda is a proprietary product. However, much information was available on Network Linda in references [17], [24], and [25].

Each of the specific approaches to interprocess communication has inherent advantages and disadvantages. The impact of the approach used on the project goals is examined in Chapter 3.

3.0 A Parallel Semi-Analytic Satellite Propagator

Chapter 3 describes the development of a parallel semianalytic orbit propagator. The parallel semianalytic orbit propagator combines an application architecture based on the PVM networking software with the Draper Semianalytic Satellite Theory (DSST). The resulting capability will be referred to as the PVM/DSST.

3.1 Software Development Goals

Developing the PVM/DSST first required identification of clear goals to guide the software development decisions. These goals originated from project requirements and years of experience in flight dynamics software development at Draper Laboratory. The goals are:

- Longevity
- Portability
- Simple Design and Interface
- Low startup costs
- Performance

Each goal is detailed in Sections 3.1.1 through 3.1.4.

3.1.1 Longevity

Many complex software systems used for satellite flight dynamics have experienced a long lifetime [27]. Examples include:

- DELTA — NORAD space surveillance system operational from the mid 60's to approximately 1980.
- AEOS — A system used by the Air Force Space Control Facility (AFSCF) in Sunnyvale, CA. Developed in the late 1960s. Software was in use until the late 1980s.
- GTDS — Developed by the mid 1970s. Still used at the present time.

- 427M — NORAD space surveillance system operational from approximately 1979 until the present time.
- TRACE — Developed by the Aerospace Corporation in the late 1970s. Still used at the present time.

These applications involve significant investments of time and money. Due to the cost and complexity of such flight dynamics systems, their useful lifetime has often been in excess of twenty years.

Draper Laboratory has been developing Flight Dynamics systems since the late 1970s under the direction of Dr. Paul Cefola [27]. The Goddard Trajectory and Determination System (GTDS) has been in use for the past twenty years [28]. Draper's version of GTDS is known as the Research and Development GTDS (R&D GTDS).

Development of GTDS began at Goddard Space Flight Center in 1970 [28]. GTDS contains a versatile set of tools and algorithms to perform flight dynamics functions for space systems [28]. Although originally developed to run on an IBM mainframe, this system has also been successfully ported to a VAX/VMS workstation, UNIX workstations including SUN and SGI, and an IBM PC [29][1]. The software has outlived the computers for which it was originally designed. As with GTDS, future software will continue to outlive the hardware; therefore new software developments should incorporate longevity into an application design. Longevity can be achieved in a number of ways including:

- Maintain software to the most current versions of hardware and operating systems available so that updates to hardware and operating systems versions will require fewer software changes.
- Develop a software system in a hardware environment predicted to have a long lifetime.

---

[1] The porting of GTDS to a VAX/VMS, SUN/UNIX, and SGI/UNIX took place at Draper Laboratory. The IBM PC version was done by Phillips Laboratory with support from Draper Laboratory.

- Develop software to a programming language standard predicted to have a long lifetime.

In reality, a combination of the above three methods for incorporating longevity into an application will have to be used. Designers should consider longevity when making software decisions, however.

Another important step to ensure a long lifetime is developing software under configuration management [78, 79, 80]. This allows future users to understand the updates that have taken place and encourage new development without fear of damaging the current system.

3.1.2 *Portability*

To be effective, software must be portable to variety of platforms. This requirement is closely tied with longevity. Software that is dependent on one platform will be ineffective once that platform is outdated. Portable software can also experience wider use, as more people can use the software without obtaining new resources. Finally, this requirement is necessary in developing a parallel application to execute on a network of heterogeneous workstations.

3.1.3 *Simple Design and Interface*

A new software application will not experience wide use if the interface is very complicated. The program flow and its capabilities must be understandable, flexible, and modifiable. Previous experience shows that a software system will be modified and expanded as requirements change. As computers go out of date, the software will have to be adapted to new architectures, which can be a tedious task if the software architecture is not easy to understand.

3.1.4 *Low Startup Costs*

Due to time limitations the PVM/DSST must not require excessive time to develop and test. To ensure this requirement is met, it will be very important to use as much legacy software as possible. This will also ensure that a valuable product is developed without re-inventing "the wheel".

3.1.5 *Performance Increase*

The primary goal of the PVM/DSST is to increase the capability to analyze multiple satellites. Creating a parallel application allows the necessary computational requirements to be met without software changes, if the software takes advantage of the multiprocessor environment. More important than the actual performance in this analysis is the performance that can be attained by a theoretical system. Future users will undoubtedly have more and faster processors than were available during the time of development. By demonstrating what can be expected of the parallel propagator on the computers currently available, the type of computers required in the future can be scaled to meet the computational requirements.

3.2 Software Design Process

The propagation method was chosen from the beginning to be the DSST. The effectiveness of this method was thoroughly explained in Chapter 1. In addition, the algorithms and software were developed at Draper Laboratory and many experts who participated in the development were available to answer questions. Software had already been written to implement this orbit propagation theory, thus there was a significant amount of legacy software that could be used.

3.2.1 *Target Environment Selection*

The first decision made before designing a parallel version of the propagator was the selection of the target environment. The options considered were:

- Using the CM-5 at LCS exclusively
- Using the four-processor SPARC 20-514 at Draper Laboratory.
- Using a network of UNIX workstations at Draper Laboratory.

Development on the CM-5 is advantageous due to the existence of a tested system of parallel programming libraries; therefore both data parallel and message passing approaches (Chapter 2) could be considered. The four-

processor SPARC is equipped with a threads library, making it possible to develop a multi-threaded application. POSIX threads were not available to the author for this platform. In addition, a FORTRAN 90/HPF compiler could have been purchased for this platform, which could have been used in a data parallel approach.

Developing on a network of workstations limited software to the message passing approach. One of the software packages described in Chapter 2 would be used to make a parallel computer from a heterogeneous collection of workstations. Several packages were readily available to promote this development, although the maturity of these packages was inferior to that of the CM-5. Only PVM was truly considered, as it was readily available (at no cost), it had already been successfully used at Draper Laboratory, and it was supported on the available workstations. Table 3-1 delineates the development options.

Table 3-1: Development Options

Hardware	**Parallel Programming Environment**	**Associated Paradigm**
CM-5	CMMD CM-FORTRAN PVM	Message Passing Data Parallel Message Passing
SPARC 10-514	SOLARIS Threads FORTRAN-90 PVM	Multi-Threading Data Parallel Message Passing
UNIX Cluster	PVM	Message Passing

3.2.2 Chosen Design

3.2.2.1 Paradigm Choice

The first decision to be made was the paradigm to be used. Table 3-2 outlines the impact of each paradigm on the development goals.

Table 3-2: Impact of Paradigm Choice on Project Goals

Goals	Data Parallel	Message Passing	Multi-Threading
Longevity	Would expect wide use into the future.	Would expect wide use into the future.	Would expect wide use into the future.
Portability	Currently limited to machines with special compilers.	All platforms available.	Limited to multi-processor SPARC.
Ease of Use	Easy to program. Algorithms would have to be redesigned.	Difficult to program. Can use old algorithms depending on algorithm choice.	Difficult to program. Algorithms would have to be re-written.
Low Startup Cost	Forced to redesign existing code parallelism.	All legacy software can be used.	Some legacy software can be used. Some sections must be rewritten.
Performance	Expected to be very good.	Depends on match between granularity and hardware.	Expected to be very good.

As table 3-2 highlights, longevity is predicted to be sound for all three types of parallel programming, as they all experience wide usage and should continue to be supported well into the future.

Because of the systems available, message passing is the most portable paradigm. PVM programs can be used on all three hardware systems. Thus PVM brought the portability to the message passing paradigm. Of course, if the CMMD Library was used for message passing, it would only work on the CM-5.

As Tables 2-11 and 2-12 pointed out, the data parallel model is the easiest to program. However, it would require rethinking many serial algorithms to create parallel operations.

Message passing would have the lowest startup costs, as serial code can be used as is. Both multi-threading and data parallel paradigms require a change in existing code.

Performance is the most difficult goal to compare against the paradigms. Performance is much more dependent on how the paradigm is implemented than on the paradigm itself.

Because portability and low startup costs were key issues in the decision, message passing was the paradigm chosen. This choice allowed the quickest development of a usable product as the legacy software could be easily incorporated into the new system.

3.2.2.2 Message Passing System

PVM and CMMD were the options available for a message passing system. Although CMMD provided more functionality, PVM was chosen because of its portability. The target platform would be a network of loosely connected workstations within Draper Laboratory. As pointed out in Table 2-8, however, a PVM application would also work on the CM-5. Additionally, PVM provided the benefit of developing applications within Draper Laboratory, thereby allowing the author more access to the computers. Although CMMD provided more reliability and specialized functionality, the portability and accessibility of PVM made it the better choice for this application.

MPI was not fully examined as it was fairly new at the time the decision was made and PVM provided the necessary portability. MPI would enhance longevity, however, as new message passing systems conform to the MPI standard. This application can easily be updated to an MPI application by

changing the PVM function calls. A section of the MPI manual describes how to move PVM applications to MPI [21].

3.2.3 *PVM and the DSST*

The target environment for this software was a loosely connected set of workstations, which complements a coarse grain decomposition of the problem. Although this limits speed-up, as demonstrated by Amdahl's Law (eq. 2-3) the communication costs across a network of workstations could be very expensive. Communication would be particularly expensive if there is a significant amount of other network traffic while the application is executing.

3.2.4 *Software Implementations of the DSST*

Before describing the new software designed, it is important to examine the legacy DSST code available for integration into the PVM/DSST.

At Draper Laboratory, two implementations of the DSST already existed in tested software prior to this project's inception. One version was contained within GTDS; the other was a separate utility that consisted of only the propagator [32, 61].

GTDS is controlled by files known as card deck inputs. A procedure that links the card deck, data files, and output files to the appropriate files sets up the environment for a GTDS run. The commands in the card deck are then executed by GTDS. A sample card deck that would propagate a satellite ahead five years is seen in Figure 3-1 [35].

```
CONTROL   EPHEM                                                  TOPEXXX     XXXXXX
EPOCH                 950401.0            000000.000000
ELEMENT1  3  6  1     7300.0000000000000D0 4.83000000000000E-4 65.9000000000000D0
ELEMENT2              330.4000000000000D0 271.3000000000000D0 73.2000000000000D0
OUTPUT    2  2  1     1000401.0           000000.0            31570560.0
ORBTYPE   5  1  2     43200.0             1.0
OGOPT
SPSHPER   1
SCPARAM               0.0001              1000
ATMOSDEN        1
DRAG      1           1.0
MAXDEGEQ  1           21.
MAXORDEQ  1           21.
POTFIELD  1  4
END
```

Figure 3-1: Sample GTDS card deck [35]

The other software implementation of the DSST existed autonomously in FORTRAN 77 code. This software, known as the stand-alone propagator, is described in the document by Early [32], as well as a study performed by Jablonski (Boelitz) [33]. This software was written to be portable, allowing the DSST to be implemented on a variety of platforms with various driver programs. Interface into the DSST was through four subroutine calls. Setup information and options are passed in through the argument list to the propagator, although many options are hard coded throughout the software. Because the stand-alone was completed much later than GTDS and written to be FORTRAN 77 compliant, the software is much easier to work with.

The GTDS version of the DSST contained more functionality than the stand-alone, implying that a parallel GTDS could potentially accomplish more than satellite propagation. For this project, the GTDS benefits were outweighed by the ease of use of the stand-alone propagator. The GTDS system would require re-creation of the card decks to start the run. Also the output files did not easily lend themselves to multiple satellite data analysis.

3.2.5 Software Design Considerations

With the stand-alone propagator chosen as the basis for the orbit propagator and the PVM utility chosen for message passing, the top level software requirements were established. Table 3-3 describes the various software design approaches considered, ranked in order of respective granularity, from the finest grain algorithm considered to the coarsest.

Table 3-3: Advantages and Disadvantages of Approaches Considered

General Concept	Advantages	Disadvantages
1] Use a parallel numerical integration scheme.	Speedup predicted in all types of processing.	Fine grain parallelism is required.
2] Calculate all mean elements, then use the results to calculate the short periodic contributions across multiple processors.	Speedup increases with more processors as long as there are enough short periodic points to be evaluated. Could be valuable in a differential correction (DC) algorithm.	Speedup only in evaluating short periodic elements. Limited by serial mean element generation.
3] Propagate different satellites on different processors.	Little communication overhead. Can use some legacy software without modification.	No speedup for just one satellite.

The third concept in Table 3-3 was chosen for implementation because it best fit development goals. The main advantages to this approach are:

- No change required in the DSST algorithm which was already coded and tested
- Coarse grained nature promised high work time/communication ratio.
- Scalable to as many processors as desired as long as number of satellites to the number of processors ratio is high enough.

The disadvantage of this design is that one satellite could not be propagated at a greater speed. The algorithm is only useful in propagating multiple satellites.

The final design is a combination of all the limitations and goals discussed in Sections 3.1 and 3.2. The design is particularly useful in examining the long term evolution of multiple satellite constellations, a capability which is exploited in Chapter 4.

3.2.6 Load Balancing Methods for Parallel Computing

Developing parallel algorithms on a heterogeneous group of processors presents a challenging load balancing problem. It is impossible to know the speed with which different processors will perform their work prior to the creation of the next task. On a dedicated, homogeneous system such as the CM-5, all processors can be assumed to work at nearly the same speed. The software can be designed to evenly apportion the tasks among the available processors. In a heterogeneous environment of workstations, many factors enter into how fast a processor will perform a desired task:

- Clock speed of processor.
- Processor architecture, i.e. RISC, CISC, pipelining, co-processors.
- Load on processor.
- Memory/Cache usage.
- Physical location of disk and network traffic between processor and disk.

If all task assignment is done before the computation begins, a computer heavily loaded with users might be given most of the work. All the other processors would have to wait until the heavily loaded computer finished its tasks. Even harder to predict, the path between the processor and the disk with the data files could be heavily loaded, increasing disk access time. To counterbalance these problems, a load balancing technique is used.

There are many sophisticated ways to approach this problem. Some algorithms may periodically measure machine loads and distribute work based on a combination of machine capacity and load at that time. The 'manager' could later redistribute work based on load averages or the performance of a particular machine.

These algorithms can be very useful but may be difficult to implement. For this application, a much simpler but effective approach was taken. The balancing method used was known as the 'pool of tasks' algorithm [13]. The terms *process* and *task* are used in very specific ways in this discussion; therefore they should be clearly defined before continuing.

process An executing program on the CPU or in memory on a UNIX machine.

task A job waiting to be executed by a process.

In this distribution algorithm, all the tasks are controlled by a 'master' process. This process has all the information necessary to perform each task. To start, the master creates slave processes distributed among the virtual machine. More slave tasks can be assigned to faster machines, but PVM distributes the tasks evenly among the machines by default. The master process then sends out one task to each process. When a task is finished, the slave process will return a message to the master indicating that it is done as well as an ID. The master will then send this process the next task.

In this way, the processors that work the fastest will do the most work. Once the master has sent out all the tasks, it waits for those tasks to be finished before continuing. The 'pool of tasks' distribution algorithm is depicted in Figure 3-2.

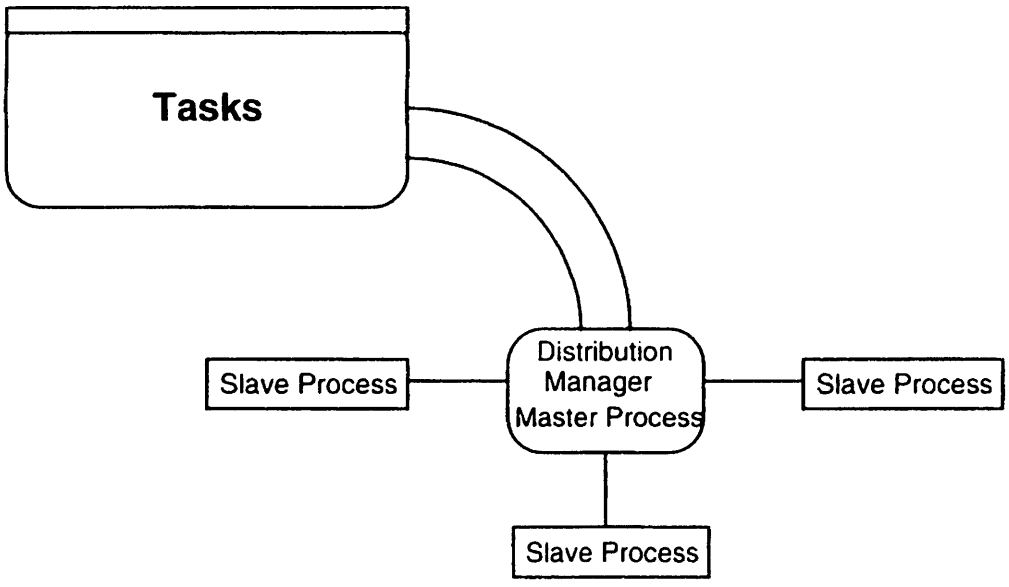


Figure 3-2: The pool of tasks algorithm.

This algorithm works most effectively when there is a high task-to-processor ratio. If there is only one task per processor, for example, the faster processors will be waiting while the slower processors finish their one job.

3.2.7 Programming Language Choice

PVM is compatible with both FORTRAN 77 and C programming languages. The DSST stand-alone propagator was written in FORTRAN 77. To minimize the interface problems with the DSST, the additional code was also written in FORTRAN 77. C could have been used, but there are occasional difficulties in calling FORTRAN routines from C programs, not the least of which is the requirement of having two different compilers to create the one executable. FORTRAN 90, a superset of FORTRAN 77, would have been considered, but the lack of compilers limited portability.

3.3 Software Description

This section describes the software implementation of the design decisions described in Section 3.2.

3.3.1 Top Level Software Design

There are two different methods of writing the required software: keeping just one executable for the 'master' and 'slave' or dividing the code into two different executables. Keeping just one executable is also known as the 'hostless' programming model; dividing the code into two executables is also known as the 'host-node' model.

The hostless programming model is depicted in figure 3-3.

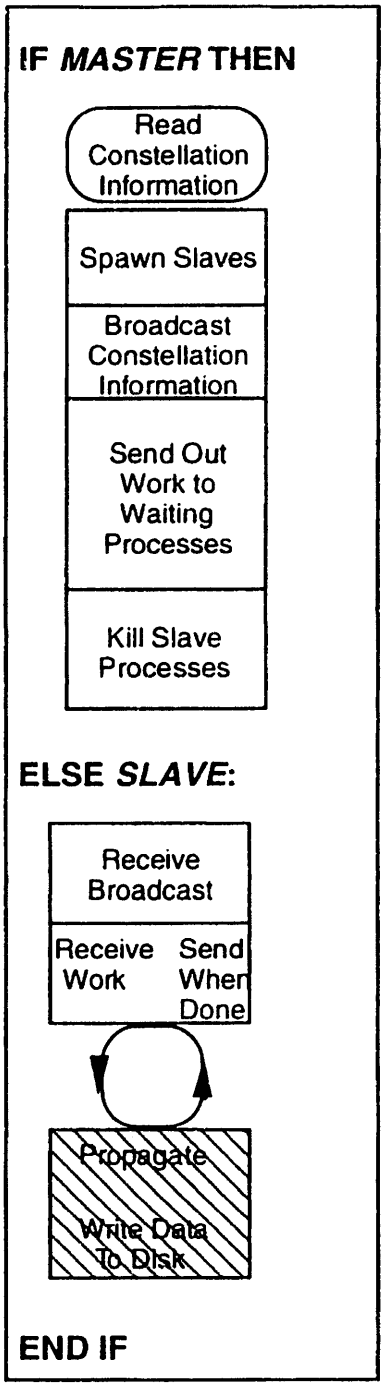


Figure 3-3: Program Flow for the Hostless Programming Model

The software used the PVM function *pvmfparent()* to determine if a particular process was the master or slave. This function is not unique to PVM; most all message passing facilities have such a capability. If the process decides it is the master process, it creates several slave processes. It then continues to manage the tasks and slave processes. Upon completion, the

master finally kills all its slaves before exiting. If a process decides it is a slave, it waits to receive data before beginning computation.

The host-node programming model is depicted in figure 3-4.

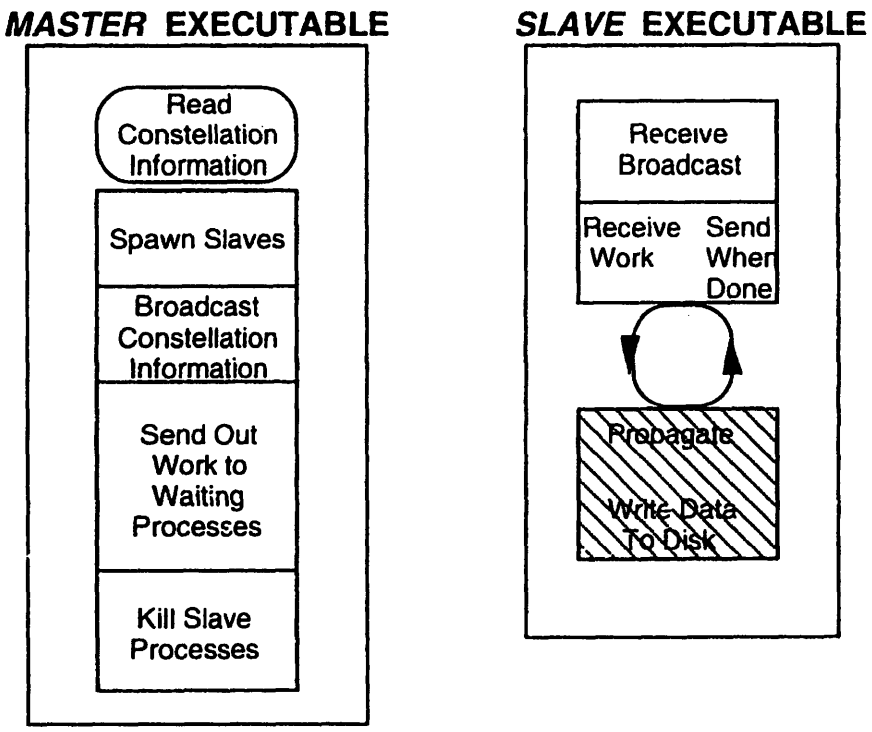


Figure 3-4: Program Flow for the Host-Node Programming Model

Dividing the code into two pieces made the project slightly more difficult to manage, but simplified the building process for the master. The master did not need to be linked with the functionality of the orbit propagator, making it a much smaller program than the slave executable.

Keeping all functionality in one executable was easier to maintain when the software was being developed. Software changes in one function often required changes in the other. Once the software was developed and tested, dividing the software into two executables was more efficient; the master executable did not have to be linked with the DSST software. For the initial version of the PVM/DSST just one executable was created. When used with the optimization tool (Chapter 4) two executables were used.

Figure 3-5 depicts the overall structure of the parallel orbit propagator written using the hostless programming model.

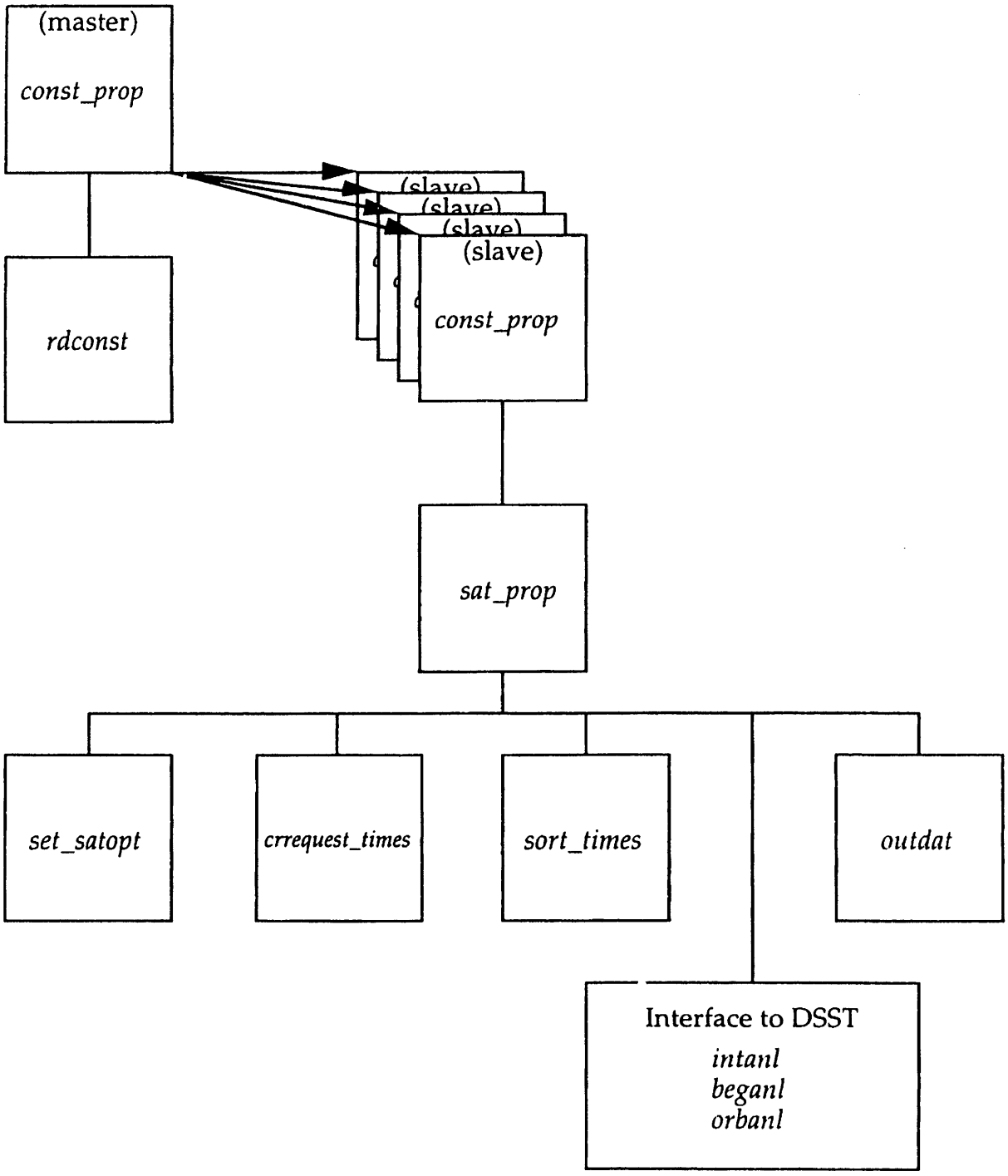


Figure 3-5: PVM/DSST Structure with the Hostless Programming Model

A more detailed program flow of the PVM/DSST is depicted in figure 3-6.

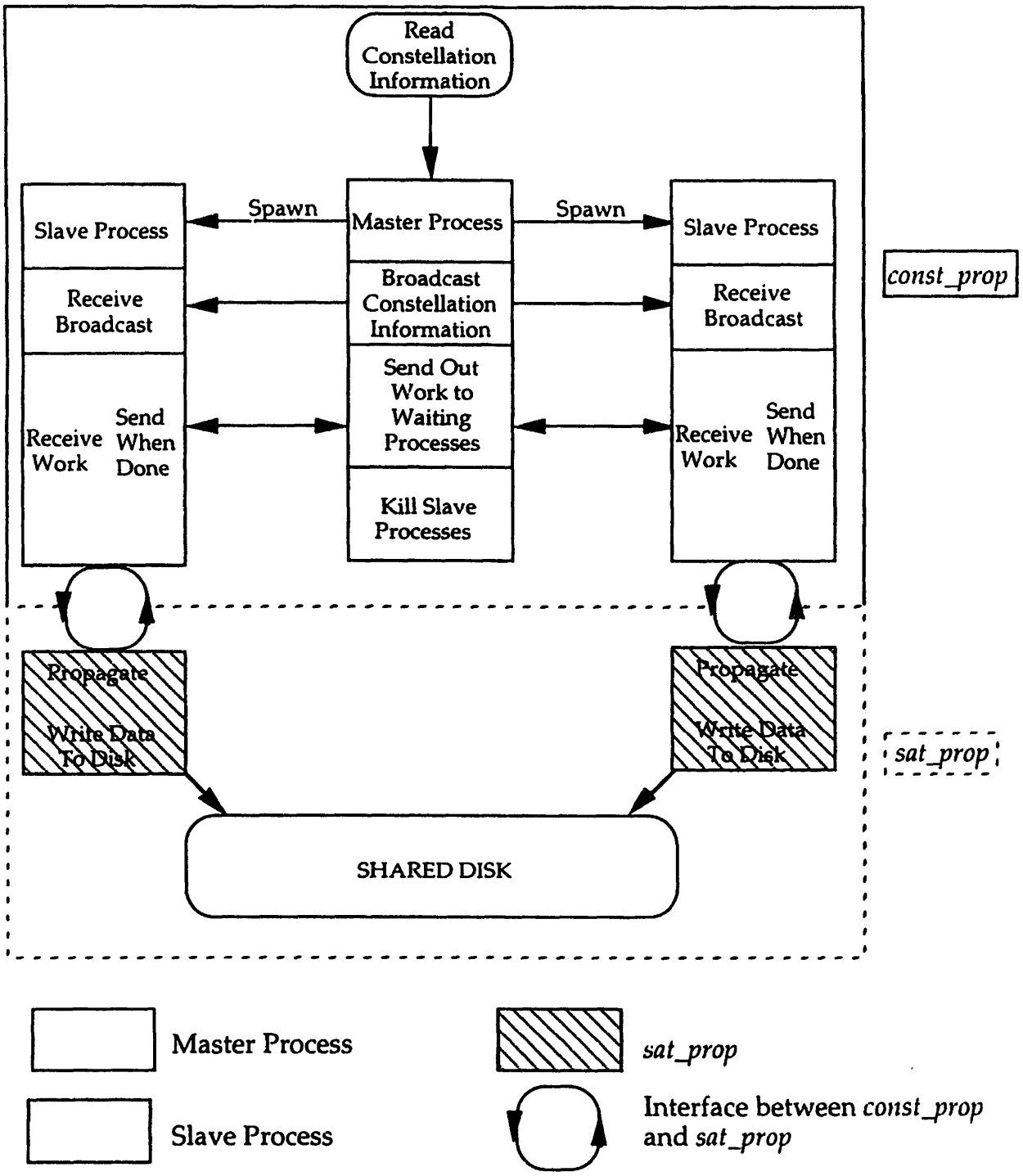


Figure 3-6: Flow of the parallel orbit propagator

3.3.2 *Process Distribution Manager: const_prop*

The process distribution manager, *const_prop* (constellation propagator) creates multiple copies of itself across multiple processors. Each slave then calls the subroutine *sat_prop*. The subroutine *sat_prop*, does not execute in parallel; it will have a single thread of control and proceed through the satellite propagation serially. The distribution manager uses the pool of tasks algorithm, discussed previously, to best distribute the jobs among the available processors.

In creating *const_prop*, a decision had to be made as to which of the *sat_prop* variables would be specified to be the same across the entire constellation, a constellation global parameter, or specific to an individual satellite, a satellite local parameter. PVM provides a global broadcast capability for more efficient communication of one message to all processes. Additionally, a global message is only sent once at the beginning of a propagation run rather than with every satellite. Therefore, it is desirable to move as much data as possible into the constellation global parameters to reduce communication costs. All the necessary data and a description of each of the global and local data items is shown in Tables 3-4 and 3-5.

Table 3-4: Constellation Global Data

Data Item	Description	Name Given in *const_prop* Input File (Fig. 3-10)
Number of Satellites	Total Number of satellites in the constellation	N Satellites
Element Type	Description of the input element set	ElType
Number of Intervals	Number of time intervals through which the satellite is propagated. In each interval, an equal time step is used	Nintervals
Time Intervals	Time of interval. Beginning Time, Ending Time, and a Timestep must be given for each interval	Begin Interval End Interval Deltat
Number of Burns	Total number of burns	Nburns

Table 3-5: Satellite Local Data

Data Item	Description	Name Given in *const_prop* Input File (Fig. 3-10)
Satellite Number	Unique number identifying each satellite	Satellite Number
Epoch Time	Time information given for the satellite's epoch date and time	Epoch Date Epoch Time
Satellite State	The element set in either Keplerian or Equinoctial elements	Equinoctial Elements or Keplerian Elements
Coefficient of Drag	Coefficient of drag	CD
Rho One	The value of Rho	Rho One
Spacecraft Mass	Mass of the spacecraft in kg	S/C Mass
Spacecraft Area	Area of satellite that sees drag effects	S/C Area
Integrator Stepsize	Step size used for numerical integration in seconds	Integrator Step
Retrograde Factor	Retrograde factor for equinoctial elements	Retro
Atmospheric Model	Describes which atmospheric model to use	Atmos Mdl
Potential Model	The model for the spherical harmonics of the Earth	Potent Mdl
Maximum Degree	Maximum degree of the central body spherical harmonic used in propagation	Nmax
Maximum Order	Maximum order of the central body spherical harmonic used in propagation	Mmax
Central Body Zonal Harmonic Averaging Option	Whether to use: 1) Analytic Averaging 2) Numerical Averaging 3) Off method of averaging	Izonal
J2 Squared Effect	Whether to include J2 squared effect 1 - Yes 2 - No	IJ2J2
Maximum resonant order	Maximum resonant order	Nmaxrs
Maximum resonant degree	Maximum resonant degree	Mmaxrs
Third Body Perturbation	Whether to use: 1) Analytic Averaging 2) Numerical Averaging 3) Off method of averaging the third body contributions	Ithird
Atmospheric Drag	Whether to include atmospheric drag 1 - Yes 2 - No	Ind Drg
J2 Height Correction for Drag	Whether to compute ISZAK's height correction for atmospheric drag 1 - Yes 2 - No	Iszak
Solar Radiation Pressure	Whether to include solar radiation pressure 1 - Yes 2 - No	Ind Sol

This division into constellation global and satellite local data was useful for the work predicted to be done during the author's time at Draper Laboratory. It may need to be changed in future applications. Both the propagator shell and the distribution manager will require code changes, which simply requires moving the send commands in the distribution manager between local and global positions in the program. Similarly, the receive commands at the top of the propagator shell will have to be moved between global and local positions. These changes should be fairly straightforward.

3.3.3 *Propagator Shell:* sat_prop

The propagator shell was designed to provide flexibility when used with a variety of applications. As described earlier, the stand-alone DSST legacy code was used for orbit propagation, and the propagator shell was designed around this software to implement the propagator. A previous implementation of the propagator was used as a starting point for the shell design. This shell, known as ORBIT_PROPAGATOR_SERVICES (OPS), was used by Draper Laboratory as a mean element propagator for maneuver planning purposes. It accepted a keyword and the necessary data for that keyword, and returned the values requested. This shell was written by David Carter at Draper Laboratory for the Landsat 6 project [34]. Much of the input information, however, was read in from a precision mean element file (PME file) and then loaded into the appropriate common block before implementing the propagator. Figure 3-7 describes the external interface to OPS.

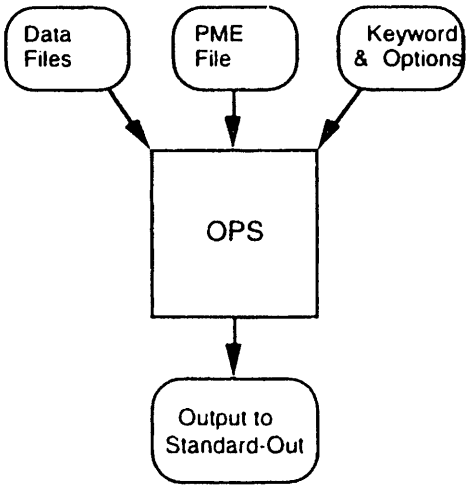


Figure 3-7: External Interface to OPS.

To make the interface between *sat_prop* and other applications, such as *const_prop*, simpler, all options were passed in through the argument list. Data files were still necessary to run the orbit propagator, but the file describing the satellite and a few propagation options, the PME file in OPS, was removed. Figure 3-8 describes the interface to *sat_prop*.

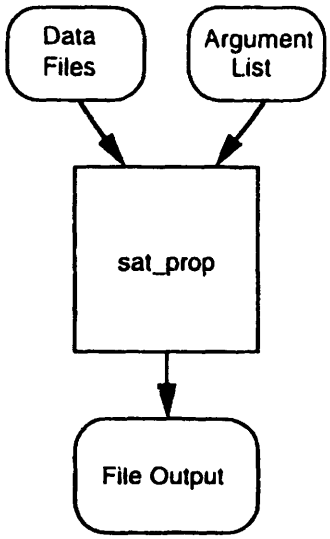


Figure 3-8: External interface to *sat_prop*.

An additional change was made to the propagator shell to reduce the amount of data that would have to be sent from the master to each slave. Rather than specifying particular request times in seconds from epoch, an interval with a start time, stop time, and time step was used. Multiple intervals could be passed as well. This functionality turned out to be very valuable when using the propagator.

