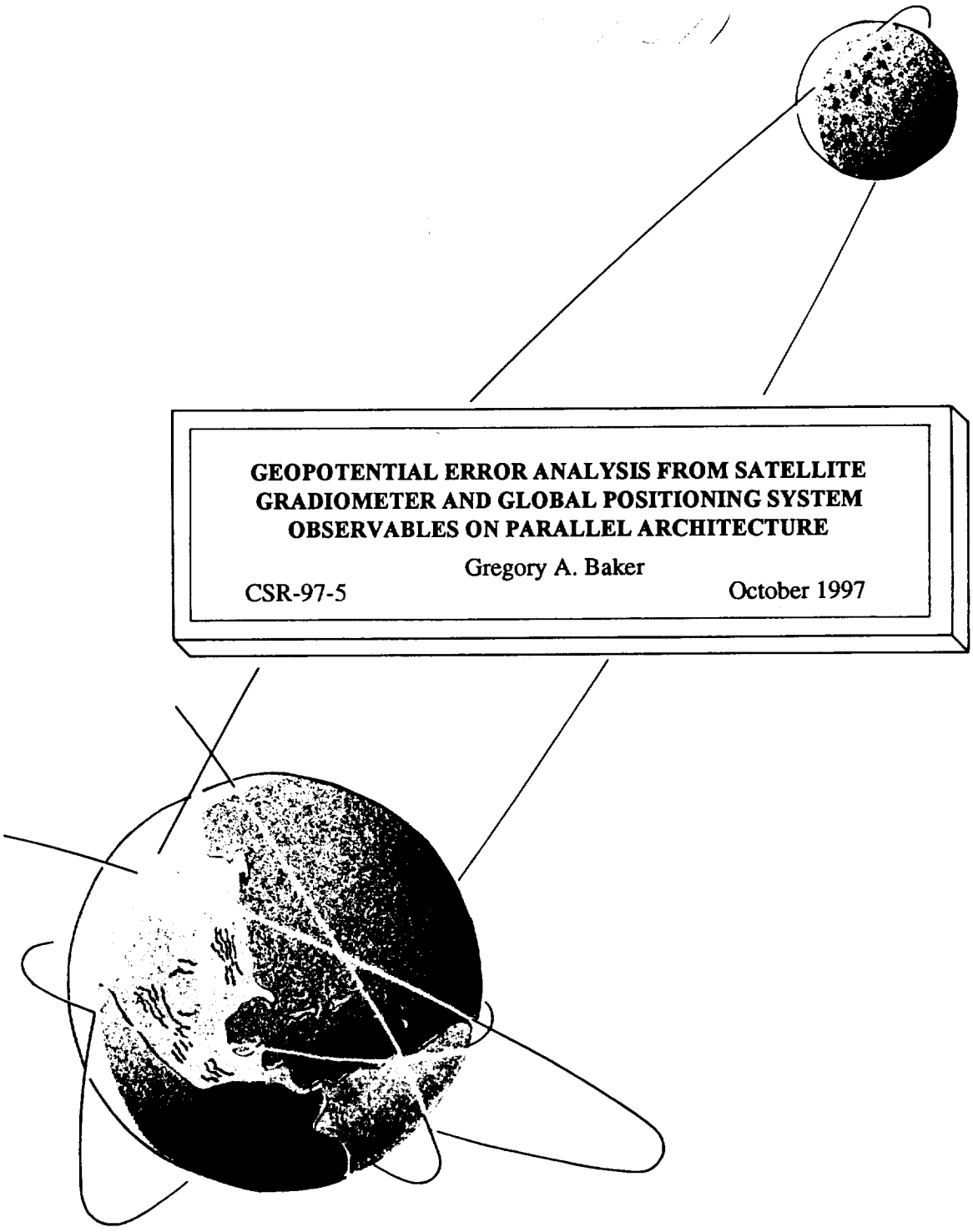


GEOPOTENTIAL ERROR ANALYSIS FROM SATELLITE GRADIOMETER AND GLOBAL POSITIONING SYSTEM OBSERVABLES ON PARALLEL ARCHITECTURE

Gregory A. Baker

CSR-97-5 October 1997

CENTER FOR SPACE RESEARCH

THE UNIVERSITY OF TEXAS AT AUSTIN AUSTIN, TEXAS

GEOPOTENTIAL ERROR ANALYSIS FROM SATELLITE GRADIOMETER AND GLOBAL POSITIONING SYSTEM OBSERVABLES ON PARALLEL ARCHITECTURE

by

Gregory A. Baker

Center for Space Research

The University of Texas at Austin

October 1997

CSR-97-5

Final report for the
National Aeronautics and Space Administration
Grant No. NAG5-2511

by the
Center for Space Research
The University of Texas at Austin
Austin, TX 78712

Principal Investigator:
Dr. Bob E. Schutz

ESS Guest Computational Investigator

ABSTRACT

The recovery of a high resolution geopotential from satellite gradiometer observations motivates the examination of high performance computational techniques. The primary subject matter addresses specifically the use of satellite gradiometer and GPS observations to form and invert the normal matrix associated with a large degree and order geopotential solution. Memory resident and out-of-core parallel linear algebra techniques along with data parallel batch algorithms form the foundation of the least squares application structure. A secondary topic includes the adoption of object oriented programming techniques to enhance modularity and reusability of code. Applications implementing the parallel and object oriented methods successfully calculate the degree variance for a degree and order 110 geopotential solution on 32 processors of the Cray T3E. The memory resident gradiometer application exhibits an overall application performance of 5.4 Gflops, and the out-of-core linear solver exhibits an overall performance of 2.4 Gflops. The combination solution derived from a sun synchronous gradiometer orbit produce average geoid height variances of 17 millimeters.

1. Introduction

The recovery of a high resolution geopotential model from satellite observations motivates the examination of high performance computational techniques. The recently accepted Gravity Recovery and Climate Experiment (GRACE) mission will soon provide direct observations of the gravity field from an orbiting platform [NASA, 1997]. Previous studies demonstrate that the rigorous, single point analyses as defined in Section 1.1.2 produce the best gravity field solutions [Bettadpur, 1993; Koop, 1993]. Computational cost and memory limitations of serial computer architectures restrict the rigorous analysis of gravity field models to approximately 100 kilometer resolution. Next-generation data analysis techniques that resolve the high frequency components of the gravity field must be developed in preparation of future dedicated gravity missions [CIGAR, 1996]. This interdisciplinary research consists of an investigation into the appropriate algorithmic design required to recover high resolution geopotential coefficients using rigorous analysis methods.

1.1 Global Gravity Field Determination

Accurate modeling of the global gravity field is of fundamental importance due to the wide variety of geodynamic processes exhibited in the

gravity signal. Gravity field anomalies provide one of the few direct manifestations of interior processes such as mantle convection and lithosphere motion. Temporal changes in the gravity field indicate changes in sea level and climate due to mass transport between the oceans, ice sheets and atmosphere [McNutt, 1990]. Oceanography requires an accurate geoid for the separation of mean circulation from the tidal and time-varying ocean circulation effects [Zlotnicki, 1990]. The ability to measure the gravity field from low orbit provides an excellent opportunity to observe these processes on a global scale. A comprehensive survey of past events in gravity field determination on local and global scales is presented by Nerem et al [1996].

1.1.1 Satellite Techniques

The development of global gravity field models relies heavily on the satellite tracking measurements. The non-uniform gravity field of the Earth perturbs the satellite motion. Range and range-rate observations to the satellite record the first and second time integral of the gravity force. The satellite tracking measurements are especially sensitive to the low frequency signal in the gravity field as the time integral smoothes over much of the gravity field's high frequency information. High precision observations performed by satellite laser ranging (SLR) enable the recovery of Earth

orientation, temporal variations in low frequency gravity field signals and allow confirmation of certain relativistic effects [Tapley, 1993]. The Global Positioning System (GPS) supplements conventional SLR tracking by providing near-continuous coverage of the Earth and low orbiting satellites.

Resolution of the higher frequency components requires the use of other measurement types. Satellite altimetry measures the local geoid height over the oceans. Measurements made on the surface of the Earth measure local accelerations. The high resolution measurements are complimentary to the low resolution satellite tracking data types in the global gravity field solution. The quality of altimetry observations from missions such as TOPEX/Poseidon depends on the knowledge of the ocean topography, atmospheric refraction and satellite trajectory [Nerem et al, 1996].

Measurements of the gravity gradient, or the spatial rate or change of gravitational acceleration, also provide high resolution observations of the gravity field. Two measurement types observe the gravity gradient. The satellite gradiometer directly measures the gradient by differencing the measurements of accelerometers mounted on a satellite platform. The satellite-to-satellite tracking observable measures the integral of acceleration differences by monitoring the inter-satellite range and range rate. Rummel [1986] provides a discussion of both concepts.

Satellite gradiometry employs pairs of linear accelerometers mounted symmetrically about the satellite center of mass. The gradient signal remains after eliminating common accelerations. Two instrument concepts are being investigated in preparation for a potential gradiometer mission. The NASA GEOID mission will employ a cryogenic gradiometer to achieve high gradient resolution [Paik, 1996]. The European Gravity and Ocean Circulation Explorer (GOCE) is a European Space Agency (ESA) mission implementing the GRADIO type instrument originally designed for the ARISTOTELES gravity mission [Rummel, 1996].

Satellite gradiometry is more sensitive to the high frequency signal of the geopotential than the satellite-to-satellite tracking method. A typical gradiometer mission at an altitude of 200 kilometers will be capable of resolving the gravity field to 50-100 km [Schrama, 1991]. Super-cooling requirements and atmospheric drag effects at low altitudes limit the duration of satellite gradiometer missions to less than one year. A gradiometer mission would provide a snapshot in time of the global gravity field. The medium to high resolution images are important to geodesist investigating solid earth physics on long temporal scales and to oceanographers studying steady-state ocean circulation patterns.

Satellite-to-satellite tracking techniques recover range or range rate information between two co-orbiting satellites separated by only a few degrees of arc. The inter-satellite measurements are time integrals of the gravity gradient between the satellites. The NASA GRACE mission scheduled to fly in 2001 employs a satellite-to-satellite microwave tracking system [Davis et al., 1996].

The satellite-to-satellite observations are less sensitive to altitude than the gradiometer observation. A higher altitude orbit allows mission lifespans of 3 to 5 years. A satellite-to-satellite mission will produce a series of images illustrating the time-varying nature of the gravity field. The examination of dynamic processes such as the movement of the water mass between the Earth, atmosphere and oceans may proceed from a new perspective.

1.1.2 Analysis Methods for Global Solutions

The global gravity field analysis computes the least squares estimate of geopotential parameters given an observation data set. The spherical harmonic series provides an accurate representation of the geopotential function. As such, the harmonic coefficients comprise the set of estimated parameters. Section 2.3 describes more fully the spherical harmonic model, and section 3.2 addresses the least squares technique.

The distribution of observations over the surface of the Earth further distinguishes the analysis approach. Single point computation methods perform the most rigorous reduction of observational data. A point-wise evaluation of the dynamic and observation models occurs along the satellite trajectory according to arbitrary dynamic and observation models. The analysis proceeds without the introduction of simplifications or assumptions. The processing cost associated with the least squares techniques may restrict the size of the single point analysis. The least squares technique requires the formation of a large, dense linear system to calculate the parameter updates. For the estimation of a large number of parameters, both the computational cost and memory requirements may become prohibitive.

Grid computation methods exploit natural symmetries in the mathematical model of the gravity field to greatly reduce computational costs. Colombo [1981] demonstrated that the normal matrix reduces to a block diagonal form by assuming an observation distribution which coincided with the equiangular points on a sphere. The same normal matrix structure results from data collected along a repeat ground track orbit with observations taken at a regular sampling interval [Koop, 1993]. The analysis of real data requires the preprocessing of irregularly sampled data taken along an imperfect repeat

orbit. The process of forming block averages and normal points introduces error into the estimate.

Grid computation methods have been useful in performing error analysis of proposed dedicated gravity missions. Many authors have illustrated the necessity of combining satellite tracking data and gradiometer observations to produce an unbiased estimate of the gravity field to high degree and order [Schrama, 1991; Koop, 1993; Visser, 1994]. Additional information is required (e.g., Kaula's rule of thumb) to recover a solution from gradiometer missions located in a non-polar orbit.

1.2 Computational Challenges

The development of computational methods capable of reducing satellite observations into useful gravity field information constitutes a significant computational challenge. The challenge arises from both the large number of observations and the large number of unknown parameters. Assuming five second sampling intervals, a six month GPS-tracked, satellite gradiometer mission would produce over eight million combined gradient and GPS observations [Schuh et al, 1996]. A geopotential expansion to degree and order 180 possesses 32,761 terms. The computer resources required to compute the linear least squares estimate of the gravity coefficients from the

described scenario are significant. Over 8 Gbytes of computer memory (64-bit precision floats) are required and approximately 10 quadrillion (10^{16}) floating point operations must be performed. Current cutting-edge distributed memory parallel architectures which execute at hundreds of Gflops require days of execution time to finish the problem. Ultimately, the rigorous analysis of such data necessitates the use teraflop and petaflop computer architectures.

Fortunately, the size of the gravity field model may be adjusted to match the performance capabilities of the available architectures. The total cost of the problem is driven primarily by the time required to accumulate the observation equations into the normal matrix. The cost of forming the dense normal equations using rigorous methods is known to be mn^2 where m is the number of observations and n is the number of geopotential parameters to be estimated. Figure 1 presents the amount of wall clock time required to accumulate one million observations into the normal matrix associated with a given maximum degree and order gravity field expansion. The different curves represent varying levels of computational performance. Figure 2 presents the amount of processor memory required to store the normal matrix associated with a given maximum degree and order expansion.

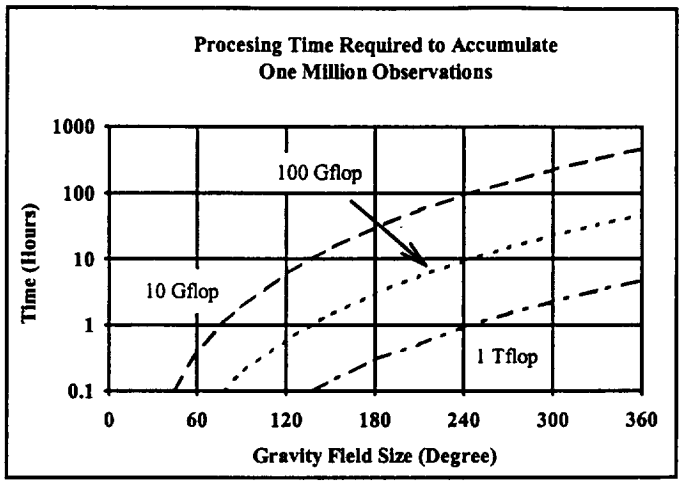


Figure 1 Processing Time for Normal Equations Accumulation

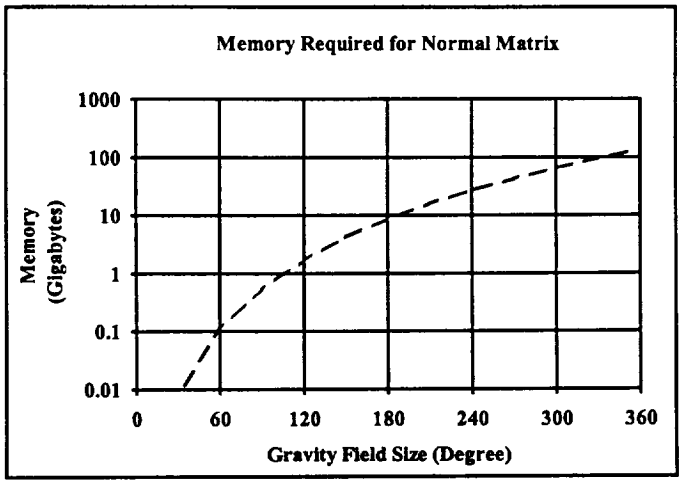


Figure 2 Memory Requirements for Normal Equations

The design of high performance satellite applications must also consider the implementation of the satellite dynamic models. The total work required to model the physical system is insignificant when compared to the total cost of the least squares operations. However, once the linear algebra

operations have been effectively optimized for performance on a parallel computer, the physical model cost dominates the wall-clock execution time of the application. An effective application must address the optimization of satellite propagation operations as well.

1.3 High Performance Computing

Parallel processing provides a practical solution to the computational cost difficulties associated with the gravity field problem. Computer industry projections predict that advances in current technology will lead to only a doubling or tripling of performance improvement in single processor technology [Astfalk, 1993]. Only the aggregate power of many processors executing concurrently will provide the performance required to solve the gravity field problem.

1.3.1 Amdahl's Law

Amdahl's Law [Amdahl, 1967] defined the early years of parallel computing. The hypothesis states that the maximum speed-up of a parallel algorithm is bound by the reciprocal of the time required to execute any serial region within the algorithm regardless of the number of processors (See Figure 3). By this measure, significant performance increases would depend on the elimination of serial regions of execution. Since many program activities and

certain algorithms are serial in nature (e.g., numerical integration), the benefits of parallel processing were thought to be minimal.

A revolution in parallel processing theory occurred after Gustafson et al [1988] exposed an implicit assumption in Amdahl's original statement. Amdahl assumes that the problem size remained constant as the parallel region is distributed across an increasing number of processors. In practice, however, the problem size generally expands to the capacity of available memory. A reformulated relationship includes the scaling of the problem size with the number processors. The scaled Amdahl's Law demonstrates that scalable algorithms, or algorithms which maintain efficiency as the number of processors increase, could be developed (See Figure 3).

Allow P to be the number of processors allocated to solve the problem.

Amdahl's Law

Allow s to be the time spent by a serial processor executing the serial region and p to be the time spent by a serial processor executing the parallel region. The speed-up (defined as the serial time divided by the parallel time) as given by Amdahl's Law is,

$$S(P) = \frac{s+p}{s+\frac{p}{P}} \tag{1}$$

which simplifies to the form,

$$S(P) = \frac{P}{s(P-1)+1} \tag{2}$$

As the number of processors approaches infinity, the speed-up is bounded by $\frac{1}{s}$.

Scaled Amdahl's Law

Allow s' to be the time spent by a parallel processor executing the serial region and p' to be the time spent by a parallel processor executing the parallel region. The speed-up as given by Scaled Amdahl's Law is,

$$S(P) = \frac{s'+Pp'}{s'+p'} \tag{3}$$

which simplifies to the form,

$$S(P) = s' + P(1-s') \tag{4}$$

The upper bound of the speed-up is now a function of the number of processors. For a perfectly parallel problem (s' equal to zero), the speed-up is equal to the number of processors.

Figure 3 Derivation of Amdahl's and Scaled Amdahl's Laws

1.3.2 Algorithmic Models

A new algorithmic model is required when moving from serial to parallel computing. Serial programs are designed according to the random-access memory (RAM) model which consists of a central processing unit and an attached memory. The term random-access refers to the ability to retrieve data elements from memory in an arbitrary order. The RAM model is not suitable for parallel algorithms since issues such as computational concurrency and interprocessor communication are not addressed [Já Já, 1992].

The development of a parallel model begins with the categorization of parallel architectures. A common method is to define architectures according to the number of instruction streams and data streams present in the model. The single instruction/single data (SISD) class issues a single sequence of instructions which operate on a single stream of data. The single instruction/multiple data (SIMD) class issues the identical sequence of instructions to multiple processors each of which operates on different streams of data. The multiple instruction/single data (MISD) class issues different sequences of instructions to multiple processors each of which must operate on the identical stream of data. The multiple instruction/multiple data (MIMD) class issues different sequences of instructions to multiple processors each of which operates on different streams of data. The SISD class

corresponds to the RAM programming model described previously. SIMD architectures experienced a wave of popularity in the late 1980's and early 1990's, and the SIMD style of programming is still prevalent. The major hardware vendors, however, have moved away from SIMD and currently produce architectures which support the more powerful MIMD processing style.

Two communication models have evolved to support the movement of data between processors in a parallel environment. The shared memory model views the architecture as a collection of processors which share access to a global memory unit. An equal algorithmic cost is assigned to the retrieval of a data element located in any position of global memory. The message passing model views the architecture as a collection of processors each of which possesses a local memory unit. Data movement between processors occurs in the form of messages. The message is initiated by a send operation on the source processor and completed by a receive operation on the destination processor. The algorithmic cost of the communication depends on the amount of data communicated and the distance between the source and destination processors.

1.3.3 Hardware

The computational performance of any algorithm ultimately depends on the hardware architecture. The primary hardware component which effects performance is the central processing unit (CPU). Two fundamental types of CPUs exist. Scalar processors perform a single operation at a time. Vector processors are fine grained parallel units which pipeline the stages of floating point computations. High performance requires the formulation of the algorithm in terms of the data access patterns best suited for each processor. The pipelining property of vector processors requires long contiguous vectors of data. The scalar processors rely on additional hardware mechanisms to insure the fast availability of data before the start of the actual computation.

A second major hardware component is RAM memory. Two types of memory organization exist. Shared memory architectures share a common memory unit and guarantee equal access time between any processor and any memory location. Distributed memory architectures allocate different physical memory units to each of the processors. Access to a processor's local memory is analogous to conventional single processor architectures. Remote memory units are accessed through network communications at a substantially higher cost.