The argument list to the subroutine *sat_prop* became:

```
subroutine sat_prop(satno, eltype, nintervals, intervals,
                    nburns, burn_list, satopt_int, satopt_dbl,
                    outfile, indata_path)
```

Table 3-6: Argument Description for Subroutine *sat_prop*

VARIABLE	TYPE	DESCRIPTION
satno	integer*4	Satellite number used to describe the output file.
eltype	integer*4	Integer flag describing the element set type. 1 - Keplerian 2 - Equinoctial
nintervals	integer*4	Number of intervals to propagate through.
intervals	real*8(5,*)	Interval description. Five numbers per interval. 1&2 - Begin Date and Time 3&4 - End Date and Time 5 - Time step
nburns	integer*4	Number of impulsive burns entered.
burn_list	real*8(4,*)	Burn information. Four numbers per burn. 1 - Burn time in seconds from epoch. 2 - Tangential burn impulse (m/sec). 3 - Normal burn impulse (m/sec). 4 - Radial burn impulse (m/sec).
satopt_int	integer*4(*)	List of integer options described in Table 3-5, satellite local data.
satopt_dbl	real*8(*)	List of real*8 options described in Table 3-5, satellite local data.
outfile	character	Output filename with path.
indata_path	character	Full path of input files.

The six necessary data files that are required in the new propagator design are seen in Table 3-7.

Table 3-7: Data Files

Name of File	Description
epotfld	Earth potential models file
jacdat	Jacchia data for drag information
slp1950	Solar, Lunar, Planetary ephemeris file in Mean of 1950 coordinates
slptod	Same as above in GTDS true-of-date coordinates
timecoef	Timing coefficients file
newcomb	Newcomb operators file

There are several other integration options that are hardwired in *sat_prop*. These options are listed in Table 3-8.

Table 3-8: Hardwired Propagator Options

Option	Variable	Current State of Option	Where The Option is Set
Mean or osculating input elements.	mean	Mean input elements	sat_prop
True of reference or mean of 1950 input coordinate system	mtod	True of Reference	sat_prop
Direction of integration	forward	Forward	sat_prop

3.3.4 Modifications to the DSST

The DSST stand-alone is a portable set of code, although a few changes were implemented when moving between platforms. Jablonski (Boelitz) [33] described how the DSST standalone could be used on a variety of platforms, including a VAX using VMS, IBM PC using DOS, a SUN SPARC station using UNIX, and an Apple Macintosh. Because the PVM/DSST software was built from the version of the DSST on the VAX/VMS and had to moved to a SPARC/UNIX environment, some small changes would have to be made.

However, when developing parallel software that works in a heterogeneous environment, it is desirable to have one set of source code that compiles on multiple platforms. One set of source code is much easier to manage, especially when developing new software, as changes only have to be integrated in one version.

The portability between platforms can be added without changing software by using a pre-processor [44]. Keywords are set that indicate the type of computer being used. This information is used by the computer to modify the software before it gets to the compiler, effectively rewriting the software for the particular platform.

The compiler used was the SPARCompiler FORTRAN 3.0, available to all the SUN machines at Draper Laboratory. This compiler applied the C-Preprocessor (despite its name, this preprocessor can be used successfully with FORTRAN 77) to all FORTRAN files ending in extension .F, converting them

to .f files, which were then compiled [45]. According to Dowd [17], this method of applying the preprocessor to FORTRAN programs is becoming standard. The C-Preprocessor functions are described in many texts, including [69].

The first change to be implemented when moving from a VAX/VMS environment to a SPARC/UNIX environment was to modify the length of direct access records. In a VAX OPEN statement, the keyword RECL is equal to the number of bytes in the record divided by four. On a UNIX platform RECL equals the number of bytes.

The preprocessor worked exceptionally well for this problem, providing code that worked on both a VAX/VMS and SPARC/UNIX platforms. This fix was made by adding the statements shown in Figure 3-9.

```
>cat setdaf.F
#include "machine.h"
#include "array_sizes.h"
      SUBROUTINE SETDAF
...

C     DEFINE FILE FOR SLP EPHEMERIS PERMANENT FILE
C     DEFINE FILE 14 (2500,566,U,ID:4)
      input_file = indata_path(1:i-1)//'slp1950'
      open(unit=14,
     1     form='unformatted',
     2      access='direct',
     3      recl=566*WORDLENGTH,
     4      file=input_file,
     5      status='old',
     6      readonly,
     7      shared)


>cat ../include/machine.h
#ifdef vax
#define WORDLENGTH 1
#endif

#ifdef unix
#define WORDLENGTH 4
#endif
```

Figure 3-9: Preprocessor modifications

At the user prompt, the command:

f77 -c setdaf.F

will first cause the C-preprocessor pass over setdaf.F before it sends it to the compiler. The commands to the C-preprocessor all start with a # in the first

column. The preprocessor then looks for the file machine.h and array_sizes.h and expands them into the file, setdaf.F. Examining machine.h, the preprocessor then sees the conditional statements '#ifdef vax' or '#ifdef unix'. If the machine is a vax, the term 'vax' will be defined; similarly for the term 'unix' on a unix machine. Assuming a unix machine was being used, the statement inside this conditional will be evaluated. The #define command replaces its first argument with the second argument everywhere it sees exactly the first argument in the code. In this case, the term WORDLENGTH is replaced with 4, so the right value is calculated for the RECL keyword on a SPARC/UNIX processor. Direct access files were opened in two places in the DSST standalone / OPS software, the subroutines *setdaf* and *satellite*. This code will, in the future, work on both the VAX and the SUN.

The other changes that had to be made included:

- Change all block data filenames to the format '....bd.for'. The SPARCompiler would not take filenames that have the same name as the block data. With this change, the software will still work on the VAX.

- The file *error_handler.for* contained many VAX specific routines. This file was not necessary for use in this project so it was commented out using the preprocessor. A VAX compilation would still make use of the error handler.

- Get rid of the VAX/VMS specific calls such as 'OPEN(SYS$INPUT)' in OPS. This was commented out using the preprocessor.

3.3.5 Support Software

Several other routines were also written to support this software effort. These routines are listed in Table 3-9. Listings of all software written is shown in Appendix B.

Table 3-9: List of Additional Software Developed

Name of Subroutine	Brief Description
crrequest_times.F	This subroutine converts a time range and interval into a list of times in seconds form epoch.
outdat.F	This subroutine writes the results of the propagation to disk.
rdconst.F	This subroutine reads the input file which contains the constellation data.
set_satopt.F	This subroutine assigns the values input through the argument list to the appropriate common block locations.
sort_times.F	This subroutine sorts the request times and burn times into increasing order. It also keeps track of which times were burns and which were requests for output.

3.4 Validation of the PVM/DSST

The job distribution logic and implementation was tested in two ways:

- The program const_prop was run through a debugger to ensure the correct messages were sent and received at the appropriate times.

- Use with the distribution manager produced the correct numerical values when compared to previous implementations of the DSST.

This implementation of the DSST was also validated in two ways. Results were first matched exactly to the test cases designed and used for verification of OPS in the Landsat 6 and Radarsat programs at Draper Laboratory. As the *const_prop* propagation software was the same as that used for OPS, these results should match to machine differences. Results were then compared with a GTDS semianalytic run.

3.4.1 *Comparison to previous tests*

The Landsat 6 tests used for comparison included [43]:

- State comparison after a four day coast
- State comparison 10,000 seconds after epoch following a tangential burn 1000 seconds after epoch.

The input file used to generate the four day coast in *const_prop* run is shown below, in Figure 3-10.

```
N Satellites:    1

nintervals:      1     ElType    2
Begin interval   1   19821025.0   000000.0
End    interval  1   19821031.0   0.00
Deltat interval  1   86400.0

nburns =         0

-----------------------------------------
Satellite Number:    1   Epoch Date: 19821025.0   Epoch Time:          0.0

Equinoctial Elements :          0.7077636704480000D+04
                                0.1564765048485586D-03
                               -0.8653247687711026D-04
                               -0.3855720457066417D-01
                                0.1154698444728130D+01
                                0.2305550252000000D+03

CD:                    2.00000000 Rho One:           0.00000000
S/C Mass:           1675.80454500 S/C Area:          0.00001379
Integrator Step: 43200.00000000

Retro:        1 Atmos Mdl:    1  Potent Mdl:  2
Nmax:        21 Mmax:        21  Izonal:      1  IJ2J2:    1
Nmaxrs:      21 Mmaxrs:      21  Ithird:      1
Ind Drg:      1 Iszak:        1  Ind Sol:     1
-----------------------------------------
```

Figure 3-10: Four day coast input file

The first, or truth runs, were performed on a VAX station 4000-90 while the tests were run on a SPARC 20-514. The comparison of the results are shown in Table 3-10 and Table 3-11.

Table 3-10: Comparison of *const_prop* against Landsat 6 test cases after Four Day Coast

Keplerian Element	Landsat 6 (Truth) VAX/VMS	*const_prop* SUN/UNIX	Absolute Difference
Mean Semimajor Axis (km)	7077.5976156445210	7077.5976156445200	1.00E-12
Mean Eccentricity	0.0003643952697747	0.0003643952697747	0.00E+00
Mean Inclination (deg)	98.2450676985650	98.2450676985646	3.98E-13
Mean Longitude of Ascending Node (deg)	2.04663721378651	2.04663721378651	0.00E+00
Mean Argument of Perigee	147.6042249337675	147.6042249337670	5.00E-13
Mean Mean Anomaly	175.4197101690098	175.4197101689990	1.08E-11

Table 3-11: Comparison of *const_prop* against Landsat 6 Test Case. Impulsive Burn 1000 Seconds After Epoch and Compare 10,000 Seconds After Epoch

Keplerian Element	Landsat 6 (Truth) VAX/VMS	*const_prop* SUN/UNIX	Absolute Difference
Mean Semimajor Axis (km)	7077.824590834495	7077.824590834480	1.5004E-11
Mean Eccentricity	0.000156411702392	0.000156411702393	1.00e-15
Mean Inclination (deg)	98.24471614261371	98.24471614261360	1.10E-12
Mean Longitude of Ascending Node (deg)	358.2020530693879	358.2020530693870	9.00E-13
Mean Argument of Perigee	124.1159642833155	124.1159642832740	4.07E-11
Mean Mean Anomaly	355.1142900226430	355.1142900226840	4.14E-11

The differences in both tables are attributable to machine differences. Differences of the same order of magnitude are apparent in different versions of GTDS [81].

3.4.2 Comparison to GTDS

To validate the results of the PVM/DSST, a GTDS run was performed using the card deck in Figure 3-1. Both the GTDS and the PVM/DSST were executed on SPARC processors. This card deck gave results in mean elements only so it would match the default setup of the PVM/DSST. The results of

this GTDS run, a five year EPHEM, are compared to the results of the *const_prop* results after five years. The input file used to generate the PVM/DSST run is shown in Figure 3-11.

```
N Satellites:   1    Eltype  1

nintervals:     1
Begin interval  1  19950401.0  0.0
End   interval  1  20000401.0  10.00
Deltat interval 1  31570559.82

nburns =        0

-----------------------------------------
Satellite Number:   1  Epoch Date: 19950401.0  Epoch Time:        0.0

Keplerian Elements:            0.730000000000000D+04
                               0.483000000000000D-03
                               0.659000000000000D+02
                               0.330400000000000D+03
                               0.271300000000000D+03
                               0.732000000000000D+02

CD:                  2.00000000 Rho One:         0.00000000
S/C Mass:               1000.00 S/C Area:        0.00010000
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    1 Iszak:       1  Ind Sol:     2
-----------------------------------------
```

Figure 3-11: PVM/DSST Input File for Validation of Software

The results, after five years, of both the GTDS and PVM/DSST test cases are shown in Keplerian Elements in Table 3-12. GTDS only outputs to 8 decimal places so the comparison was not made to the same precision as the comparison against the VAX/VMS OPS.

Table 3-12: Comparison of Results between GTDS and the PVM/DSST

Keplerian Element	GTDS Results	PVM/DSST Results	Difference
Semimajor Axis (km)	7289.671441	7289.671441	0.00
Eccentricity	0.000306449787	0.000306449787	0.00
Inclination (deg)	65.8961296	65.89961296	0.00
Longitude of the Ascending Node (deg)	14.54720186	14.54720186	0.00
Argument of Perigee (deg)	30.72517568	30.72517568	0.00
Mean Anomaly (deg)	20.02981688	20.02981683	5.0e-08

Table 3-12 shows that the PVM/DSST matches GTDS to the accuracy shown in the output files.

3.5 PVM/DSST Performance Analysis

Performance is very difficult to measure. Benchmarking computers is an involved procedure and not of primary importance to this discussion. This section concentrates on the performance of the software designed, as opposed to an analysis of the hardware. To describe the results, however, a description of the hardware environment is necessary. The hardware environment description will be followed by a description of the performance tests and results. The last section draws conclusions about the software design based on the test results.

3.5.1 Test Environment Description

Four computers were involved in the performance testing of the software. The machines all belong to Draper Laboratory and are associated with the ACME Lab within Draper Laboratory. The four computers used are described in Table 3-11 [26].

Table 3-13: Computer Description

Computer	Description
wile-e	SPARC ELC. SunOs 4.1.3 Operating System
coyote	SPARC 10-30 SunOs 4.1.3 Operating System
porky	SPARC 20-61 SunOs 4.1.3 Operating System
petunia	SPARC 20-514 Four Processors using a shared memory system. SOLARIS 2.4 Operating System.

The connection between the computers is crucial to the amount of communication overhead, as described in Chapter 2. The computers are connected as shown in Figure 3-12.

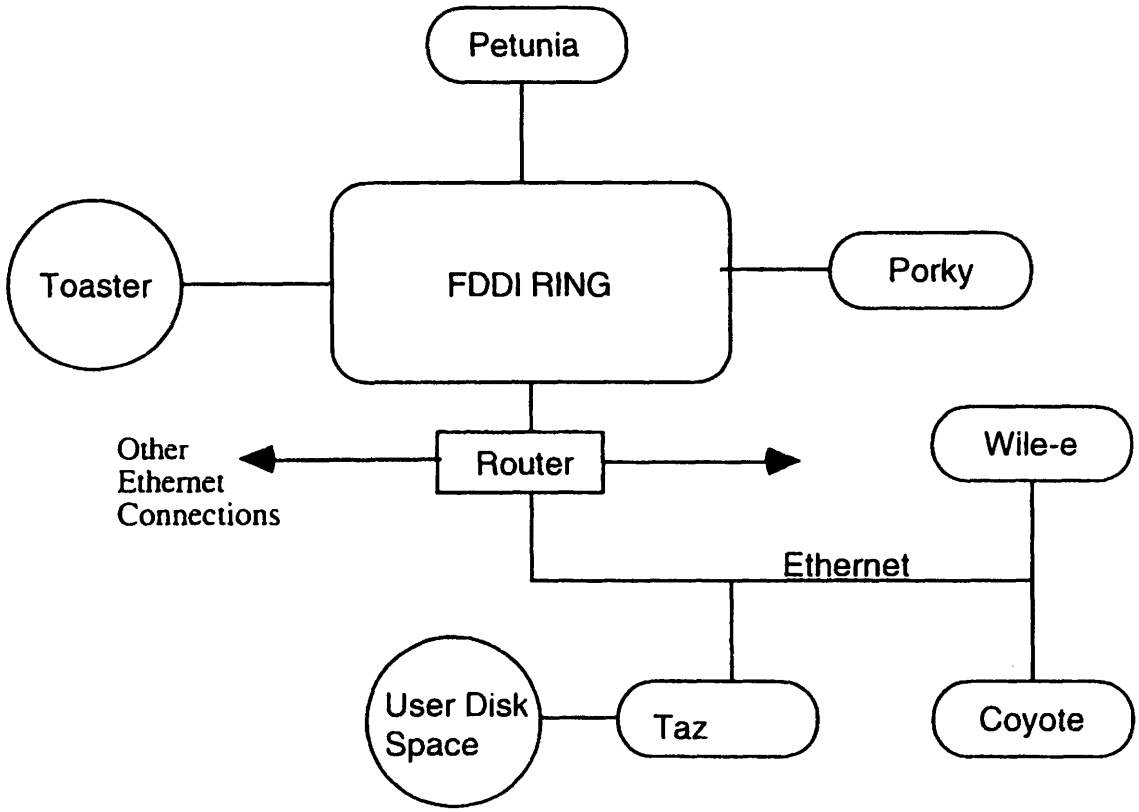


Figure 3-12: Hardware Configuration

The toaster is a network file server, and it contained the data files described in Table 3-6. The executable programs and timing results were on a disk

connected to taz, a SPARC 20-61. Otherwise, taz was not used in the performance testing.

The four computers were turned into fifteen different testing environments. Each system consisted of a combination of computers and test routines. This was done so test results could be easily associated with the correct system. The testing environments are described in Table 3-14.

Table 3-14: Description of Systems Timed

System	Computer(s)	Test Type
1	Porky	Serial
2	Coyote	Serial
3	Wile-e	Serial
4	Porky	Parallel
5	Coyote	Parallel
6	Wile-e	Parallel
7	Porky-Coyote	Parallel
8	Porky-Wile-e	Parallel
9	Coyote-Wile-e	Parallel
10	Porky-Coyote-Wile-e	Parallel
11	Petunia with 1 slave tasks	Parallel
12	Petunia with 2 slave tasks	Parallel
13	Petunia with 3 slave tasks	Parallel
14	Petunia with 4 slave tasks	Parallel
15	Petunia	Serial

Two different types of tests were run for performance analysis. The parallel test used PVM and distributed the tasks among each system. The serial test was used to perform the same calculations without the PVM overhead and using only one processor.

No system combining both the multiprocessor (petunia) and a single processor were timed as optimal process assignment on such a virtual machine would have required additional code. This mixed configuration was used for constellation analysis (Chapter 4). Additionally, the PVM implementation on the multi-processor platform was not bug free. Using the mixed configuration occasionally created problems. Problems with PVM are discussed more fully in Chapter 5.

Slight modifications were made to the programs *const_prop* and *sat_prop* to facilitate performance testing. The version of *sat_prop* was modified to become *sat_opt*, which was originally changed for constellation optimization. This program output the results of a cost function back through an argument list variable rather than writing a list of states to a file. Performance testing was more manageable using this version as no new files were created with each run. Similarly, the function *const_prop* was split into a master process, *const_opt* and its slave process *const_opt_slave*. These routines were created to run *sat_opt* and made testing easier as *const_opt* accepted the number of processes created at one time as a parameter in the argument list.

The program *timing* was used to time a test case on a particular configuration; *timing* passed the number of satellites to be propagated and the number of processes to create to the *const_opt* subroutine. The number of satellites was set as an UNIX environment variable, which could then be easily changed before running the program again [16]. The FORTRAN subroutine *getenv* was used to pass the information from the UNIX environment into the program.

The subroutine *const_opt* then created the requested number of processes evenly across the virtual machine and began sending out the satellites to each *const_opt_slave,* until each of the requested satellites had been propagated. Figure 3-13 depicts the structure of the program *timing*.

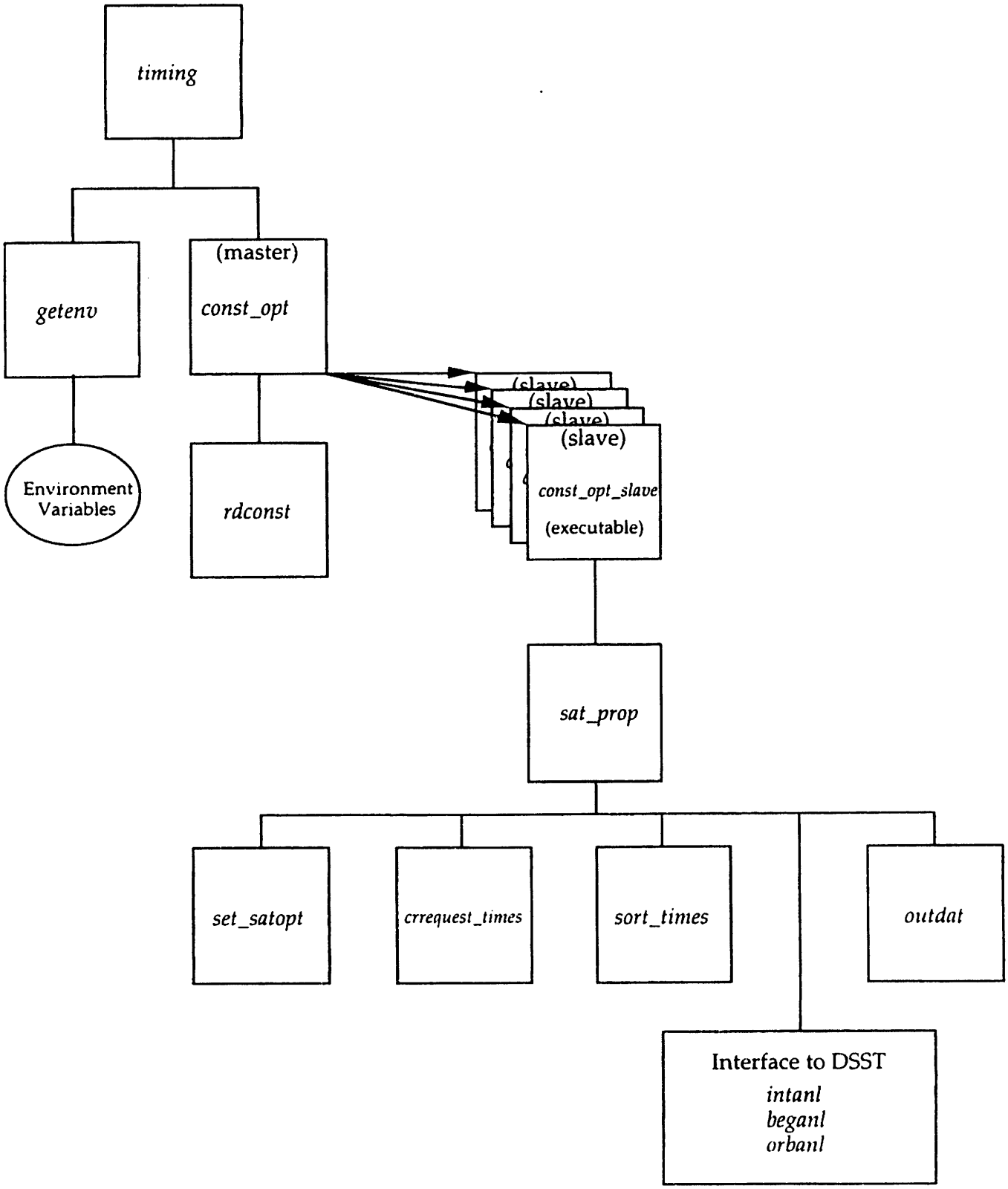


Figure 3-13: Structure of *timing*

For the timing results, the exact satellite was repeatedly propagated. The input file for the test case is shown below, in Figure 3-14.

```
N Satellites:    1    ElType  1

nintervals:      1
Begin interval   1   19950402.0  000000.0
End    interval  1   19960401.0  0.00
Deltat interval  1   86400.0

nburns =         0

-----------------------------------------
Satellite Number:    1  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7073140000000000D+04
                                0.1173029490000000D-02
                                0.9814200000000000D+02
                                0.0000000000000000D+00
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                   0.00000000 Rho One:          0.00000000
S/C Mass:             1.00000000 S/C Area:         0.00000000
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  1
Nmax:       21 Mmax:         0  Izonal:      1  IJ2J2:    2
Nmaxrs:     21 Mmaxrs:       0  Ithird:      3
Ind Drg:     2 Iszak:        2  Ind Sol:     2
-----------------------------------------
```

Figure 3-14: PVM/DSST Input File for Performance Testing

For the serial test case, *time_sat_opt*, the PVM overhead was removed by making calls directly to *sat_opt* for each satellite to be propagated. Figure 3-15 depicts the structure of the program *time_sat_opt*.

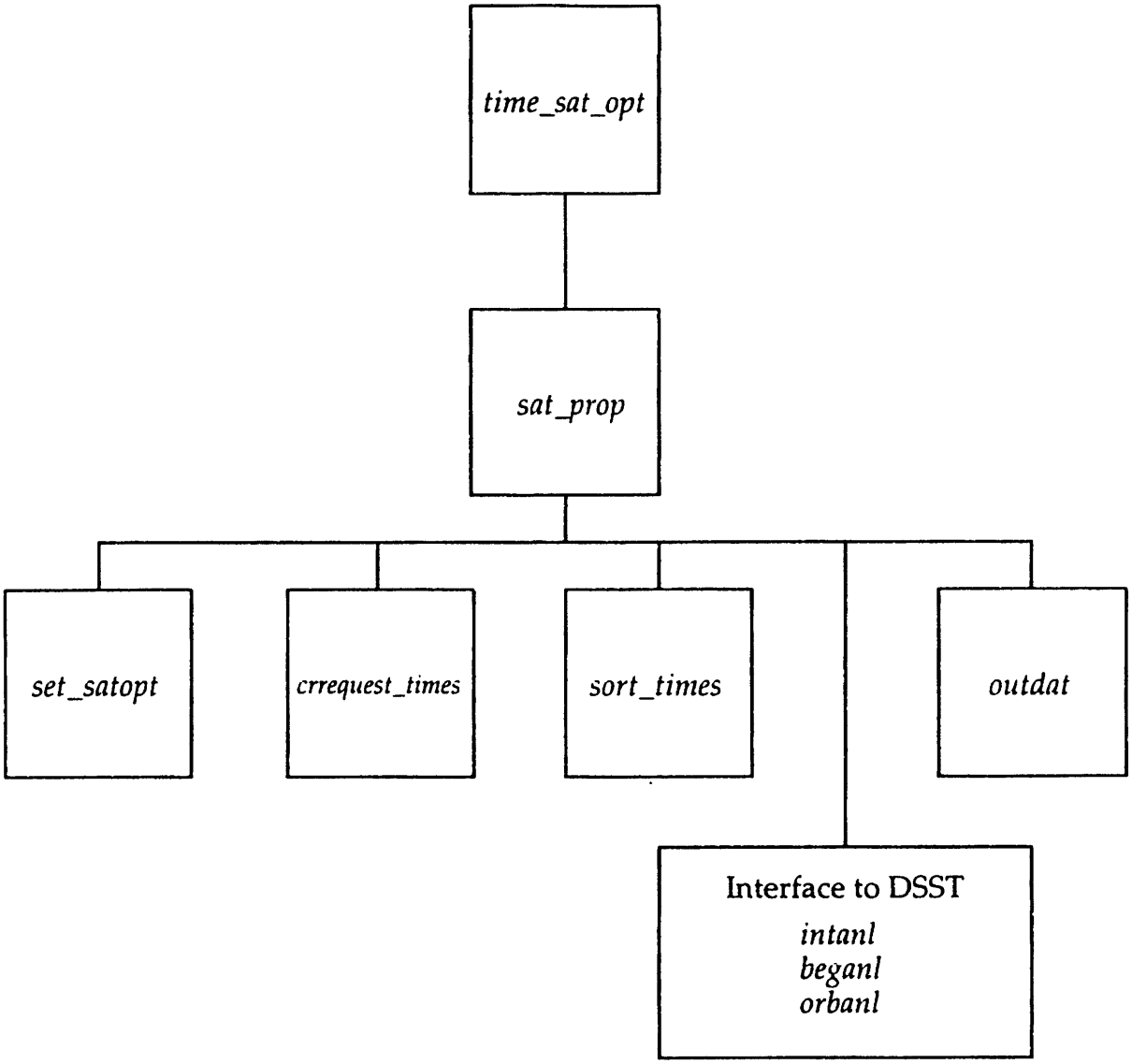


Figure 3-15: Structure of *time_sat_opt*

A script file automated the testing on a variety of configurations. An example script file used to time a constant number of satellites and vary the number of processes is shown in Figure 3-16.

```
#!/usr/local/bin/bash
#This script executes the timing
#test for testing the PVM/DSST
#$1 is the system number
#$2 is the number of slaves

SYS=$1
NSLAVES=$2
export NSLAVES

#Get the envrionment
. ${HOME}/.UserLogin Path
. ${HOME}/.UserLogin Variables

export PVM_ARCH=`/Users/taz/scott/pvm3/lib/pvmgetarch`
export PATH=${PATH}:$PVM_ROOT/bin/${PVM_ARCH}:${PVM_ROOT}/lib

DIR=${HOME}/ccm_satUtil_db/TEMP_OPT.2.0/TEMP_OPT/timing_tests/

#Halt pvm
pvm << EOF
halt
EOF

#Clear tmp of pvm files
rm -f /tmp/pvm*.10995

#Start pvm with appropriate hostfile
pvm ${DIR}/hostfiles/sys${SYS} << EOF
quit
EOF

#Print current virtual machine configuration
echo My system is ${SYS}
pvm << EOF
conf
EOF


#Run the test case
for i in 1 2;
 #Create the filename
  do FILE=${DIR}/perfData/s${SYS}_`date '+%d%H%M%S'`.dat
  for j in 1 2 4 8 16 32 64 128 256;
    do export NSATS=${j};
#      echo 'Running Timing Test With '${NSATS}' satellite(s).';
       ONE=`timing`
      echo ${ONE} |awk '{print  $3}' >> ${FILE}
    done;
done

#Halt pvm
pvm << EOF
halt
EOF

#Clear pvm tmp files
rm -f  tmp pvm*.10995
```

Figure 3-16: Example script to perform timing tests

The UNIX script file was written with the help of Roseman and Beaupre [44,16]. Kernighan is an excellent reference for writing UNIX scripts [70]. The commands for starting and stopping of PVM were necessary for this script to be executed automatically by the UNIX *cron* utility. The utility *cron* executes any UNIX commands in *crontab* files on a regular basis, once per day at 5 PM, for example. The *cron* capability was especially important for timing tests as each test could then be run many times, at the same time every day to ensure consistency. Figure 3-17 lists an example *crontab* file.

```
30 23 * * *
/Users/taz/scott/ccm_satUtil_db/TEMP_OPT,2.0/TEMP_OPT/timing_tests/parallel_time_test 5 1 ;
/Users/taz/scott/ccm_satUtil_db/TEMP_OPT,2.0/TEMP_OPT/timing_tests/serial_time_test 2;
/Users/taz/scott/ccm_satUtil_db/TEMP_OPT,2.0/TEMP_OPT/timing_tests/parallel_time_test 9 2
```

Figure 3-17: Example *crontab* File

Statistics were then compiled on the results, to increase the accuracy of the answers. All the timing tests were performed ten times and the average result was used for performance comparison.

3.5.2 *Serial Test Case*

The same satellite was propagated 1, 2, 4, 8, 16, 32, 64, 128, and 256 times for every test. The serial test case was designed to demonstrate the overhead associated with creating a parallel program. This test case was also used to compare the relative speed of the systems described in Table 3-14 in executing the routine *sat_opt*. No conclusion is intended relative to the performance characteristics of a particular type of machine. The network configuration, the average load, and the setup of each of the computers adds many variables to the execution time. The intent was to compare the computers in their environment so more sense could be made out of the resulting parallel performance tests.

The normalized value of one processor is defined in Equation 3-1.

$$P^{\cdot} = \frac{\text{Execution Time of Fastest Processor}}{\text{Execution Time of Processor of Inter}}$$

(3-1)

Table 3-15 lists the average execution times for the entire serial test case (all 511 satellites), and the normalized value of each processor. Each test case was run ten times. The average execution time and the standard deviation are listed in table 3-15. Because this was a serial test, the normalized value of petunia represents only one of its four-processors.

Table 3-15: Serial Test Case Execution Times and Normalized Processor Values

Machine Name / System	Average Serial Test Case Execution Time (sec)	Standard Deviation (sec)	Normalized Value of One Processor (p*)
porky / 1	594.04	11.10	1.00
petunia / 15	783.95	19.85	0.76
coyote / 2	1078.34	6.82	0.55
wile-e / 3	2578.72	43.34	0.23

When using a heterogeneous network of processors, the sum p^* can be summed to calculate the total number of normalized processors in a virtual machine. This sum will take the place of p in Equation 2-2 used to calculate the efficiency of a parallel execution.

3.5.3 *Overhead*

Overhead is defined for these tests as the amount of work introduced by turning a serial application into a parallel application. To demonstrate the overhead created in developing the PVM/DSST, the serial execution times were compared to the parallel execution times, using only one computer. Systems four through six were compared to systems one through three, respectively. This test could not be successfully performed on petunia, as the computer will automatically distribute the master and slave tasks to different processors. No command existed within PVM to insure that processes were spawned on a particular processor within the multi-processing system.

The overhead is predicted to consist of two portions: a constant value, independent of the number of satellites propagated, and a value that increases linearly with number of satellites. The constant value originates from the time required to create a new process and enroll the processes in PVM. Each satellite directly corresponds to an additional message, so the time required to generate and send each message should appear to increase linearly with the number of satellites propagated.

An additional metric used to demonstrate overhead is the efficiency of the one processor system (Eq. 2-2). The normalized value of each processor is then used to calculate the appropriate value for p. This ratio represents the performance loss in executing the test case on one processor as two separate communicating processes. When the efficiency is one, no overhead is introduced by sending messages. Ratios less than one indicate the time lost in overhead. The communication time for this test is very small, as all the communication will take place on one computer. However, the extra work involved in using PVM to create a new process and send information between processes is demonstrated.

Figures 3-18 through 3-20 visually demonstrate the experimental results of the overhead tests. The first graph in each figure plots execution time vs. number of satellites. Both the serial and parallel execution times are shown. The second plot in each figure graphs the difference between the parallel and serial execution times. The times used are the mean times from the ten separate tests. Note that in all cases, the parallel execution took slightly longer than the serial execution on the same computer.

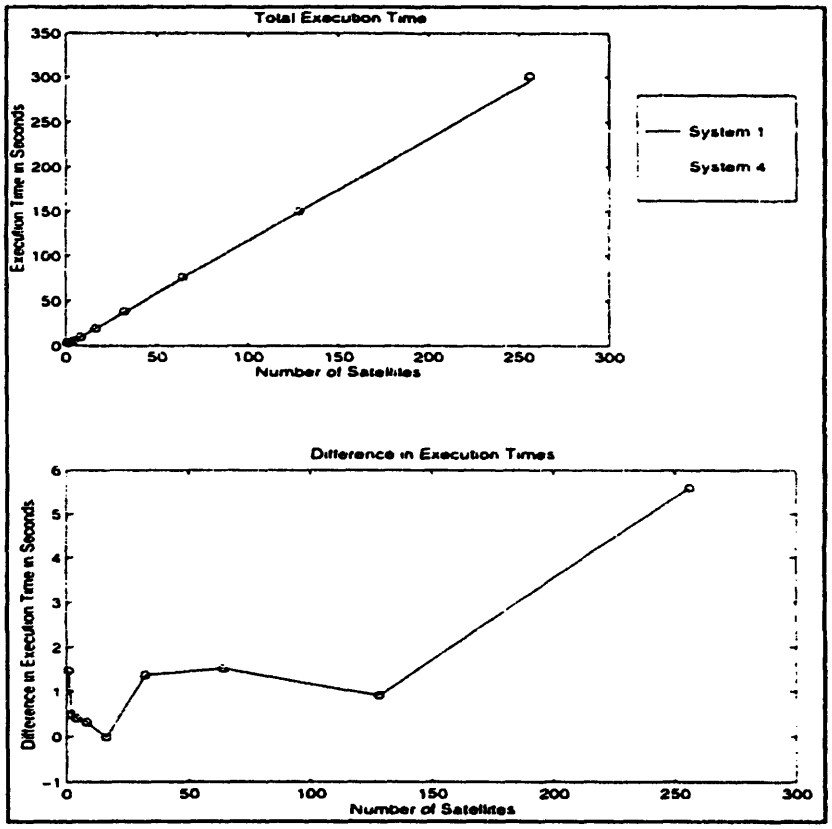


Figure 3-18: Overhead Comparison: System 1 and System 4

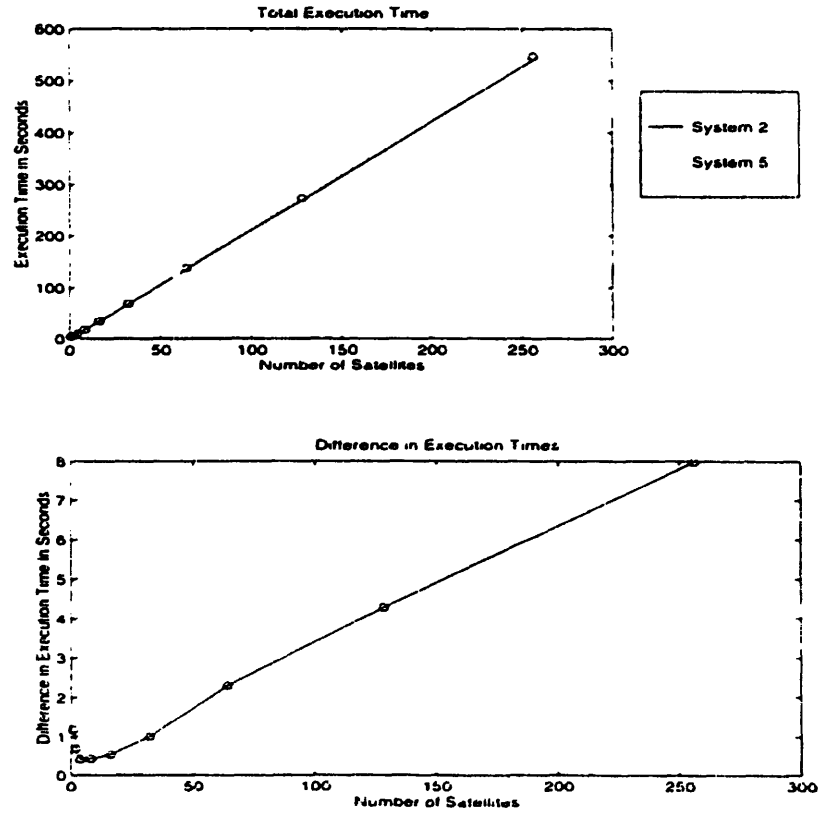


Figure 3-19: Overhead Comparison: System 2 and System 5.

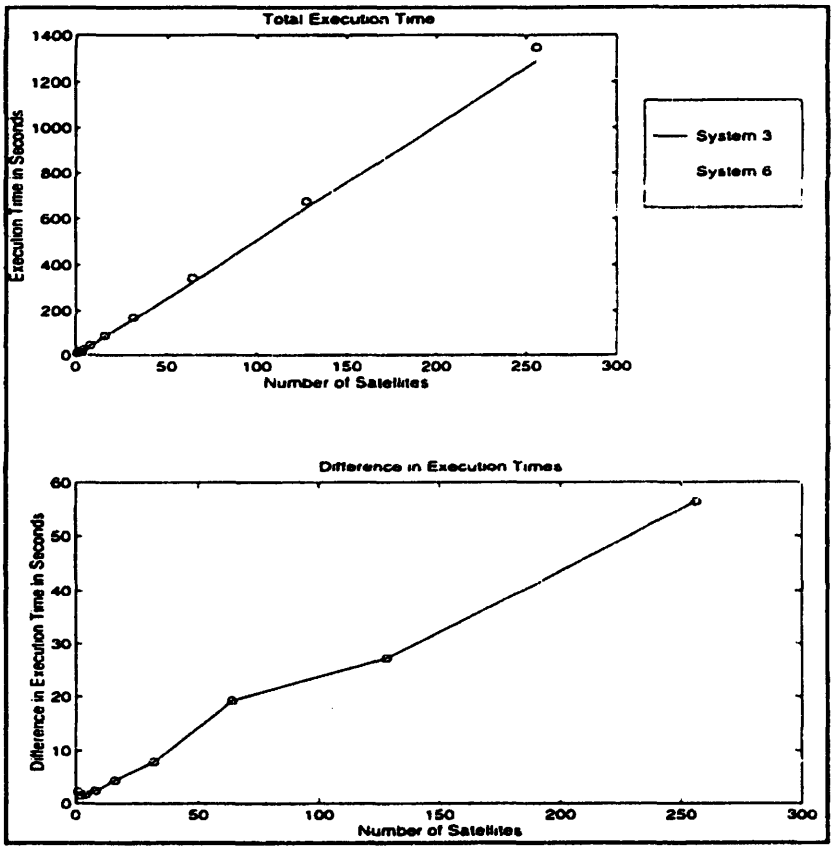


Figure 3-20: Overhead Comparison: System 3 and System 6.

Table 3-16 quantifies the overhead information in the previous three figures by giving the best-fit line to the second graph in each of the figures. These values describe the constant (satellite-independent) and satellite dependent costs associated with the PVM/DSST.

Table 3-16: Overhead Values per Machine

Machine / System	Overhead Constant (sec)	Additional Overhead Cost / Satellite (sec/sat)
Porky/1,4	0.3507	0.0174
Coyote/2,5	0.4142	0.0294
Wile-e/3,6	1.446	0.2144

The efficiency (equation 2-2) on one processor also gives an indication of the overhead. The normalized value of each processor is used for p. The efficiencies of the single machine cases are shown in Table 3-17.

Table 3-17: Efficiencies of the PVM/DSST on One Machine

Machine / System	Efficiency
Porky/1,4	0.980
Coyote/2,5	0.985
Wile-e/3,6	0.956

The efficiencies demonstrate the performance loss in propagating the same number of satellites with the PVM/DSST as compared to using *time_sat_opt*, which executes the DSST without the PVM overhead.

The numbers shown in Table 3-17 only represent communication between processes on the same machine; therefore, messages sent between different machines will have a higher overhead and lower efficiencies. The amount of work done on each satellite, the granularity, will also impact the overhead.

Whether or not overhead has a significant impact on performance depends on the exact application of the PVM/DSST. The results for this test case showed that the PVM/DSST worked very well. Very little setup time is required in comparison to the time required for computation. Because there are many variables affecting performance, it is difficult to generalize these positive results to other applications or other environments. The results do show PVM has the potential to *develop* effective distributed applications.

3.5.4 Speed-Up and Efficiency

Speed-up and efficiency of a parallel algorithm are defined in equations 2-1 and 2-2. Table 3-15 was used to determine the normalized processor value for a network of heterogeneous workstations. This value was used to calculate the efficiency across a network of heterogeneous processors. Efficiencies equal to one demonstrate the best possible performance (indicates that no processing time was lost to communication or other overhead).

As the PVM/DSST on one processor showed overhead (Section 3.5.3), computation across multiple processors will introduce even larger overheads resulting in lower efficiencies. Systems 11 through 14, which only use the

multi-processor platform, will have lower communication times but are also sharing resources, so the efficiencies are difficult to predict.

Figures 3-21 and 3-22 give a qualitative measure of the relative speed-up of the parallel systems. The mean execution time vs. the number of satellites is plotted in both figures.

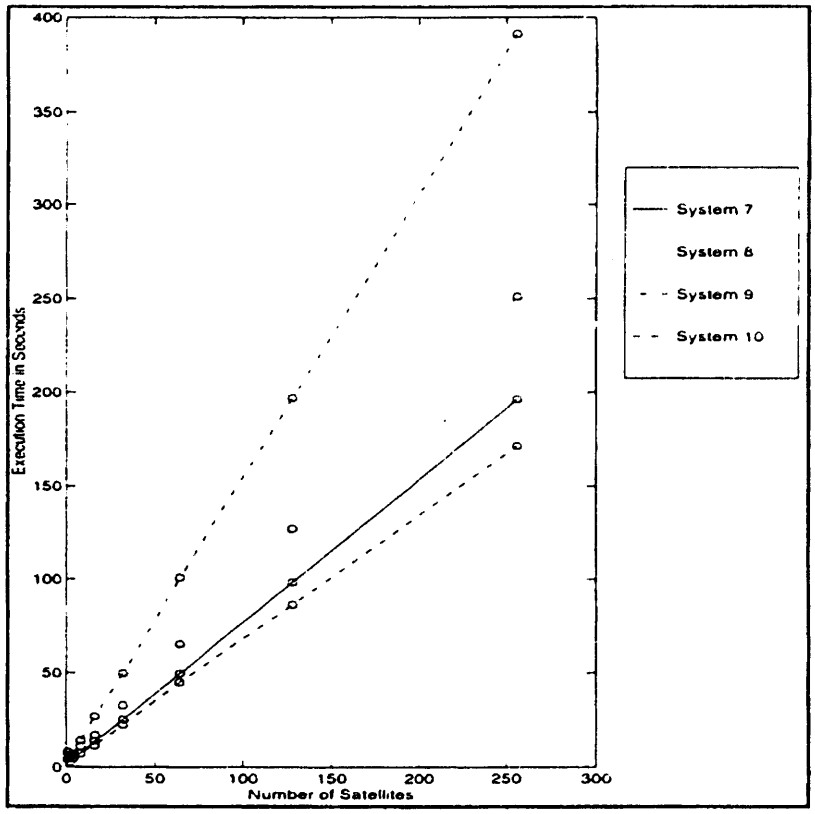


Figure 3-21: Execution times vs. number of satellites for systems 7-10

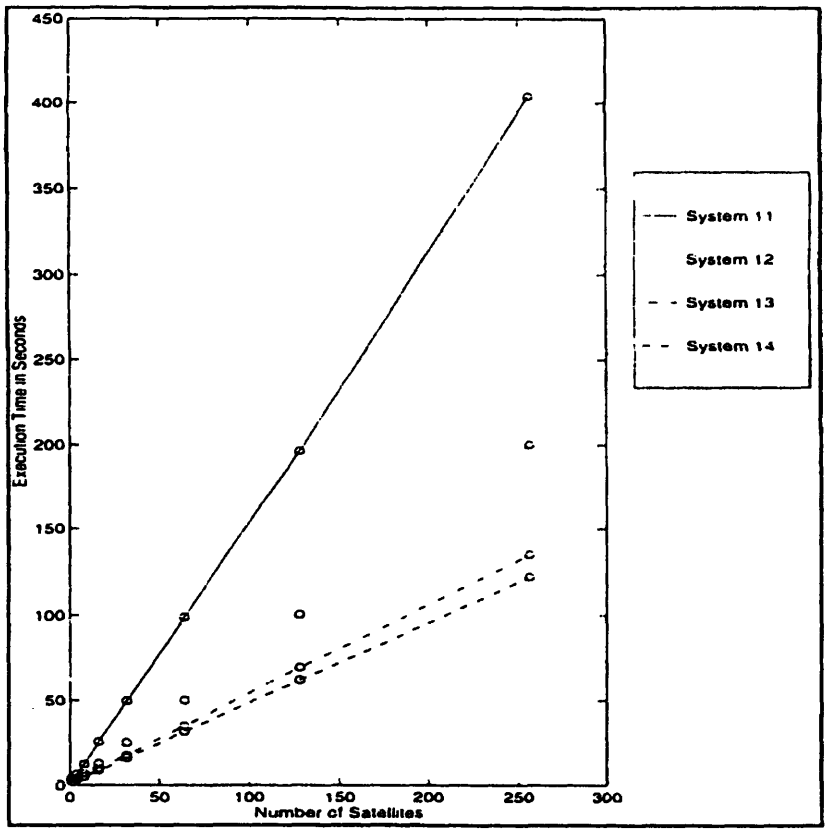


Figure 3-22: Execution times vs. number of satellites for systems 11-14

Figures 3-21 and 3-22 demonstrate that using a network of computers decreases execution time. Multiple processors are effectively used to speed up the execution.

In addition, the small speed increase in going from system 13 to system 14 in figure 3-21 indicates that the multiprocessing platform, petunia, moved the master process to an unloaded CPU. Systems 11 through 13 spawned fewer tasks than the number of processors available. The master task then proceeded to run on an unloaded CPU. System 14 spawned as many slave tasks as there are processors, so the master process shared a CPU with the slave task. Therefore the resulting difference between systems 13 and 14 is not proportional to the difference between 11 and 12.

To quantify the gain achieved, equations 2-1 and 2-2 are applied to the results shown in figures 3-21 and 3-22. Using the normalized value of processors to calculate p (Table 3-15), speed-up and efficiency are calculated. Systems 11 through 13 could not be shown, however, because the correct value of p could

not be calculated. Because petunia automatically spread its work among the available processors, p can only be calculated if the entire machine is used. The speed-up and efficiency results are presented in Table 3-18 and in Figures 3-23 and 3-24.

Table 3-18: Speed-up and efficiency of the PVM/DSST

System Number	Normalized Value of System	Speedup (Compared to System 1)	Efficiency
7	1.55	1.50	0.9678
8	1.23	1.16	0.9432
9	0.78	0.751	0.9606
10	1.78	1.70	0.9564
11	0.76	NA	NA
12	1.52	NA	NA
13	2.28	NA	NA
14	3.04	2.39	0.7874

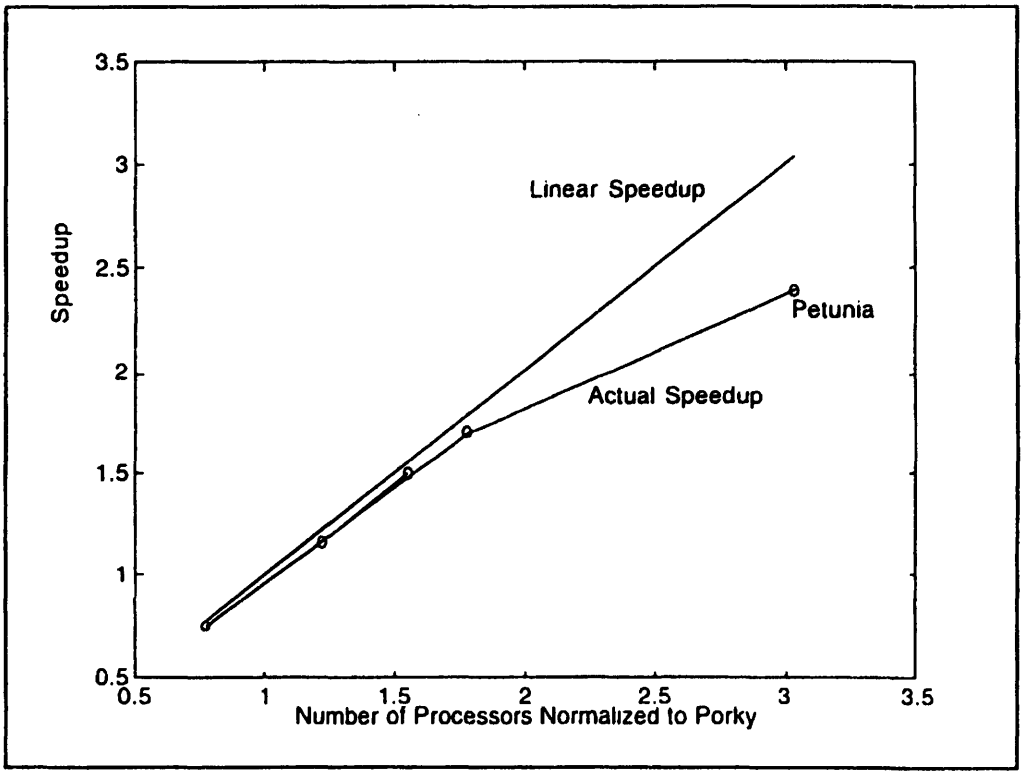


Figure 3-23: Actual Speed-up

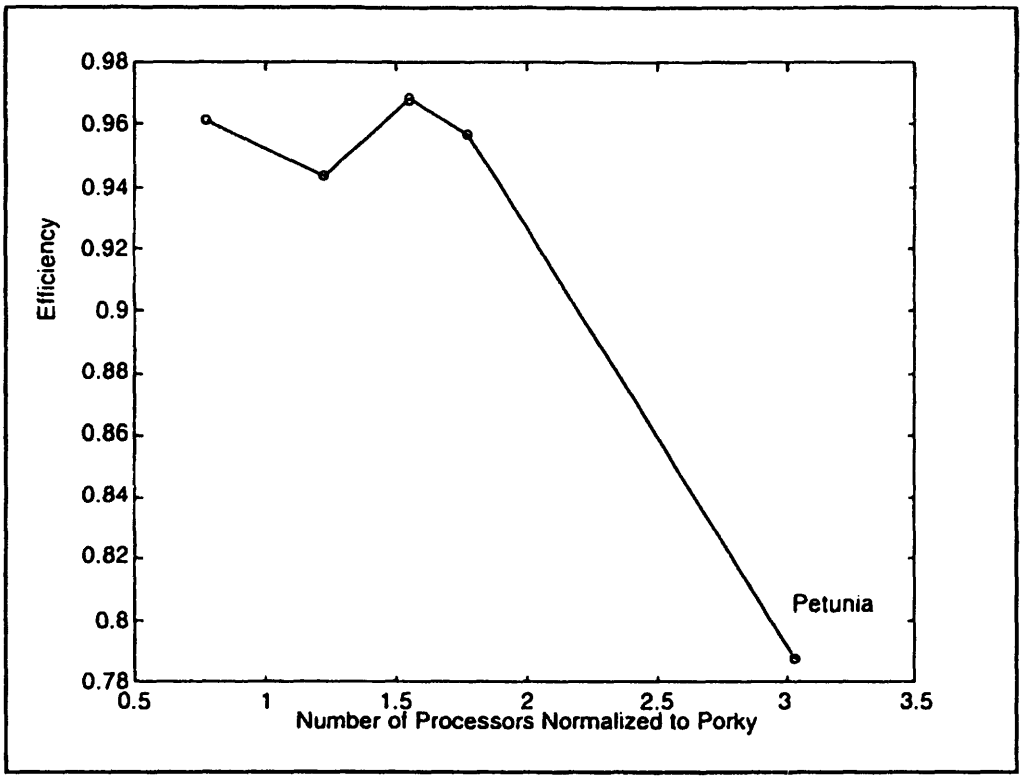


Figure 3-24: Efficiency

3.5.5 *Performance Conclusions*

The PVM/DSST worked well for the test case. Communication overhead affected the performance as predicted. The overall effect of communication was relatively small compared to the potential gain. Speed-up and efficiency showed that the algorithm worked exceptionally well for the distributed network. A comparison between tables 3-18 and 3-17 show a small decrease in efficiency in moving from one machine to a distributed processing environment, as would be expected.

Figures 3-23 and 3-24 show that the multi-processing system did not perform as well as expected, however. The multi-processing platform had lower efficiency than any of the distributed systems, despite the requirement for network communication on a distributed processing system. The most likely reasons for the degraded efficiency on the four-processor machine include:

- The work required to manage the shared resources reduced the effective CPU available.

- The shared resources caused parts of the application to be 'serialized'. Memory writes, for example, are sequentialized because two processors cannot write at the same time [53].

- The system has not been correctly tuned for performance.

- The PVM interface to the native communication system degrades performance.

The efficiency decrease in the four-processor machine was not predicted but is understandable. This machine must perform extra work to manage the four-processors competing for common resources. It is impossible from these series of tests to deduce exactly what caused the reduced efficiency.

Figure 3-23 shows that the speed-up appears to degrade as more processors are added. However, the only point showing significant loss in speed-up is the four-processor machine. If more distributed machines were added to the computing environment, speed-up should not decrease significantly, to a point. Eventually too many machines will saturate the network and overload the *pvmd*'s. At this point, the management and communication requirements of an additional task will require more work than the benefit of adding a new machine. Not enough machines were available to approach this performance limit, however. For a limited number of machines, this algorithm will continue to scale well if the size of the problem is large enough.

4.0 Satellite Constellation Design

The PVM/DSST (Chapter 3) employs a network of computers to make multiple satellite orbit propagation efficient and practical. Combining the PVM/DSST with an optimization algorithm provides a powerful orbit design tool which is easily applied to satellite constellations. This chapter discusses constellation design, the integration of a genetic algorithm (GA) optimization method with the PVM/DSST, and an example application of the optimization tool to the Teledesic satellite constellation.

The constellation design problem has been addressed by many engineers. Walker has presented perhaps the most well-known descriptions of the problem and possible solutions [56]. Other authors have presented studies on designing orbits and constellations [57,58]. These studies have looked at designing constellations to maximize performance characteristics based on the geometry of the initial constellation and the dynamics of orbital motion.

The goal of this study is to refine and automate a portion of the constellation design process so that an initial orbit can be chosen to better meet system requirements in the presence of orbital perturbations. This addition to the design process should help the engineer develop more effective satellite constellations.

Section 4.1 describes the constellation design problem and metrics used to evaluate satellite constellations. Section 4.2 goes on to discuss the orbit optimization tool, which is used in Section 4.3 to automate frozen orbit selection. Finally, Section 4.4 demonstrates the capabilities of the orbit optimization tool in performing an analysis of the Teledesic orbit.

4.1 Design of Homogeneous Satellite Constellations

"The design of a system represents a decision about how resources should be transformed to meet some objectives [54]." Satellite orbits are designed to meet specific requirements. Requirements are balanced to meet mission

objectives. The sun-synchronous orbit described in section 1.2.4 is an example of an orbit designed to meet specific objectives.

Larson and Wertz present a checklist for orbit design, acknowledging that orbit design has no "absolute rules [55]." The checklist is shown in figure 4-1.

1.	Establish orbit types
2.	Establish orbit-related mission requirements
3.	Assess applicability of specialized orbits
4.	Evaluate a single satellite vs. a constellation
5.	Do mission orbit design trades
6.	Assess launch and retrieval or disposal options
7.	Evaluate constellation growth and replenishment
8.	Create ΔV budget
9.	Document orbit parameters, selection criteria, and allowed ranges
10.	Iterate as needed

Figure 4-1: "Checklist " for Orbit / Constellation Design [55]

A satellite constellation is normally used instead of a single satellite when coverage over the Earth is the key criteria in the system design [55]. The term 'coverage' describes how often a satellite system can be accessed from the ground. Because coverage is important to constellations, coverage can be used as a measurement of the performance of a satellite constellation. Therefore, the ability of satellite constellations to provide Earth coverage is an important metric for constellation evaluation.

4.1.1 Satellite Communication Systems

Communication systems use satellite constellations for their ability to provide access to some or all of the entire Earth. The most recent proposals, specifically the systems mentioned in Figure 1-1, plan to use constellations to provide *continuous* and *worldwide* access to communication and data. These systems are of primary interest in this description.

4.1.1.1 Requirements of Communication Satellites

There are two requirements for establishing a communication link between a satellite and a ground user:

- The ground user has a line of sight view to the satellite.
- The communications link between the satellite and the ground has the appropriate signal to noise ratio.

Signal to noise ratios are calculated using a link budget [2]. Both Gordon [2] and Agrawal [3] thoroughly discuss link budgets for communication satellites.

Elevation angles, described in Section 4.1.1.2, determine whether the user has a line-of-sight connection to the satellite. In addition, elevation angles indirectly enter into the link budget calculation.

The recently proposed satellite constellations for mobile communications must maintain a minimum elevation angle above the Earth's surface to ensure users will always have communication access. A minimum elevation angle is required for a particular communication system because:

- Distance from the ground to the satellite increases as elevation angles decrease.
- Obstructions on the horizon prevent a line of sight connection to the satellites.
- Antenna orientation may favor higher elevation angles.
- Atmospheric interference is greater at low elevation angles.

Because the GPCS communication satellites are interested in continuous, worldwide coverage, the minimum elevation angles over the entire Earth for a period of time are of interest. Many metrics can be used to analyze constellations. These metrics include:

Metric	Description
Coverage	Percent of time above the minimum necessary elevation angle for a selection of grid points.
Maximum Coverage Gap [55]	The longest length of time a point on the Earth is below the minimum elevation angle.
Mean Coverage Gap [55]	Average length of time a point on the Earth is below the minimum elevation angle.
Minimum Elevation Angle	The minimum elevation angle at any time for a point on the Earth.

Although all of these metrics are important for constellation design, only the minimum elevation angle metric was used to examine the effects of perturbations on constellations.

Calculation of the elevation angles and the minimum elevation angle constellation design metric is discussed in the next section.

4.1.1.2 Elevation Angles

The elevation angle (E) is measured from the projection of the station-to-spacecraft vector on the local tangent plane to the vector itself. This angle is positive when the spacecraft is above the horizon [49]. Figure 4-2 depicts the geometry of the elevation angle.

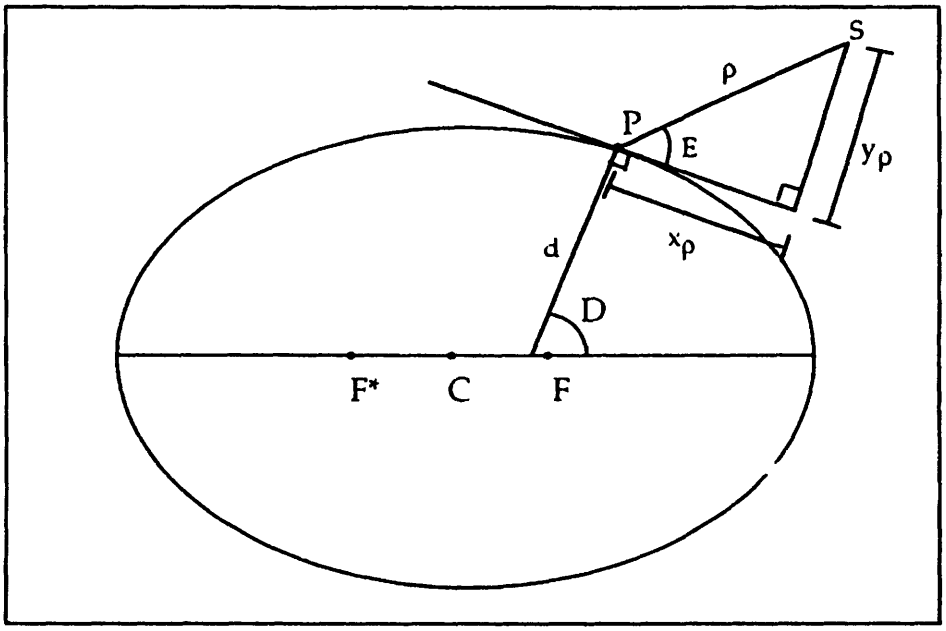


Figure 4-2: Elevation Angle Calculation

where: F and F* are the foci of the reference ellipsoid.

C is the center of the reference ellipsoid (the geocenter).

S is the instantaneous position of the satellite.

P is the location of a point on the Earth's surface.

E is the elevation angle.

d is the vector from the equatorial plane to the normal to the surface of the reference ellipsoid passing through point P.

D is the acute angle between the equatorial plane and the vector d (geodetic latitude).

ρ is the vector from P to S.

x_ρ is the projection of the vector ρ on the local tangent.

y_ρ is the projection of the vector ρ on the unit vector normal to the local tangent.

Practical calculation of the elevation angle uses the spherical Earth assumption. The errors introduced into the elevation angle calculation as a result of this assumption are discussed in Section 4.4.3. The geometry of the elevation angle on a spherical Earth is shown in figure 4-3.

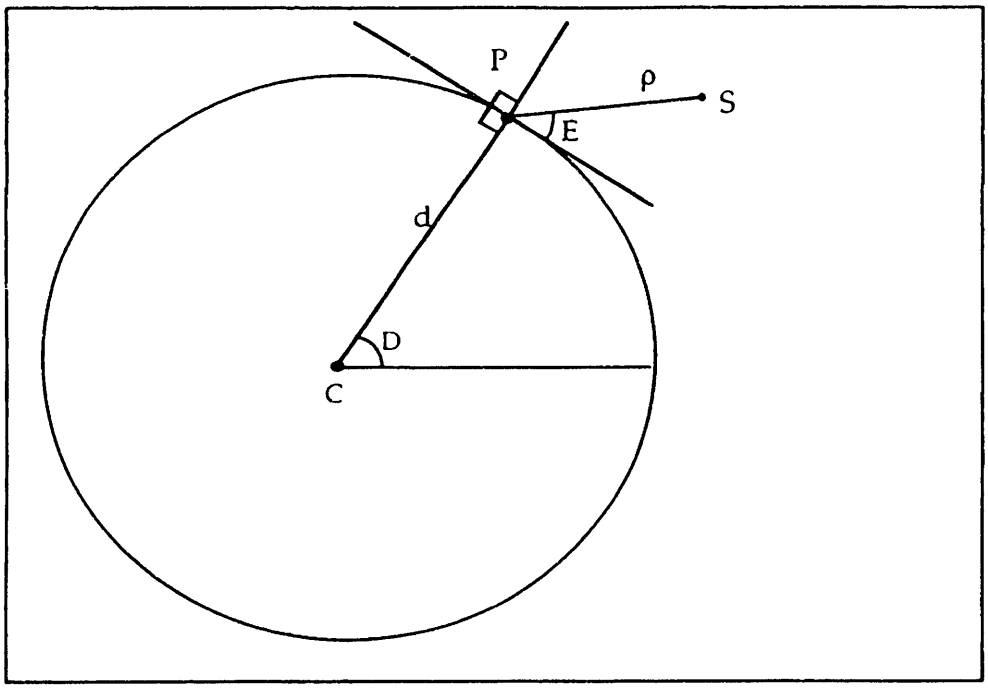


Figure 4-3: Elevation Angle Calculation using the Spherical Earth Assumption

From the above picture, the elevation angle E is 90° minus the angle between d and ρ. Equation 4-1 describes the calculation of the angle E.

$$E = \frac{\pi}{2} - \arccos\left(\frac{\rho \bullet d}{|\rho||d|}\right) \tag{4-1}$$

The elevation angle calculated by equation 4-1 solves for an elevation angle at one time at one point over the Earth. As the satellite constellations are interested in continuous coverage over the entire Earth, the same calculation must be performed for a grid latitude and longitude of points over a period of time. The minimum elevation angle metric is calculated with the following algorithm:

```
For each time DO

    For each grid point DO
        Calculate the elevation angle to each satellite.
        Keep the largest elevation angle
    End DO

    Keep the smallest elevation angle for each grid point and all times
        calculated.

End DO
```

However, due to the satellite and ground station dynamics, a numerical minimum elevation plot can definitively calculate only the upper bound of the minimum elevation angle at each grid point. With a numerical evaluation, the claim can be made that the minimum elevation angle is at least this small. In addition, the values calculated are only valid for each time step and each grid point, not for the time span and the area of the grid points.

To make use of the minimum elevation angle metric for constellation design, the maximum errors must be estimated. Section 4.4.3 quantifies the errors introduced in creating minimum elevation plots.

4.2 Orbit Optimization Design Tool

The minimum elevation metric described in Section 4.1 is one way to measure the effectiveness of a satellite constellation. With a performance metric established, an optimization method can be used to design a constellation that best satisfies the metric. This section describes the development of the orbit optimization tool, which couples the PVM/DSST with a genetic algorithm (GA) optimization method, designed and implemented by Schott [64].

4.2.1 *Genetic Algorithm Optimization Method*

Two definitions are necessary before continuing in this section:

cost function The function to be minimized.

parameters Variables that the GA modifies to find the minimal cost function. There must be some relationship between the parameters and the cost function, but the relationship may not be analytically defined.

A genetic algorithm optimization method was chosen for this optimization problem because:

- GA's only require parameter ranges and a cost function. No derivative information is necessary.
- GA's provide a good global answer to the optimization problem. Global is defined as the parameter space.
- GA's can make use of parallel cost function evaluations.
- Ongoing work at Draper by Schott [64] and Schor [65] provided an excellent source of expertise in the use of GA's.

A well known reference on the GA optimization method is Goldberg [63]. Forrest [68] presents a brief overview of GA's :

"The basic idea of a genetic algorithm is very simple. First, a population of individuals is created in a computer (typically stored as binary strings in the computers memory), and then the population is evolved with use of the principles of variation, selection, and inheritance."

For the GA used in the orbit optimization tool, each member of the population represents a different combination of initial orbital elements. Each member is used to evaluate the cost functions, which are found by

propagating the orbit forward in time. The members are then modified using genetic operators, so the orbit which minimizes the cost function is chosen.

4.2.2 *Software Description*

The GA used was designed for a Master's thesis by Schott [64]. It was developed within the Design Optimizer / Markov Evaluator software, written at Draper Laboratory [65]. All the software is written in FORTRAN 77.

4.2.2.1 Interface to Genetic Algorithm Software

The interface between the GA software requires that the cost function evaluations be performed by a subroutine call. This subroutine was a modified version of *const_prop*, known as *const_opt*. A combination of the GA software and *const_opt* became the master process. For every series of cost function evaluations required, a call to *const_opt* was made. The subroutine *const_opt* enrolled itself as a PVM task, spawned slave processes across the virtual machine, and sent a member of the population to each slave process where the cost function evaluation was calculated in parallel. The interface between the PVM/DSST and the GA is shown in figure 4-4.

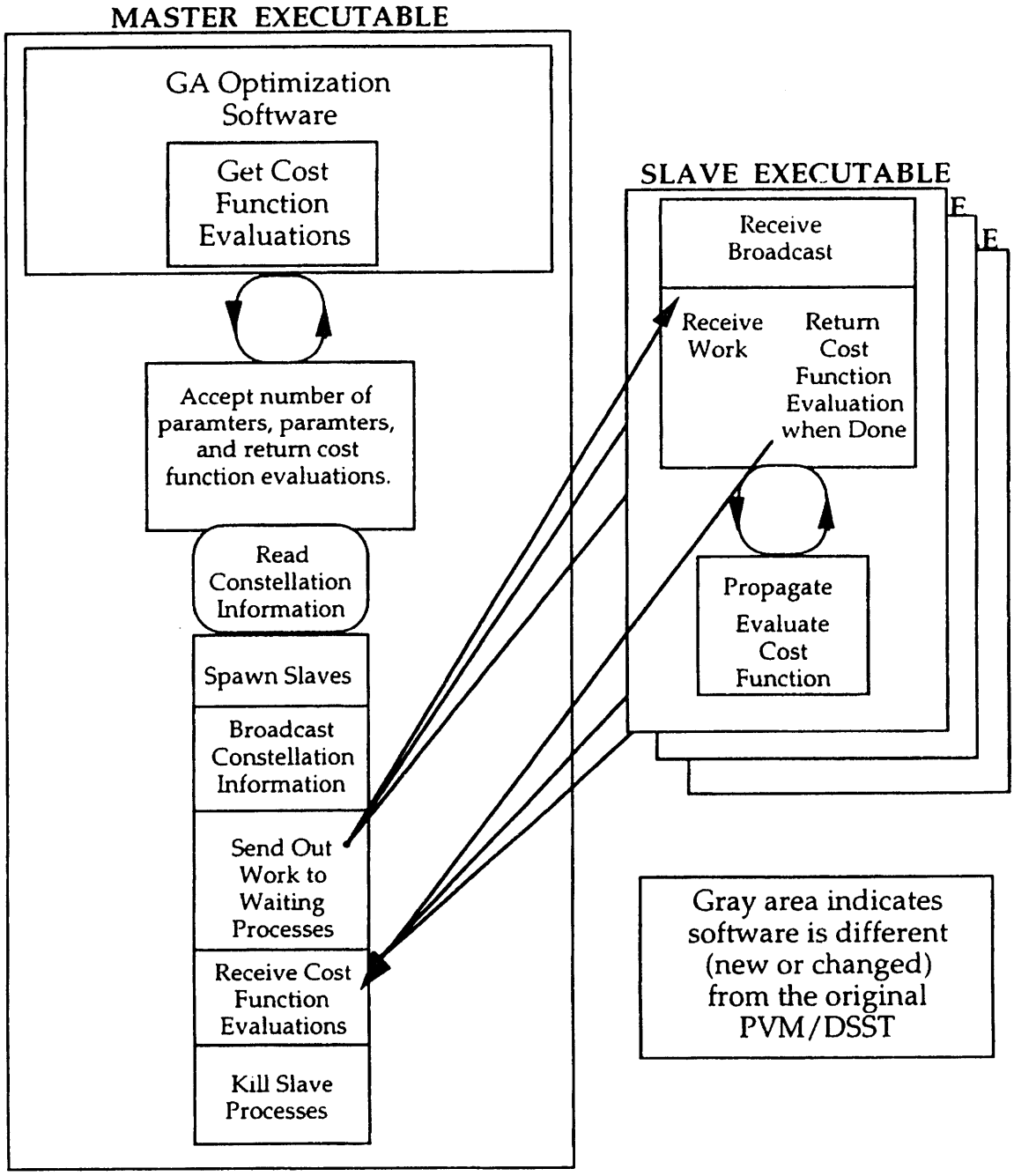


Figure 4-4 : Interface Between GA and PVM/DSST

The slave executable is detailed in Figure 4-5.

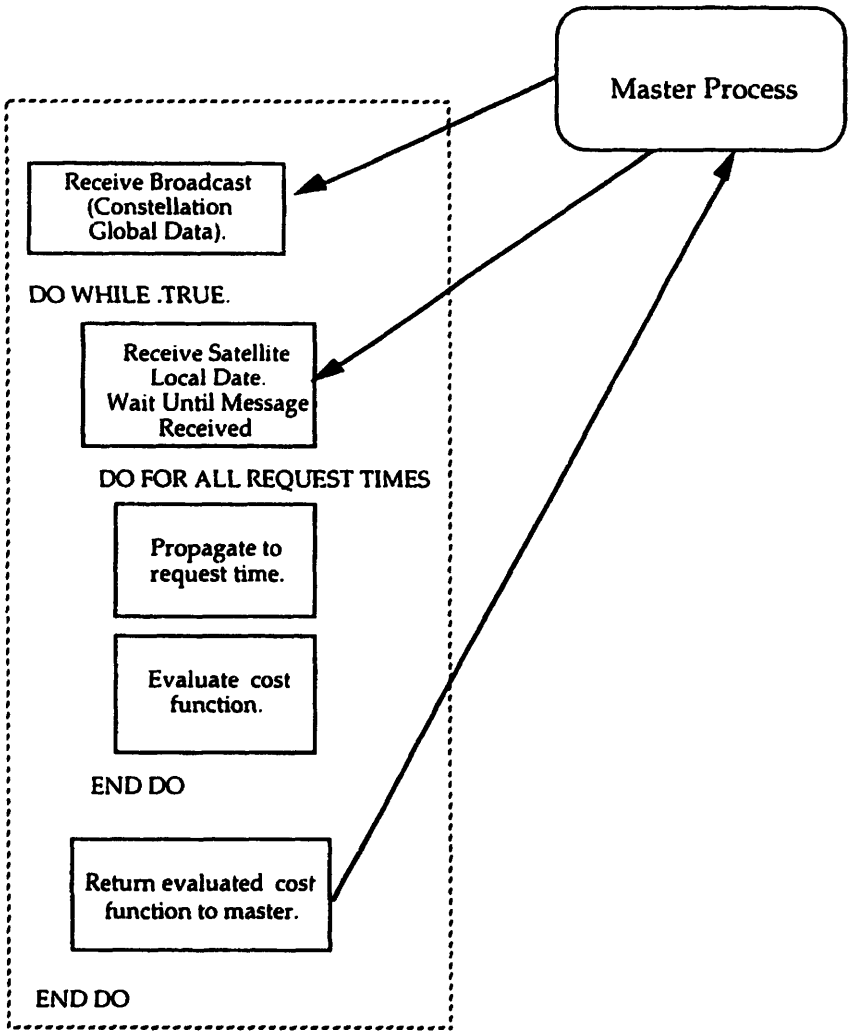


Figure 4-5: Slave Executable

One of the main advantages in using the GA for the optimization technique is its capability to make use of parallel cost function evaluations. Other optimization techniques only operate on one set of parameters; after every cost function evaluation, new parameter values are chosen. There is no concept of a population requiring multiple cost function evaluations at the same time.