Shared memory, vector processor architectures reflect the state of the art in conventional supercomputing. Computers such as the Cray Y-MP and Cray T90 have a long record of reliability and superior performance. However, the shared memory computer suffers from an inherent lack of scalability due to the equal cost restriction on memory access [Astfalk, 1993]. As a result, the shared memory architectures will not achieve the level of performance required for next generation, gravity field applications. Distributed memory, scalar processor architectures reflect the next generation in supercomputing. Scalable demonstration architectures such as the Cray T3D and Intel iPSC/2 lead to the development of teraflop production systems such as the Cray T3E and Intel Paragon.

1.3.4 Software

Given the wide variety of parallel systems, software should be developed in a portable and maintainable manner. Portability refers to the ability to execute the same high level algorithm on any machine without code modifications. Maintainability refers to the ease with which the software may be modified and enhanced. In some respects, maintainability is a measure of the complexity of the code in terms of number of lines, readability, etc. The discussion of portability leads to an interesting paradox. As mentioned

previously, the algorithm must match the hardware to guarantee performance. However, portability requires that no architecture specific language appear in the algorithm. These two seemingly mutual exclusive items are reconciled in the advanced programming concepts of standardized libraries and object oriented programming.

1.3.4.1 Standardized Libraries

A standardized library establishes an interface to a set of well-defined computational primitives. Architecture dependent parameters are purposely omitted from the interface to facilitate implementation of the library on a variety of different architectures. The standard provides numerous advantages to application developers and hardware vendors. Applications which are designed around standards can be assured of executing efficiently on any platform which supports the standard. Hardware vendors may implement the standard in such a way to exploit the performance characteristic of their machine. The gravity field problem benefits from two specific standards.

The Basic Linear Algebra Subprograms (BLAS) provide single processor implementations of dense linear algebra operations. The original set of subroutine calls described by Hanson [1972] proved insufficient to fully exploit the performance capabilities of different architecture types. Higher

level operations capable of encapsulating architecture specific performance characteristics have been established [Dongarra, 1988] and will be presented in later chapters. For architectures on which these routines have been optimized for performance, the BLAS comprise the components of highly effective libraries and applications. FORTRAN source code is available for architectures which do not posses optimized BLAS calls. The software libraries EISPACK, LINPACK and LAPACK are examples of serial libraries built upon the BLAS [Dongarra, 1992].

The Message Passing Interface (MPI) library establishes a standard to facilitate portability for message passing parallel applications [Snir et al, 1996]. MPI provides a common set of communication routines upon which higher level routines and libraries may be layered. While a relatively new standard, virtually all major distributed architectures support an MPI implementation. In addition, a generic MPI implementation is available [Gropp, 1996].

1.3.4.2 Object Oriented Programming

The object oriented programming (OOP) style establishes a framework within which highly useable and maintainable software may be developed. Application development centers around the manipulation of language abstractions called objects. The objects are programming language constructs

which represent the mathematical or physical components of the problem. A distinguishing characteristic of OOP is the degree to which program complexity is hidden from the programmer. The data and algorithms associated with the object's functionality are hidden, or encapsulated, within a single, modular construct. Interactions between the object and an application program are restricted to well-defined, user interface routines, or methods, which reflect the natural functionality of the modeled system component.

OOP provides several significant advantages over conventional structured programming techniques. Subroutines in large and complex applications usually require long call sequences and/or many global variables to pass the necessary input and output data. The organization of the data into objects reduces the number of call sequence parameters and the dependency on global variables. The resulting code is more readable and easier to maintain. Also, since objects are specifically designed according to the definition of system components, the code more closely resemble the natural expression of the algorithm. The encapsulation of data and functionality creates a modularity which isolates higher level algorithms from programming errors and changes in object implementation. The modularity is also ideally suited for incorporation into software libraries. The programming flexibility

provided by such a library permits the rapid development of new and different applications.

The implementation of OOP using a structured programming language such as C or FORTRAN requires self-discipline in the use of conventional programming constructs. Data structures contain the object's data while subroutines provide the object's functionality. However, protection of hidden object components from direct access by high level routine cannot be enforced by the compiler as is the case in object oriented languages such as C++ and SmallTalk. In general, any access of object components in a manner other than specified by the object methods will lead to unpredictable results.

The development of an Object Oriented Precision Orbit Determination (OOPOD) library begins with the establishment of two general types of objects, or classes. The physical class abstracts in a generic manner the physical entities which comprise the satellite environment. The properties of a physical class object are completely specified by a user-supplied description of the modeled object and its environment. For example, the properties of a satellite object correspond to the physical characteristics of the satellite (mass, dimensions, moments of inertia, etc.) and the forces acting on the satellite (Earth gravity, drag, moon, etc.). The mathematical class abstracts in a generic manner the mathematical techniques used to manipulate data derived

from the physical system. Mathematical class properties are partially specified by a user-supplied description of the mathematical technique. Additional information must be extracted from associated physical objects to complete the mathematical object description. For example, the properties of a multistep numerical integration routines include the characteristics solely associated with the integrator (starting convergence criteria, grid size, order of integration method, etc.) and data extracted from the "to be propagated" satellite object (dimension of integration vector, initial state, equations of motion derived from list of forces, etc.)

The existence of OOPOD library objects permits the rapid development of application code. Given a conceptual description of a precision orbit determination problem, the development of the corresponding OOPOD application consists of four steps.

1. Model the physical entities in the problem by creating physical class objects using the appropriate physical object creation methods according to the desired model parameters.
2. Initialize the mathematical techniques by creating mathematical class objects using the appropriate mathematical object creation methods according to desired object functionality.

3.Fully realize the mathematical class objects through associations with physical objects using the appropriate mathematical object realization methods.

4.Specify the POD algorithm in terms of the physical and mathematical object methods according to desired application functionality.

Chapter 2 and Chapter 3 describe the physical and mathematical object in conjunction with the description of application components. Prototypes of the object methods use standard C syntax. A complete list of all object methods as proposed by this research in provided in Appendix C.

1.3.5 Parallel Linear Algebra

A parallel linear algebra library must fulfill a significant list of expectations. A library implementation must include the necessary interprocessor communication without sacrificing portability and high performance. The library must also possess a flexible interface which can support a wide variety of applications which distribute data in very different ways. Two groups contributing significantly to the development of parallel linear algebra libraries are the ScaLAPACK project and the PLAPACK project.

The ScaLAPACK project is a combined effort between the University of Tennessee Computer Science Department and Oak Ridge National Laboratory [Dongarra, 1997]. The ScaLAPACK package is a FORTRAN library built upon the BLAS. Matrix elements are distributed over the processors in a block-cyclic manner. Communication between processors occurs via the Basic Linear Algebra Communication Subprograms (BLACS), a proposed communication standard developed specifically for the ScaLAPACK library. The ScaLAPACK project began in 1989.

The PLAPACK project originates from the University of Texas at Austin [van de Geijn, 1997]. The PLAPACK package is a C library built upon the BLAS. Matrix elements are distributed over the processor array according to the principle of Physically Based Matrix Distribution (PBMD) [Edwards, 1995]. PBMD permits the abstraction of different data distributions which are naturally related by collective communication operations. The PLAPACK project began in 1996.

1.3.5.1 Parallel Out-of-Core Processing

The speed of parallel computers permit the solution of problem sizes which exceed the capacity of available memory. As a result, interest in parallel out-of-core (OOC) processing techniques has increased. Many out-of-core dense linear solvers have been developed [Klimkowski, 1995], and

parallel linear algebra library developers are beginning to advertise out-of-core functionality as part of their packages [Dongarra, 1997].

Out-of-core linear algebra implementations may be differentiated by the shape of the submatrix brought into memory for processing. Slab-based algorithms decompose the matrix one-dimensionally and bring an entire row or column panel into memory. While beneficial for pivoting operations in non-symmetric dense solvers, the slab-based approach possesses a greater I/O overhead for large problem sizes. Block-based algorithms decompose the matrix two-dimensionally and read approximately square matrix blocks into memory for processing. While hindering pivoting operations, the block based algorithms minimizes the I/O to computation ratio for large problems.

Another consideration is the amount of the data to read into memory at a given time. Conventional approaches to OOC functionality emphasize the communication of data from disk to memory concurrent with computational activity. This overlapping of communication and computation hides much of the I/O traffic. However, an additional level of complexity is added to the program due to double buffering operations. Overlapping communication also requires memory to be allocated to I/O operations instead of computations which leads to deteriorated algorithmic performance. Klimkowski [1995] demonstrated a linear solver which allocated a significant portion of memory

to computational activity and exposed much of the I/O. Performance results demonstrated the I/O cost in such a scheme to be approximately 10 to 20% of the total computational cost.

1.4 Related Interdisciplinary Work

This work builds directly upon the dissertation research of Dr. Srinivas Bettadpur at the University of Texas at Austin. Dr. Bettadpur examines the implementation of a high performance satellite gradiometer application on shared-memory vector processor. The fine grained nature of the vector processors permit the investigation of high performance spherical harmonic synthesis algorithms and least squares estimation techniques. The research results demonstrate a 97.8% efficiency for memory resident applications and a 89.6% efficiency for out-of-core applications executed on an 8 processor Cray Y-MP.

The propagation of satellite trajectories is perhaps the most common component among all satellite applications. The parallelism of the trajectory propagation process may be the most difficult as well due to the fine granularity of the computations. Two groups published results from applications which implement parallel propagation techniques. The Naval Research Laboratory exploits parallel computing to enhance capabilities of

correlating observations with objects in the Space Command database of orbiting objects [Coffey, 1996]. A brute force approach computes the Lambert solution of every uncorrelated pair of observations in the search for orbits which correspond to previously known objects or which match other uncorrelated observation to within some user-specified criteria. A master processor distributes uncorrelated pairs of observations to slave processors which perform the computations. Results on the IBM SP2 located at the Maui High Performance Computing Center (MHPCC) demonstrate satisfactory results both in the number of recovered orbits and in the efficiency of the method through 128 processors.

Draper Labs uses parallel computing to search for stable configurations for large satellite constellations [Wallace, 1995]. Populations of constellation configurations are evaluated using genetic algorithms. A master-slave algorithm distributes the work of both the analytic trajectory propagation and the genetic algorithm cost evaluation. Results on a heterogeneous workstation network at Draper Labs demonstrate good efficiency over a small number of processors.

The above projects possess similar advantages and disadvantages. The master-slave paradigm inherently performs dynamic load balancing to keep all the processors active. However, the method is not scalable to a large number

of processors due to the communication requirements between each slave processor and the master. Also, the problem size of the individual tasks are limited by the performance capabilities of the individual processors. The master-slave paradigm is useful in the distribution of many small tasks to a moderate number of processors, but the method becomes ineffective for large problem sizes or large number of processors.

The Jet Propulsion Laboratory (JPL) performed research similar to this work by completing the determination of a Venus gravity field complete to degree and order 90 on the 256-processor Cray T3D [Konopliv, 1995]. Data from the Magellan and Pioneer Venus Orbiter missions was processed using a modified version of JPL's Orbit Determination Program (ODP). The global processor array was partitioned into groups with each group responsible for observation equation generation and estimation of sub-arc parameters. Information relevant to global parameters was accumulated using Given's rotations into a information matrix wrapped by rows across a one-dimensional mapping of the global processor array. Parallel accumulation was accomplished by pipelining the communication of single observation arrays along the one-dimensional mapping. Performance results demonstrated a two order of magnitude speed-up over serial processing techniques although

absolute performance figures were not presented nor could be calculated from the available information.