The majority of the computation required for an optimization algorithm is in the cost function evaluation. The ability to perform this step in parallel results in a significant performance improvement.

4.2.2.2 Modification of Propagator

The following changes had to be made to the PVM/DSST (Chapter 3) so it would work with the GA software:

- The slave program, *sat_opt*, had to send the cost function evaluation to the master process.

- The master program, *const_prop*, had be written as a subroutine. The argument list passed the parameters from the GA, the number of parameters to evaluate, and returned the cost function evaluations.

- Because the cost functions were evaluated in parallel, the order in which the cost functions were evaluated did not necessarily match the order of the parameters. An extra value had to be sent between the master and the slave. This number identified the slave process to the master so the correct parameters could be matched with their respective cost function evaluations. (Ref. Section 2.3.4)

4.3 Frozen Orbit Design

This section describes an example use of the orbit optimization tool. The example applies the orbit optimization tool to the frozen orbit design problem. Use of the optimization tool is described in detail in Appendix D.3.4.2.

4.3.1 Use of the Optimization Tool

Two steps are required before using the optimization tool. The two steps are:

- Develop a cost function that the optimization tool will minimize. The cost function must include all factors going into the orbit design as the tool will neglect any concerns that do not appear in the cost function.

- Determine which parameters to vary and the range of each of the parameters.

4.3.2 The Frozen Orbit

The goal of a frozen orbit is to maintain a constant argument of perigee and eccentricity [62]. Many satellites require frozen orbits. Earth observation satellites need to be at the same altitude over the same place on the Earth to obtain several pictures for comparison over time [27]. Frozen orbits also reduce fuel consumption in station keeping [27]. In addition, both Ellipso and Teledesic GPCS are using frozen orbits [66, 50].

The central body non-sphericity causes the largest changes in the argument of perigee and eccentricity. The changes in argument of perigee and eccentricity due to the J_2 and J_3 zonal harmonics are shown in equations 4-2 [62]:

$$\begin{aligned}
\frac{de}{dt} &= -\frac{3nR_e J_3 \sin i}{2a^3(1-e^2)^2}\left(1-\frac{5}{4}\sin^2 i\right)\cos\omega \\
\frac{d\omega}{dt} &= \frac{3nJ_2 R_e^{\,2}}{a^2(1-e^2)}\left(1-\frac{5}{4}\sin^2 i\right)\Theta \\
\Theta &= 1+\frac{J_3 R_e}{2J_2 a(1-e^2)}\left(\frac{\sin^2 i - e\cos^2 i}{\sin i}\right)\frac{\sin\omega}{e}
\end{aligned} \tag{4-2}$$

Because J2 is the dominant zonal perturbation (Table 1-3), equations 4-2 will provide a good estimate of a frozen orbit. Further refinement must be accomplished in the presence of a full zonal field.

Analyzing equations 4-2 reveals the methods to achieve a frozen orbit. There are three methods to null the eccentricity rate:

- Place the orbit in the critical inclination [$\left(1-\frac{5}{4}\sin^2 i\right)=0$]
- Place the satellite in an equatorial orbit. [$i=0^\circ$]
- Set the argument of perigee to 90° or 270°.

To remove the argument of perigee variation, the satellite must be in a critical inclination orbit or Θ must be set to zero [62].

4.3.3 Frozen Orbit Design using the Orbit Optimization Tool

A nominal satellite is given with near frozen starting conditions. This orbit is taken from the Teledesic constellation (Section 4.4) [66]. The satellite orbital elements are shown in Table 4-1.

Table 4-1: Satellite Keplerian Elements used for Frozen Orbit Determination [66]

Element	Value
Semimajor Axis (km)	7073.14
Eccentricity	0.00118
Inclination (deg)	98.142
Longitude of Ascending Node (deg)	0.0
Argument of Perigee (deg)	90

This orbit achieves its frozen state by using a argument of perigee equal to 90° and choosing the appropriate value for eccentricity where Θ is zero. Due to other constraints the semimajor axis and inclination are fixed, therefore the critical inclination cannot be used to achieve the frozen orbit. To numerically depict the frozen orbit, the PVM/DSST can be used to generate element histories over time. The input file used to generate the element histories is shown in figure 4-6.

```
N Satellites:    1   ElType  1

nintervals:      1
Begin interval   1  19950401.0  000000.0
End   interval   1  19951001.0  1.00
Deltat interval  1  86400.0

nburns =         0

-----------------------------------------
Satellite Number:    1  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7073140000000000D+04
                                0.1180000000000000D-02
                                0.9814200000000000D+02
                                0.0000000000000000D+00
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                   2.20000000 Rho One:          0.00000000
S/C Mass:           800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:         0  Izonal:      1  IJ2J2:    2
Nmaxrs:     21 Mmaxrs:       0  Ithird:      3
Ind Drg:     2 Iszak:        2  Ind Sol:     2
-----------------------------------------
```

Figure 4-6 Input file for Generating Element Histories from the Nominal Satellite State

The zonal harmonics through degree 21 were the only perturbation used in this analysis. However, developing the frozen orbit in the presence of other perturbations only requires modification of the satellite input file. The PVM/DSST propagated the satellite six months, outputting mean elements once per day. The resulting argument of perigee and eccentricity element histories are shown in figure 4-7.

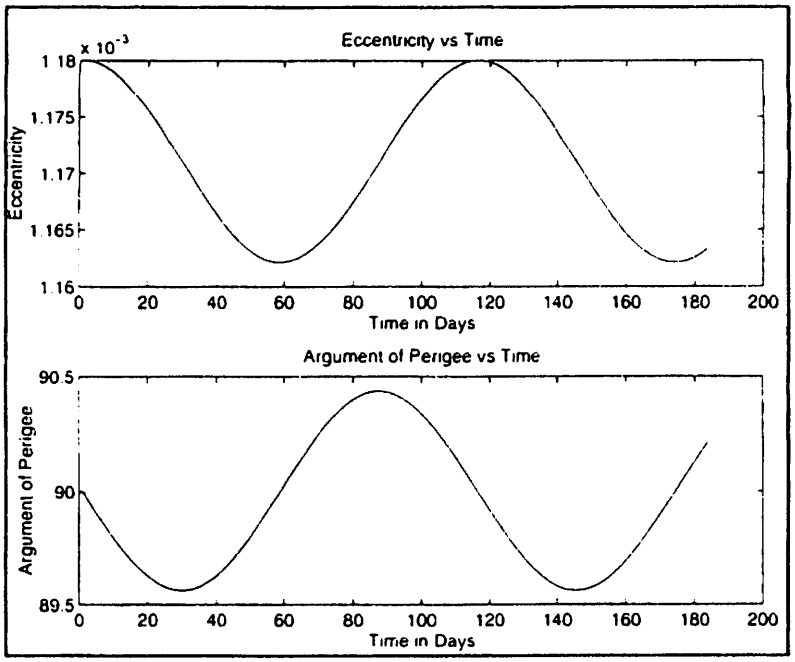


Figure 4-7: Element Histories of Nominal Satellite

It is also useful to plot a phase plane, the eccentricity versus the argument of perigee.

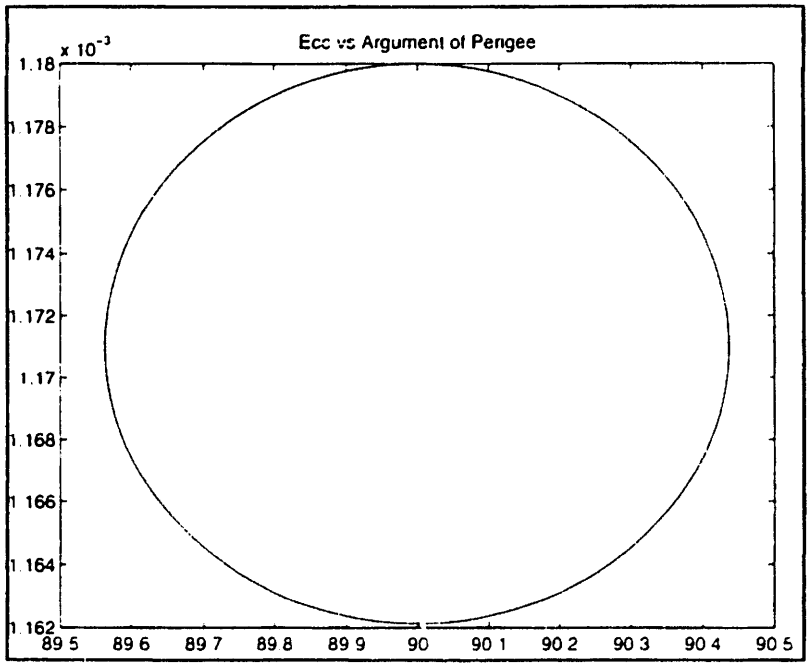


Figure 4-8: Argument of Perigee Vs eccentricity

A simple cost function was then developed to reduce the variation in argument of perigee and eccentricity. The cost function calculated the total

variation from the initial eccentricity. This was accomplished by adding the absolute value of the eccentricity deviation from the initial value at every output time. The output time step (86400.0 seconds) is shown in the input file, figure 4-6. Because deviations in the eccentricity and argument of perigee are directly related, there was no need to add the argument of perigee variations to the cost function. Although this problem has a very simple cost function, more complex problems will require more complex cost functions. Multiple parameters in the cost function require normalization and weighting of each parameter, for example.

The final step was to choose which parameters to vary. For this problem the choice was very simple. Only the eccentricity could be varied to achieve the frozen orbit. All other parameters were fixed by other constraints or the equations in 4-2.

The optimization software used the following input file (titled *dome.in*) to find the best frozen orbit.

```
Choose the most frozen eccentricity
1                              itest
250 0.07,                      iopt,maxitr,epsiln
11985 50,1,                    kseed,mpopsize,ncomp
1 1 1 0,0,2,0,                 Opts:  constr,clones,Popt,Ropt,Topt,ishr
1                              fixed parameters
1                              continuous parameters
1 1 1                          it chooses initial conditions
1 .001000,                     min of continuous
1 .001200,                     max of continuous
1                              discrete parameters (3 failure rates)
4                              number of bins for each discrete parameter
                               initial discrete (ga: param# ie. #1)
 1 ..e4 .001171,.001173,.001175,
```

Figure 4-9: Example GA input file

The orbit and perturbation options in figure 4-6 were used to describe the nominal conditions. The eccentricity values are generated by the genetic algorithm within the range 0.0010 and 0.0012, as specified in the GA input file, figure 4-9. The GA generates 255 discrete values from the one 'continuous' parameter. For this problem, it is trivial to evaluate all the 255 possible combinations of values. With just two parameters, the number of combinations would rise to 255^2 or 65,025 function evaluations. The real

power of this optimization method is in finding the region of the optimal multi-criteria answer without evaluating all possible functions. The GA will not continue to narrow its focus or 'zero in' on the best value beyond the initial discretization of the problem. However, the chosen values will be in the area of the best solution. In order to come up with the best value the parameter range must be narrowed manually. This process will be illustrated in this example.

The output of the GA is given in two files, *DO* and *Dz*. Figure 4-10 and table 4-2 show the output generated using the input files shown in figures 4-6 and 4-9.

```
****  DOME BEGAN ON 27-Apr-95 AT 23:04:35    ****
Run ID: Choose most frozen eccentricity

* Optimization method:    9 *
Optimization search stopping criterion:          7.0000E-02
Maximum number of optimization iterations:              250
Genetic Algorithm:
population size:          50  random number seed:        20985
crossover:              0.80  per bit mutation:         0.0040
markov model states:       1  fixed parameters:              0
continuous:                1  discrete parameters:           0

continuous     initial        lower         upper
variable       value         bound         bound
     0.0000E+00    1.0000E-03    1.2000E-03
cfe#     139         ** stop due to population convergence **
Parameters reverted to original:         0
Total cost function evaluations:       139
Evaluation of minimum value:            50
Algorithm elapsed time:           101.4633

Function value                              Parameter values
                      2             3             4             5
1.64641633E-05   1.17098039E-03
****  DOME TERMINATED ON 27-Apr-95 AT 23:06:17    ****
```

Figure 4-10: DO Output Report.

Table 4-2: Dz Output File[1].

Number of Function Evaluations Performed	Population Size	Number of identical members of the population	Cost Function Evaluation	'Best' Parameter Value	Convergence Factor
50	50	0	1.64641633E-05	1.17098039E-03	1.53222466E-02
95	50	11	1.64641633E-05	1.17098039E-03	4.34377119E-02
139	50	14	1.64641633E-05	1.17098039E-03	8.91743973E-02

The resulting eccentricity is close to the value given in the initial design. However, small changes in the initial parameters have a dramatic effect on the element histories. The first value chosen by the GA was used to narrow the eccentricity range so that the optimization calculation could be repeated. Three more refinements were made. Table 4-3 lists the ranges used, the 'best' eccentricity found, and the value of the cost function evaluation for each successive iteration.

Table 4-3: Optimization Results for Iterations 2,3 and 4.

Iteration	Range	Best Eccentricity	Cost
2	0.001170-0.001175	0.00117105874	1.3255e-06
3	0.0011710-0.0011711	0.00117106584	4.6564e-08
4	0.00117106-0.00117107	0.00117106561	1.5760e-09

The effect of the eccentricity chosen by the fourth iteration is shown in figures 4-11A and 4-11B.

[1]The first row of text has been added to this file for explanation

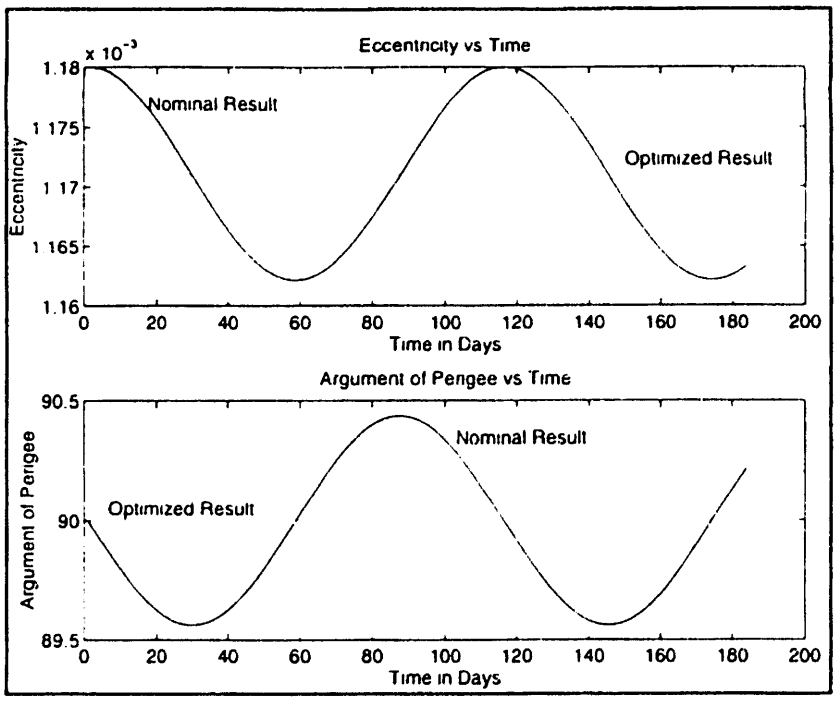


Figure 4-11A: Nominal and Optimized Element Histories

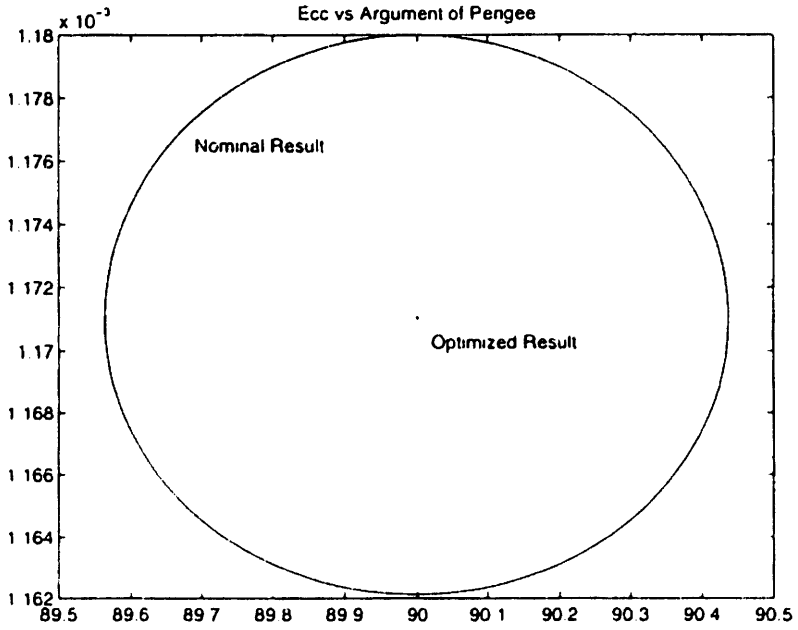


Figure 4-11B: Nominal and Optimized Argument of Perigee Vs Eccentricity

Figures 4-11 depict the improvement in reducing argument of perigee and eccentricity variations. These results are seen more clearly in figure 4-12. This figure shows the difference between the maximum and minimum values of eccentricity and argument of perigee, plotted on a $\log_{10}$ scale.

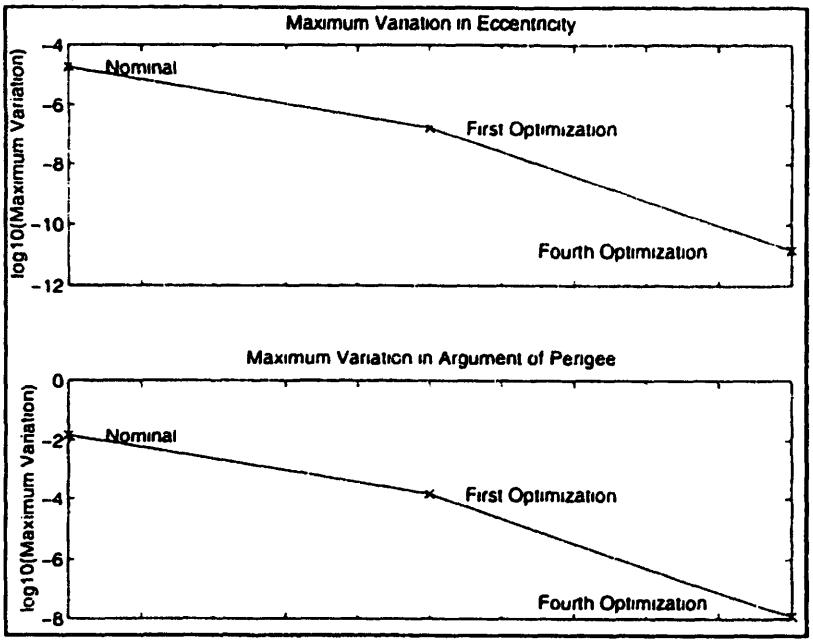


Figure 4-12: Maximum Variations

The element histories of the optimized orbit demonstrate the effectiveness of the optimization tool applied to this problem.

An advantage of using the orbit optimization tool for frozen orbit determination is its ability to include arbitrary perturbations. Propagating the 'optimized' orbit described in figure 4-6 and table 4-3 in the presence of tesseral harmonics, $(J_2)^2$, third body, and solar radiation pressure generates figure 4-13. A year interval, as opposed to the six month interval shown previously, was used to generate figure 4-13.

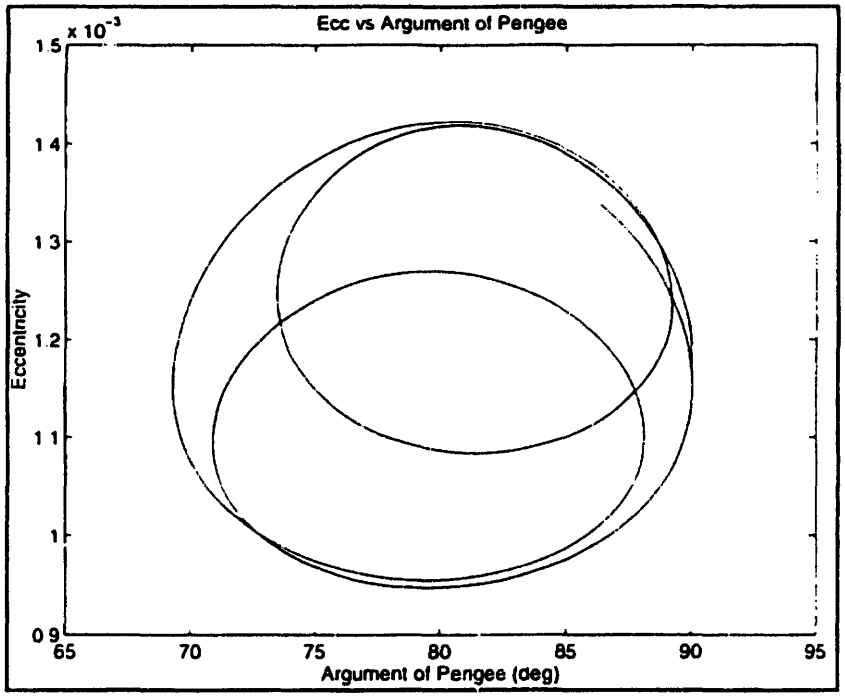


Figure 4-13: Argument of Perigee Vs Eccentricity with Perturbations

Further attempts to achieve a more frozen orbit by adjusting eccentricity showed negligible improvement in both the eccentricity and argument of perigee histories. Any future attempts to achieve a more frozen orbit will require modification of additional orbital elements.

An improvement to the optimization tool would use the GA to find the region of the best values and use other optimization methods to refine the solution.

4.4 Application of the PVM/DSST and the Optimization Tool: The Teledesic System

The Teledesic Corporation has proposed the construction of a communication satellite constellation to "provide interactive broadband information services to people in rural and remote parts of the United States and the World [66]." Teledesic plans to offer fixed satellite services. The Teledesic target market is remote and rural areas of the world, where access to broadband information services do not already exist. Unique to Teledesic is the size of the constellation proposed. As shown in figure 1-1, Teledesic is

planning to operate 840 satellites. As stated in Chapter 1, this constellation alone is proposing to operate more satellites than are currently *operating* in space.

4.4.1 *Overview of Satellite System Design*

The Teledesic satellite is depicted in figure 4-14.

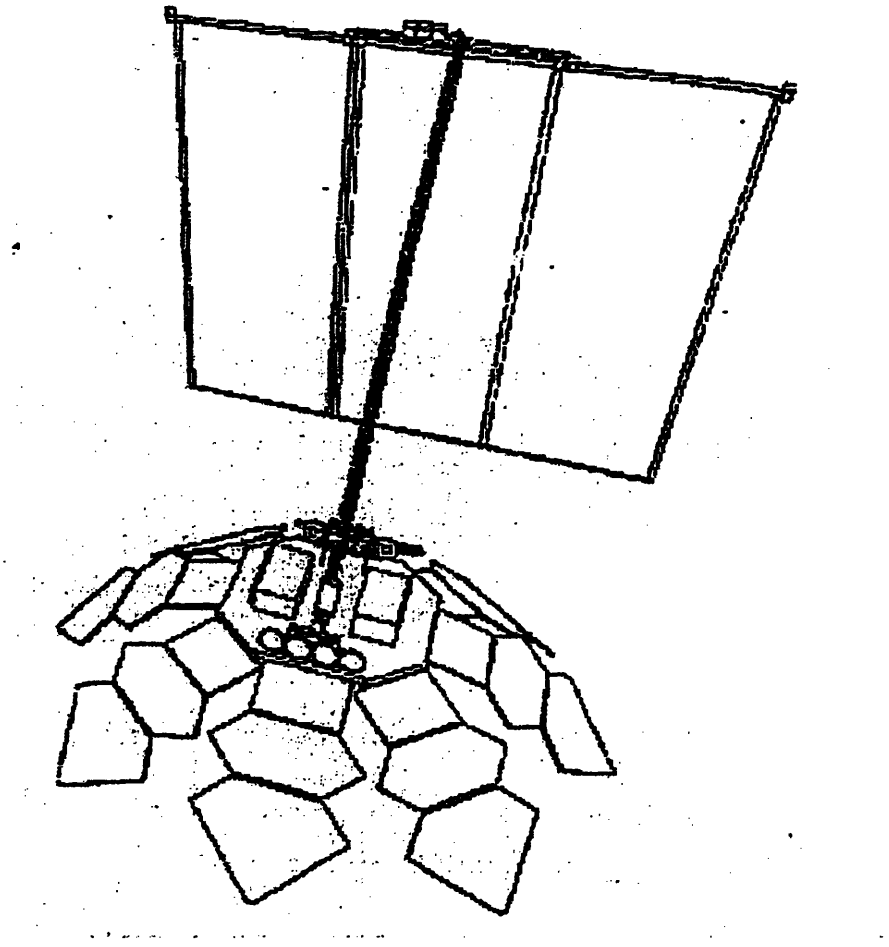


Figure 4-14: The Teledesic Satellite

The constellation consists of twenty-one evenly spaced planes, each plane separated by 9.5°. Each plane will contain forty-four near circular satellite orbits. Forty of the satellites will be operational and four will be used as on-orbit spares [66]. The satellites are in a sun synchronous, frozen orbit. The constellation plans to provide a minimum elevation angle of 40° between ±72° latitude. The constellation Keplerian elements are listed in table 4-4.

Table 4-4: Teledesic Orbital Parameters [66]

Plane No.	Number of Satellites	Altitude (km)	Angle of Perigee	Arc (deg)	Right Ascension of Ascending Node (deg)	Eccentricity	Inclination (deg)
1	40 to 44	695.0	90	360	0.0	0.00118	98.142
2	40 to 44	695.5	90	360	9.5	0.00118	98.144
3	40 to 44	696.0	90	360	19.0	0.00118	98.146
4	40 to 44	696.5	90	360	28.5	0.00118	98.148
5	40 to44	697.0	90	360	38.0	0.00118	98.150
6	40 to 44	697.5	90	360	47.5	0.00118	98.152
7	40 to 44	698.0	90	360	57.0	0.00118	98.154
8	40 to 44	698.5	90	360	66.5	0.00118	98.156
9	40 to 44	699.0	90	360	76.0	0.00118	98.158
10	40 to 44	699.5	90	360	85.5	0.00118	98.160
11	40 to 44	700.0	90	360	95.0	0.00118	98.162
12	40 to 44	700.5	90	360	101.5	0.00118	98.164
13	40 to 44	701.0	90	360	114.0	0.00118	98.166
14	40 to 44	701.5	90	360	123.5	0.00118	98.168
15	40 to44	702.0	90	360	133.0	0.00118	98.170
16	40 to 44	702.5	90	360	142.5	0.00118	98.172
17	40 to 44	703.0	90	360	152.0	0.00118	98.174
18	40 to 44	703.5	90	360	161.5	0.00118	98.176
19	40 to 44	704.0	90	360	171.0	0.00118	98.178
20	40 to 44	704.5	90	360	180.5	0.00118	98.180
21	40 to 44	705.0	90	360	190.0	0.00118	98.182

4.4.2 Assumptions

Several assumptions were made in analysis of the Teledesic satellite constellation. Teledesic has staggered the orbital altitudes to prevent collision between satellites [66]. To simplify the refinement of the constellation, this requirement was removed from the design constraints.

Secondly, because long time spans (5 years) were used in analyzing the constellation, the effects of drag were not studied. The satellite has a higher than average area/mass ratio (0.18 m^2/kg), so drag will have a significant impact on the satellite [67]. Note that this area/mass ratio is a worst case for this satellite. Drag studies will require modeling the effective area of the satellite. However, neglecting drag is a valid assumption if drag make-up

maneuvers maintain the nominal semimajor axis of the orbit. The no-drag assumption leads to additional assumptions in the analysis of the orbit.

The frozen orbit constraint requires the eccentricity and argument of perigee to remain constant. However, the Keplerian VOP equations demonstrate that drag make-up maneuvers can also be used to control the variations in the argument of perigee and eccentricity [5]. Therefore, changes in the initial constellation were only constrained to maintain the original amount of variation in argument of perigee and eccentricity. Although obtaining the minimum variation in argument of perigee and eccentricity was desirable, it was not accomplished in this project.

Element histories are presented per plane, with the implied assumption that the perturbative effects are the same for every satellite in the plane. This assumption is not valid for the tesseral harmonics, as these perturbations are dependent on the ground track of the satellite. The minimum elevation angle plots, however, do not use this assumption as all 840 satellites are propagated individually. Because the satellites have a 100 minute period, the in-plane differences in third body and solar radiation pressure perturbations are negligible.

Finally, the DSST was assumed to accurately predict the future state of the satellites.

4.4.3 Error Sources in Elevation Angles

In order to use minimum elevation angles as a constellation design metric, the maximum errors in the evaluation process of these angles must be determined. If the error is not determined, the minimum elevation angles for different constellations cannot be compared. The error could be larger than the differences between the metrics, making a comparison meaningless.

Due to the process error, the calculated minimum elevation angle will have different upper and lower bounds. The upper and lower bounds are described in equation 4-3.

$$E - \varepsilon_{lb} < E < E + \varepsilon_{ub} \tag{4-3}$$

where E is the elevation angle.

ε_{lb} is the maximum error below the numerically calculated minimum elevation angle.

ε_{ub} is the maximum error above the numerically calculated minimum elevation angle.

$E - \varepsilon_{lb}$ is the lower bound for the minimum elevation.

$E + \varepsilon_{ub}$ is the upper bound for the minimum elevation.

There are four sources of error in the minimum elevation angle calculation:

1] Spherical Earth assumption.

2] Error in satellite position.

3] Length of time between each angle evaluation.

4] Grid spacing.

Because finding the *minimum* elevation angle is of interest, the upper bound is easily calculated. The upper bound is found by correcting the calculated elevation angle for the error in numerical evaluation (errors 1 and 2). Error introduced due to the time or position of evaluation (errors 3 and 4) will not factor into determination of the upper bound.

Calculation of the lower bound requires subtracting all four error sources from the calculated minimum elevation angle. The third error source is necessary for the minimum elevation angles to be generalized over the duration of the time interval. If this error source is ignored, the minimum elevation angles are only valid for the exact time of calculation. Calculation of the fourth error source allows the elevation angles to be generalized for the area between the grid points. Ignoring this error makes the minimum elevation angles valid only for the exact locations calculated.

4.4.3.1 Spherical Earth Assumption

The spherical Earth assumption adds error to the calculated minimum elevation angle. The worst case situation is used to calculate the maximum error introduced into the minimum elevation angle evaluation. For this error analysis, an ellipsoidal Earth model that varies with latitude will be used as truth. No longitude dependent errors enter into the calculation.

The error due to a spherical earth assumption is important only in finding the 'true' minimum elevation angle. When using the minimum elevation angle to compare constellations, this error can be neglected as it is the same for each angle evaluation.

The spherical Earth error can be broken into two parts. The two parts are:

- The local topocentric coordinate system (LTCS) has an incorrect orientation.
- The (LTCS) has an incorrect origin.

The first error is depicted in figure 4-15.

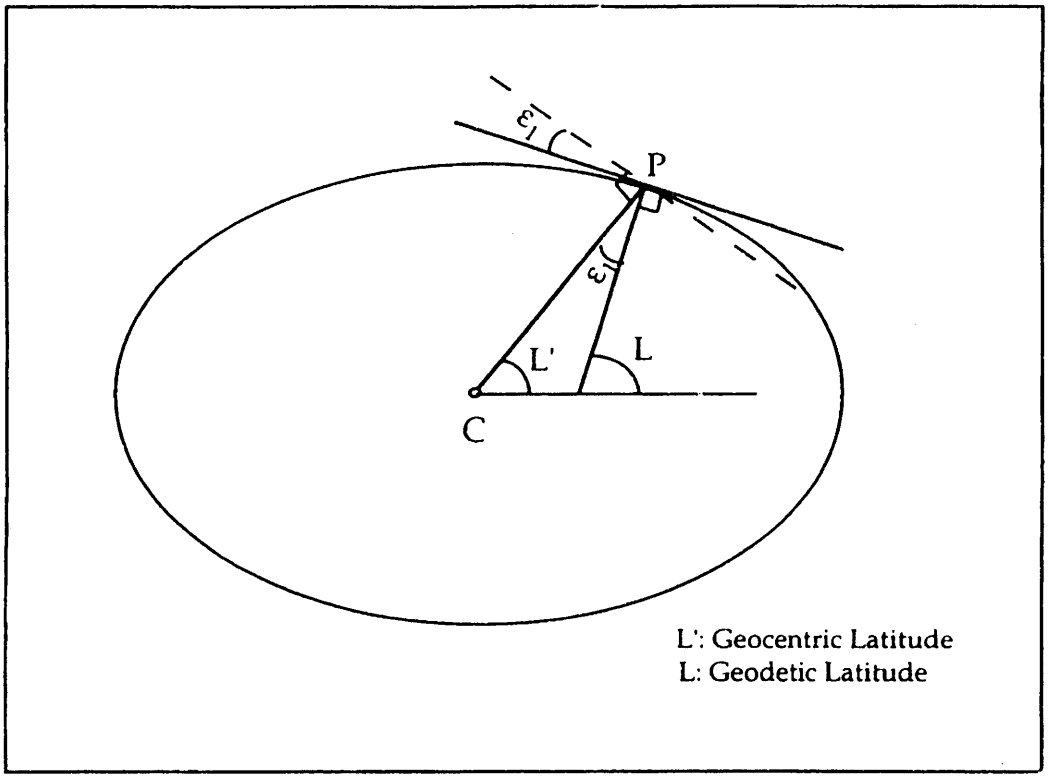


Figure 4-15: Error Generated by Ignoring the Difference Between Geodetic and Geocentric Latitude .

Geocentric and geodetic latitude are depicted in figure 4-15 [38]. In calculating the elevation angle, the vector from the center of the Earth (*C*) to the grid point (*P*) is assumed to be perpendicular to the local horizon. Because the geodetic latitude describes the angle perpendicular to the local horizon, an error of magnitude ε is introduced into the elevation angle evaluation. The quantity ε_1 is simply the difference between the geocentric and geodetic latitudes. The maximum ε_1 can be found using equation 4-4 [49]

$$L' = \arcsin\left[\frac{R_e(1-e^2)\sin L\sqrt{1-e^2\cos^2 L}}{R_p\sqrt{1-e^2\sin^2 L}}\right] \tag{4-4}$$

where: e is the eccentricity of the Earth

R_e is the equatorial radius

R_p is the polar radius

The maximum difference occurs at L=45°. Using :

$R_e = 6378.137$ km $R_p = 6356.753$ km $e = 0.08182$

gives a maximum error of ε_1=0.1917°.

The change in elevation angle due to the error in the origin of the LTCS is created by assuming a spherical Earth of radius R_e. This difference is depicted in figure 4-16.

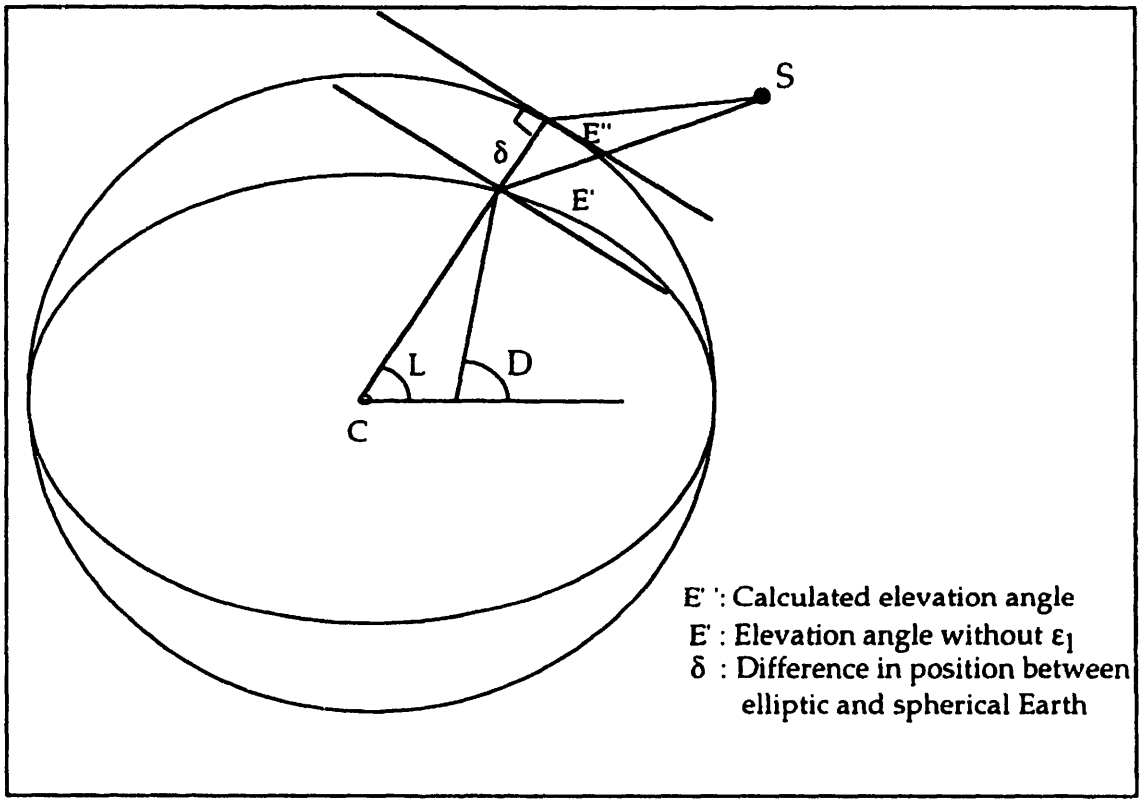


Figure 4-16: Difference in Elevation Angle Due to Site Position Difference.