The European Space Agency (ESA) through the work of the Consortium for the Investigation of Gravity Anomaly Recovery (CIGAR) recognizes the processing difficulties associated with the gravity field problem, but suggest an alternative approach to overcoming the computational difficulties. The group postulates that future computational capabilities will be insufficient to formulate a set of normal equations for the expected problem sizes [Schuh, 1996]. A preconditioned conjugate gradient (CG) method would avoid the explicit formation of normal equations by operating directly on the linear system of observation equations (e.g., $y = Hx$). This method can perform the single point computations required for a rigorous analysis of data. Results demonstrate the successful recovery of geopotential information to degree and order 50 on serial processors from simulated combination gradiometer and GPS observables.

The ability to perform single point computations and reduced memory cost justify further study of the CG method. However, certain difficulties may preclude use of this approach. The CG method is a iterative method which requires the generation of the full set of observation equations on each iteration. The cost of forming the observation equations is significant for the

number of observations expected from the dedicated gravity missions. Also, covariance information which provides an important measure of the formal variance of the estimate may not be easily extracted from this method.

1.5 Research Goals and Contributions

The development of proof-of-concept software tools capable of analyzing satellite gradiometer data motivates this research. Conventional analysis tools are implemented on serial architectures but suffer the processing cost and memory restrictions described previously. New software tools designed to exploit high performance distributed memory parallel architectures will permit rigorous gravity field analyses. The rigorous single point computation methods are necessary to recover the most accurate gravity field solution.

This research seeks the following goals:

1. The understanding of the implementation techniques required to develop high performance satellite applications on scalar architectures.
2. The understanding of the implementation techniques required to develop high performance satellite applications on distributed memory architectures.

3.The understanding of the implementation techniques required to develop high performance OOC satellite algorithms on distributed memory architectures.

4.The production of a gradiometer application capable of performing covariance error analyses for high resolution gravity fields.

5.The verification of computational techniques by performing an error analysis of combination gradiometer and GPS data.

The contributions of this research reflect the interdisciplinary nature of the investigation. The integration of high performance computational techniques into a satellite application requires an understanding of computational performance issues and their impact on traditional satellite algorithms. The satellite application produced by this research implements distributed memory parallel accumulation and linear system solve algorithms which effectively addresses all performance issues from single processor execution to out-of-core methods. This work presents an effective implementation of data parallel concurrency in the numerical integration of satellite trajectories and generation of gravity field observation equations. A generalized object oriented framework for satellite applications encapsulates the software complexity and permits the expression of satellite algorithms in a natural manner.

Separate work in the individual disciplines of computational science and satellite geodesy complements the interdisciplinary contributions. The development of PLAPACK Virtual Object functionality as implemented by this research yields a generalized out-of-core library for dense linear algebra. Covariance error analyses for combination gravity field solutions conducted by rigorous methods verify previous studies using grid methods.

1.6 Solution Metrics

1.6.1 Speed-Up and Efficiency

The performance of the parallel application is measured in terms of speed-up and efficiency. Speed-up quantifies the gain in processing speed of a parallel application relative to a serial application that performs the same function. Formally, speed-up is defined as the wall-clock time required to complete the serial application divided by the serial wall-clock time required to complete the parallel application. The parameter n specifies the problem size, and the parameter p specifies the number of processors.

$$S(n,p) = \frac{T^{*}_{serial}(n)}{T_{parallel}(n,p)} \tag{5}$$

Efficiency quantifies the effective use of processor resources relative to the serial algorithm. Formally, efficiency is defined as the speed-up divided by the number of processors.

$$E(n,p) = \frac{S(n,p)}{p} \tag{6}$$

As problem sizes grow large, limits on computational resources prohibit the use of serial applications as benchmarks for parallel applications. In these cases, speed-up and efficiency are often reported in terms of operations per second. The wall-clock time for the parallel application is measured and divided by the operation count of the algorithm to yield the aggregate processing speed. The per processor processing speed is recovered by dividing the aggregate processing speed by the number of processors. The application performance is compared against the peak speed of the processor to provide speed-up and efficiency information.

1.6.2 Error Degree Variance and Degree RMS

The variances recovered from the inverted normal matrix for gravity field solutions are commonly reported in terms of degree variances. The error degree variance presented in Equation (7) is expressed in terms of the sum of coefficient variances corresponding to degree *l*.

$$\sigma_l^2 = \sigma_{l,0}^2(\overline{C}) + \sum_{m=1}^{l} \sigma_{l,m}^2(\overline{C}) + \sigma_{l,m}^2(\overline{S}) \tag{7}$$

The error degree variance is divided by the number of coefficients at degree l to yield the error degree-order variance as presented in Equation (8).

$$\overline{\sigma}_{lm}^2 = \frac{\sigma_l^2}{2l+1} \tag{8}$$

The square root of the error degree-order variance yields the error RMS per coefficient per degree.

1.6.3 Geoid Height Error

Brun's formula specifies the difference in geoid heights between two geopotential fields at a certain location on the surface of the Earth [Vanicek, 1986]. The parameter γ_0 represents the normal gravity on the reference ellipsoid, and ϕ and λ represent the latitude and longitude of the subsurface point on the Earth.

$$\Delta h(\phi,\lambda) = \frac{U_2(\phi,\lambda) - U_1(\phi,\lambda)}{\gamma_0} \tag{9}$$

Equation (9) may be expressed in terms of the geopotential coefficient differences $\Delta\overline{C}_{lm}$ and $\Delta\overline{S}_{lm}$.

$$\Delta h(\phi,\lambda) = \frac{1}{\gamma_0}\frac{\mu}{r}\sum_{l=0}^{l_{max}}\sum_{m=0}^{l}\left(\frac{a_e}{r}\right)^l \overline{P}_{lm}(\sin\phi)\left[\Delta\overline{C}_{lm}\cos m\lambda + \Delta\overline{S}_{lm}\sin m\lambda\right] \tag{10}$$

On the surface of the Earth, $r = a_e$ and $\gamma_0 = \frac{\mu}{a_e^2}$. Equation (10) reduces to a simpler form.

$$\Delta h(\phi,\lambda) = a_e \sum_{l=0}^{l_{max}} \sum_{m=0}^{l} \overline{P}_{lm}(\sin\phi)\left[\Delta\overline{C}_{lm}\cos m\lambda + \Delta\overline{S}_{lm}\sin m\lambda\right] \tag{11}$$

To derive the expression for the variance in geoid height in terms of the coefficient covariance, define α_{lm} and β_{lm} as in Equation (12).

$$\begin{aligned} \alpha_{lm} &= a_e \overline{P}_{lm}(\sin\phi)\cos m\lambda \\ \beta_{lm} &= a_e \overline{P}_{lm}(\sin\phi)\sin m\lambda \end{aligned} \tag{12}$$

The difference in geoid heights and the square of the difference may be expressed in terms of a dot product between the parameters α_{lm} and β_{lm} and the geopotential coefficient differences $\Delta\overline{C}_{lm}$ and $\Delta\overline{S}_{lm}$.

$$\Delta h(\phi,\lambda) = \begin{bmatrix}\alpha & \beta\end{bmatrix}\begin{bmatrix}\Delta\overline{C} \\ \Delta\overline{S}\end{bmatrix} = \mathrm{A}^T\delta x \tag{13}$$

$$\begin{aligned} \Delta h^2(\phi,\lambda) &= \left(\mathrm{A}^T\delta x\right)\left(\mathrm{A}^T\delta x\right)^T \\ &= \mathrm{A}^T\delta x\delta x^T\mathrm{A} \end{aligned} \tag{14}$$

The variance of the geoid height is determined by applying the expectation operator to Equation (14). The matrix P specifies the covariance matrix associated with the estimate of the geopotential coefficients.

$$E\left[\Delta h^2(\phi,\lambda)\right] = E\left[\mathrm{A}^T \delta x \delta x^T \mathrm{A}\right] = \mathrm{A}^T E\left[\delta x \delta x^T\right] \mathrm{A} \tag{15}$$

$$E\left[\Delta h^2(\phi,\lambda)\right] = \mathrm{A}^T P \mathrm{A} \tag{16}$$

The remainder of this document presents a detailed description of the mathematical and computational methods used in this research. Chapter 2 presents the relationships describing the dynamical models and physical objects. Chapter 3 presents the mathematical and computational techniques required to manipulate and recover model parameters. Chapter 4 focuses on the parallel methods as applied to the least squares problem and the observation generation process. Chapter 5 documents the design of the parallel analysis software. Chapter 6 describes the error analysis for the satellite gradiometer mission scenarios. Finally, Chapter 7 discusses the conclusions and recommendations of this research.

2. Simulation Environment

Simulation of the environment traversed by satellites in low Earth orbit presents a complicated and challenging task. The system is inherently non-deterministic due to the countless numbers of perturbing forces and the effects of processes best described in a stochastic manner. Even so, the problem is tractable if the researcher permits a statistically quantified and measurable level of error in the analysis. The resulting physical model can provide insight into the reactions and interactions of an actual satellite orbiting the Earth.

The comprehensive modeling of the physical environment is beyond the scope of this research; however, a certain degree of complexity in the physical model validates the new computational techniques. This research asserts that demonstration of single point processing capabilities satisfy complexity requirements. Single point processing assumes nothing about spatial or temporal symmetries within the environment. Satellite states are determined according to the numerical integration of arbitrarily complex dynamic models. Observation models also implement arbitrarily complex expressions. The use of sophisticated models as required for the processing of real data would require no modifications in the processing methodology.

The physical model adopted by this research consists of a non-spherical gravity field attached to a non-uniformly rotating Earth. The body-fixed frame is determined in part by the stochastic processes of non-uniform rotation and polar motion which must be evaluated from the interpolation of tabular values (see Appendix A). The GPS and gradiometer satellites orbit the Earth and collect observations according to the point-wise calculations of the gravity.

2.1 Physical Objects

Physical objects abstract in a generic manner the physical entities which comprise the satellite environment. Regardless of the degree of specialization, each physical object shares a common set of properties which specify the functionality of the object within the simulated environment. The observable property specifies whether the object may be treated as an observable quantity in the estimation of model parameters. The dynamic property specifies whether the object's state must be determined by numerical integration methods. The force property specifies whether the object influences the motion of an orbiting body. Property values are set at creation and may be accessed by the inquiry methods provided in Appendix C.

All physical objects are created and fully realized via a single call to the object's creation method. In the following discussion of physical entities, the associated physical classes and a sampling of object methods will be described. A complete list of all object methods as proposed by this research is provided in Appendix C.

2.2 Reference Frames and Coordinate Transformations

The development of mathematical equations to describe the relationship between physical objects requires the establishment of appropriate coordinate systems. The equations of motion for a dynamical body are defined within an inertial coordinate system. An approximation of an inertial reference system is the celestial reference frame (CRF). The CRF is a spatial coordinate system realized by a catalogue of extra-galactic radio sources which show no proper motion [Bock, 1996]. A second reference system is used to specify the positions and velocities of objects located on or near the surface of the Earth. The terrestrial reference frame (TRF) is realized by a catalogue of station positions located on the surface of the Earth. The station locations form a time-varying polynomial. The configuration of the polynomial at the chosen epoch defines the instantaneous TRF. The TRF will be referred to in this work as the geocentric coordinate system.

The conversion between the CRF and the TRF is accomplished through a series of plane rotations accounting for the effects of general precession, nutation, time-varying rotation and polar motion. Deformation effects of the Earth are neglected in this study. A summary of the rotation process is presented in Appendix A.

Other reference systems commonly referenced are the topographic (ENU) system, the radial, transverse, normal (RTN) system, and the gradiometer system. The topographic system is defined on the surface of the Earth at longitude λ and latitude ϕ. The coordinate axes point in the direction of increasing longitude (East), increasing latitude (North) and increasing altitude (Up). The RTN system is defined at the center of mass of a satellite in orbit. The coordinate axes point along the instantaneous radius vector, in the plane of the orbit orthogonal to the radius vector and normal to the orbit plane. The gradiometer system is defined by the orientation of the gradiometer with respect to the satellite. The gradiometer system is offset from the satellite frame by constant Euler angles ψ, θ and ϕ.

The coordinate transformation operation is a fundamental component for satellite application software. Two objects provide the functionality of the operation. The first object encapsulates the table look-up process required for

the Earth orientation parameters. The second object encapsulates the creation of matrix rotations required for the transformation of state vectors.

int32 EopTable_create	**(char ***	**filename,**
	int32	**file_format,**
	EopTable *	**table);**
int32 EopTable_free	**(EopTable ***	**table);**

The **EopTable** object contains the tabular information necessary to determine the polar motion and sidereal time at a given epoch. The creation method reads the parameters from the specified file **filename** with specified format **file_format.**

int32 EopTable_calculate	**(EopTable**	**table,**
	float64	**epoch);**

The **EopTable_calculate** method performs the table look-up and caches the results within the object for future use. The data is retrieved via calls to the inquiry methods listed in Appendix C.

int32 RefFrame_create	**(char ***	**filename,**
	int32	**file_format,**
	RefFrame *	**frame);**
int32 RefFrame_free	**(RefFrame ***	**frame);**

The **RefFrame** object contains the information and algorithms necessary to generate the rotation matrices required to transform position and velocity

vectors between the geocentric, true-of-date, mean-of-date, and J2000 coordinate systems. The creation routine internally creates an **EopTable** according to the specified file **filename** with specified format **file_format**.

```
int32 RefFrame_calculate    ( int32      set_tod,
                              float64    epoch,
                              RefFrame   frame );
```

The **RefFrame_calculate** method calculates the Earth orientation parameters and rotation matrices for the specified epoch and caches the results within the object for future use. If **set_tod** is set to **SET_TOD**, the specific epoch will be considered the true-of-date epoch and the precession and nutation parameters will be calculated for the epoch. If **set_tod** is set to **NO_SET_TOD**, only the parameters associated with Earth orientation will be calculated. The data is retrieved via calls to the inquiry methods listed below and in Appendix C.

```
int32 RefFrame_J2000_to_meanofdate      ( RefFrame   frame,
                                          float64 *  a );
int32 RefFrame_meanofdate_to_trueofdate ( RefFrame   frame,
                                          float64 *  a );
int32 RefFrame_trueofdate_to_geocentric ( RefFrame   frame,
                                          float64 *  a,
                                          float64 *  b );
```

The above **RefFrame** methods return rotation matrices between the J2000 inertial system, the mean-of-date coordinate system and the true-of-date

coordinate system. The rotation matrices are placed in **a** and **b** indexed in column-major order.

```
int32 RefFrame_geocentric_to_topographic ( float64    radius,
                                           float64    lambda,
                                           float64    phi,
                                           float64 *  a );

int32 RefFrame_geocentric_to_rtn         ( float64 *  pos,
                                           float64 *  vel,
                                           float64 *  a );
```

The above **RefFrame** methods return rotation matrices based on the geocentric position. **RefFrame_geocentric_to_topographic** creates the rotation matrix from the current geocentric position expressed in spherical coordinates to the topographic position defined by the spherical coordinates. **RefFrame_geocentric_to_rtn** creates the rotation matrix from the current position and velocity vectors defined in geocentric coordinates into the radial, transverse, normal coordinates. The rotation matrices are placed in **a** indexed in column-major order.

```
int32 RefFrame_general                  ( float64      psi,
                                          float64      theta,
                                          float64      phi,
                                          float64 *    a );
int32 RefFrame_cartesian_to_spherical   ( float64 *    x,
                                          float64 *    radius,
                                          float64 *    lambda,
                                          float64 *    phi );
int32 RefFrame_spherical_to_cartesian   ( float64      radius,
                                          float64      lambda,
                                          float64      phi,
                                          float64 *    x );
```

The above **RefFrame** methods facilitate common coordinate conversions. **RefFrame_general** creates a general rotation matrix based on Euler angles. The rotation matrix is placed in **a** indexed in column-major order. **RefFrame_cartesian_to_spherical** and **RefFrame_spherical_to_cartesian** perform the conversion between spherical and cartesian coordinate systems.

```
int32 RefFrame_merge                    ( int32        side,
                                          int32        trans,
                                          float64 *    a,
                                          float64 *    b );
```

The **RefFrame_merge** method merges the two rotation matrices **a** and **b** into a single rotation matrix overwriting **a**. The valid values of **side** are **LEFT** and **RIGHT**. The valid values of **trans** are **TRANS** and **NO_TRANS**. Table 1 presents a summary of operations performed by **RefFrame_merge**.

Side	Trans	Operation
LEFT	NO_TRANS	$a \leftarrow ba$
LEFT	TRANS	$a \leftarrow b^T a$
RIGHT	NO_TRANS	$a \leftarrow ab$
RIGHT	TRANS	$a \leftarrow ab^T$

Table 1 Summary of Operations for *RefFrame_merge*

```
int32 RefFrame_vector          ( int32      trans,
                                 float64 *  a,
                                 float64 *  x );
int32 RefFrame_tensor          ( int32      trans,
                                 float64 *  a,
                                 float64 *  g );
int32 RefFrame_vector_partials ( int32      trans,
                                 float64 *  a,
                                 float64 *  xp,
                                 int32      length );
int32 RefFrame_tensor_partials ( int32      trans,
                                 float64 *  a,
                                 float64 *  gp,
                                 int32      length );
```

The above **RefFrame** methods perform the coordinate transformation of vector **x** or tensor **g**. The valid values of **trans** are **TRANS** and **NO_TRANS**. **RefFrame_vector_partials** and **RefFrame_tensor_partials** permit the transformation of the partial derivatives associated with the vector or tensor quantity. The number of partial derivatives is specified by **length**. Table 2 and Table 3 present a summary of operations performed by the above member functions.

The partials are assumed packed in a contiguous array according to the ordering given in the tables.

Trans	Operation	Storage
NO_TRANS	$x \leftarrow Ax$ $\frac{\partial x}{\partial \alpha_i} \leftarrow A\frac{\partial x}{\partial \alpha_i}, all\ i$	$x \in \left[x_0 \quad x_1 \quad x_2\right]$ $\frac{\partial x}{\partial \alpha_i} \in \left[\frac{\partial x_0}{\partial \alpha} \quad \frac{\partial x_1}{\partial \alpha} \quad \frac{\partial x_2}{\partial \alpha}\right]$
TRANS	$x \leftarrow A^T x$ $\frac{\partial x}{\partial \alpha_i} \leftarrow A^T\frac{\partial x}{\partial \alpha_i}, all\ i$	$x \in \left[x_0 \quad x_1 \quad x_2\right]$ $\frac{\partial x}{\partial \alpha_i} \in \left[\frac{\partial x_0}{\partial \alpha} \quad \frac{\partial x_1}{\partial \alpha} \quad \frac{\partial x_2}{\partial \alpha}\right]$

Table 2 Summary of Operations for *RefFrame_vector* and *RefFrame_vector_partials*

Trans	Operations	Storage
NO_TRANS	$g \leftarrow AgA^T$ $\frac{\partial g}{\partial \alpha_i} \leftarrow A\frac{\partial g}{\partial \alpha_i}A^T, all\ i$	$g \in \left[g_{0,0} \quad g_{1,0} \quad g_{2,0} \quad g_{1,1} \quad g_{2,1} \quad g_{2,2}\right]$ $\frac{\partial g}{\partial \alpha_i} \in \left[\frac{\partial g_{0,0}}{\partial \alpha} \quad \frac{\partial g_{1,0}}{\partial \alpha} \quad \frac{\partial g_{2,0}}{\partial \alpha} \quad \frac{\partial g_{1,1}}{\partial \alpha} \quad \frac{\partial g_{2,1}}{\partial \alpha} \quad \frac{\partial g_{2,2}}{\partial \alpha}\right]$
TRANS	$g \leftarrow A^T gA$ $\frac{\partial g}{\partial \alpha_i} \leftarrow A^T\frac{\partial g}{\partial \alpha_i}A, all\ i$	$g \in \left[g_{0,0} \quad g_{1,0} \quad g_{2,0} \quad g_{1,1} \quad g_{2,1} \quad g_{2,2}\right]$ $\frac{\partial g}{\partial \alpha_i} \in \left[\frac{\partial g_{0,0}}{\partial \alpha} \quad \frac{\partial g_{1,0}}{\partial \alpha} \quad \frac{\partial g_{2,0}}{\partial \alpha} \quad \frac{\partial g_{1,1}}{\partial \alpha} \quad \frac{\partial g_{2,1}}{\partial \alpha} \quad \frac{\partial g_{2,2}}{\partial \alpha}\right]$

Table 3 Summary of Operations for *RefFrame_tensor* and *RefFrame_tensor_partials*

The following example illustrates the use of the coordinate transformation routines. x is a position vector expressed in the RTN coordinate system at epoch t. The following code fragment demonstrates how the vector would be converted to the true of date coordinate system.

```
float64 x[3], a[9], b[9];
RefFrame frame = NULL;
RefFrame_create ( "EOP", CSR_EOP_FORMAT, & frame );
/*
user code
*/
RefFrame_calculate ( NO_SET_TOD, t, frame );
RefFrame_trueofdate_to_geocentric ( frame, a, b, );
RefFrame_geocentric_to_rtn ( pos, vel, b );
RefFrame_merge ( RIGHT, NO_TRANS, a, b );
RefFrame_vector ( TRANS, a, x );
```

2.3 Geopotential

Many mathematical representations of the geopotential have been suggested and the associated algorithms were investigated by Bettadpur [1993] in the context of high performance vector processors. The linear access patterns required for base function evaluations and series summations do not lend themselves to high performance on scalar microprocessors. However, the $\vartheta(l^2)$ computational cost where l is the maximum degree and order of the geopotential expansion comprises only lower order terms in the

overall cost formulation of the gravity field problem. No single processor optimization of the spherical harmonic synthesis will be examined, and the traditional spherical harmonic expression of the geopotential expressed in Equation (17) will be used.

$$U(r,\lambda,\phi) = \frac{\mu}{r}\sum_{l=0}^{\infty}\sum_{m=0}^{l}\left(\frac{a_e}{r}\right)^l \overline{P}_{n,m}(\sin\phi)\left[\overline{C}_{l,m}\cos\lambda + \overline{S}_{l,m}\sin\lambda\right] \tag{17}$$

The gravity field is fixed to the rotating Earth. The evaluation of the potential and its first and second partial derivatives occurs within the geocentric reference frame as expressed in spherical coordinates. Perturbing force and gradient operations require the expression of the potential directional derivatives in the cartesian frame. An effective conversion from the spherical coordinates to the topographic system is described by Bettadpur [1992]. The implementation of the conversion using BLAS operations is presented in Appendix B.