The maximum difference in the elevation angle calculation will occur at the North and South poles. At the poles the difference in position creates a maximum difference in elevation angle shown in equation 4-5.

$$\varepsilon_2 = abs(E - E') = \arcsin\left(\frac{\delta}{\rho}\right) \tag{4-5}$$

where: ρ is the minimum distance from the ground to the satellite.

The quantity ρ is evaluated at the minimum ρ as the error reaches a maximum at this point.

The maximum δ is 21.384 km, when the values for R_e and R_p shown above are used. The quantity ρ depends on the satellite orbit.

The error ε_2 will only affect the upper bound of the minimum elevation angle. Using the equatorial radius for the spherical Earth radius will cause the assumed LTCS origin to be father from the center of the Earth than the actual origin (equal at the equator). Therefore, this error source will cause the calculated elevation angles to be less than or equal to the actual elevation angles.

The upper and lower bounds due to a spherical Earth assumption are given in equation 4-6.

$$\varepsilon_{lb} = \varepsilon_1 + E \quad \varepsilon_{ub} = \varepsilon_1 + \varepsilon_2 + E \tag{4-6}$$

where: E is the calculated elevation angle.

4.4.3.2 Error in satellite position

The worst case difference in elevation angle caused by an error in the satellite position is described by equation 4-7.

$$\varepsilon_{\nu p} = \arcsin\left(\frac{\Delta}{\rho}\right) \tag{4-7}$$

where: $\varepsilon_{\nu p}$ is the maximum error in the minimum elevation angle calculation due to error in satellite position.

Δ is the maximum difference between the actual and calculated satellite position.

ρ is the minimum distance from the ground to the satellite.

This value depends on the maximum error for a particular orbit and the propagation technique used.

Using the DSST (Chapter 1) without the contribution of the short periodic functions results in a maximum position error of 10 km for a low Earth, near circular satellite [27]. With a ρ of 690.3 km using the mean elements only gives an $\varepsilon_{\nu p}$ of ±0.83°.

4.4.3.3 Length of time between each angle evaluation

The maximum change in elevation angle between each time step must be calculated to generalize the minimum elevation angle calculation over the time interval from the first to the last evaluation. The minimum elevation angle to one satellite changes monotonically over a time step, unless the satellite passes through its maximum value in between the time steps. Assuming the elevation angle is at the predicted constellation minimum at time t_1 and monotonically decreases with a constant rate until time t_2, the maximum deviation from a calculated elevation angle will occur halfway between two time steps.

As the elevation angle rate depends on the elevation angle, the constant rate assumption is not accurate. However, the absolute value of the elevation angle rate decreases with the elevation angle, so the rate at time t_1 is larger than time t_2. Therefore, this assumption is conservative in generating the maximum deviation in elevation angle.

The maximum error is then found by integrating the maximum elevation angle rate over half a time step.

The elevation angle rate is calculated from the worst case geometry. The worst case assumes the following:

- The satellite is moving directly away from the point of interest (P) on the Earth. For the satellite to be moving directly away from the point of interest on the sphere, the orbital plane must intersect P.
- For the development of the elevation angle rate equations, the spherical Earth and the circular orbit assumptions are made.
- For eccentric orbits, the elevation angle rate is larger near perigee. If elevation angles of a highly eccentric orbit is of interest, the satellite velocity at perigee can be used for the worst case central angle rate, $\frac{d\phi}{dt}$ (see figure 4-17).
- All coordinates are in ECEF, so the quantity $\frac{d\phi}{dt}$ must reflect the maximum difference between the satellite velocity and the rotation rate of the Earth.

The geometry of the worst case is shown in figure 4-17.

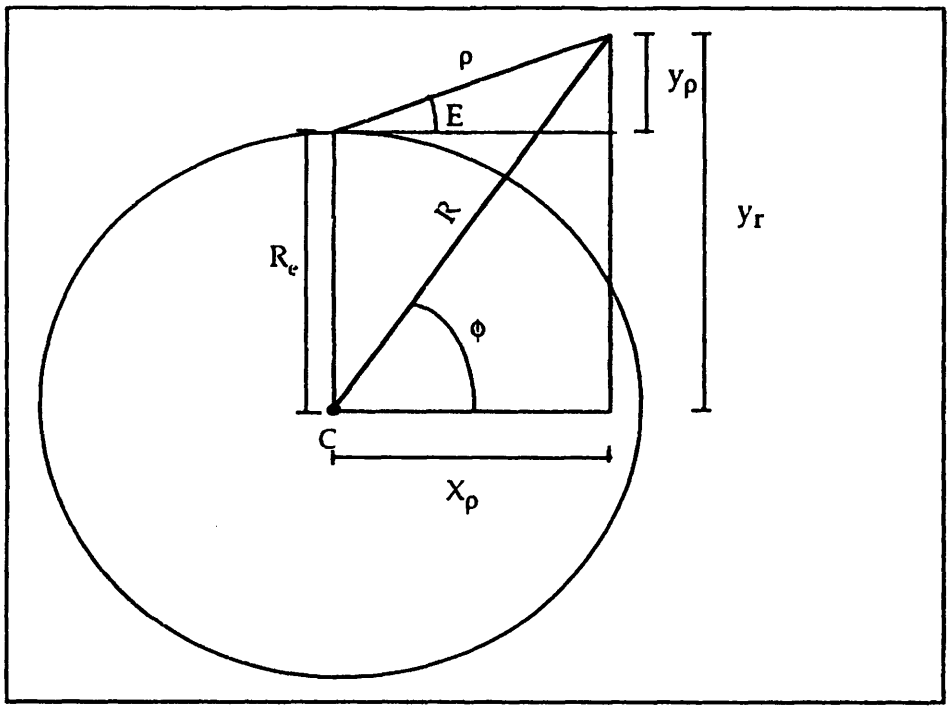


Figure 4-17: Geometry of Elevation Angle Rate Calculation

where: R is the vector from the center of the Earth to the satellite.

ϕ is the angle between the projection of ρ on the local tangent plane and the vector R.

θ is the angle between the vector x_ρ and ρ.

y_ρ is the projection of the radius vector of the satellite on the normal to the local tangent.

R_e is the spherical Earth radius.

Equation 4-8 calculates the elevation angle from figure 4-17.

$$\tan E = \left(\frac{y_r - R_e}{x_\rho} \right) = \tan \phi - \frac{R_e}{a} \sec \phi \qquad (4\text{-}8)$$

where: a is the semi-major axis of the satellite

Taking the derivative of both sides and simplifying results in equation 4-9 for the elevation angle rate.

$$\frac{dE}{dt} = \frac{\frac{d\phi}{dt}(a^2 - y_\rho * R_c)}{\rho^2} \qquad x_p \neq 0 \tag{4-9}$$

where: $\frac{d\phi}{dt}$ is a constant for the circular satellite.

As ϕ, y_ρ, and ρ are functions of E, equation 4-9 demonstrates that the rate of change of the elevation angle is a function of the elevation angle. When solving for the maximum change in elevation between time steps, the constellation predicted minimum elevation for the constellation is used as the initial condition. This is again the worst case. Assuming that the satellite is at the desired minimum elevation angle at the evaluation time and the elevation will continue to decrease until the next evaluation will result in the error in elevation angle. As described above, integrating equation 4-9 for a half a time step with the initial elevation angle equal to the constellation minimum elevation angle results in the maximum change in the minimum elevation angle.

This error only appears in the change in the lower bound of the minimum elevation angle. The error is shown in equation 4-10.

$$\varepsilon_{ts} = \int_0^{\frac{t}{2}} \left(\frac{dE}{dt}\right) dt \tag{4-10}$$

where: ε_{ts} is the error due to the time step size.

t is the time step size.

4.4.3.4 Grid spacing

In order to calculate the maximum change in the minimum elevation angle between grid points, figure 4-17 and equation 4-8 are used. The maximum difference in minimum elevation angle due to grid spacing will occur between grid points. As with the error introduced from generalizing over

time, the constellation minimum elevation angle is used to determine the worst case impact. This will determine the maximum change between grid points below the minimum elevation angle. Starting with the constellation minimum elevation angle, equation 4-8 is used to calculate the angle ϕ. The angle ϕ is then increased until E changes by the maximum error desired. The change in ϕ is used to describe the necessary spacing between grid points. With this error source calculated, the minimum elevation angles for a grid of points over the Earth can be generalized to include the area between the grid points. The error due to grid spacing will be referred to as ε_{gs}

4.4.4 Impact of Perturbations on Nominal System

The minimum elevation angle metric was used to evaluate the impact of perturbations on the nominal constellation. As described in Section 4.4.3 , to make use of the minimum elevation angle metric, the maximum error in numerically calculating the angles must be determined.

4.4.4.1 Error in Minimum Elevation Angle Metric

The elliptical Earth model was used to calculate the ECEF position vectors on the Earth. However, the vector from the center of the Earth was assumed to be perpendicular to the local horizon at the surface of the Earth. From equation 4-4, this assumption introduced a maximum error of ±0.19°.

Mean elements were used to generate the satellite positions. The maximum difference between mean and osculating positions for the Teledesic orbit is 10 km [27]. From equation 4-5, the worst case error in satellite position creates a maximum error in the minimum elevation angle of ±0.83°.

To keep the error bound within -2.0° a grid spacing of 0.4° was necessary. Because the metric calculated the *minimum* elevation angles, the grid spacing and time step errors can only increase the lower error bound. The satellite position and spherical Earth assumptions, however, affect the upper and lower bounds.

To place an evenly spaced grid over the entire Earth would require approximately 257,600 grid points[2]. These points are evenly spaced on the surface of the Earth, every 0.4° in central angle. The grid spacing requirement generated too many points to numerically evaluate, so only one longitude was evaluated. Calculating the upper and lower bounds of the minimum elevation angle for one longitude was still effective to demonstrate the effect of perturbations on the constellation.

The final error source involves calculating the maximum possible elevation rate. Figure 4-18 shows the elevation rate versus elevation angle calculated using equation 4-8. The worst case (lowest altitude) Teledesic satellite was used and a circular orbit assumption was made.

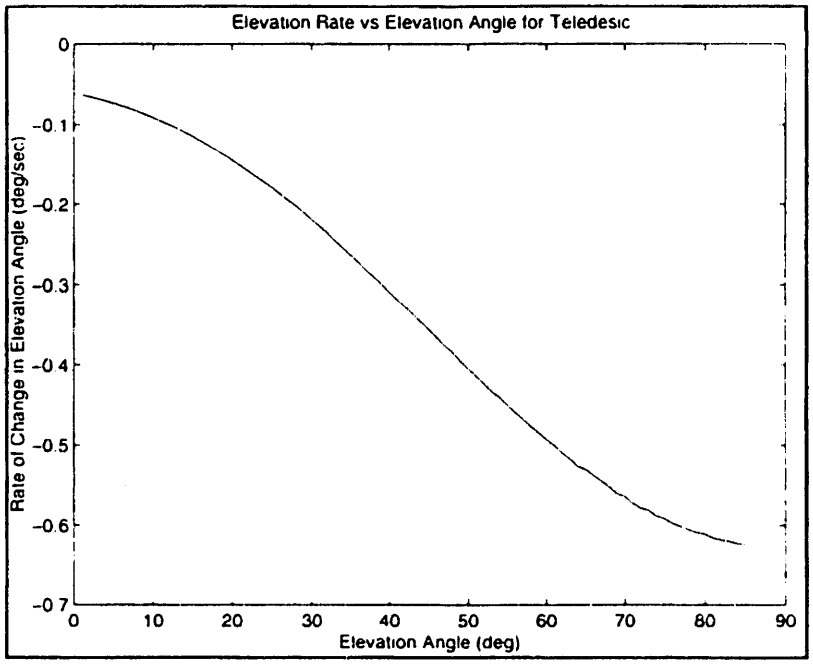


Figure 4-18: Elevation Angle Rate Vs Elevation Angle

Figure 4-18 shows that the elevation angle rate from E=40° to E=30° varies from -0.3 deg/sec to -0.2 deg/sec. To achieve the desired -2° maximum error would result in evaluating the elevation angles every 14.4 seconds. A 14.4 second time step would require excessive calculation times on the computer systems used. Therefore, the minimum elevation plots could not be

[2]This number is found in generating equally spaced points over the Earth, as opposed to points equally spaced in latitude and longitude.

generalized to include all times between each evaluation. The metric is only valid for the time of each evaluation, so comparison between plots at different times do not necessarily demonstrate a worse constellation. A 60 second time step for two days was eventually used to generate the minimum elevation plots.

Table 4-5 summarizes the maximum errors and assumptions in the minimum elevation angle plots.

Table 4-5: Error and Assumption Summary

Error Source	Error / Assumption
Vector from center of the Earth to grid point is perpendicular to the surface.	±0.19°
Satellite Position	±0.83°
Grid Spacing	-2°
Length of time between each elevation angle evaluation	Elevation maps only describe elevation angles at time steps, not time between evaluations.
LOWER BOUND	**E - 3.02°**(E is the calculated elevation angle)
UPPER BOUND	**E + 1.02°**(E is the calculated elevation angle)

The numbers in table 4-5 are important as they describe how accurately the calculated values describe the true minimum elevation angles.

When comparing the minimum elevation angle between constellations, the spherical Earth errors are removed. The upper and lower bounds for comparing minimum elevation angles between constellations are shown in table 4-6.

Table 4-6: Error Bounds for Comparing Constellations

LOWER BOUND	**E - 2.83°**(E is the calculated elevation angle)
UPPER BOUND	**E + 0.83°**(E is the calculated elevation angle)

4.4.4.2 Minimum Elevation Angles

The minimum elevation angle metric was used to compare the constellation at epoch and five years after epoch to quantify the impact of perturbations on the constellation. A summary of the perturbations and metric evaluation conditions is shown in table 4-7.

Table 4-7: Summary of Perturbations and Metric Evaluation Conditions

Epoch Date	April 1995
Comparison Date	April 2001
Perturbations	Geopotential (21X21 JGM2) Third Body Solar Radiation Pressure
Propagation Method	PVM/DSST
Points Evaluated	Longitude. ±90° degrees latitude. Points every 0.4° latitude.
Frequency and Duration of Elevation Angle Evaluation.	Evaluated Every 60 Seconds for 2 Days.
Number of Satellites used to Generate Element Histories	21
Number of Satellites Propagated to Generate Minimum Elevation Angle Plots	840

Propagating the constellation two days after epoch gives the minimum elevation plot shown in figure 4-19.

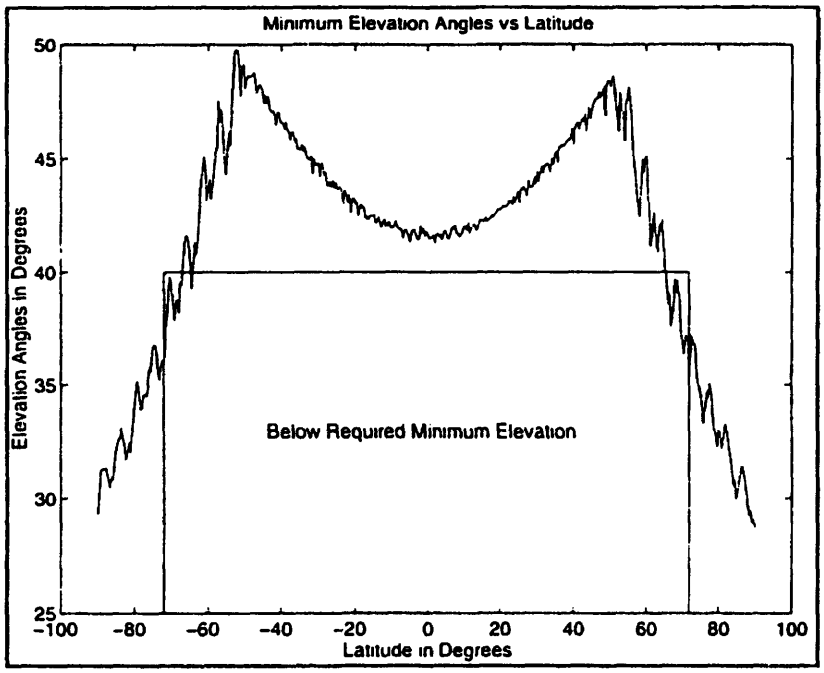


Figure 4-19: Initial Minimum Elevation Angles Vs Latitude for the Nominal Constellation

The nominal Teledesic constellation at epoch for the times sampled is very close to meeting its minimum elevation angle requirements.

The impact of perturbations on the nominal constellation are shown as maximum variations in Keplerian elements over five years. The input file containing the twenty-one nominal satellites is included in Appendix B. Generating these element histories took approximately 2 hours and 30 minutes, using two SPARC 20's, one SPARC 10, and a SPARC ELC.

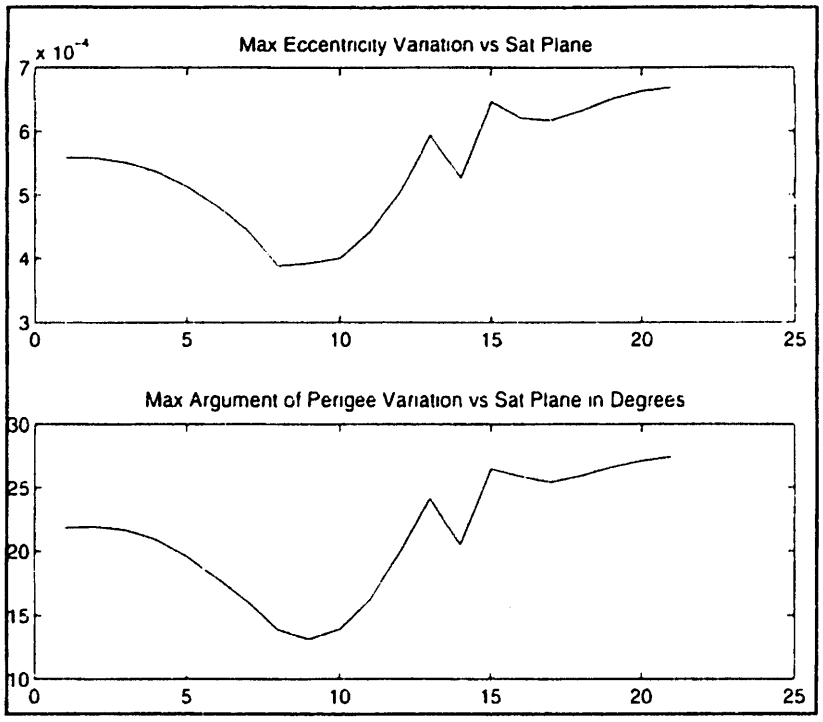


Figure 4-20: Nominal Constellation Element Histories

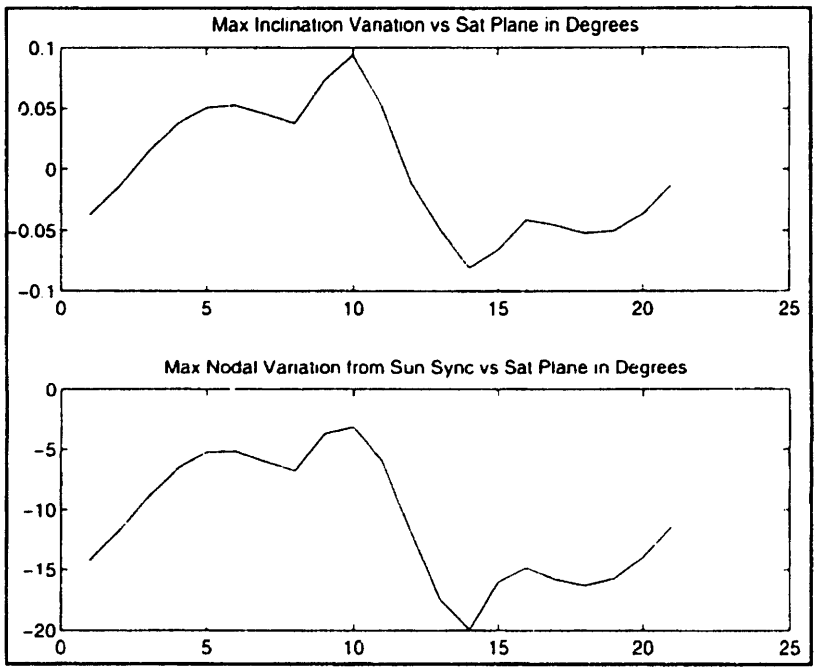


Figure 4-21: Nominal Constellation Element Histories

The element histories lead to two conclusions about the Teledesic satellite constellation:

- The orbits are not frozen over time.
- The ascending nodes vary from the sun-synchronous rate.

The minimum elevation angles after five years are shown in figure 4-22. As stated in table 4-4, the minimum elevation angles depicted in figure 4-22 cannot be directly compared to the nominal elevation plots because the elevation angles were not calculated at a high enough frequency to remove reasonable errors from the metric evaluation. However, minimum elevation plots evaluated at the same times can be compared.

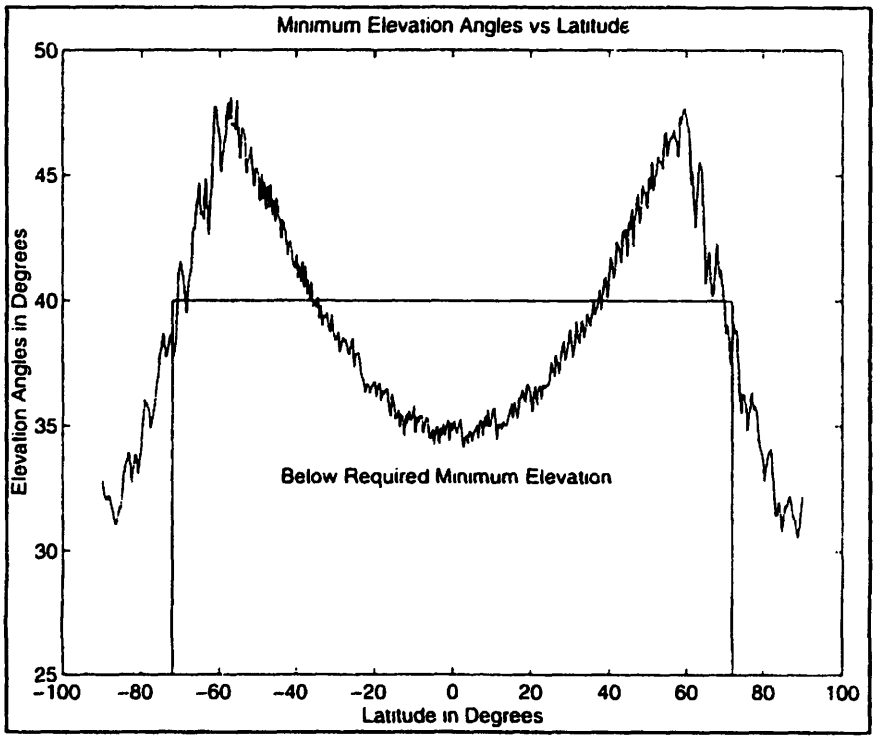


Figure 4-22: Minimum Elevation Angles of Nominal Constellation Five Years after Epoch.

The minimum elevation angles in figure 4-22 were generated by propagating for five years after epoch and then outputting satellite positions every minute for two days. Figure 4-22 shows that the angles drop well below the required 40° minimum elevation. Section 4.4.5 describes the process in which the initial constellation was modified with the goal of better achieving the constellation requirements.

In summary, the nominal constellation propagated five years has the following problems with respect to orbital requirements.

- The orbits are not frozen over time.
- The ascending nodes vary from the sun-synchronous rate.
- The minimum elevation angles at the comparison times fall below the elevation angle requirements.

4.4.5 Constellation Modifications

To understand what was causing the deviation from orbital requirements seen in the previous section, the solar radiation pressure was removed and the same plots were generated again. The element histories and minimum elevation plots are shown in figures 4-23 through 4-25.

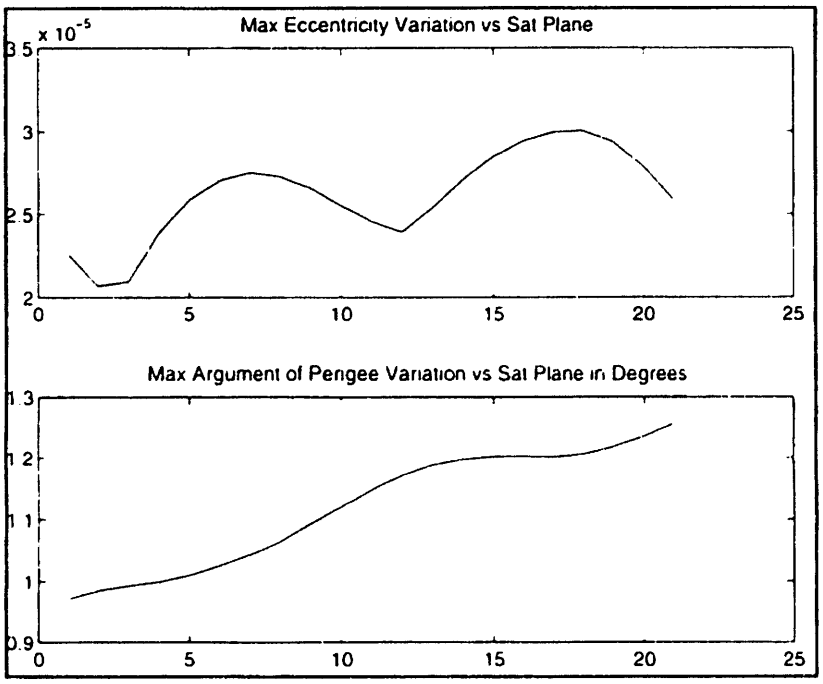


Figure 4-23: Element Histories Without Solar Radiation Pressure

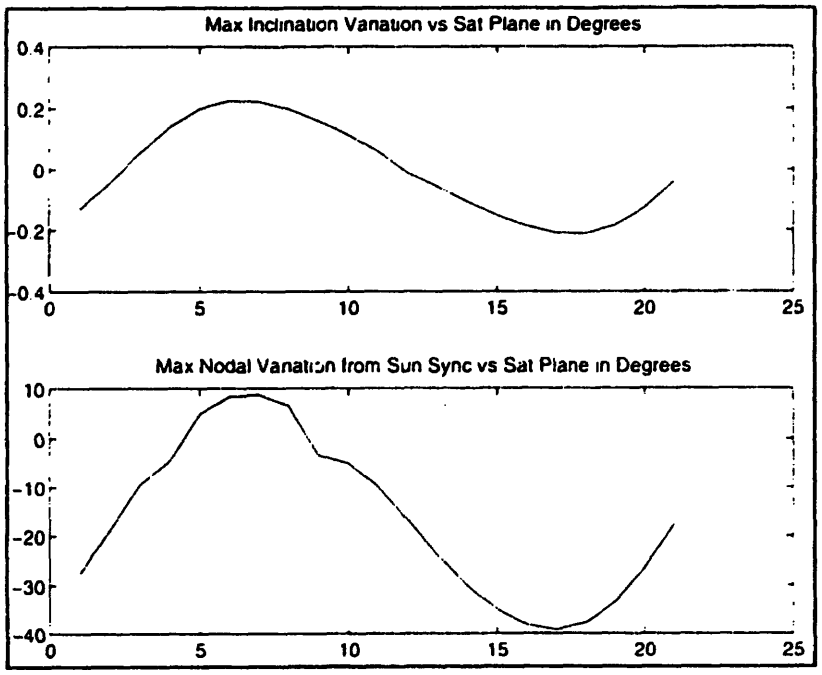


Figure 4-24: Element Histories Without Solar Radiation Pressure

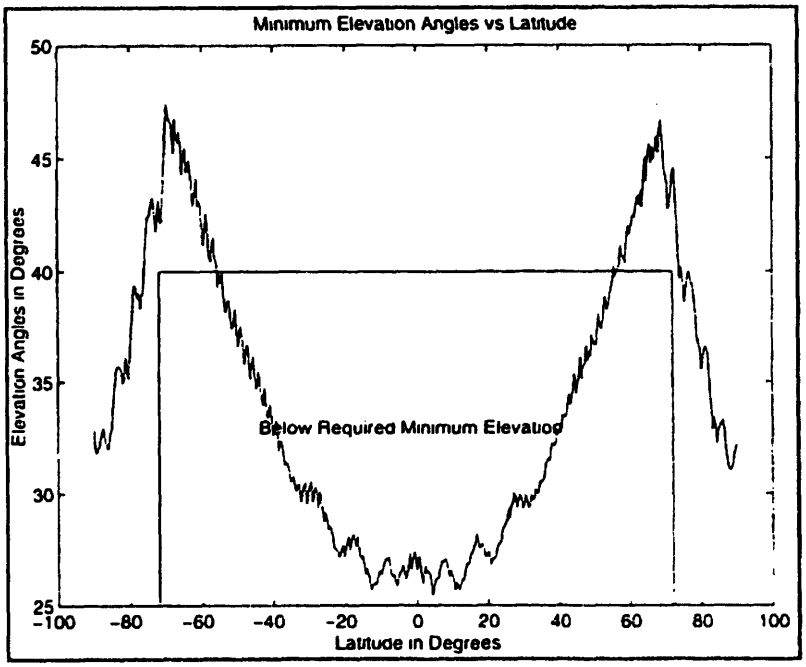


Figure 4-25: Element Histories Without Solar Radiation Pressure

The following hypotheses are generated from a comparison of figures 4-23 through 4-25 against figures 4-20 through 4-22.

- Because the satellite area/mass ratio is much larger than average, the solar radiation pressure is a significant perturbing force on the satellite. Solar

radiation pressure results in the largest variation in eccentricity and argument of perigee.

- Solar radiation pressure and third body create large inclination changes which effect the nodal rate. The solar radiation pressure counteracts the solar point mass effects on the inclination and the nodal rate.

- The less than required elevation angles result from differing nodal rates. If the nodal rates can be made more consistent across all the planes, the minimum elevation angles will not decrease.

4.4.5.1 Initial Cost Function Design

From the hypotheses, the constellation requirements, and trial and error, the cost function shown in equation 4-11 was developed in order to make use of the GA to perform orbit optimization.

$$\text{cost} = \frac{abs(e - e_{nom})}{\max(\Delta e)} + \frac{abs(\omega - \omega_{nom})}{\max(\Delta \omega)} + \frac{3 * abs(\Omega - \Omega_{sun-sync})}{\max(\Delta \Omega_{sun-sync})} \qquad (4\text{-}11)$$

One satellite in each plane was then modified by the orbit optimization tool to minimize the cost function shown in equation 4-11. Each orbit was treated as a separate optimization problem. The entire process of optimizing all twenty-one orbits took approximate twenty-four hours using two SPARC 20's, one SPARC 10, and one SPARC ELC. The *dome.in* file used is shown in figure 4-26.

```
Firs t run of optimizing satellite orbits
0,                              itest
9,35000,0.1,                    iopt,maxitr,epsiln
42,1000,1,                      kseed,mpopsize,ncomp
1,0,0,0,0.2,0,                  Opts:  constr,clones,Popt,Ropt,Topt,ishr
0,                              fixed parameters
3,                              continuous parameters
0,0.0,                          it chooses initial conditions
7071.14,0.00100,98.1020,        min of continuous
7085.14,0.00136,98.2220,        max of continuous
0,                              discrete parameters (3 failure rates)
4,                              number of bins for each discrete parameter
0,                              initial discrete (ga: param# ie. #1)
.001169,.001171,.001173,.001175,
```

Figure 4-26: *dome.in* for Constellation Optimization

The semimajor axis, inclination, and eccentricity were the parameters the optimization tool modified. The initial semi-major axis, eccentricity, and inclination chosen by the optimization run are shown in figure 4-27.

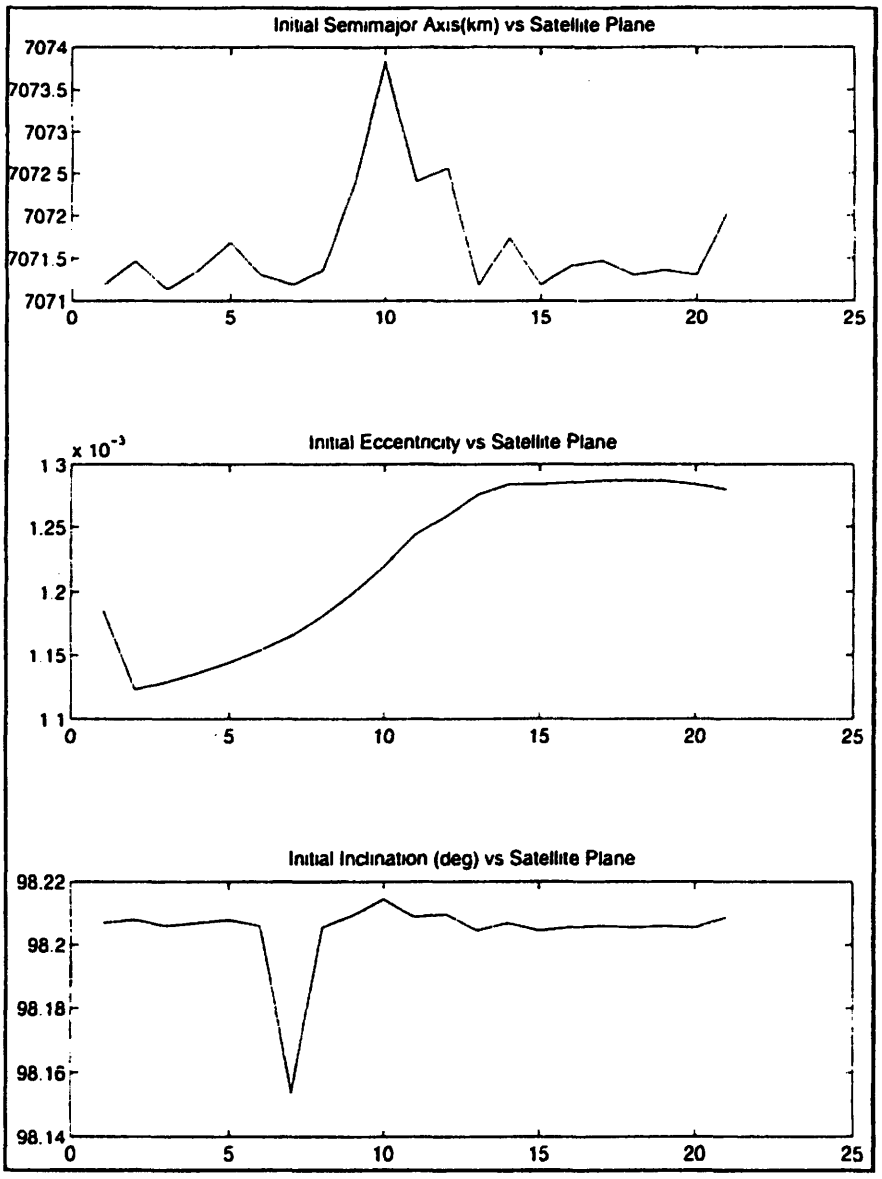


Figure 4-27: 'Optimized' Elements at Epoch

The element histories of the optimized constellation are shown in figures 4-28 through 4-29. Note that these element histories correspond to the nominal element histories shown in figures 4-20 and 4-21.

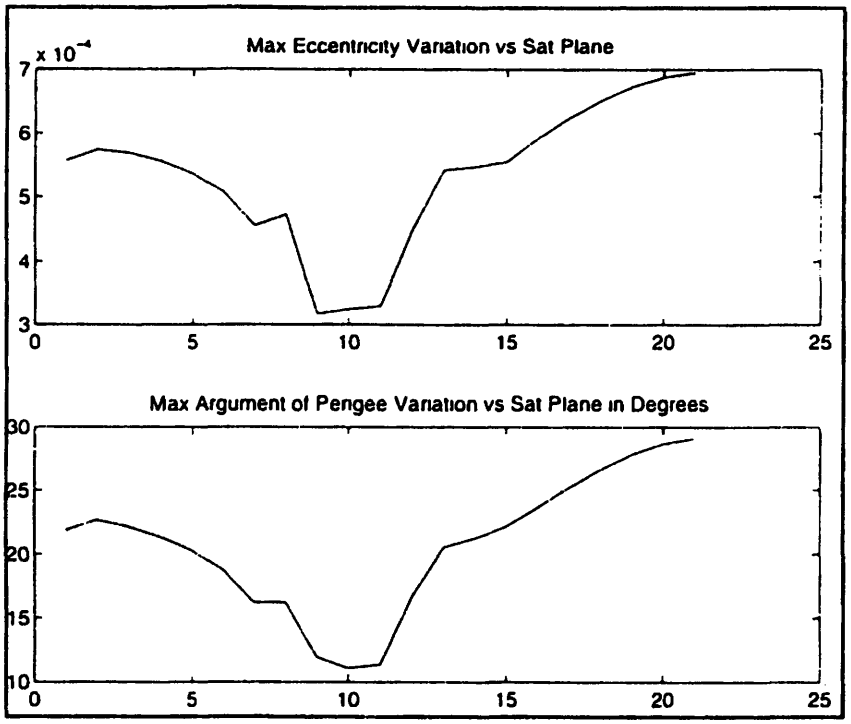


Figure 4-28: 'Optimized' Constellation Element Histories

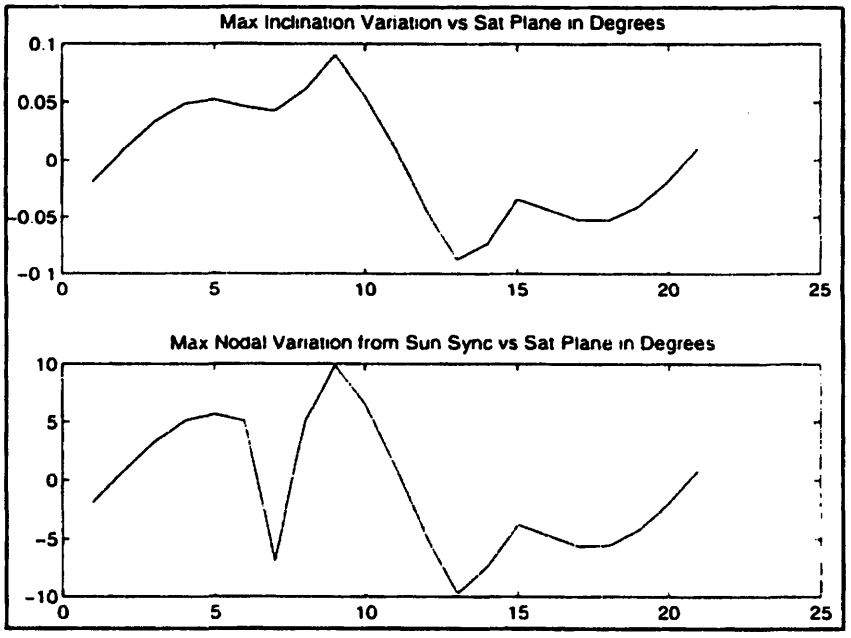


Figure 4-29: 'Optimized' Constellation Element Histories

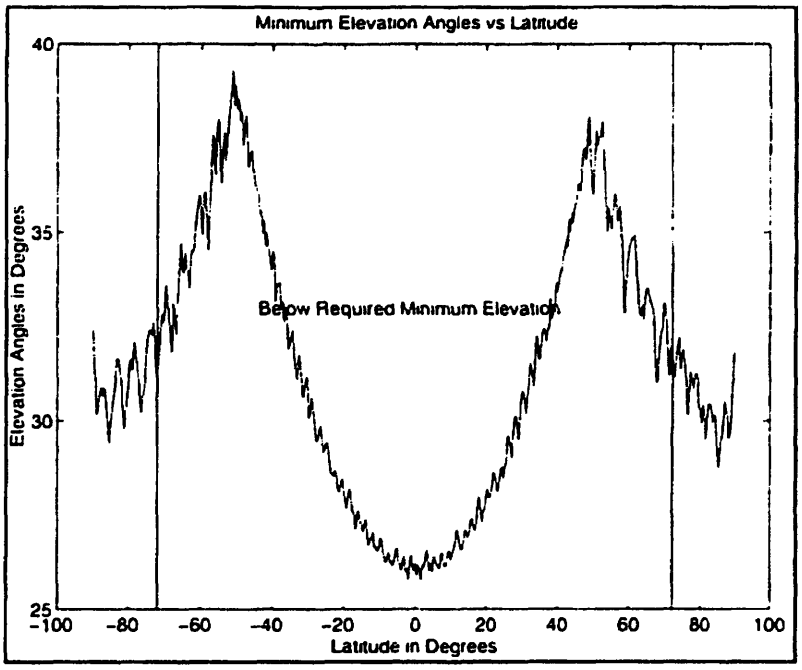


Figure 4-30: 'Optimized' Minimum Elevation Angles

The resulting 'optimized' constellation had some positive and negative features. The argument of perigee and eccentricity variations were kept to near their original value. The inclination variations were also similar to the nominal variations.

The nodal rate was much closer to the sun synchronous rate. This can be seen more clearly in figure 4-31.

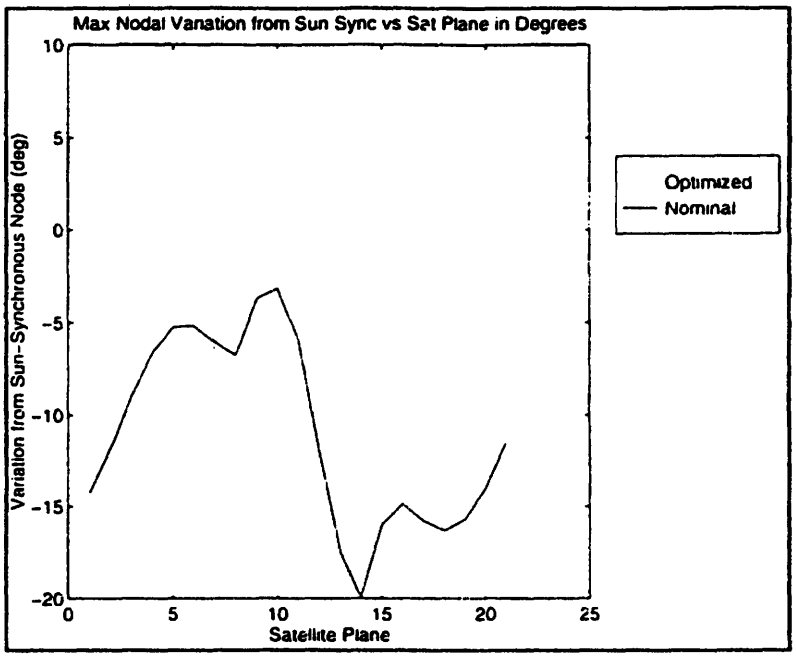


Figure 4-31: Maximum Deviation from Sun Synchronous Node for Nominal and 'Optimized' Constellations

The optimization algorithm chose elements for the constellation that best satisfied the cost function. In doing so, the maximum deviation from the sun-synchronous value varied more quickly in adjacent planes. Because of this problem, larger gaps in coverage occurred and the minimum elevation plot was actually worse. This is seen more clearly in figure 4-32.

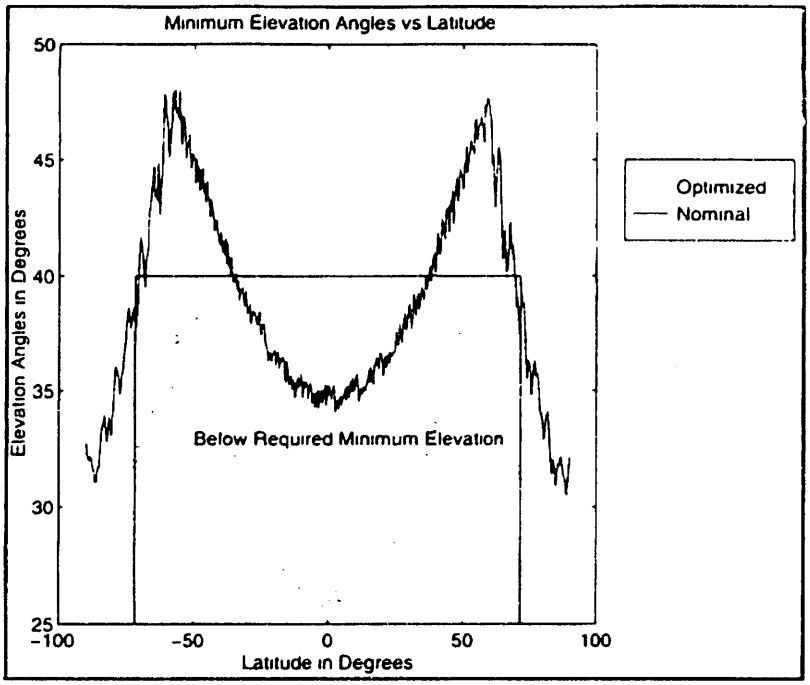


Figure 4-32: Minimum Elevation for Nominal and Optimized System

The optimized constellation has a much worse elevation angle five years from epoch, approximately 10° at every latitude. This comparison demonstrates a significant degradation in performance as the errors calculated in table 4-6 are much smaller than the average difference.

4.4.6 Conclusions

Maintenance of the Teledesic constellation presents a great number of technical challenges. Different planes will require different maneuver planning budgets to make up for the inclination changes induced by the third body perturbation. All satellites will have to be designed to carry the 'worst case' amount of fuel so that each satellite can be produced identically. Certain planes will require much more frequent inclination maintenance maneuvers. These problems all have impact on the system design.

If the sun-synchronous requirement were removed from the Teledesic system, it is possible the variation in nodal rates would decrease dramatically. The orbit optimization tool could be configured to choose a new nodal rate and 'design' a constellation that has a more consistent node rate across each plane.

The orbit design tool is an effective tool for the orbit designer. Because the GA is operating in a parallel environment with the DSST, an enormous amount of computation can be performed. Careful design of the cost function is critical to the result achieved with the tool. Any concerns or requirements not present in the cost function or the parameter constraints will be ignored, which may lead to unwanted results.

5.0 Conclusions and Future Work

5.1 Conclusions

5.1.1 PVM/DSST

Satellite constellations are increasing in number for a variety of reasons, the most important of which are:

- Ability to provide worldwide communications.
- Market potential of mobile communication and information services.
- Technological developments in communication systems.

All the satellite constellations proposed will require significant computing resources to track, control, and maintain. The demand for scalable, portable, and flexible flight dynamics software will continue to grow as many of these systems are being proposed by commercial ventures interested in cost efficient use of resources. Parallel computing can provide the necessary computing resources required for such systems with cost efficiency. However, software must be designed to take advantage of the parallel hardware.

PVM was chosen from the available methods for implementing a parallel orbit propagator, as it provided the most capability in the shortest amount of time. It's main advantages included:

- Portability
- Ease of using legacy code
- Ability to use on a network of workstations

The data parallel and multi-threaded approaches may have produced more performance but would have required re-writing more software. With PVM the legacy software was used almost 'as is'.

The PVM/DSST is an initial step in the development of a constellation flight dynamics system. Combining the message passing approach to interprocess communication with a FORTRAN 77 programming environment proved to be an effective method for creating a parallel application from existing serial code. The development of the PVM/DSST required very little new software in comparison to the amount of existing code used. The use of this legacy software created a much more powerful application.

The PVM/DSST was used to effectively propagate multiple satellites. Adding additional processors demonstrated speed-up and efficient use of all available hardware. Several factors make the PVM/DSST practical for examining multiple satellite constellations. These factors include:

- A simple approach to the work division and task management.
- A versatile orbit propagator with the capability to produce mean elements using a variety of perturbation models.
- An easily reconfigurable networking system with low setup costs and efficient communication.
- A network of computers using shared disk resources.
- The ability to produce a wide variety of output data in different formats.

The only significant limitation in the design of the PVM/DSST was its inability to demonstrate speedup in propagating a single trajectory.

Because the goal was not to produce an operational system, the error handling capabilities of the UNIX programming environment and the PVM system were not exploited. Lack of error handling development occasionally caused failures while using the PVM/DSST. For instance, processes would remain running after program completion or PVM would not start correctly. In such a situation the following actions were taken:

- Halting pvm, if possible.

- Killing all active processes.
- Deleting all the /tmp/pvm[dl].[uid] files.

The four processor SPARC, known to PVM as a SUNMP architecture, had more problems than any of the single processor machines. Use of PVM on the four processor machine required tuning of kernel parameters. These parameters are located in the /etc/system file.

On the multiprocessor platform, the problems seen were most likely due to:

- The PVM implementation on the SUNMP architecture is not completely error free.
- The four processor machine was not administered for optimal parallel application execution. This job would require thorough knowledge of the architecture as well as the operating system.

These problems may also contribute to the lower efficiency achieved on the multi-processor platform. Despite the lower efficiency, the computation-to-cost ratio of this machine is still very good.

5.1.2 Orbit Optimization Tool

The use of Schott's genetic algorithm (GA) optimization method proved to be an excellent match with the PVM/DSST [64]. GA's are not as computationally efficient as other approaches, requiring more cost function evaluations to reach the optimal answer. However, by operating on a population instead of just one set of parameters, GA's can take advantage of parallel cost function evaluations. Using a GA in a parallel computing environment reduces the impact of its computational inefficiencies. If the hardware is available to perform the necessary computation, the simple interface to the GA makes this a powerful optimization method for orbit design. Cost functions are easily developed and no derivative information is necessary.

Unfortunately, the effort was not made to automate the process of minimizing the cost function to within a given tolerance. However,

excellent results were attained by manually reducing the interval and repeating the optimization algorithm in a smaller parameter space.

The capabilities of the GA combined with the PVM/DSST were demonstrated in Section 4.3.3. Frozen orbit determination in the presence of a 21-by- 0 zonal field was achieved to an arbitrary level of accuracy. A more difficult problem was examined in the attempt to optimize the Teledesic orbit to better meet requirements. This problem is discussed in the next section.

5.1.3 Teledesic

The Teledesic satellite communication system is an enormous project. There are many factors that complicate the development of this system. One of largest technological challenges is constructing the 840 satellite constellation.

Analysis of the 840 satellite Teledesic constellation was performed, in part, to stretch the computational capability of the PVM/DSST and the optimization tool.

A system of distributed workstations using 2 SPARC 20's, a SPARC 10, and a SPARC ELC was able to propagate the 840 satellite orbits for five years in approximately 2 hours and 15 minutes. All available perturbation models except drag (21-by-21 spherical harmonics, solar radiation pressure, and third body) were used in this analysis.

The orbits initially chosen for the constellation represent a unique approach to satellite constellation design. Because Teledesic uses a high inclination orbit and does not control the phasing of satellites in adjacent planes, different semimajor axes are required for each plane to prevent collision. Because of the sun-synchronous and frozen orbit constraints, each plane will require slightly different orbital elements. With very tight tolerances on collision and different elements for each of the planes, orbital perturbations to the constellation will be a significant part of the orbit refinement procedure.

The following assumptions were made in the analysis of the Teledesic constellation:

- Drag effects were canceled by satellite maneuvers.
- The possibility of collision was not addressed.
- Mean elements were used in the analysis.

From the analysis discussed in Chapter 4, the Teledesic system provides the desired minimum elevation angle with the nominal system. However, perturbations will have different effects on each of the satellite planes.

The initial orbit design does not passively meet the sun-synchronous requirement in the presence of perturbations. Although thrusting maneuvers could be used to maintain the sun-synchronous requirement, it appeared that different initial conditions could be chosen to better maintain the orbits. Using the genetic algorithm optimization method, a new system was designed that more closely meets the sun-synchronous requirement with the same perturbations. However, the new system did demonstrate a lower minimum elevation angle after five years. This was due to increased nodal spacing between adjacent planes.

Both the nominal and optimized systems exhibit large variations in inclination. To maintain the nominal constellation minimum elevation angles, inclination must be kept within a narrow tolerance. Because the inclination variations are plane dependent, the fuel budget will be different per plane. This result may alter the optimal design of a common satellite for all planes, as each satellite will be forced to carry the 'worst case' amount of fuel for out of plane maneuvers.

5.2 Future Work

5.2.1 PVM/DSST

There is an enormous amount of future work in the area of parallel computing and astrodynamics. If algorithms and software are created to be efficient and scalable, the amount of computation capability available will increase dramatically. This could impact many areas of astrodynamics.

Specific ideas for future work include:

- The current software is written to be portable to the CM-5. A CM-5 implementation would provide more computing capability than feasible within the current environment. The CM-5 would also be a stable platform for testing that would give a direct comparison between the computation levels achieved using a network of workstations and a supercomputer.

- Move the software to an IBM PC using the LINUX operating system to demonstrate high level performance on a network of personal computers. This implementation would demonstrate the power of networking low cost computers.

- Examine the cost/performance ratio for additional multi-processor workstations. Some machines are currently being offered with sixteen processors per workstation (SGI), which could represent enormous computing capability for the cost if the efficiency of these machines remains relatively high when executing parallel applications.

- Develop a GUI interface to the current system with the concept of expanding it into a constellation flight dynamics interface.

- Examine other workstation networking products such as Network Linda. These tools may provide a better parallel programming environment to develop a more comprehensive parallel and scalable flight dynamics system.

- Redesign and rewrite sections of the stand-alone to perform vector calculations with a data parallel language such as FORTRAN 90/HPF. This effort is currently ongoing at the Charles Stark Draper Laboratory with support from Phillips Lab/VTA.

- Implement a different algorithm for orbit propagation using the DSST. For example, calculate the mean elements for an interval and then spawn processes to calculate the short periodic contributions in parallel. This concept could produce significant speed-up when a high density of accurate state evaluations are needed in a short amount of time, as in a batch orbit determination.

- Develop a full flight dynamics system on top of a networking system such as PVM. Keeping the software scalable and efficient would allow the system to increase its computing capability by simply adding more computers instead of redesigning the software.

These ideas do not include all the future work in combining parallel computing and astrodynamics.

5.2.2 Orbit Optimization Tool

There are many possible future applications of the orbit optimization tool, from calculating optimal maneuvers that minimize fuel expenditures to a more comprehensive optimization of a satellite constellation.

The constellation design problem could be approached more thoroughly if the constellation was examined as one optimization problem. Each satellite could be represented as an additional parameter to be optimized. Other than machine capacity, the GA has no limits on the number of parameters that can be solved for. Cost functions could be designed for the entire constellation, including direct evaluation of metrics such as the minimum elevation angle. This could be particularly helpful in refining current designs to perform optimally in the presence of perturbations.

In addition, the design of the orbit optimization tool can be improved. Many different types of GA's are available for solving a variety of problems. A thorough investigation may show other GA's will solve the problems more efficiently. Additionally, the combination of the GA with different optimization methods could be used to automatically refine an answer within a given tolerance.

5.2.3 *Teledesic*

The perturbative effects on the Teledesic constellation should be examined if the nodal rate is changed from the sun synchronous rate. This can easily be done using the orbit optimization tool, by optimizing the constellation to minimize nodal deviations to the desired rate. If the relaxation of the sun-synchronous constraint significantly decreases the variations between planes, Teledesic may have to weigh the advantages of a sun-synchronous orbit against a decreased fuel budget.

The Teledesic constellation will undoubtedly undergo further revisions to its initial constellation. The next level of analysis should examine the area-to-mass ratio to determine how to best model the drag and solar radiation pressure effects. The perturbative variations due to the natural solar cycles should be analyzed for the lifetime of the constellation.

Appendix A: Keplerian and Equinoctial Elements

This appendix describes the Keplerian and equinoctial elements. The Keplerian elements are well known because they geometrically describe the two-body orbit.

The Keplerian elements are described in table A-1. The descriptions apply to elliptical orbits only in an inertial reference system (**I J K**).

Table A-1: Description of Keplerian Elements [38]

ELEMENT	DESCRIPTION
Semimajor Axis (a)	One half the major axis of the ellipse.
Eccentricity (e)	The shape of the ellipse. e=0 is a circle e=1 is a parabola The eccentricity vector ($\underline{e}$) points in the direction of periapsis.
Inclination (i)	The angle between the vector normal to the plane and the **K** vector.
Longitude of Ascending Node (Ω)	The angle between **I** and the point where the orbit crosses the (**I J**) plane in a northerly direction (**-K to +K**).
Argument of Perigee (ω)	The angle between the ascending node and the periapsis measured in the orbital plane.
True Anomaly (υ)	The angle between the eccentricity vector and the position of the satellite.

Figure A-1 depicts the Keplerian orbital elements.

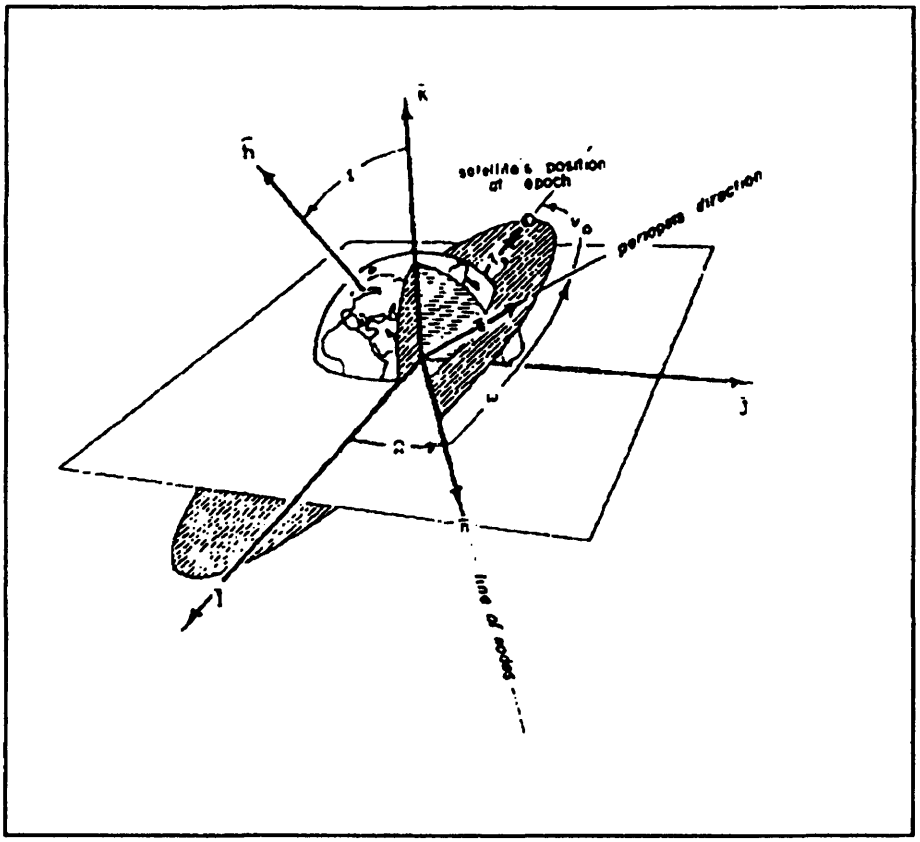


Figure A-1: Geometry of Keplerian Elements [38]

The non-singular equinoctial elements can be defined in terms of the Keplerian elements. The equinoctial elements are described by equations given in table A-2.

The retrograde factor (I) is necessary to describe the equinoctial elements. If the wrong retrograde factor is used, the equinoctial element set is singular for equatorial orbits. For direct equatorial orbits, the retrograde factor must be set to +1; for retrograde equatorial orbits, it must be -1.

Table A-2: Equinoctial Elements [49]

Equinoctial Element	Keplerian Element
a	a
h	$e\sin(\omega+I\Omega)$
k	$e\cos(\omega+I\Omega)$
p	I=+1 $\tan(i/2)\sin(\Omega)$ I=-1 $\cot(i/2)\sin(\Omega)$
q	I=+1 $\tan(i/2)\cos(\Omega)$ I=-1 $\cot(i/2)\cos(\Omega)$
λ	$M+\omega+I\Omega$
I	Retrograde Factor I=+1 for $0^\circ \leq i < 180^\circ$ Direct Elements I=-1 for $0^\circ < i \leq 180^\circ$ Retrograde Elements

Appendix B: Software Listings

The first two sections of this appendix contain software listings. Section B.1 contains the listings for the *const_prop* software. This software contains all the routines that were used to perform communication between processes. The difference between *const_prop* and *sat_prop*, is demonstrated in figures 3-5 and 3-6.

The routines listed Section B.1 are:

- const_prop.F
- const_opt.F
- rdconst.F

Section B.2 lists the software written to interface directly to the DSST. This software contains no PVM calls and can be used without a message passing environment. This software is used to execute the DSST (figure 3-15) and also interface below the constellation software (figure 3-5).

The routines listed in Section B.2 are:

- sat_prop.F
- sat_opt.F
- set_satopt.F
- crrequest_times.F
- sort_times.F

Section B.3 lists an example PVM/DSST input file. The input file contains one satellite per plane from the nominal Teledesic constellation.

B.1 Message Passing Listings

B.1.1 Program const_prop

```
#include "array_sizes.h"
#define MAX_NPROCS      200
#define NTASK_PER_HOST 4
#define MSGTAG          10

c-----------------------------------------------------------------------
      program const_prop
c
c     const_prop.F - a FORTRAN program that distributes
c     itself among a pvm virtural machine to run multiple instances
c     of the DSST
c
c     Scott T Wallace, LT, USAF
c     Master's Student, MIT Aero/Astro
c
c-----------------------------------------------------------------------
c
      implicit none
c
c     Include the FORTRAN PVM header file
      include '/Users/taz/scott/pvm3/include/fpvm3.h'
c
      character*18 nodename, host(MAX_NUM_HOSTS)
      character*8  arch
      character*12 env_input, env_output
      character*MAX_PATH_LENGTH indata_path, outdata_path
      character*MAX_PATH_LENGTH const_file, satdat_file
c
      integer*4 mytid, info
      integer*4 tids(0:MAX_NPROCS)
      integer*4 i, info, nproc, nhost
      integer*4 mytid, ptid, dtid
      integer*4 speed, narch, ntask
      integer*4 bufid
      integer*4 njobs, jobs_rec, jobs_sent
      integer*4 numt, k
      integer*4 const_size, nintervals
      integer*4 nburns, satno
      integer*4 constopt_int(INT_OPT_SIZE,MAX_NUM_SATS)
      integer*4 satopt_int(INT_OPT_SIZE)
      integer*4 eltype

      logical   fileex

      real*8    intervals(5,MAX_NUM_INTERVALS)
      real*8    burn_list(4,MAX_NUM_BURNS)
      real*8    constopt_dbl(REAL_OPT_SIZE,MAX_NUM_SATS)
      real*8    satopt_dbl(REAL_OPT_SIZE)
c     ----------------------------------------------------------------
c
c     Get the pathnames for the data files
      satdat_file = 'satdata'
      env_input   = 'CONST_INPUT'
      env_output  = 'CONST_OUTPUT'
      call getenv(env_input,  indata_path)
```

```
      call getenv(env_output, outdata_path)
      print *,outdata_path
c
c     Enter this process in PVM
      call pvmfmytid( mytid )
c
c     If I am the parent process then read in data start
c     and manage other programs
      call pvmfparent( ptid )
      if (ptid .eq. pvmnoparent) then
c
c     Get the name of the input file
      write(*,*)'Please enter the name of the constellation file:'
      read(*,*)  const_file
c      const_file='teledesic21'
c
c     Check to make sure the input file is there
c     Remove spaces at end of path
      i = 1
      do while(indata_path(i:i).ne.' ')
      i = i+1
      end do
      const_file = indata_path(1:i-1)//const_file
c
      inquire(FILE=const_file,EXIST=fileex)
      if (.NOT.fileex) then
         write(*,*)'This file is not located in the CONST_INPUT dir'
         stop
      end if

c     Read in the satellite data
         call rdconst(const_size, eltype, nintervals, intervals,
     1     nburns, burn_list, constopt_int, constopt_dbl,
     2     const_file)
         njobs = const_size
c
         nhost = MAX_NUM_HOSTS
         do i=1,nhost
            call pvmfconfig( nhost, narch, dtid, host(i), arch,
     2           speed, info )
d            print *,'My name was ',host(i), dtid
d            print *, 'I have ',nhost,' hosts'
         end do
         ntask = NTASK_PER_HOST*nhost
c
c     Check to make sure ntask is not larger than the njobs
         if (ntask.gt.njobs) then
            ntask = njobs
         end if
c
c     If  arch is set to '*' then ANY configured machine is acceptable
         nodename = 'const_prop'
         arch = '*'
         if (ntask.gt.0) then
            call pvmfspawn( nodename, PvmTaskDefault, arch, ntask,
     1           tids, numt)
         else
            write(*,*) 'No jobs to spawn'
            stop
         end if

c     Check for spawning problems
d        do 100 i=0, ntask
d           print *,'tid',i,tids(i)
```

```
d 100     continue

          if( numt .lt. nproc ) then
             print *, 'trouble spawning ',nodename
             print *, ' Check tids for error code'
             call shutdown( numt, tids )
          endif
c
cc     Send constellation data
          call pvmfinitsend(PVMDEFAULT, bufid)
          call pvmfpack(INTEGER4, eltype, 1, 1, info)
          call pvmfpack(INTEGER4, nintervals, 1, 1, info)
          call pvmfpack(REAL8, intervals, nintervals*5, 1, info)
          call pvmfpack(INTEGER4,nburns,1,1,info)
          call pvmfpack(REAL8, burn_list, nburns*4, 1, info)
          call pvmfmcast(ntask,tids,MSGTAG,info)

c      Setup for keeping track of jobs
          jobs_rec  = 0
          jobs_sent = 0
          k         = 0

c      Start loop to
c      1) Send out jobs to all processors
c      2) Wait til a job comes in and send out the next job
c      3) Collect jobs not received
c
          do while (jobs_rec.lt.njobs)
c
c      If I have already sent enough jobs
             if (jobs_sent.ge.ntask) then
                call pvmfrecv(-1,-1,bufid)
                call pvmfunpack(INTEGER4, k, 1, 1, info)
                jobs_rec = jobs_rec + 1
                write(*,*) 'I received from ',host(k+1)
             end if


cc     If I need to send a job
c      Note: Jobs_sent = satno
             if (jobs_sent.lt.njobs) then
                jobs_sent = jobs_sent + 1
                call pvmfinitsend(PVMDEFAULT, bufid)
                call pvmfpack(INTEGER4, k, 1, 1, info)
                call pvmfpack(INTEGER4, jobs_sent, 1, 1, info)
                call pvmfpack(INTEGER4, constopt_int(1,jobs_sent),
     1               INT_OPT_SIZE, 1, info)
                call pvmfpack(REAL8, constopt_dbl(1,jobs_sent),
     1               REAL_OPT_SIZE, 1, info)
                call pvmfsend(tids(k), MSGTAG, info)
                write(*,*) 'I sent satellite', jobs_sent,' to '
     2               ,host(k+1)
                k = k +1
             end if
c
          end do
c
c      Kill the slaves I spawned and then exit pvm myself
          call shutdown(numt,tids)

c      If I was a slave receive the data and do work
       else
c
c      Generate the output filename with path
```

```
         i = 1
         do while(outdata_path(i:i).ne.' ')
            i = i+1
         end do
         satdat_file = outdata_path(1:i-1)//satdat_file
         print *,satdat_file
c
cc     Receive the global broadcast data
         call pvmfrecv(ptid,MSGTAG,bufid)
         call pvmfunpack(INTEGER4, eltype, 1, 1, info)
         call pvmfunpack(INTEGER4, nintervals, 1, 1, info)
         call pvmfunpack(REAL8, intervals, nintervals*5, 1, info)
         call pvmfunpack(INTEGER4,nburns,1,1,info)
         call pvmfunpack(REAL8, burn_list, nburns*4, 1, info)
c
c      Do this loop always until I am killed
         do while (.TRUE.)
c        .
c      c    Receive the local satellite data
             call pvmfrecv(ptid, MSGTAG, bufid)
             call pvmfunpack(INTEGER4, k, 1, 1, info)
             call pvmfunpack(INTEGER4, satno, 1, 1, info)
             call pvmfunpack(INTEGER4, satopt_int,
     1            INT_OPT_SIZE, 1, info)
             call pvmfunpack(REAL8, satopt_dbl,
     1            REAL_OPT_SIZE, 1, info)
c
cc     Perform work
             call sat_prop(satno, eltype, nintervals, intervals, nburns,
     2            burn_list, satopt_int, satopt_dbl, satdat_file,
     3            indata_path)
c
cc     Send back my id so I can get more work
             call pvmfinitsend(PVMDEFAULT, bufid)
             call pvmfpack(INTEGER4, k, 1, 1, info)
             call pvmfsend(ptid, MSGTAG, info)
c
         end do
       end if
       stop
       end
c
c
       subroutine shutdown( nproc, tids )
       integer nproc, tids(*)
c
c      Kill all tasks I spawned and then myself
c
       do 10 i=1, nproc
c          write(*,*) 'Tid ', i,' was ',tids(i)
          call pvmfkill( tids(i), info )
  10   continue
       call pvmfexit( info )
       return
       end
```

B.1.2 Program const_opt

```
#include "array_sizes.h"
#define NPARAMETERS     3
#define MAX_NPROCS      200
#define NTASK_PER_HOST 4
#define MSGTAG          10

c----------------------------------------------------------------------
      subroutine const_opt(njobs,params,answers,ntask)
c
c     const_prop.F - a FORTRAN subroutine that distributes
c     itself among a pvm virtural machine to run multiple instances
c     of the DSST
c
c     Scott T Wallace, LT, USAF
c     Master's Student, MIT Aero/Astro
c
c----------------------------------------------------------------------
c
      implicit none
c
c     Include the FORTRAN PVM header file
      include '/Users/taz/scott/pvm3/include/fpvm3.h'
c
      character*18 nodename, host(MAX_NUM_HOSTS)
      character*8  arch
      character*12 env_input
      character*MAX_PATH_LENGTH indata_path
      character*MAX_PATH_LENGTH const_file, satdat_file
c
      integer*4 mytid, info
      integer*4 speed, narch, ntask
      integer*4 tids(MAX_NPROCS)
      integer*4 i, info, nproc, nhost
      integer*4 mytid, ptid, dtid
      integer*4 bufid
      integer*4 njobs, jobs_rec, jobs_sent
      integer*4 numt, k
      integer*4 const_size, nintervals
      integer*4 nburns, satno
      integer*4 constopt_int(INT_OPT_SIZE,MAX_NUM_SATS)
      integer*4 satopt_int(INT_OPT_SIZE)
      integer*4 eltype, jobno

      logical   fileex

      real*8    intervals(5,MAX_NUM_INTERVALS)
      real*8    burn_list(4,MAX_NUM_BURNS)
      real*8    constopt_dbl(REAL_OPT_SIZE,MAX_NUM_SATS)
      real*8    satopt_dbl(REAL_OPT_SIZE)
      real*8    params(NPARAMETERS,*), answers(*)
c     ----------------------------------------------------------------
c
c     Get the pathnames for the defualt constellation
         env_input   = 'CONST_INPUT'
         call getenv(env_input,  indata_path)
c
c     Enter this process in PVM
      call pvmfmytid( mytid )
c
c     If I am the parent process then read in data start
```

```
c      and manage other programs
       call pvmfparent( ptid )
c
c       if (ptid .eq. pvmnoparent) then
          env_input = 'OPT_FILE'
          call getenv(env_input,const_file)

c          const_file='tel_opt.14'
c
c      Check to make sure the input file is there
c      Remove spaces at end of path
          i = 1
          do while(indata_path(i:i).ne.' ')
             i = i+1
          end do
          const_file = indata_path(1:i-1)//const_file
c
          inquire(FILE=const_file,EXIST=fileex)
          if (.NOT.fileex) then
             write(*,*)'This file is not located in the CONST_INPUT dir'
             stop
          end if

c      Read in the general satellite data
          call rdconst(const_size, eltype, nintervals, intervals,
     1         nburns, burn_list, constopt_int, constopt_dbl,
     2         const_file)
c
c          do i=1,MAX_NUM_HOSTS
c             call pvmfconfig( nhost, narch, dtid, host(i), arch,
c     2            speed, info )
d             print *,'My name was ',host(i), dtid
d             print *, 'I have ',nhost,' hosts'
c          end do
c          ntask = NTASK_PER_HOST
c
c      Check to make sure ntask is not larger than the njobs
          if (ntask.gt.njobs) then
             ntask = njobs
          end if
c
c      If  arch is set to '*' then ANY configured machine is acceptable
          nodename = 'const_opt_slave'
          arch = '*'
          if (ntask.gt.0) then
            do i=1,ntask
              tids(i)=0
            end do
             numt=0
             call pvmfspawn( nodename, PVMTASKDEFAULT, arch, ntask,
     1            tids, numt)
          else
             write(*,*) 'No jobs to spawn'
             stop
          end if

c      Check for spawning problems
d          do 100 i=0, ntask
d             print *,'tid',i,tids(i)
d 100      continue

          if( numt .lt. nproc ) then
             print *, 'trouble spawning ',nodename
             print *, ' Check tids for error code'
```

```
            call shutdown( numt, tids )
          endif
c
cc      Send constellation data
          do i=1,ntask
            call pvmfinitsend(PVMDEFAULT, bufid)
            call pvmfpack(INTEGER4, eltype, 1, 1, info)
            call pvmfpack(INTEGER4, nintervals, 1, 1, info)
            call pvmfpack(REAL8, intervals, nintervals*5, 1, info)
            call pvmfpack(INTEGER4,nburns,1,1,info)
            call pvmfpack(REAL8, burn_list, nburns*4, 1, info)
            call pvmfsend(tids(i), MSGTAG, info)
          enddo

c       Multicast has problems on petunia
c       call pvmfmcast(ntask,tids,MSGTAG,info)

c       Setup for keeping track of jobs
          jobs_rec  = 0
          jobs_sent = 0
          k         = 1

c       Start loop to
c       1] Send out jobs to all processors
c       2] Wait til a job comes in and send out the next job
c       3] Collect jobs not received
c
          do while (jobs_rec.lt.njobs)
c
c       If I have already sent enough jobs
            if (jobs_sent.ge.ntask) then
              jobs_rec = jobs_rec + 1
              call pvmfrecv(-1,-1,bufid)
              call pvmfunpack(INTEGER4, k, 1, 1, info)
              call pvmfunpack(INTEGER4, jobno, 1, 1, info)
              call pvmfunpack(REAL8, answers(jobno),1,1,info)
             call pvmffreebuf(bufid, info)
d              write(*,*) 'I received from ',host(k+1)
            end if
c
cc      If I need to send a job
c       Note: Jobs_sent = satno
            if (jobs_sent.lt.njobs) then
c       Add in the appropriate parameters
c         1,4 is the eccentricity. The rest can be found in set_satopt.F
              jobs_sent = jobs_sent + 1
              constopt_dbl(3,1) = params(1,jobs_sent)
              constopt_dbl(4,1) = params(2,jobs_sent)
              constopt_dbl(5,1) = params(3,jobs_sent)
              call pvmfinitsend(PVMDEFAULT, bufid)
              call pvmfpack(INTEGER4, k, 1, 1, info)
             call pvmfpack(INTEGER4, jobs_sent, 1, 1, info)
              call pvmfpack(INTEGER4, jobs_sent, 1, 1, info)
              call pvmfpack(INTEGER4, constopt_int(1,1),
     1             INT_OPT_SIZE, 1, info)
              call pvmfpack(REAL8, constopt_dbl(1,1),
     1             REAL_OPT_SIZE, 1, info)
              call pvmfsend(tids(k), MSGTAG, info)
d              write(*,*) 'I sent satellite', jobs_sent,' to '
d     2              ,host(k+1)
              k = k +1
            end if
c
          end do
```

```
c
c     Kill the slaves I spawned and then exit pvm myself
         call shutdown(numt,tids)
c
      return
      end
c
c
      subroutine shutdown( nproc, tids )

      implicit none

      integer*4 info
      integer*4  nproc, tids(*)
      integer*4 i
c
c     Kill all tasks I spawned and then myself
c
      do 10 i=1, nproc
c         write(*,*) 'Tid ', i,' was ',tids(i)
         call pvmfkill( tids(i), info )
         if (info.ne.0) then
         print *,'Error in pvmfexit ',info
         end if
  10  continue
      call pvmfexit( info )
      if (info.ne.0) then
       print *,'Error in pvmfexit ',info
      end if
      return
      end
```