2.3.1 Legendre Associated Functions

The Legendre associated base functions must be evaluated before the summation of the spherical harmonic series. The Legendre associated functions and their derivatives are computed in terms of first and second order linear recursions. Lundberg [1986] recommends two computationally stable

algorithms for computing the Legendre associated functions to high degree and order. The first calculates column ordered recursions along increasing degree. The second calculates row ordered recursions along increasing order. The column ordered variant was selected since the column major ordering corresponds to ordering of matrix elements in FORTRAN. The recursion formulas are presented in Equation (18) and Equation (19) where l and m represent the degree and order of the function element.

$$
\begin{aligned}
\overline{A}_{l,l} &= \sqrt{\left(\frac{1}{2-\delta_{0,l-1}}\right)\left(\frac{2l+1}{l}\right)} \\
\overline{A}_{l+1,l} &= \sqrt{2l+3} \\
\overline{A}_{l,m} &= \sqrt{\frac{4l^2-1}{l^2-m^2}} \\
\overline{B}_{l,m} &= -\sqrt{\left(\frac{2l+1}{2l-3}\right)\left(\frac{(l-1)^2-m^2}{l^2-m^2}\right)} \\
\overline{F}_{l,m} &= \sqrt{\frac{2-\delta_{0,m}}{2}(l-m)(l+m+1)}
\end{aligned}
\tag{18}
$$

$$
\begin{aligned}
\overline{P}_{o,o}(\sin\phi) &= 1 \\
\overline{P}_{l,l}(\sin\phi) &= \overline{A}_{l,l}\cos\phi\,\overline{P}_{l-1,l-1}(\sin\phi) \\
\overline{P}_{l+1,l}(\sin\phi) &= \overline{A}_{l+1,l}\sin\phi\,\overline{P}_{l,l}(\sin\phi) \\
\overline{P}_{l,m}(\sin\phi) &= \overline{A}_{l,m}\sin\phi\,\overline{P}_{l-1,m}(\sin\phi) - \overline{B}_{l,m}\overline{P}_{l-2,m}(\sin\phi) \\
\frac{d\overline{P}_{l,m}(\sin\phi)}{d\phi} &= \overline{F}_{l,m}\overline{P}_{l,m+1}(\sin\phi) - m\frac{\sin\phi}{\cos\phi}\overline{P}_{l,m}(\sin\phi) \\
\frac{d^2\overline{P}_{l,m}(\sin\phi)}{d\phi^2} &= \overline{F}_{l,m}\frac{\sin\phi}{\cos\phi}\overline{P}_{l,m+1}(\sin\phi) - \\
&\quad \left[\overline{F}_{l,m}^2 + \frac{\left(m - m^2\sin^2\phi\right)}{\cos^2\phi}\right]\overline{P}_{l,m}(\sin\phi)
\end{aligned}
\tag{19}
$$

int32 Legendre_create	**(**	**int32**	**shape,**
		int32	**degree,**
		int32	**order,**
		int32	**deriv,**
		Legendre *	**legendre);**
int32 Legendre_free	**(**	**Legendre ***	**legendre);**

The **Legendre** object contains the information and algorithms necessary to evaluate the Legendre associated functions. The input argument **shape** specifies the choice of a triangular or trapezoidal representation of the gravity field model. The input arguments **degree** and **order** specify the maximum degree and order of the expansion. **order** is significant only for trapezoidal fields. The input argument **deriv** specifies the whether the first and second derivatives of the Legendre associated functions should be calculated.

```
int32 Legendre_calculate    ( float64    u,
                              Legendre   legendre );
```

The **Legendre_calculate** method calculates the Legendre associated functions according to the information contained in **legendre** and caches the data within the object for future use. The value **u** is the input argument of the functions. The data is retrieved via calls to the inquiry methods listed below and in Appendix C.

```
int32 Legendre_index    ( Legendre   legendre,
                          int32      l,
                          int32      m );
```

The **Legendre_index** method returns the linear array index for the degree *l* and order *m* element. This routine permits the user to access data elements contained in the objects without any knowledge of the internal mapping of the data elements.. For example, if **p** specifies the beginning address of the array containing the Legendre function evaluations, the value corresponding to degree *l* and order *m* would be accessed through the following code fragment.

```
    float64 * p = NULL;
    Legendre legendre = NULL;
    Legendre_create ( TRIANGULAR, 30, 30, NO_DERIVATIVES, & legendre );
    /*
    user code
    */
/* extract array containing computed values */
    Legendre_p ( legendre, & p );
/* value of l, m legendre function assigned to value */
    value = p [ Legendre_index ( legendre, l, m ) ];
```

2.3.2 Gravity Field Expansion

```
int32 GravityField_create      ( int32          shape,
                                 int32          degree,
                                 int32          order,
                                 int32          measurement,
                                 int32          coordinates,
                                 char *         filename,
                                 int32          file_format,
                                 GravityField * gfield );
int32 GravityField_free        ( GravityField * gfield );
```

The **GravityField** object contains the information and algorithms necessary to evaluate gravity field functions. The input argument **shape** specifies the choice of a triangular or trapezoidal representation of the gravity field model. The input arguments **degree** and **order** specify the maximum degree and order of the expansion. **order** is significant only for trapezoidal fields. The input argument **measurement** specifies which series summation to

calculate. The input argument **coordinates** specify the reference frame in which to return the calculated values. Gravity field parameters are read from **filename** with specified format **file_format**.

int32 GravityField_calculate	**(float64**	**radius,**
	float64	**lambda,**
	float64	**phi,**
	GravityField	**gfield);**

The **GravityField_calculate** method calculates the gravity field series summations according the information contained in **gfield** and caches the data within the object for future use. The geocentric location of the evaluation is specified by spherical coordinates **radius, lambda,** and **phi.** The data is retrieved via calls to the inquiry functions listed in Appendix C.

int32 GravityField_partial_length	**(GravityField**	**gfield,**
	int32	**estim_param,**
	int32 *	**length);**
int32 GravityField_extract_partials	**(GravityField**	**gfield,**
	int32	**which_set,**
	int32	**estim_param,**
	float64 *	**partials);**

The above **GravityField** methods facilitate the extraction of the partial derivatives with respect to the geopotential coefficients. The input argument **estim_param** specifies the subset of geopotential coefficients to be considered. The input argument **which_set** specifies the choice of gravity field summation

(potential, accelerations, or gradients) from which to extract the partial derivatives. The number of partial derivatives is returned in **length**. The partial derivatives are copied into the vector **partials** which must be at least **length** elements long.

2.4 Orbital Dynamics Model

The dynamics governing orbital motion can be represented as a system of ordinary differential equations. The satellite trajectory is obtained through the solution of the associated initial value problem. Each physical system effecting the satellite trajectory is represented by an additional acceleration term on the right hand side of the system of ODE's. The only force considered in this study is the gravitational accelerations associated with the non-spherical gravity field fixed to the non-uniformly rotating Earth.

$$\ddot{\vec{r}} = \mathrm{T}_{geocentric \rightarrow trueofdate} \nabla U(r, \lambda, \phi) \tag{20}$$

The motion of the pole due to precession and nutation is negligible over the arc length of the examined mission scenarios. Therefore, the true-of-date system corresponding to the initial arc epoch will be defined as the inertial reference frame. The gravity field accelerations defined in the TRF must be transformed to the true-of-date system to yield the correct

accelerations. The transformation is denoted by $T_{geocentric\_trueofdate}$ in Equation (20).

Trajectory propagation illustrates a complication in the development of object oriented routines for satellite applications. The satellite is an example of an abstraction which at first glance appears to exist concurrently as a physical object and as a mathematical object. As a physical object, the satellite comprises a part of the physical system. As a mathematical object, it appears the satellite is integrated according to the equations of motion. Upon closer look, the trajectory generation process involves not only the satellite but also requires the several different physical objects which define the equations of motion. Also, information recovered from the integration process such as the state transition information is purely mathematical and would not be considered a defining characteristic of the satellite object. An abstraction was chosen that allows for the existence of a purely mathematical integration object which derives part of its structure from the physical system. The remainder of this section presents the physical satellite object. A discussion of the integrator object will appear in the next chapter.

```
int32 Satellite_create      ( float64      epoch,
                              int32        coordinates,
                              float64 *    state,
                              Satellite *  satellite,
                              int32        number_forces,
                              ...          /* vargs */ );
int32 Satellite_free        ( Satellite *  satellite );
```

The **Satellite** object contains the information and algorithms necessary to manipulate the physical satellite object. The input argument **coordinates** specifies the input coordinates of the satellite position and velocity specified by **state** at the time index specified by **epoch**. The input argument **number_forces** specifies the number of physical objects which perturb the satellite trajectory. The list of physical objects completes the variable argument list. Valid physical objects must have their force property set to **TRUE**. Data contained within the **Satellite** object is retrieved via calls to the inquiry methods listed in Appendix C.

The following code fragment illustrates the creation of a polar orbiting satellite object moving under the influence of a non-spherical gravity field to degree and order 120.

```
int32           coordinates = ORBITAL_ELEMENTS;
float64         initial_epoch = J1990,
                initial_state [ 6 ] = {  7000.0e+0,
                                         1.0e-2,
                                         PI_OVER_2,
                                         0.0e+0,
                                         0.0e+0,
                                         0.0e+0 };
GravityField    gfield = NULL;
Satellite       satellite = NULL;

GravityField_create  ( TRIANGULAR,
                       120,
                       120,
                       ACCELERATION,
                       TOPOGRAPHIC,
                       "JGM3.GEO",
                       CSR_GEO_FORMAT,
                       & gfield );

Satellite_create  ( initial_epoch,
                    coordinates,
                    initial_state,
                    & satellite,
                    1, gfield );

/*
user code
*/

Satellite_free ( & satellite );
GravityField_free ( & gfield );
```

2.5 Observation Models

Two observation datatypes are used in this study. The first datatype is the satellite gradiometer observation which is sensitive to the medium to high frequency components of the gravity field. The second type is the GPS high-low satellite range observation which is sensitive to the low frequency components of the gravity field. The combination of the two datatypes

provides a complimentary set of observations for the recovery of a global geopotential model.