B.1.3 *Program rdconst*

```
#include "array_sizes.h"
      subroutine rdconst(const_size, eltype, nintervals, intervals,
     1     nburns, burn_list, constopt_int, constopt_dbl,
     2    const_file)

c-----------------------------------------------------------------------c
c
c     subroutine rdconst   - reads the constellation file
c
c     This program reads  a constellation file
c     for use in the multiple satellite propagator
c
c     Jan 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
      implicit none
c
      character*(*) const_file

      integer*4  unitnum, numsats, nintervals, intervalnum
      integer*4  nburns, satno, eltype
      integer*4  const_size, is(INT_OPT_SIZE)
      integer*4  constopt_int(INT_OPT_SIZE,MAX_NUM_SATS)
      integer*4  i, j
      integer*4 status
c
      logical   unit_unavailable
c
      real*8     rs(REAL_OPT_SIZE)
      real*8     constopt_dbl(REAL_OPT_SIZE,MAX_NUM_SATS)
      real*8     intervals(5,MAX_NUM_INTERVALS)
      real*8     burn_list(4,MAX_NUM_BURNS)
c
      include 'const_format'
c
c     Find the first available unit
         unitnum  = 10
         unit_unavailable = .true.
         do while  ( unit_unavailable )
            unitnum = unitnum + 1
            inquire ( unit=unitnum, opened=unit_unavailable,
     2           iostat=status)
         end do
c
c     Open the constellation file
         open(unit=unitnum, file=const_file, status='old')
c
c     Read the initial, global, data
      read(unitnum,110) const_size, eltype
      read(unitnum,*)
      read(unitnum,100) nintervals
      do i=1,nintervals
         read(unitnum,200) intervalnum, intervals(1,intervalnum),
     1        intervals(2,intervalnum)
         read(unitnum,200) intervalnum, intervals(3,intervalnum),
     1        intervals(4,intervalnum)
         read(unitnum,300) intervalnum, intervals(5,intervalnum)
      end do
```

```
      read(unitnum,100) nburns
      do i=1,nburns
         read(unitnum,400) burn_list(1,i), burn_list(2,i),
     1         burn_list(3,i), burn_list(4,i)
      end do
      read(unitnum,*)
c
c     Read the data for each satellite
      do i=1,const_size
         read(unit=unitnum,900)
         read(unit=unitnum,1000) satno, rs(1), rs(2)
         read(unit=unitnum,2000) rs(3),rs(4),rs(5),rs(6),rs(7),rs(8)
         read(unit=unitnum,3000) rs(9), rs(10)
         read(unit=unitnum,4000) rs(11), rs(12)
         read(unit=unitnum,5000) rs(13)
         read(unit=unitnum,6000) is(1), is(2), is(3)
         read(unit=unitnum,7000) is(4), is(5), is(6), is(7)
         read(unit=unitnum,8000) is(8), is(9), is(10)
         read(unit=unitnum,9000) is(11), is(12), is(13)
         read(unit=unitnum,900)
c
         do j=1,INT_OPT_SIZE
            constopt_int(j,i)=is(j)
         end do
         do j=1,REAL_OPT_SIZE
            constopt_dbl(j,i)=rs(j)
         end do
c
      end do
c
      close(unitnum)
c
      return
      end
```

B.2 DSST Shell Listings

B.2.1 Subroutine sat_prop.F

```
c-----------------------------------------------------------------------c
c
c     subroutine sat_prop   - propagates the satellite described
c                             in the argument list using the DSST.
c
c     This subroutine invokes Draper semianalytic satellite theory
c     to provide satellite precision mean elements and element rates
c     at user request intervals. All input is through the argument
c     list.  Output is directly into a file
c
c     Jan 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
c
c  Include files / This file requiures preprocessing
c
c  machine.h includes the machine specific definitions
c  maxArrays.h includes the maximum array sizes
c
#include  "machine.h"
#include  "array_sizes.h"
#define   NDATA_ITEMS 3
c
      subroutine sat_prop (satno, eltype, nintervals, intervals, nburns,
     2     burn_list, satopt_int, satopt_dbl,outfile, indata_path)
c
c  ---------------------------------------------------------------------
c     Argument list definitions
c
c          name           i/o  meaning
c
c          satno           i   satellite identification number
c          nintervals      i   number of intervals in interval array
c          intervals(5,*)  i   array containing times and deltas
c                            intervals(begining:  yyyymmdd(1,*), hhmmss(2,*)
c                            end: yyyymmdd(3,*), hhmmss(4,*), deltat(5,*))
c          nburns          i   number of burns in the burn array
c          burn_list(4,*)  i   array containing impulsive burns deltas
c          satopt_int(*)   i   an array of integer values for satellite
c                              propagation global to the constellation
c          satopt_dbl(*)   i   an array of double precision values
c                              for satellite propagation global to the
c                              constellation
c          burn_list (1,j) i   time of impulsive velocity manuever
c          burn_list (2,j) i   delta velocity in the x (r of rtn) direction
c          burn_list (3,j) i   delta velocity in the y (t of rtn) direction
c          burn_list (4,j) i   delta velocity in the z (n of rtn) direction
c  ---------------------------------------------------------------------
c
c          iatmos_preburn  i   selector for preburn drag modification
c                              -1 => no drag,       0 => overestimate drag
c                              +1 => nominal drag, +2 => underestimate drag
c          iatmos_postburn i   selector for postburn drag modification
```

```
c                               -1 => no drag,       0 => overestimate drag
c                               +1 => nominal drag, +2 => underestimate drag
c          rho_one_hi       i   drag modification: overestimation  percentage
c          rho_one_low      i   drag modification: underestimation percentage
c          epoch_ymd        o   epoch of mean elements file (yymmdd.)
c          epoch_hms        o   epoch of mean elements file (hhmmss.ssss)
c
c  ---------------------------------------------------------------------
c
c     subroutines called ============================================
c
c     intanl          - helps initialize draper semianalytic theory
c     beganl          - starts the draper semianalytic theory
c     orbanl          - propagates using draper semianalytic theory
c     kepeqn          - makes kepler elements from equinoctials
c     julpak          - converts packed calendar time to julian date
c     calpak          - converts julian date to packed calendar time
c
c     impulsive_burn_propagator - propagates with an
c                                 impulsive burn model
c
c     aldiff          - computes the time difference a.1 - utc
c     ddiv            - division with remainder
c     read_epot       - read the potential model matching the number
c                       used in the pme file
c
c
c     data types ====================================================
c
c     no implicit types
      implicit     none
c
c     Character Variables ===========================================
c
      character * (60)  filename
      character * (18)  buffername          /  'uninitialized'  /
      character * (72)  text
      character * (1)   blank               /  ' '              /
      character * (1)   comment             /  'c'              /
      character * (*)   outfile, indata_path

      integer*4          satopt_int(*)
      integer*4    burn_cntr,       data_cntr,        i
      integer*4    nburns,          nintervals,       nrequest_times
      integer*4    satno,           eltype
      integer*4    equin,           mtod,             mean
      integer*4    status,          unitnum,          k
c
      logical*4    unit_unavailable
      logical*4    burn_logical(MAX_NUM_TIMES)
      logical*4    setrtr
c
      real*8       intervals(5,*)
      real*8       satopt_dbl(*)
      real*8       request_times(MAX_NUM_TIMES)
      real*8       times(MAX_NUM_TIMES)
      real*8       burn_times(MAX_NUM_BURNS)
      real*8       burn_list(4,*)
      real*8       data(NDATA_ITEMS,MAX_NUM_TIMES)
      real*8       kepler(6)

      real*8       elmint(6),       ymdint,           hmsint
      real*8       pos(3),          vel(3),           oscelm(6)
      real*8       avrelm(6),       pvdrv(6,300),     avrdrv(6,300)
```

```
      real*8        endorb,          avrkep(6),         avrate(6)
      real*8        epoch_date,      epoch_tod,         obstim
      real*8        offset
      real*8        burn_delta_x,    burn_delta_y,      burn_delta_z
      real*8        dayjul0,         secjul0,           ymd
      real*8        hms,             dayjul,            secjul
      real*8        leapseconds,     quot
      real*8        aldiff
      real*8        radians,         degrees,           infinity
      real*8        rfactor,         forward
      real*8        ymd_vec(MAX_NUM_TIMES), hms_vec(MAX_NUM_TIMES)
      real*8        gha,             ecefR(3)
      real*8        calc_gha
c
c     constants ===============================================================
c
      parameter  (  radians       = 57.295779513082321    d00 )
      parameter  (  degrees       =  0.017453292519943296 d00 )
      parameter  (  infinity      = 99999999.             d20 )
      parameter  (  forward       = 1. d0    )
      parameter  (  equin         = 3        )
      parameter  (  mtod          = 2        )
      parameter  (  mean          = 2        )


c
c     FORTRAN include modules ==================================================
c
c     include the satellite epoch data buffer
      include 'PMERN.CMN'
c
c     BEGIN PROGRAM ============================================================
c
c     Convert to equinoctial elements
      if (eltype.eq.1) then
         rfactor = satopt_int(1)
         setrtr  = .false.
         kepler(1)     = satopt_dbl(3)
         kepler(2)     = satopt_dbl(4)
         kepler(3)     = satopt_dbl(5) /radians !Convert to radians
         kepler(4)     = satopt_dbl(6) /radians
         kepler(5)     = satopt_dbl(7) /radians
         kepler(6)     = satopt_dbl(8) /radians
         call eqnkep(elmint,rfactor,kepler,setrtr)
         satopt_dbl(3) = elmint(1)
         satopt_dbl(4) = elmint(2)
         satopt_dbl(5) = elmint(3)
         satopt_dbl(6) = elmint(4)
         satopt_dbl(7) = elmint(5)
         satopt_dbl(8) = elmint(6) / degrees    !Convert to degrees
      end if
c
c     Call the routine which will take the options input in the argument
c     list and put them in the PMERN common area and read in the
c     potential field
      call set_satopt(satopt_dbl, satopt_int, status)

c     setup satellite at epoch
      pme_cd      =  pme_cd * ( 1.d0 + pme_rho_one )
      pme_rho_one =  0.d0
c
c     find the first available unit
      unitnum  = 10
      unit_unavailable = .true.
```

```
      do while  ( unit_unavailable )
         unitnum = unitnum + 1
         inquire ( unit=unitnum, opened=unit_unavailable,
     2        iostat=status, err=999 )
      end do
c
c     open the gravity field file
      k = 1
      do while(indata_path(k:k).ne.' ')
      k = k+1
      end do
      filename = indata_path(1:k-1)//'epotfld'
c      print *, filename
      open (unit=unitnum, file=filename, status='old',
     2     form='unformatted', access='direct',
     3     recl = 1050*WORDLENGTH )
c
c     Call read_epot to read new gravity model and update common
      call read_epot(unitnum,status)
      if    ( status .ne. 0 )     goto 999
c
c     Close the input earth file
      close(unitnum)

c     Extract epoch from the buffer and adjust the century
      epoch_date = pme_date
      epoch_tod  = pme_time
      ymdint     = epoch_date-19000000.d0
      hmsint     = epoch_tod
c
c     Call julpak to obtain julian date at epoch
      call julpak (dayjul0,secjul0,ymdint,hmsint)
c
c     Extract epoch equinoctial elements from the buffer
      elmint (1)  = pme_els_equin (1)
      elmint (2)  = pme_els_equin (2)
      elmint (3)  = pme_els_equin (3)
      elmint (4)  = pme_els_equin (4)
      elmint (5)  = pme_els_equin (5)
      elmint (6)  = pme_els_equin (6) * degrees
      rfactor     = pme_retro
c
c     Call intanl to initialize force models
      call intanl (elmint,rfactor,equin,mtod,mean,ymdint,hmsint)
c
c     Call beganl to start the semianalytic integrator
      call beganl (forward)
c
c     Set integrator time to zero
      offset          = 0. d0
c
c     Make a list of request times (in seconds from epoch)
c     Check all request times to insure they come after the epoch
      call crrequest_times(dayjul0, secjul0, intervals, nintervals,
     1     request_times, nrequest_times, status)
c
c     Assign all the burn times to a vector
      do i=1,nburns
         burn_times(i)=burn_list(4,i)
      end do
c
c     Sort the request times & burn times together
      call sort_times(burn_times, nburns, request_times,
     $     nrequest_times,times, burn_logical, nrequest_times+nburns,
```

```
     $     status)
c
c     Create a separate cntr to keep track of burns and amount of output
      burn_cntr = 1
      data_cntr = 1
c
c     For each request time and burn time
      do i=1,nrequest_times+nburns
c
c     call orbanl to propagate to the next time (a.1 offset)
         secjul     = secjul0 + times(i)
         call ddiv(quot,secjul,secjul,86400.d0,43200.d0)
         dayjul     = dayjul0 + quot
         leapseconds= aldiff(dayjul,secjul) - aldiff(dayjul0,secjul0)
         obstim     = times(i) + leapseconds
         call orbanl ( pos,vel,oscelm,avrelm,avrate,
     2        pvdrv,avrdrv,endorb,obstim-offset )
c
c     If time was a burn, do a burn and restart propagator
c     at the burn time
         if (burn_logical(i)) then
c
c     extract burn parameters from the burn list
               burn_delta_x        = burn_list(1,burn_cntr) / 1000.0d0
               burn_delta_y        = burn_list(2,burn_cntr) / 1000.0d0
               burn_delta_z        = burn_list(3,burn_cntr) / 1000.0d0
c
c     call impulsive_burn_propagator to add the delta_v to the averaged
c     elements
               call impulsive_burn_propagator(burn_delta_x,
     1              burn_delta_y,
     2              burn_delta_z, avrelm)
c
c     call calpak for utc calendar time
               secjul     = secjul0 + obstim - leapseconds
               call   ddiv (quot,secjul,secjul,86400.d0,43200.d0)
               dayjul     = dayjul0 + quot
               leapseconds= aldiff(dayjul,secjul) - aldiff(dayjul0,secjul0)
               call calpak (ymd,hms,dayjul0,secjul0+obstim-leapseconds)
c
c
c     Call intanl to reinitialize force models at utc time
c     and reset propagator epoch to burn time
               call intanl (avrelm,rfactor,equin,mtod,mean,ymd,hms)
c
c     Call beganl to restart the semianalytic integrator
               call beganl (forward)
c
c     Set integrator time to time at end of burn, as we just restarted
c     it.
               offset = obstim
c
c     Add one to the burn counter
               burn_cntr = burn_cntr + 1

c     End the work for a burn, return to next time
            end if
c
c     if time was a request time store state
            if (.not.burn_logical(i)) then
c
c     Call kepeqn to obtain classical elements at request time
               call kepeqn (avrkep,avrelm,rfactor)
c
```

```
c     Call calpak for utc calendar time to output back to the user
            secjul     = times(i)
            call   ddiv (quot,secjul,secjul,86400.d0,43200.d0)
            dayjul     = dayjul0 + quot
            leapseconds= aldiff(dayjul,secjul) - aldiff(dayjul0,secjul0)
            call calpak (ymd,hms,dayjul0,secjul0+obstim-leapseconds)
c
c     Evaluate the GHA angle
            gha=calc_gha(ymd,hms)
c
c     Rotate to ecef coordinates
            ecefR(1) = cos(gha)*pos(1) + sin(gha)*pos(2)
            ecefR(2) = -1.0*sin(gha)*pos(1) + cos(gha)*pos(2)
            ecefR(3) = pos(3)
c
c     Output the data
c
c             data(1,data_cntr) = avrkep(1)
c             data(2,data_cntr) = avrkep(2)
c             data(3,data_cntr) = avrkep(3)
c             data(4,data_cntr) = avrkep(4)
c             data(5,data_cntr) = avrkep(5)
c           data(6,data_cntr) = avrkep(6)
c             data(7,data_cntr) = ymd
c           data(8,data_cntr) = hms
c
       data(1,data_cntr) = ecefR(1)
       data(2,data_cntr) = ecefR(2)
       data(3,data_cntr) = ecefR(3)
c
       if (satno.eq.1) then
           ymd_vec(data_cntr) = ymd
           hms_vec(data_cntr) = hms
        end if
c
            data_cntr = data_cntr + 1
c
c     End the request time option
         end if
c
c     Return to propagate to the next time
      end do
c
c     Send the data to the data file
      call outdat(satno,data,nrequest_times,NDATA_ITEMS,outfile)
c
c     Write out the time information
      if (satno.eq.1) then
         open(unit=37,file='ymdhms',status='unknown')
         do i=1,data_cntr
           write(37,'(2f25.16)')ymd_vec(i),hms_vec(i)
        end do
        close(37)
      end if
c
c     mark buffer undefined
 999  buffername = 'uninitialized'
c
c     return with error status
      text = 'i/o error in orbit_propagator_services. status = '
      write (*,'(i4)')   status
      return
c
      end
```

B.2.2 *Subroutine sat_opt.F*

```
#include  "machine.h"
#include  "array_sizes.h"
#define    MAXDELTANOD 0.43630d0
#define    MAXDELTAARG 0.52360d0
#define    MAXDELTAECC 0.00070d0
#define    SSRATE      0.0172027910d0
#define    TWOPI       6.2831853070d0
#define    PIE         3.14159270d0
c-----------------------------------------------------------------------c
c
c     subroutine sat_opt   - propagates the satellite described
c                            in the argument list using the DSST.
c
c     This subroutine invokes Draper semianalytic satellite theory
c     to provide satellite precision mean elements and element rates
c     at user request intervals. All input is through the argument
c     list.  Output is through the argument list.
c
c     Jan 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
c
c  Include files / This file requiures preprocessing
c
c  machine.h includes the machine specific definitions
c  maxArrays.h includes the maximum array sizes
c
c
      subroutine sat_opt (satno, eltype, nintervals, intervals, nburns,
     2     burn_list,lc_satopt_int,lc_satopt_dbl,outfile,indata_path,
     3     optval)
c
c  ---------------------------------------------------------------------
c     Argument list definitions
c
c          name           i/o  meaning
c
c          satno           i   satellite identification number
c          nintervals      i   number of intervals in interval array
c          intervals(5,*)  i   array containing times and deltas
c                            intervals(begining:  yyyymmdd(1,*), hhmmss(2,*)
c                            end: yyyymmdd(3,*), hhmmss(4,*), deltat(5,*))
c          nburns          i   number of burns in the burn array
c          burn_list(4,*)  i   array containing impulsive burns deltas
c          satopt_int(*)   i   an array of integer values for satellite
c                              propagation global to the constellation
c          satopt_dbl(*)   i   an array of double precision values
c                              for satellite propagation global to the
c                              constellation
c          burn_list (1,j) i   time of impulsive velocity manuever
c          burn_list (2,j) i   delta velocity in the x (r of rtn) direction
c          burn_list (3,j) i   delta velocity in the y (t of rtn) direction
c          burn_list (4,j) i   delta velocity in the z (n of rtn) direction
c  ---------------------------------------------------------------------
c
c          iatmos_preburn  i   selector for preburn drag modification
c                              -1 => no drag,       0 => overestimate drag
c                              +1 => nominal drag, +2 => underestimate drag
```

```
c          iatmos_postburn i   selector for postburn drag modification
c                              -1 => no drag,       0 => overestimate drag
c                              +1 => nominal drag, +2 => underestimate drag
c          rho_one_hi      i   drag modification: overestimation  percentage
c          rho_one_low     i   drag modification: underestimation percentage
c          epoch_ymd       o   epoch of mean elements file (yymmdd.)
c          epoch_hms       o   epoch of mean elements file (hhmmss.ssss)
c
c  ---------------------------------------------------------------------
c
c     subroutines called ============================================
c
c     intanl          - helps initialize draper semianalytic theory
c     beganl          - starts the draper semianalytic theory
c     orbanl          - propagates using draper semianalytic theory
c     kepeqn          - makes kepler elements from equinoctials
c     julpak          - converts packed calendar time to julian date
c     calpak          - converts julian date to packed calendar time
c
c     impulsive_burn_propagator - propagates with an
c                                 impulsive burn model
c
c     aldiff          - computes the time difference a.1 - utc
c     ddiv            - division with remainder
c     read_epot       - read the potential model matching the number
c                       used in the pme file
c
c
c     data types ====================================================
c
c     no implicit types
      implicit     none
c
c     Character Variables ===========================================
c
      character * (60)  filename
      character * (18)  buffername          /  'uninitialized'  /
      character * (72)  text
      character * (1)   blank               /  ' '              /
      character * (1)   comment             /  'c'              /
      character * (*)   outfile, indata_path

      integer*4          satopt_int(INT_OPT_SIZE),lc_satopt_int(INT_OPT_SIZE)
      integer*4     burn_cntr,       data_cntr,        i
      integer*4     nburns,          nintervals,       nrequest_times
      integer*4     satno,           eltype
      integer*4     equin,           mtod,             mean
      integer*4     status,          unitnum,          k
c
      logical*4     unit_unavailable
      logical*4     burn_logical(MAX_NUM_TIMES)
      logical*4     setrtr
c
      real*8        intervals(5,*)
      real*8        satopt_dbl(REAL_OPT_SIZE)
      real*8        lc_satopt_dbl(REAL_OPT_SIZE)
      real*8        request_times(MAX_NUM_TIMES)
      real*8        times(MAX_NUM_TIMES)
      real*8        burn_times(MAX_NUM_BURNS)
      real*8        burn_list(4,*)
      real*8        kepler(6)

      real*8        elmint(6),       ymdint,           hmsint
      real*8        pos(3),          vel(3),           oscelm(6)
```

```
      real*8        avrelm(6),      pvdrv(6,300),     avrdrv(6,300)
      real*8        endorb,         avrkep(6),        avrate(6)
      real*8        epoch_date,     epoch_tod,        obstim
      real*8        offset
      real*8        burn_delta_x,   burn_delta_y,     burn_delta_z
      real*8        dayjul0,        secjul0,          ymd
      real*8        hms,            dayjul,           secjul
      real*8        leapseconds,    quot
      real*8        aldiff
      real*8        radians,        degrees,          infinity
      real*8        rfactor,        forward
      real*8        optval
      real*8        ideal,          noddev
c
c     constants ===================================================
c
      parameter  (  radians       = 57.295779513082321    d00 )
      parameter  (  degrees       =  0.017453292519943296 d00 )
      parameter  (  infinity      = 99999999.             d20 )
      parameter  (  forward       = 1. d0    )
      parameter  (  equin         = 3        )
      parameter  (  mtod          = 2        )
      parameter  (  mean          = 2        )


c
c     FORTRAN include modules =====================================
c
c     include the satellite epoch data buffer
      include 'PMERN.CMN'
c
c     BEGIN PROGRAM ===============================================
c
c     Copy argument list into local variables
       do i=1,REAL_OPT_SIZE
          satopt_dbl(i)= lc_satopt_dbl(i)
        end do

       do i=1,INT_OPT_SIZE
          satopt_int(i)= lc_satopt_int(i)
        end do

c     Convert to equinoctial elements
      if (eltype.eq.1) then
         rfactor = satopt_int(1)
         setrtr  = .false.
         kepler(1)     = satopt_dbl(3)
         kepler(2)     = satopt_dbl(4)
         kepler(3)     = satopt_dbl(5) /radians !Convert to radians
         kepler(4)     = satopt_dbl(6) /radians
         kepler(5)     = satopt_dbl(7) /radians
         kepler(6)     = satopt_dbl(8) /radians
         call eqnkep(elmint,rfactor,kepler,setrtr)
         satopt_dbl(3) = elmint(1)
         satopt_dbl(4) = elmint(2)
         satopt_dbl(5) = elmint(3)
         satopt_dbl(6) = elmint(4)
         satopt_dbl(7) = elmint(5)
         satopt_dbl(8) = elmint(6) / degrees    !Convert to degrees
      end if
c
c     Call the routine which will take the options input in the argument
c     list and put them in the PMERN common area and read in the
c     potential field
```

```
      call set_satopt(satopt_dbl, satopt_int, status)

c     setup satellite at epoch
      pme_cd      =  pme_cd * ( 1.d0 + pme_rho_one )
      pme_rho_one =  0.d0
c
c     find the first available unit
      unitnum  = 20
      unit_unavailable = .true.
      do while  ( unit_unavailable )
         unitnum = unitnum + 1
         inquire ( unit=unitnum, opened=unit_unavailable,
     2        iostat=status)
      end do
c
c     open the gravity field file
      k = 1
      do while(indata_path(k:k).ne.' ')
      k = k+1
      end do
      filename = indata_path(1:k-1)//'epotfld'
      open (unit=unitnum, file=filename, status='old',
     2     form='unformatted', access='direct',
     3     recl = 1050*WORDLENGTH,IOSTAT=k)
c
c     Call read_epot to read new gravity model and update common
      call read_epot(unitnum,status)
      if   ( status .ne. 0 )  then
       print *,'Error in opening epotfld'
       stop
      end if
c
c     Close the input earth file
      close(unitnum)

c     Extract epoch from the buffer and adjust the century
      epoch_date = pme_date
      epoch_tod  = pme_time
      ymdint     = epoch_date-19000000.d0
      hmsint     = epoch_tod
c
c     Call julpak to obtain julian date at epoch
      call julpak (dayjul0,secjul0,ymdint,hmsint)
c
c     Extract epoch equinoctial elements from the buffer
      elmint (1)  = pme_els_equin (1)
      elmint (2)  = pme_els_equin (2)
      elmint (3)  = pme_els_equin (3)
      elmint (4)  = pme_els_equin (4)
      elmint (5)  = pme_els_equin (5)
      elmint (6)  = pme_els_equin (6) * degrees
      rfactor     = pme_retro
c
c     Call intanl to initialize force models
      call intanl (elmint,rfactor,equin,mtod,mean,ymdint,hmsint)
c
c     Call beganl to start the semianalytic integrator
      call beganl (forward)
c
c     Set integrator time to zero
      offset          = 0. d0
c
c     Make a list of request times (in seconds from epoch)
c     Check all request times to insure they come after the epoch
```

```
      call crrequest_times(dayjul0, secjul0, intervals, nintervals,
     1     request_times, nrequest_times, status)
c
c     Assign all the burn times to a vector
      do i=1,nburns
         burn_times(i)=burn_list(4,i)
      end do
c
c     Sort the request times & burn times together
      call sort_times(burn_times, nburns, request_times,
     $     nrequest_times,times, burn_logical, nrequest_times+nburns,
     $     status)
c
c     Create a separate cntr to keep track of burns and amount of output
      burn_cntr = 1
      data_cntr = 1
c
c     For each request time and burn time
      do i=1,nrequest_times+nburns
c
c     call orbanl to propagate to the next time (a.1 offset)
         secjul     = secjul0 + times(i)
         call ddiv(quot,secjul,secjul,86400.d0,43200.d0)
         dayjul     = dayjul0 + quot
         leapseconds= aldiff(dayjul,secjul) - aldiff(dayjul0,secjul0)
         obstim     = times(i) + leapseconds
         call orbanl ( pos,vel,oscelm,avrelm,avrate,
     2        pvdrv,avrdrv,endorb,obstim-offset )
c
c     If time was a burn, do a burn and restart propagator
c     at the burn time
         if (burn_logical(i)) then
c
c     extract burn parameters from the burn list
            burn_delta_x        = burn_list(1,burn_cntr) / 1000.0d0
            burn_delta_y        = burn_list(2,burn_cntr) / 1000.0d0
            burn_delta_z        = burn_list(3,burn_cntr) / 1000.0d0
c
c     call impulsive_burn_propagator to add the delta_v to the averaged
c     elements
            call impulsive_burn_propagator(burn_delta_x,
     1           burn_delta_y,
     2           burn_delta_z, avrelm)
c
c     call calpak for utc calendar time
            secjul     = secjul0 + obstim - leapseconds
            call   ddiv (quot,secjul,secjul,86400.d0,43200.d0)
            dayjul     = dayjul0 + quot
            leapseconds= aldiff(dayjul,secjul) - aldiff(dayjul0,secjul0)
            call calpak (ymd,hms,dayjul0,secjul0+obstim-leapseconds)
c
c
c     Call intanl to reinitialize force models at utc time
c     and reset propagator epoch to burn time
            call intanl (avrelm,rfactor,equin,mtod,mean,ymd,hms)
c
c     Call beganl to restart the semianalytic integrator
            call beganl (forward)
c
c     Set integrator time to time at end of burn, as we just restarted
c     it.
            offset = obstim
c
c     Add one to the burn counter
```

```
            burn_cntr = burn_cntr + 1

c     End the work for a burn, return to next time
         end if
c
c     if time was a request time store state
         if (.not.burn_logical(i)) then
c
c     Call kepeqn to obtain classical elements at request time
            call kepeqn (avrkep,avrelm,rfactor)
c
c     Call calpak for utc calendar time to output back to the user
            secjul     = secjul0 + obstim - leapseconds
            call   ddiv (quot,secjul,secjul,86400.d0,43200.d0)
            dayjul     = dayjul0 + quot
            leapseconds= aldiff(dayjul,secjul)
     &           - aldiff(dayjul0,secjul0)
            call calpak (ymd,hms,dayjul0,secjul0+obstim-leapseconds)
c
c     Record the total change in eccentricity
c
c       Calculate the ideal sunsync value
         ideal = 0.0d0
         ideal = kepler(4)+(SSRATE)*((dayjul-dayjul0)+
     &           (secjul-secjul0)/86400.0d0)
         ideal = dmod(ideal,TWOPI)
c
c       Calculate the deviation
         noddev = abs(ideal-avrkep(4))
         if (noddev.gt.PIE) then
           noddev = TWOPI - noddev
         end if

c        Evaluate the cost function
c         optval = abs(avrkep(2)-kepler(2))/MAXDELTAECC
c     &        +   abs(avrkep(5)-kepler(5))/MAXDELTAARG
c     &        +   3*noddev/MAXDELTANOD
c     &        +   optval
c
c       Evaluate the cost function
         optval = noddev/MAXDELTANOD + optval
c        Evaluate the cost function
c          optval = abs(avrkep(3)-kepler(3))
c
c     End the request time option
         end if
c
c     Return to propagate to the next time
      end do
c

      return
c
      end
```

B.2.3 *Subroutine set_satopt.F*

```
      subroutine set_satopt(satopt_dbl, satopt_int, status)
c
c-----------------------------------------------------------------------c
c
c     subroutine set_satopt    - Sets the values in common area PMERN
c                                to run the satellite propagator
c
c     set_satopt is necessary to keep all the options in two arrays
c     in higher level programs
c
c     Jan 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
c
      real*8    satopt_dbl(*)
      integer*4 satopt_int(*)
      integer*4 status
c
      implicit none
c
c
      include 'PMERN.CMN'

c-----------------------------------------------------------------------c
      pme_date              = satopt_dbl(1)
      pme_time              = satopt_dbl(2)
      pme_els_equin(1)      = satopt_dbl(3)
      pme_els_equin(2)      = satopt_dbl(4)
      pme_els_equin(3)      = satopt_dbl(5)
      pme_els_equin(4)      = satopt_dbl(6)
      pme_els_equin(5)      = satopt_dbl(7)
      pme_els_equin(6)      = satopt_dbl(8)
      pme_cd                = satopt_dbl(9)
      pme_rho_one           = satopt_dbl(10)
      pme_scmass            = satopt_dbl(11)
      pme_scarea            = satopt_dbl(12)
      pme_stepsize          = satopt_dbl(13)
c
      pme_retro             = satopt_int(1)
      pme_atmos_model       = satopt_int(2)
      pme_potential_model   = satopt_int(3)
      pme_nmax              = satopt_int(4)
      pme_mmax              = satopt_int(5)
      pme_izonal            = satopt_int(6)
      pme_ij2j2             = satopt_int(7)
      pme_nmaxrs            = satopt_int(8)
      pme_mmaxrs            = satopt_int(9)
      pme_ithird            = satopt_int(10)
      pme_inddrg            = satopt_int(11)
      pme_iszak             = satopt_int(12)
      pme_indsol            = satopt_int(13)
c
      return
      end
```

B.2.4 *Subroutine crrequest_times.F*

```
      subroutine crrequest_times(dayjul0, secjul0, intervals,
     $     nintervals, request_times, nrequest_times, status)
c
c-----------------------------------------------------------------------c
c
c     subroutine crrequest_times - Create Request times
c
c     This subroutine creates the request times in seconds from
c     epoch from the intervals given in the intervals argument.
c
c     Jan 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
c
      real*8 dayjul0
      real*8 secjul0
      real*8 intervals(5,*)
      real*8 request_times(*)
      real*8 current_time
      real*8 daybeg, dayend
      real*8 secbeg, secend
      real*8 deltat
      real*8 begint_sec, endint_sec
      real*8 day_seconds
c
      integer          nintervals
      integer          nrequest_times
      integer          status
      integer          i
      integer          time_cntr
c
      parameter (day_seconds = 86400.0)
c
      implicit none
c
c-----------------------------------------------------------------------c
c


c
c     Initialize variables
      time_cntr = 1
      nrequest_times=0
c
c     For each interval
      do i=1,nintervals

c     Call julpak to obtain julian date at interval beginning and end
         call julpak (daybeg, secbeg, intervals(1,i)-19000000.0D0,
     $        intervals(2,i))
         call julpak (dayend, secend, intervals(3,i)-19000000.0D0,
     $        intervals(4,i))

         deltat = intervals(5,i)

         begint_sec = (daybeg-dayjul0)*DAY_SECONDS + secbeg-secjul0
         endint_sec = (dayend-dayjul0)*DAY_SECONDS + secend-secjul0

         current_time = begint_sec
```

```
   do while (current_time.lt.endint_sec)
      request_times(time_cntr)=current_time
      current_time = current_time + deltat
      nrequest_times=time_cntr
      time_cntr = time_cntr+1
   end do
end do
end
```

B.2.5 *Subroutine sort_times.F*

```
#ifdef DEBUG
#define NATEST 3
#define NBTEST 2
#define NTESTTIMES (NATEST+NBTEST)
      program test_times

      integer nra,    nrb,    nout, i, status
c
      real*8  ra(NATEST),  rb(NBTEST),  rout(NTESTTIMES)
c
      logical lout(NTESTTIMES)
c
      implicit none
c
      nra = NATEST
      nrb = NBTEST
      nout = nra+nrb

      do i = 1,NATEST
         ra(i)=i*i
      end do

      do i = 1,NBTEST
         rb(i)=i*i*i - i
      end do


      call sort_times(ra, nra, rb, nrb, rout, lout,
     $     nout, status)

      end

#endif
      subroutine sort_times(ra, nra, rb, nrb, rout, lout,
     $     nout, status)
c-----------------------------------------------------------------------c
c
c     subroutine sort_times   - Puts two arrays into one long array
c     sorts the long array along with a logical array describing where
c     the array came from (TRUE if first array, FALSE if second)
c
c     JAN 95
c
c     Scott T Wallace, Lt, USAF
c     MIT / Aero Astro Dept/ Draper Fellow
c-----------------------------------------------------------------------c
c
      integer nra, nrb,    nout, i, status

      real*8  ra(*),  rb(*),  rout(nout)

      logical lout(nout)
c
c-----------------------------------------------------------------------c
c
      implicit none
c
      do i = 1,nra
         rout(i) = ra(i)
         lout(i) = .TRUE.
      end do
```

```
      do i = nra+1,nout
         rout(i) = rb(i-nra)
         lout(i) = .FALSE.
      end do

      call sort2(nout, rout, lout)

      return
      end

c-----------------------------------------------------------------------c

      SUBROUTINE SORT2(N,RA,RB)

      integer n,l,ir,i,j
      real*8 ra,rra
      logical rb,rrb

      implicit none

      DIMENSION RA(N),RB(N)
      L=N/2+1
      IR=N
 10   CONTINUE
      IF(L.GT.1)THEN
         L=L-1
         RRA=RA(L)
         RRB=RB(L)
      ELSE
          RRA=RA(IR)
          RRB=RB(IR)
          RA(IR)=RA(1)
          RB(IR)=RB(1)
          IR=IR-1
          IF(IR.EQ.1)THEN
            RA(1)=RRA
            RB(1)=RRB
            RETURN
          ENDIF
        ENDIF
        I=L
        J=L+L
20      IF(J.LE.IR)THEN
          IF(J.LT.IR)THEN
            IF(RA(J).LT.RA(J+1))J=J+1
          ENDIF
          IF(RRA.LT.RA(J))THEN
            RA(I)=RA(J)
            RB(I)=RB(J)
            I=J
            J=J+J
          ELSE
            J=IR+1
          ENDIF
        GO TO 20
        ENDIF
        RA(I)=RRA
        RB(I)=RRB
      GO TO 10
      END

c-----------------------------------------------------------------------c
```