2.5 1 Satellite Gravity Gradiometry

A gradiometer measures of the spatial rate of change of acceleration. The satellite gradiometer observation is modeled mathematically by forming the acceleration difference between two points symmetric to the satellite center of mass [Rummel, 1989]. Linearization of the difference about the satellite center of mass yields the fundamental equation of the satellite gradiometer expressed in Equation (21). U represents the gradiometer potential, ω represents the instantaneous rotation vector of the satellite, δx represents the baseline of the accelerometers, and δa represents the gradiometer observation.

$$\delta\bar{a} = \left[-\nabla\nabla U\big|_{cm} + \dot{\bar{\omega}} \times (\cdot) + \bar{\omega} \times \left(\bar{\omega} \times (\cdot)\right)\right]\delta\bar{x} \tag{21}$$

A gradiometer instrument consists of pairs of linear accelerometers each of which are sensitive along a single axis. Combinations of the accelerometer pairs may be formed to generate measurements of the different spatial gradients.

Certain assumptions are made concerning the gradiometer model used in this study. The satellite is assumed to rotate such that the satellite reference

frame always aligns to the RTN reference frame. The gradiometer reference frame maintains a constant offset from the satellite frame according to user input Euler angles. The rotation effects associated with maintaining the alignment of the satellite with the RTN system are neglected.

int32 Gradiometer_create	**(**	**GravityField**	**gfield,**
		Satellite	**satellite,**
		float64	**psi,**
		float64	**theta,**
		float64	**phi,**
		Gradiometer *	**gradio);**
int32 Gradiometer_free	**(**	**Gradiometer ***	**gradio);**

The **Gradiometer** object contains the information and algorithms necessary to model a satellite gradiometer. The input argument **gfield** specifies the **GravityField** object to be observed by the gradiometer. The input argument **satellite** specifies the **Satellite** object upon which the gradiometer is mounted. The input arguments **psi, theta,** and **phi** specify the Euler angles by which the gradiometer is offset from the satellite reference frame.

int32 Gradiometer_calculate	**(**	**Gradiometer**	**gradio);**

The **Gradiometer_calculate** method calculates the satellite gradiometer observation according the information contained in **gradio** and caches the data

within the object for future use. The data is retrieved via calls to the inquiry methods listed in Appendix C.

2.5.2 Satellite-to-Satellite Ranging

The satellite tracking observable used in this study consists of simplified GPS measurement which returns an unbiased inter-satellite range measurement. The satellite range information allows the determination of differential accelerations acting on the high-low satellite system. The instantaneous slant range observation is expressed in Equation (22) where r represents the satellite state vector and x, y, z represent the inertial components of the state vector.

$$\begin{aligned} \rho(t) &= \left\| \bar{r}_{High}(t) - \bar{r}_{Low}(t) \right\| \\ \rho(t) &= \sqrt{\left(x_{High}(t) - x_{Low}(t)\right) + \left(y_{High}(t) - y_{Low}(t)\right) + \left(z_{High}(t) - z_{Low}(t)\right)} \end{aligned} \tag{22}$$

The recovery of geopotential information from the slant range observable requires the evaluation of the partial derivatives with respect to the initial satellite state and the geopotential coefficients. The derivation of the partials with respect to the initial satellite states is expressed in Equations (23) and (24).

$$\frac{\partial \rho(t)}{\partial \vec{r}_{High}(t_0)} = \frac{\partial \rho(t)}{\partial \vec{r}_{High}(t)} \frac{\partial \vec{r}_{High}(t)}{\partial \vec{r}_{High}(t_0)}$$
$$\frac{\partial \rho(t)}{\partial \dot{\vec{r}}_{High}(t_0)} = \frac{\partial \rho(t)}{\partial \vec{r}_{High}(t)} \frac{\partial \vec{r}_{High}(t)}{\partial \dot{\vec{r}}_{High}(t_0)} \tag{23}$$

$$\frac{\partial \rho(t)}{\partial \vec{r}_{Low}(t_0)} = \frac{\partial \rho(t)}{\partial \vec{r}_{Low}(t)} \frac{\partial \vec{r}_{Low}(t)}{\partial \vec{r}_{Low}(t_0)}$$
$$\frac{\partial \rho(t)}{\partial \dot{\vec{r}}_{Low}(t_0)} = \frac{\partial \rho(t)}{\partial \vec{r}_{Low}(t)} \frac{\partial \vec{r}_{Low}(t)}{\partial \dot{\vec{r}}_{Low}(t_0)} \tag{24}$$

The $\frac{\partial \vec{r}(t)}{\partial \vec{r}(t_0)}$ and $\frac{\partial \vec{r}(t)}{\partial \dot{\vec{r}}(t_0)}$ terms specify the changes in the satellite state given a small perturbation in the initial conditions. This corresponds to the state transition information between initial epoch t_0 and current epoch t.

$$\frac{\partial \vec{r}(t)}{\partial \vec{r}(t_0)} = \Phi_{rr}(t, t_0)$$
$$\frac{\partial \vec{r}(t)}{\partial \dot{\vec{r}}(t_0)} = \Phi_{rr}(t, t_0) \tag{25}$$

The derivation of partials with respect to the geopotential coefficients is expressed in Equations (26), (27) and (28).

$$\frac{\partial \rho(t)}{\partial \alpha} = \frac{\partial \rho(t)}{\partial \vec{r}_{High}(t)} \frac{\partial \vec{r}_{High}(t)}{\partial \alpha} + \frac{\partial \rho(t)}{\partial \vec{r}_{Low}(t)} \frac{\partial \vec{r}_{Low}(t)}{\partial \alpha} \tag{26}$$

$$\frac{\partial \rho(t)}{\partial \bar{r}_{High}(t)} = \frac{1}{\rho}\left[x_{High}(t) - x_{Low}(t) \quad y_{High}(t) - y_{Low}(t) \quad z_{High}(t) - z_{Low}(t)\right]$$
$$\frac{\partial \rho(t)}{\partial \bar{r}_{Low}(t)} = -\frac{1}{\rho}\left[x_{High}(t) - x_{Low}(t) \quad y_{High}(t) - y_{Low}(t) \quad z_{High}(t) - z_{Low}(t)\right] \tag{27}$$

$$\frac{\partial \rho(t)}{\partial \bar{\alpha}} = \frac{\partial \rho(t)}{\partial \bar{r}_{High}(t)}\left[\frac{\partial \bar{r}_{High}(t)}{\partial \bar{\alpha}} - \frac{\partial \bar{r}_{Low}(t)}{\partial \bar{\alpha}}\right] \tag{28}$$

The $\frac{\partial \bar{r}(t)}{\partial \bar{\alpha}}$ term specifies the change in the satellite state given a small perturbation in the model parameters. This corresponds to the state transition information between initial epoch t_0 and current epoch t.

$$\frac{\partial \bar{r}(t)}{\partial \bar{\alpha}} = \Phi_{r\alpha}(t, t_0) \tag{29}$$

The expression for the partials derivatives of the slant range observations with respect to the geopotential coefficients, $\bar{\alpha}$, is given in Equation (30). The evaluation of the range partials is expressed in the inertial reference frame.

$$\frac{\partial \rho(t)}{\partial \bar{\alpha}} = \frac{\partial \rho(t)}{\partial \bar{r}_{High}(t)}\left[\Phi_{r_{High}\alpha}(t, t_0) - \Phi_{r_{Low}\alpha}(t, t_0)\right] \tag{30}$$

The perturbations of the range observable are caused by errors in the initial conditions as well as the error in the geopotential coefficients. Both sets of parameters must simultaneously be estimated to recover the best update

of the geopotential. However, information derived from the high satellite provides only a small contribution. The results presented in Chapter 7 neglect the contribution of the high satellite.

int32 SatTrack_create	**(float64**	**elevation_mask,**
	MsilibABFS	**integ_high,**
	MsilibABFS	**integ_low,**
	SatTrack *	**sst);**
int32 SatTrack_free	**(SatTrack ***	**sst);**

The **SatTrack** object contains the information and algorithms necessary to calculate the satellite-to-satellite tracking observable. The input argument **elevation_mask** specifies the elevation mask with respect to the low satellite below which no range measurements may be taken. The input arguments **integ_high** and **integ_low** specify the numerical integration objects associated with the two satellites.

int32 SatTrack_calculate	**(SatTrack**	**sst);**

The **SatTrack_calculate** method calculates the high-low satellite tracking observation according the information contained in **gradio** and caches the data within the object for future use. The data is retrieved via calls to the inquiry methods listed in Appendix C.

3. Mathematical Environment

The mathematical environment of the satellite application consists of the operations which manipulate satellite states and recover model parameters. A discussion of the least squares method as applied to satellite applications and the numerical methods used to propagate the satellite trajectory will be presented.

3.1 Mathematical Objects

Mathematical objects abstract in a generic manner the mathematical techniques used to manipulate data derived from the physical system. Each mathematical object shares a common set of properties which specify the functionality of the object. The function property specifies the general mathematical technique. The method property (not to be confused with an object method) specifies the specific manner in which the function is implemented. Object property values are set at object creation and may be accessed by the appropriate inquiry method.

All mathematical objects require a two step creation/realization process. The creation method produces an instance of the object according to the parameters describing the mathematical technique. The realization method associates the mathematical object with appropriate physical objects. In the

following discussion of mathematical techniques implemented in this research, the associated mathematical classes and a sampling of their methods will be described. A complete list of all object methods as proposed by this research is provided in Appendix C.

3.2 Least Squares Estimation

The least squares estimation process recovers physical model parameters which best fit a set of true observations. The estimation process begins with the design of a physical system model which approximates the true physical system. The observations are recreated in the simulated world and compared to the true observations. Observational and dynamic parameters are adjusted to minimize the sum of the squares of the observation residuals. Non-linear physical models require an iterative adjustment until a specified convergence is satisfied. Observation residuals may also be examined to identify systematic errors which may then be incorporated into the physical model.