B.3 Example PVM/DSST Input File

```
N Satellites:   21   ElType  1

nintervals:     1
Begin interval  1  19950401.0  000000.0
End   interval  1  20050401.0  0.00
Deltat interval 1  432000.0

nburns =        0

-----------------------------------------
Satellite Number:   1  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7073140000000000D+04
                                0.1180000000000000D-02
                                0.9814200000000000D+02
                                0.0000000000000000D+00
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:       21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:     21  Ithird:      1
Ind Drg:     2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   2  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7073640000000000D+04
                                0.1180000000000000D-02
                                0.9814400000000000D+02
                                0.9500000000000000D+01
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:       21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:     21  Ithird       1
Ind Drg:     2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   3  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7074140000000000D+04
                                0.1180000000000000D-02
                                0.9814600000000000D+02
                                0.1900000000000000D+02
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000
```

```
Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   4  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7074640000000000D+04
                               0.1180000000000000D-02
                               0.9814800000000000D+02
                               0.2850000000000000D+02
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:         0.00000000
S/C Mass:          800.00000000 S/C Area:        0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   5  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7075140000000000D+04
                               0.1180000000000000D-02
                               0.9815000000000000D+02
                               0.3800000000000000D+02
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:         0.00000000
S C Mass:          800.00000000 S/C Area:        0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   6  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7075640000000000D+04
                               0.1180000000000000D-02
                               0.9815200000000000D+02
                               0.4750000000000000D+02
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:         0.00000000
S/C Mass:          800.00000000 S/C Area:        0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
```

```
Satellite Number:   7  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7076140000000000D+04
                                0.1180000000000000D-02
                                0.9815400000000000D+02
                                0.5700000000000000D+02
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:       21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:     21  Ithird:      1
Ind Drg:     2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   8  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7076640000000000D+04
                                0.1180000000000000D-02
                                0.9815600000000000D+02
                                0.6650000000000000D+02
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:       21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:     21  Ithird:      1
Ind Drg:     2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:   9  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7077140000000000D+04
                                0.1180000000000000D-02
                                0.9815800000000000D+02
                                0.7600000000000000D+02
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:       21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:     21  Ithird:      1
Ind Drg:     2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  10  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7077640000000000D+04
                                0.1180000000000000D-02
                                0.9816000000000000D+02
                                0.8550000000000000D+02
                                0.9000000000000000D+02
```

```
                              0.0000000000000000D+00

CD:                  0.00000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:    1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  11  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7078140000000000D+04
                               0.1180000000000000D-02
                               0.9816200000000000D+02
                               0.9500000000000000D+02
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:    1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  12  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7078640000000000D+04
                               0.1180000000000000D-02
                               0.9816400000000000D+02
                               0.1045000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:    1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  13  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7079140000000000D+04
                               0.1180000000000000D-02
                               0.9816600000000000D+02
                               0.1140000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
```

```
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  14  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7079640000000000D+04
                               0.1180000000000000D-02
                               0.9816800000000000D+02
                               0.1235000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  15  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7080140000000000D+04
                               0.1180000000000000D-02
                               0.9817000000000000D+02
                               0.1330000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  16  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :         0.7080640000000000D+04
                               0.1180000000000000D-02
                               0.9817200000000000D+02
                               0.1425000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:      1 Atmos Mdl:   1  Potent Mdl:  4
Nmax:      21 Mmax:       21  Izonal:      1  IJ2J2:   1
Nmaxrs:    21 Mmaxrs:     21  Ithird:      1
Ind Drg:    2 Iszak:       2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  17  Epoch Date: 19950401.0  Epoch Time:   0.00
```

```
Keplerian   Elements :          0.7081140000000000D+04
                               0.1180000000000000D-02
                               0.9817400000000000D+02
                               0.1520000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  18  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7081640000000000D+04
                               0.1180000000000000D-02
                               0.9817600000000000D+02
                               0.1615000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  19  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7082140000000000D+04
                               0.1180000000000000D-02
                               0.9817800000000000D+02
                               0.1710000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00

CD:                  2.20000000 Rho One:          0.00000000
S/C Mass:          800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:      1  IJ2J2:   1
Nmaxrs:     21 Mmaxrs:      21  Ithird:      1
Ind Drg:     2 Iszak:        2  Ind Sol:     1
-----------------------------------------
-----------------------------------------
Satellite Number:  20  Epoch Date: 19950401.0  Epoch Time:   0.00

Keplerian   Elements :          0.7082640000000000D+04
                               0.1180000000000000D-02
                               0.9818000000000000D+02
                               0.1805000000000000D+03
                               0.9000000000000000D+02
                               0.0000000000000000D+00
```

```
CD:                   2.20000000 Rho One:          0.00000000
S/C Mass:           800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:       1  IJ2J2:    1
Nmaxrs:     21 Mmaxrs:      21  Ithird:       1
Ind Drg:     2 Iszak:        2  Ind Sol:      1
-----------------------------------------
-----------------------------------------
Satellite Number:  21  Epoch Date: 19950401.0  Epoch Time:    0.00

Keplerian   Elements :          0.7083140000000000D+04
                                0.1180000000000000D-02
                                0.9818200000000000D+02
                                0.1900000000000000D+03
                                0.9000000000000000D+02
                                0.0000000000000000D+00

CD:                   2.20000000 Rho One:          0.00000000
S/C Mass:           800.00000000 S/C Area:         0.00014400
Integrator Step: 43200.00000000

Retro:       1 Atmos Mdl:    1  Potent Mdl:  4
Nmax:       21 Mmax:        21  Izonal:       1  IJ2J2:    1
Nmaxrs:     21 Mmaxrs:      21  Ithird:       1
Ind Drg:     2 Iszak:        2  Ind Sol:      1
-----------------------------------------
```

Appendix C: Data Files

This appendix documents the data files that were used for each of the program executions performed in conjunction with this thesis.

All the tests using the PVM/DSST used data files are stored in the Continuus Configuration Management system. Instructions on the use of this system can be found in [73]. The data files used for the PVM/DSST can be found in:
Database: satUtil_db
Project: BSD,1.1

C.1 Software Validation Tests

C.1.1 Comparison to Orbit_Propagator_Services (OPS)

Table B-1: Data Files used for OPS to PVM/DSST Comparison

epotfld	radarsat_earthfld.dat
jacdat	jacchia.data_sun
slp1950	de96_slp1950.dat
slptod	de96_slptod.dat
timecoef	de96 _timcoef.dat
newcomb	N/A[1]

C.1.2 Comparison to GTDS

Table B-2: Data Files used for GTDS to PVM/DSST Comparison

epotfld	radarsat_earthfld.dat
jacdat	jr_schatten_nom.dat
slp1950	orbit.slp.mn1950.dat
slptod	orbit.slp.tod1950.dat
timecoef	orbit.slp.timcof.dat
newcomb	N/A

[1] This file was not needed for any of these tests.

C.2 Performance Analysis

Table B-3: Data Files used for Performance Analysis

epotfld	radarsat_earthfld.dat
jacdat	jr_schatten_nom.dat
slp1950	orbit.slp.mn1950.dat
slptod	orbit.slp.tod1950.dat
timecoef	orbit.slp.timcof.dat
newcomb	N/A

C.3 Teledesic Analysis

Table B-4: Data Files used in the Teledesic Analysis

epotfld	radarsat_earthfld.dat
jacdat	jr_schatten_nom.dat
slp1950	orbit.slp.mn1950.dat
slptod	orbit.slp.tod1950.dat
timecoef	orbit.slp.timcof.dat
newcomb	N/A

Appendix D: Using the PVM/DSST

This Appendix provides a description of the software, how to access the current version and how to execute it from the Draper Laboratory environment.

Section D.1 describes the different executables currently built from the software. Section D.2 describes the input files for the various executables. Section D.3 details test case execution of the software.

D.1 Executable Description

The software is currently written to generate the executables described in table D-1.

Table D-1: Executable Description

Executable Name	Description
test_sat_prop	Executes the DSST for one satellite. The orbit and satellite are hard-coded in the file 'test_sat_prop.F'.
const_prop	Propagates the satellites described in the input file. Prompts user for input file. Requires environment variable CONST_INPUT be set to the directory with the links to the input files. The output, ECEF positions, are written in the CONST_OUTPUT (also an environment variable) to the files satdata?, where ? is the satellite number.
const_prop_kep	Same as *const_prop* except the output is written in Keplerian elements.
ga32	Executes the genetic algorithm optimization software. Currently set to find the best frozen orbit (minimize changes from initial eccentricity). Input data files must be located in the CONST_INPUT directory. The environment variable OPT_FILE describes the name of the input file. GA output files are put in the current working directory. Uses the *const_opt_slave* executable to perform propagation. The cost function is located in sat_opt.F. To change the number of modifiable parameters, (currently set to one) the following changes must be made: declare.inc (GAOPT project) : Set *mxalfa* and *mxcont* to the number of parameters and recompile the software. const_opt.F (DSST_SHELL) : The values sent to the slave task must contain the values passed in through the GA software.
const_opt_slave	Used by *ga32* to evaluate the cost functions. Must be a spawned process as the required input be sent via PVM.

D.2 Input File Description

D.2.1 const_prop Input Files

The following data files are necessary to execute the propagator.

Table D-2: Data Files Description

Name of File	Description
epotfld	Earth potential models file.
jacdat	Jacchia data for drag information.
slp1950	Solar, Lunar, Planetary ephemeris file in Mean of 1950 coordinates.
slptod	Same as above in GTDS true-of-date coordinates.
timecoef	Timing coefficients file.
newcomb	Newcomb operators file.

The operator is prompted for the orbit input file or the OPT_FILE environment variable is used (see Table D-1).

The output is controlled by the CONST_OUTPUT environment variable.

D.2.2 Genetic Algorithm (GA) Input Files

The environment variable OPT_FILE describes the input file. The input path is given by CONST_INPUT. This variable also describes the location of the data files.

The GA requires the file 'dome.in' to be in the current working directory (CWD). A typical 'dome.in' file is shown in figure D-1 and explained in table D-3.

```
:  .t the most frozen eccentricity
                              itest
. 11. . 7                     iopt,maxitr,epsiln
11961 5..1.                   kseed,mpopsize,ncomp
1 0.1 1 0.2.0.                Opts:  constr,clones,Popt,Ropt,Topt,ishr
1                             fixed parameters
1                             continuous parameters
1 0.0                         it chooses initial conditions
0.001000.                     min of continuous
0.001200.                     max of continuous
1                             discrete parameters  1 failure rates
4                             number of bins for each discrete parameter
1                             initial discrete (ga: param# ie. #1
 0.1165 .001171,.001173,.001175,
```

Figure D-1: 'dome.in' Input File

Table D-3: 'dome.in' Description[2]

Parameter	Description
iopt	Optimization method. 9-Traditional GA 10- 'Improved' GA [64]
maxitr	Maximum number of iterations
epsiln	Convergence tolerance, where convergence describes how sure the GA is of the answer. Typical values range from 0.1, 0.9 (0<epsiln<1). 0 - Easy to converge 1 - Difficult to converge
kseed	Random number seed.
mpopsize	Population size.
continuous parameters	Number of continuous parameters. Discrete parameters are parameters for which only specific values can be chosen.
it chooses...	A zero followed by a comma is needed for every parameter.
min of continuous max of continuous	Parameter ranges.

D.3 Executing the PVM/DSST

This Section describes how to access the software developed for this thesis. The user is assumed to have access to the Continuus Configuration Management Tool (CCM), MATLAB, and be working within the BASH shell. In addition, to execute the entire test suite without repeating sections, all commands must be executed on the same type of computer.

The following convention will be used in the next three sections:

- The operator is the individual running the tests.
- The symbol . . . indicates there will output coming from the computer that was not listed in this document.
- The > symbol was the prompt in the environment used to generate the tests.
- `Courier font represents text taken directly off the computer screen.`

[2]Only the parameters described in Table D-3 were used. Other parameters did not need to be modified. Information concerning these parameters can be found in [65]

- **Bold courier describes information that must be entered exactly as shown.**

D.3.1 Environment Setup

Before executing the software, the operator will need to copy two setup files into their home directory. These files are automatically executed at login and will create the environment for the rest of the tests.

If these files already exist in the operator's home directory, they should be renamed to a different file before continuing; otherwise they will be overwritten.

The first commands shown copy the necessary files into the operator's home directory.

```
>cp  /Users/taz/scott/.ccmdefaults   .
>cp  /Users/taz/scott/.UserLogin     .
```

The operator should now completely logoff and then log in to the computer.

D.3.2 Building PVM

If the operator does not have PVM installed, it must be installed and built as described in this section before continuing. The parallel virtual machine is very easy to build. General instructions can be found in [13]. The instructions in this section are specific to the Draper environment.

PVM can be installed by root such that everyone has access to the same *pvm* and *pvmd* executables. However, PVM can also be installed in the operator's home directory, so that root privileges are not required.

PVM, along with many other useful utilities and information, is kept on the lab-wide file server fs1. If PVM is not found on the fs1, it can be obtained over the Internet through anonymous ftp to netlib2.cs.utk.edu.

To get PVM type:

```
>cd
>cp  /nfs/fs1/ftp/source/hpc/pvm/pvm3.3.7.tar.gz  ~/.
```

The environment variable, PVM_ROOT, must be set to before building PVM. If PVM is installed in the operator's home directory, PVM_ROOT is set in the login files copied in Section D.3.1. Otherwise, PVM_ROOT must be set manually.

To build PVM in the operator's home directory type:

```
>cd
>tar  -xzf  pvm3.3.7.tar.gz
>cd  pvm3
>make
...
```

D.3.3 Starting the Configuration Management Tool

CCM projects a copy of its file system into the user's directory using soft links. All the work for this thesis is contained in the satUtil_db database.

A database contains projects and a project contains the software. The software for this thesis was divided into projects as much as practical so that it was easier to work with. Dividing up the original stand alone-code DSST into functional projects would have been desirable but represented a significant effort that was not accomplished as a part of this thesis.

The software for this thesis is divided into the following projects:

Table D-4: Project Descriptions

Project	Brief Description
PDSST-2.0	Highest level project. Contains all the other projects and makefiles.
GAOPT-2.0	Genetic algorithm optimization software.
DSST_SHELL-2.0	Software for performing constellation propagation.
DSST_BASE-2.0	The stand-alone DSST software.
BSD-1.1	Binary data files.

The configuration management tool will be used here without a graphical user interface (GUI). This is done so that the description presented here is complete.

```
>cd
>ccm start -nogui
Starting Continuus/CM...
...
>ccm sync PDSST-2.0
Personal workarea update starting for /Users/taz/scott/ccm_satUtil_db/
...
        Updating /Users/taz/scott/ccm_satUtil_db/PDSST-2.0/...
        Updating /Users/taz/scott/ccm_satUtil_db/BSD-1.1/...
        Updating /Users/taz/scott/ccm_satUtil_db/DSST_BASE-2.0/...
        Updating /Users/taz/scott/ccm_satUtil_db/DSST_SHELL-2.0/...
        Updating /Users/taz/scott/ccm_satUtil_db/GAOPT-2.0/...
Personal workarea update complete.
```

At this point, the projects are projected into the operator's account.

D.3.4 Executing the Software

Two different tests are performed to demonstrate that the software is fully tested. The first test is the serial test case described in Chapter 3. PVM is not used in this test.

D.3.4.1 Serial Test Case

After completing sections D.3.1, D.3.2 and D.3.3 type:

```
>cd
>cd  ccm_satUtil_db/DSST_BASE-2.0/DSST_BASE
>ls
Makefile.aimk  include        test
data_files     source
```

The script *aimk* comes with the PVM distribution. It executes the UNIX *make* facility after creating a directory based on the architecture and operating system of the computer. The object files are placed into this directory, so heterogeneous platforms using a shared disk can safely build the same executable. Note that the SUN4SOL2 in the next line describes the platform used to generate these tests. This will be different dependent on the platform the operator is using.

```
>aimk  test_sat_prop
making in SUN4SOL2/ for SUN4SOL2
...
>export  CONST_INPUT=./test/
```

The next command will run the DSST using the input files described in ./test/ directory. The output file generated, 'test_sat_prop.out', is also placed into the ./test/ directory.

```
>test_sat_prop
   0
>cd test
>matlab
...
>> verif_sat_prop

Your results are :
   1.0e+03 *

   7.07759761564452
   0.00000036439527
   0.09824506769856
   0.00204663721379
   0.14760422493377
   0.17541971016900

>>quit
```

Note that these results match the numbers given in table 3-10.

This completes the serial test case.

D.3.4.2 GA Test Case

This test case executes the genetic algorithm optimization software, set up to find a frozen orbit. This test case executes on two computers. It is assumed that the second computer is a different type according to PVM, so PVM will also be built on the second computer.

```
>cd
>cd  ~/ccm_satUtil_db/PDSST-2.0/PDSST
>export  CONST_INPUT=$HOME/ccm_satUtil_db/PDSST-2.0/PDSST/test/
>aimk all
making in SUN4SOL2/ for SUN4SOL2
...

>rsh porky
Last login: ...
>cd
>cd pvm3
>make
...
>cd  ~/ccm_satUtil_db/PDSST-2.0/PDSST

>aimk all
making in SUN4/ for SUN4
...

>exit
>pvm
pvm> add porky
1 successful
                    HOST     DTID
                   porky    80000
pvm> quit

pvmd still running.
>cd test
>ls
dome.in     loadmats.m  nom_sat.in  opt_sat.in
```

This directory contains the input files necessary to execute the optimization algorithm.

The next commands link the appropriate data files for use by the propagator. The commands each take two lines to describe but should be entered into the computer as a single line.

```
>ln  -s  ~/ccm_satUtil_db/PDSST-2.0/PDSST/BSD/sun_binary_data/
radarsat_earthfld.dat  epotfld
```

```
>ln  -s  ~/ccm_satUtil_db/PDSST-2.0/PDSST/BSD/sun_binary_data/
jr_schatten_nom.dat  jacdat
>ln  -s  ~/ccm_satUtil_db/PDSST-2.0/PDSST/BSD/sun_binary_data/
orbit.slp.mn1950.dat  slp1950
>ln  -s  ~/ccm_satUtil_db/PDSST-2.0/PDSST/BSD/sun_binary_data/
orbit.slp.tod1950.dat  slptod
>ln  -s  ~/ccm_satUtil_db/PDSST-2.0/PDSST/BSD/sun_binary_data/
orbit.slp.timcof.dat  timecoef
>export  OPT_FILE=nom_sat.in
```

The next command starts the optimization process, where the cost function evaluation takes place on two processors.

```
>ga32
                                        0
                                       50
                                       95
...

>more  Dz
  50     50         0 1.64641633E-05 1.17098039E-03 1.53222466E-02
  95     50        11 1.64641633E-05 1.17098039E-03 4.34377119E-02
 139     50        14 1.64641633E-05 1.17098039E-03 8.91743973E-02
```

(The times and dates indicated in the following file are not important)

```
>more  DO
 ****  DOME BEGAN ON 11-May-95 AT 06:28:22    ****
Run ID: Choose most frozen eccentricity

 * Optimization method:    9 *
 Optimization search stopping criterion:          7.0000E-02
 Maximum number of optimization iterations:              250
 Genetic Algorithm:
 population size:          50  random number seed:        20985
 crossover:              0.80  per bit mutation:         0.0040
 markov model states:       1  fixed parameters:              0
 continuous:                1  discrete parameters:           0

 continuous    initial       lower        upper
  variable      value        bound        bound
     1       0.0000E+00   1.0000E-03   1.2000E-03
     cfe#    139        ** stop due to population convergence **
  Parameters reverted to original:        0
  Total cost function evaluations:      139
  Evaluation of minimum value:           50
  Algorithm elapsed time:          100.5320

  Function value                          Parameter values
     1              2              3             4             5
   1.64641633E-05   1.17098039E-03
 ****  DOME TERMINATED ON 11-May-95 AT 06:30:03    ****
```

D.3.4.3 *const_prop* Test Case

This test case propagates the two orbits described in the file 'opt_sat.in'. This file contains the same orbit and satellite information as 'nom_sat.in' for the first satellite. The second satellite is identical except for the eccentricity is the value chosen by the GA execution in section D.3.2. The results, in the form of two MATLAB plots, are output to the screen as well as encapsulated post script files.

```
>const_prop_kep
 /Users/taz/scott/ccm_satUtil_db/PDSST-2.0/PDSST/test/
 Please enter the name of the constellation file:
opt_sat.in
 I sent satellite  1 to taz
 I sent satellite  2 to taz
 I received from taz
 I received from taz
>matlab
...
>>loadmats
...
>>quit
>
```

The plots generated by the **loadmats** command are depicted in figures D-2 and D-3.

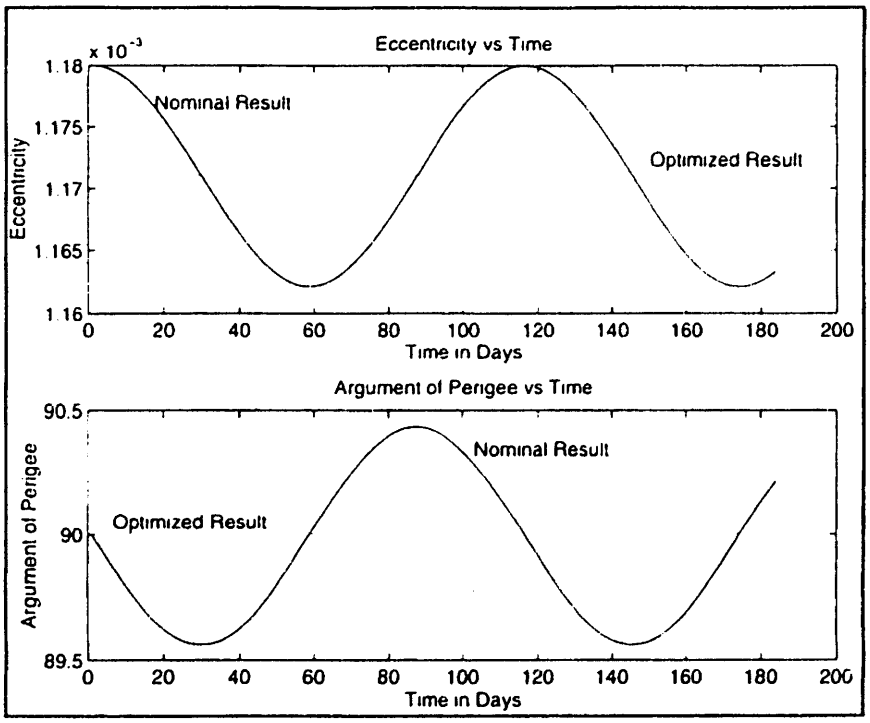


Figure D-2: Nominal vs. Optimized Eccentricity and Argument of Perigee

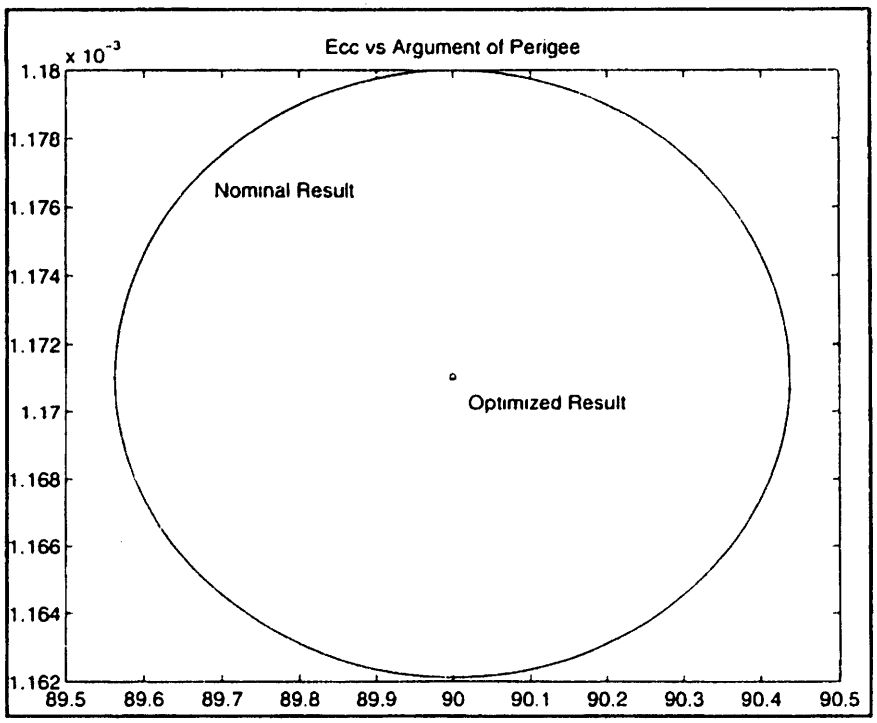


Figure D-3: Argument of Perigee vs. Eccentricity

References

[1] Thurston, Robin G. "The State of the Space Catalog." *Proceedings of the Air Force Space Command Space Control Workshop* held in Colorado Springs. 23-25 August 1994.

[2] Gordon, Gary D. *Principles of Communication Satellites*. New York: John Wiley and Sons, Inc. 1993.

[3] Agrawal, Brij N. *Design of Geosynchronous Spacecraft*. Washington D.C.: Prentice-Hall, Inc. 1986.

[4] Roy, A. E. *Orbital Motion*. New York: Adam Hilger. 1991.

[5] Battin, Richard H. *An Introduction to the Mathematics and Methods of Astrodynamics*. New York: AIAA Education Series. 1987.

[6] König-Lopex, Orly. "Are We Trashing the Heavens." *Via Satellite*. January 1995. p. 32-39.

[7] Kreyzig, Erwin. *Advanced Engineering Mathematics*. 6th Edition. New York: John Wiley & Sons. 1988.

[8] Danielson, D.A., B. Neta, and L.W. Early. *Semianalytic Satellite Theory (SST): Mathematical Algorithms*. Naval Postgraduate School, Report Number NPS-MA-94-001, January 1994.

[9] McClain, Wayne D. "A Semianalytic Artificial Satellite Theory." Vol 1. 1992. Copy available through Wayne D. McClain at Charles Stark Draper Laboratory.

[10] Cefola, Paul J. and Roger Broucke. "On the Formulation of the Gravitational Potential in Terms of Equinoctial Variables." AIAA Preprint 75-9. AIAA 13th Aerospace Sciences Meeting. Pasadena, CA. January, 1975.

[11] Fonte, Daniel. "An Introduction to Perturbation Theory." A report for 16.601, an advanced special topics course at MIT's department of Aeronautics and Astronautics. Copy available through Dan Fonte or Draper Laboratory.

[12] Lovell, Bob. Lecture on the Orbcomm Satellite System given at the Massachusetts Institute of Technology. November 1994.

[13] Giest, Al, et al. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA: The MIT Press. 1994. Postscript copy of this text also available via anonymous ftp at *netlib2.cs.utk.edu*.

[14] Shaver, Jeffrey Scott. *Formulation and Evaluation of Parallel Algorithms for the Orbit Determination Problem*. Doctor of Philosophy Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. CSDL T-709. March 1980.

[15] Simon, Horst D. "High Performance Computing: Architecture, Software, Algorithms." NASA report number RNR-93-018. December 1993

[16] Beaupre, James. Personal Discussion. Charles Stark Draper Laboratory, Inc. (617)-258-1000. September 1994 through May 1995.

[17] Dowd, Kevin. *High Performance Computing*. Sebastopol, CA: O'Reilly & Associates, Inc. 1993.

[18] Bertsekas, Dimitri. *Distributed and Parallel Computing*. Englewood Cliffs, New Jersey: Prentice Hall. 1989.

[19] Blitzer, Leon. "Handbook of Orbital Perturbations." Professor of Physics, University of Arizona. Partially reprinted for Astrodynamics 422 course at the United States Air Force Academy (Spring 1993). Copy available from Department of Astronautics, USAFA.

[20] Ferguson, Jack. Notes taken in Astrodynamics 422 at the United States Air Force Academy. Copy available from Scott Wallace.

[21] Gropp, William, et al. *Using MPI.* Cambridge, MA: The MIT Press. 1994.

[22] *SunOS™ 5.3 Guide to Multithread Programming.* Mountain View, CA: SunSoft. 1994.

[23] Mueller, Frank "A Library Implementation of POSIX Threads under UNIX." 1993 Winter USENIX. San Diego, CA. January 25-29, 1993.

[24] Turcotte, Louis. "A survey of Software Environments for Exploiting Networked Computing Resources." Available by anonymous ftp as bulldog.wes.army.mil:/pub/report.ps.Z.

[25] Petrie, Ann and Ron Kerr. "A Qualitative Comparison of Network Linda and PVM." University of Newcastle upon Tyne. 30th September 1993.

[26] McDonald, Kyle. Personal Discussions. September 1993 through April 1995. Northeastern University.

[27] Cefola, Paul. Personal Discussions. August 1993 through May 1995.

[28] Cappellari, J. O., C.E. Velez and A. J. Fuchs. Editors. *Mathematical Theory of The Goddard Trajectory Determination System.* Greenbelt, Maryland: Goddard Space Flight Center. April 1976.

[29] Fonte, Daniel J. "PC Based Orbit Determination." Paper presented at the AIAA/AAS Astrodynamic Conference. August 1-3, 1994. Scottsdale, AZ.

[30] *arches.* Document Included with PVM 3.3.7. Available by anonymous ftp to netlib2.cs.utk.edu.

[31] *CM VIEW*. On-line Documentation for the CM-5. Accessed over the Internet at scout@mit.lcs.edu. For account information on the CM-5 contact project SCOUT at the Laboratory for Computer Science at MIT.

[32] Early, L. W. *A Portable Orbit Generator Using Semianalytic Satellite Theory*. AIAA/AAS Astrodynamics Conference. Williamsburg, VA. August 1986.

[33] Jablonski (Boelitz), Carole. *Application of Semianalytic Satellite Theory to Maneuver Planning*. Master of Science Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. CSDL T-1086. May 1991.

[34] Carter, David. Personal Discussions. August 1993 through October 1994.

[35] Carter, Scott S. GTDS Card Deck developed in support of this work. Charles Stark Draper Laboratory.

[36] Wackernagel, H.B. "Orbit Representation." SCC Development Division. Memorandum for Record. 2 October 1975. Copy available from Dr. Paul Cefola, CSDL.

[37] Comparetto, Gary M. "A Technical Comparison of Several Global Mobile Satellite Communications Systems." *Space Communications* 11 (1993). 97-04.

[38] Bate, Rodger R., Donald D. Mueller and Jerry E. White. *Fundamentals of Astrodynamics*. New York: Dover Publications.

[39] Battin, Richard. Class notes from 16.347 Astrodynamics II. 5 April 1994. Copy available through Scott Wallace.

[40] Fonte, Daniel John. *Implementing a 50 X 50 Gravity Field Model in an Orbit Determination System.* Master of Science Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. CSDL-T-1169. June 1993.

[41] Kozai, Yoshihide. "Analytical Orbital Theories for Satellites." *The Use of Artificial Satellites for Geodesy and Geodynamics.* Proceedings of the International Symposium on the use of Artificial Satellites for Geodesy and Geodynamics. Athens, Greece. May 1973.

[42] Lindeburg, Michael R. *EIT Training Manual.* Belmont, CA: Professional Publications, Inc. 1992. 8th Ed.

[43] "Software Test Description for the Flight Dynamics CSCI of the Radarsat MSC Program." CSDL-306787. November 12, 1993.

[44] Stu Roseman. nineCo unlimited. Personal Discussions. September 1993 through April 1995.

[45] *SPARCompiler FORTRAN 3.0. User's Guide.* Mountain View, CA: SunPro. 1993.

[46] Cefola, Paul J., et al. "The RADARSAT Flight Dynamics System: An Extensible, Portable, Workstation-based Mission Support System." Paper presented at the AIAA/AAS Astrodynamics Conference. August 1-3, 1994. Scottsdale, AZ.

[47] Smith, Dan. "Efficient Mission Control for the 48 Satellite Globalstar Constellation." Loral Aero Sys. Space Ops 94. November 16, 1994.

[48] Neal D Hulkower, Ph.D. "A Reevaluation of Ellipso(TM), Globalstar, IRIDIUM (TM/SM) and Odyssey (TM)." Talk given at Volpe Center for Transportation. Sponsored by the New England Section of the AIAA. October 1994.

[49] Long, A.C., et al. *Goddard Trajectory and Determination System (GTDS) Mathematical Theory.* NASA's Operational GTDS Mathematics Specification. Revision 1. Edited by Computer Sciences Corporation and NASA Goddard Space Flight Center. GSFC Code 550. July 1989.

[50] Sabol, Chris. *Application of Sun-Synchronous Critically Inclined Orbits to Global Personal Communications Systems.* Master of Science Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. CSDL T-1235. Nov 1994.

[51] Livingston, Marilynn, and Quentin Stout. "Introduction to Parallel Computing." Transcript from a tutorial presented at Supercomputing 92 on November 16, 1992.

[52] Hwang, Kai and Fayé A. Briggs. *Computer Architecture and Parallel Processing.* New York: McGraw-Hill Book Company. 1984.

[53] Chiou, Derek. "Symmetric MultiProcessors (SMP's)." Notes for course CS-722, Parallel Architecture, at Boston University. Fall 1994.

[54] Neufville, Richard de. *Applied Systems Analysis.* New York: McGraw-Hill, Inc. 1990.

[55] Wertz James R. and Wiley J. Larson. *Space Mission Analysis and Design.* Boston: Kluwer Academic Publishers. 1991.

[56] Walker, J. G. "Satellite Constellations." Royal Aircraft Establishment, Farnborough. 1983.

[57] Gordon, K. J. "The Computation of Satellite Constellation Range Characteristics." AIAA-94-3704-CP.

[58] Karrenberg, H. K., E. Levin, and R. D. Lüders. "Orbit Synthesis." *The Journal of the Astronautical Sciences.* 17 (Nov-Dec 1969). p. 129-177.

[59] Telephone Conversation with SUN Microsystems. 24 April 1995.

[60] Metzinger, Richard. RADARSAT Interview for D-Notes, to be published May 1995.

[61] Not Used.

[62] McClain, Wayne D. "Eccentricity Control and the Frozen Orbit Concept for the Navy Remote Ocean Sensing System (NROSS) Mission." Presented at AAS/AIAA Astrodynamics Conference, Kalispell Montana. August 1987.

[63] Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company. New York. 1989.

[64] Schott, Jason. *Fault Tolerant Design Using Single and Multi-Criteria Genetic Algorithm Optimization*. Master of Science Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology. CSDL T-1251. May 1995.

[65] Hammett, Robert C. et al. "Design Optimizer / Markov Evaluator (DOME) Version 1.0 -- User's Manual." Charles Stark Draper Laboratory. Analysis and Software Department of the Control and Decision Systems Division. CSDL-R-2409. May 1992.

[66] "Application of Teledesic Corporation for a Low Earth Orbit Satellite System in the Domestic and International Fixed Satellite Service." Presented before the Federal Communications Commission. Washington D.C. File No. DSS-P/L-94.

[67] Uphoff, Chauncey. Telephone interview. 20 January 1995.

[68] Forrest, Stephanie. "Genetic Algorithms: Principles of Natural Selection Applied to Computation." *Science*. Vol. 26. 13 August 1993.

[69] Qualline, Steve. *Practical C.* O'Reilly & Associates, Inc. Sebastopol, CA. 1993.

[70] Kernighan, Brian W, and Bob Pike. *The UNIX Programming Environment.* Prentice Hall, Inc. New Jersey. 1984.

[71] Aho, Alfred V. et al. *Compilers Principles, Techniques, and Tools.* Addison-Wesley Publishing Company. Reading, MA. 1986.

[72] Hwang, Kai. *Advanced Computer Architecture.* McGraw-Hill, Inc. New York. 1993.

[73] *Command Reference.* Continuus Software Corporation. Irvine, CA. Copy available from Charles Stark Draper Laboratory. Cambridge, MA.

[74] Cefola, P. J. and R. A Broucke. "On the Equinoctial Orbit Elements." *Celestial Mechanics.* 1972: 303-310.

[75] Brouwer, D. and Hori, G. "Theoretical Evaluation of Atmospheric Drag Effects in the Motion of an Artificial Satellite." *Astronomical Journal.* Vol. 66. No. 1290. June 1961.

[76] Computer Sciences Corporation and Systems Development and Analysis Branch (GSFC). *Research and Development Goddard Trajectory Determination System. User's Guide.* July 1978.

[77] "Detailed Descriptions of the Draper Semianalytic Satellite Theory GTDS Keyword Cards". Appendix A to the *Research and Development Goddard Trajectory Determination System. User's Guide.* Copy available through Draper Laboratory.

[78] Jameson, Kevin. *Multi-Platform Code Management.* Sebastopol, CA: O'Reilly and Associates, Inc. 1994.

[79] Whitgift, David. *Methods and Tools for Software Configuration Management.* New York: John Wiley and Sons. 1991.

[80] Babich, Wayne A. *Software Configuration Management Coordination for Team Productivity*. Reading, Massachusetts: Addison-Wesley Publishing Company. 1986.

[81] Metzinger, Richard W. "Validation of the Workstation Version of R&D GTDS." Cambridge, MA: Charles Stark Draper Laboratory. February 24, 1993.