Typical physical processes require complex, non-linear models. The estimation problem is simplified by converting the non-linear problem into a linear problem. The linearization process begins by representing the true physical system by a system of non-linear ordinary differential equations. $\mathcal{X}^{True}$

is partitioned into subvectors of dynamic variables and dynamic model parameters.

$$\begin{aligned} \dot{\mathcal{X}}^{True} &= \mathcal{F}^{True}\left(\mathcal{X}^{True}, t\right) \\ \mathcal{X}^{True} &= \begin{bmatrix} X^{True} \\ \alpha^{True} \end{bmatrix} \\ \mathcal{F}^{True}\left(\mathcal{X}^{True}, t\right) &= \begin{bmatrix} F\left(X^{True}, t; \alpha^{True}\right) \\ 0 \end{bmatrix} \end{aligned} \tag{31}$$

The non-linear system may be converted into a linear system through the introduction of a reference system of similar form.

$$\begin{aligned} \dot{\mathcal{X}} &= \mathcal{F}(\mathcal{X}, t) \\ \mathcal{X} &= \begin{bmatrix} X \\ \alpha \end{bmatrix} \\ \mathcal{F}(\mathcal{X}, t) &= \begin{bmatrix} F(X, t; \alpha) \\ 0 \end{bmatrix} \end{aligned} \tag{32}$$

The dynamics of the true system are expanded in a Taylor series about the reference system. If the two systems are sufficiently close in space and time, the higher order terms of the expansion may be neglected.

$$\dot{\mathcal{X}}^{True}(t) = \mathcal{F}(\mathcal{X}, t) + \frac{\partial \mathcal{F}(\mathcal{X}, t)}{\partial \mathcal{X}}\left(\mathcal{X}^{True} - \mathcal{X}\right) \tag{33}$$

The difference between the dynamic systems $x = \mathcal{X}^{True} - \mathcal{X}$ may now be expressed in terms of a linear system of ordinary differential equations.

$$\dot{x}(t) = A(\mathcal{X},t)x(t)$$
$$A(\mathcal{X},t) = \frac{\partial \mathcal{F}(\mathcal{X},t)}{\partial \mathcal{X}} = \begin{bmatrix} \frac{\partial F(X,t;\alpha)}{\partial X} & \frac{\partial F(X,t;\alpha)}{\partial \alpha} \\ 0 & 0 \end{bmatrix} \tag{34}$$

This homogeneous ODE possesses a solution which may be formulated in terms of the state transition matrix.

$$x(t) = \Phi(t,t_0)x(t_0)$$
$$\Phi(t,t_0) = \begin{bmatrix} \frac{\partial X(t)}{\partial X(t_0)} & \frac{\partial X(t)}{\partial \alpha} \\ 0 & \mathrm{I} \end{bmatrix} \tag{35}$$

The solution of the state transition matrix is derived through substitution into Equation (34). The resulting ODE is usually evaluated by numerical integration methods.

$$\dot{\Phi}(t,t_0) = A(\mathcal{X},t)\Phi(t,t_0)$$
$$\dot{\Phi}(t_0,t_0) = \mathbf{I} \tag{36}$$

The observation-state relationship in Equation (37) may be linearized in a similar manner to produce the set of linear observation equations in Equation (38). The value ε_i represents combined effect of all systematic and random errors in the measurement.

$$Y_i^{True} = G^{True}\left(\mathcal{X}^{True}(t_i);\beta^{True}\right) + \varepsilon_i'^{True}$$
$$Y_i = G\left(\mathcal{X}(t_i);\beta\right) + \varepsilon_i' \tag{37}$$

$$y_i = \tilde{H}\left(\chi(t_i);\beta\right)x(t_i) + \varepsilon_i$$
$$\tilde{H}\left(\chi(t_i);\beta\right) = \frac{\partial G\left(\chi(t_i);\beta\right)}{\partial \chi} \tag{38}$$

The expression for the observation residuals y_i in Equation (38) is expressed in terms of the state differences at epoch t_i. Using Equation (35) the observation equations may be formulated in terms of the state differences at initial epoch t_0.

$$y_i = \tilde{H}\left(X(t_i);\beta\right)\Phi(t_i,t_0)x(t_0) + \varepsilon_i$$
$$y_i = H\left(X(t_i);\beta\right)x(t_0) + \varepsilon_i \tag{39}$$

Finally, the least squares minimization of the observational residual results in the well-known relationship for the state difference update.

$$\hat{x} = \left(H^T H\right)^{-1} H^T y \tag{40}$$

Equation (40) represents the normal equation formulation of the least squares solution. The solution may also be formulated in terms of orthogonal transformations which yield greater precision benefits. Two reasons motivate the use of normal equations over orthogonal transformations. First, the parallel implementation of the normal equation expression is simpler than that of the orthogonal transform. Secondly, different sets of normal equations are quickly combined via an $\vartheta(n^2)$ addition operation as compared to an $\vartheta(n^3)$ accumulation operation for orthogonal transformations.

As described in Section 2.5.2, the geopotential information derived from the satellite tracking measurement requires the estimate of the low satellite intial conditions in addition to the geopotential coefficients. A geopotential estimate including the contribution due to the initial state adjustment at each subarc does not require an explicit state estimate. Appendix J presents the methodology used to form the geopotential covariance from satellite tracking information.

int32 Batch_create	**(int32** **Batch ***	**batch_size,** **estimator);**
int32 Batch_free	**(Batch ***	**estimator);**

The **Batch** object contains the information and algorithms necessary to calculate the normal equation formulation of the least squares estimate. The input argument **batch_size** specifies the number of observation equations to cache before performing the accumulation operation. As described in Section 1.3.4.2, the creation routine only initializes an instance of the **Batch** object. The initialization is completed via a call to the realization routine.

int32 Batch_initialize_parameters	**(Batch**	**estimator,**
	int32	**n_parm_types,**
	...	**/* vargs */);**

The **Batch_initialize_parameters** method informs the **Batch** object of the parameters to be estimated. The input argument **n_parm_types** specifies the number of different dynamic and observation parameter types. The list of parameter types and subarc information (if necessary) completes the variable argument list. The method must be invoked prior to the realization method. Logically, the functionality is part of the object realization and should be combined with the realization method, but language complexities associated with the variable argument list contribute to the use of a separate method.

int32 Batch_realize	**(Batch**	**estimator,**
	int32	**which_obs,**
	PhysObj	**observation,**
	float64	**sigma);**

The **Batch_realize** method extract information from the simulated environment to complete the realization process. The input argument **which_obs** specifies the type of physical object. The physical object must have its observable property set to **TRUE**. The input argument **sigma** specifies the *a priori* variance of the observation.

int32 Batch_increment_subarc	**(Batch**	**estimator);**
int32 Batch_accumulate	**(Batch PhysObj**	**estimator, observation);**
int32 Batch_solve	**(Batch**	**estimator);**

The above **Batch** methods provide the functionality of the **Batch** object. The **Batch_increment_subarc** methods informs the **Batch** object to begin processing the next subarc. The **Batch_accumulate** method extracts the current observation residual and partial derivative information from the physical object **observation**. The normal equations will not be immediately updated, but rather **batch_size** number of partials will be cached inside the object to permit the use of a matrix-matrix multiplication. The **Batch_solve** method performs the linear system solve of the normal equations and caches the information within the object. Data contained within the **Batch** object is retrieved via calls to the inquiry methods listed in Appendix C.

3.3 Numerical Integration

The propagation of a satellite trajectory requires the numerical integration of the satellite's equations of motion. Many different methods of numerical integration exist with each method possessing certain computational cost and numerical accuracy attributes. Of the different methods, multistep integration routines have been shown to be ideally suited

to the smooth trajectories associated with satellite motion [Lundberg, 1981]. The multistep methods require knowledge of the integration state values at multiple epochs in order to advance the solution. Lundberg [1981] describes many different implementations of the multistep method as applied to the satellite propagation problem.

The Center for Space Research numerical integration software library MSILIB [Lundberg, 1991] provides the numerical integration functionality for this research. The library implements four different multistep methods. Of the methods, the Adams-Bashforth-Moulton algorithm integrates the satellite equations of motion and the state transition matrix. The Adams-Bashforth-Moulton algorithm is a Class I formulation which integrates first order differential equations. The MSILIB Class II formulations operate directly on second order differential equations providing greater accuracy for long integration intervals. The familiarity with first order systems of equations influenced the choice of a Class I formulation. The MSILIB library was written in FORTRAN. A numerical integration object written in C calls the library to perform the numerical integration.

int32 MsilibABFS_create	**(int32**		**state_transition,**
	int32		**nord,**
	int32		**start_nloop,**
	float64		**start_alim,**
	float64		**mesh_size,**
	MsilibABFS *		**integrator);**
int32 MsilibABFS_free	**(MsilibABFS ***		**template);**

The **MsilibABFS** object contains the information and algorithms necessary to propagate a satellite trajectory and calculate satellite state transition information. The input argument **state_transition** specifies the computation of state transition information. The input argument **nord** specifies the order of the multistep integration. The input argument **start_nloop** specifies the maximum number of iterations in the starting procedure, and the input argument **start_alim** specifies the starting convergence. The input argument **mesh_size** specifies the spacing in the multistep mesh. The creation routine only initializes an instance of the **MsilibABFS** object. The initialization is completed via a call to the realization routine.

int32 MsilibABFS_initialize_parameters	**(MsilibABFS**		**integrator,**
	int32		**n_parm_types,**
	...		**/* vargs */);**

The **MsilibABFS_initialize_parameters** method informs the **MsilibABFS** object of the parameters against which state transition information should be calculated. The input argument **n_parm_types** specifies the number of different

dynamic and observation parameter types. The list of parameter types completes the variable argument list. The method must be invoked prior to the realization method. Logically, the functionality is part of the object realization and should be combined with the realization method, but language complexities associated with the variable argument list contribute to the use of a separate method.

```
int32 MsilibABFS_realize    ( MsilibABFS    integrator,
                              int32         nsats
                              ...           /* vargs */ );
int32 MsilibABFS_free       ( MsilibABFS *  integrator );
```

The **MsilibABFS_realize** method extract information from the simulated environment to complete the realization process. The input argument **nsat** specifies the number of physical objects to be propagated. The list of physical objects complete the argument list. The physical objects must have their dynamic properties set to **TRUE**.

```
int32 MsilibABFS_propagate  ( MsilibABFS    integrator,
                              float64       tout );
int32 MsilibABFS_restart    ( MsilibABFS    integrator );
```

The above **MsilibABFS** methods provide the functionality for the **MsilibABFS** object. The **MsilibABFS_propagate** method integrates the trajectory according to the forces associated with the satellite objects. The input

argument **tout** specifies the final epoch of the integration with respect to the initial epoch of the satellite state. The **MsilibABFS_restart** method performs an integration restart operation and reinitializes the state transition matrix to identity. Data contained within the **MsilibABFS** object is retrieved via calls to the inquiry methods listed in Appendix C.

4. Computational Environment

The main focus of this research is the examination of parallel computational techniques to improve the performance of precision orbit determination and geopotential recovery applications. The development of parallel techniques requires the examination of the various layers of the computer architecture. The first issue concerns performance issues on scalar processors. The next layer directly address parallel computing and the development of high performance accumulation and matrix factorization algorithms. Finally, the parallel methods will be extended to out-of-core processing techniques. The discussion in this chapter will conclude with an examination of data-parallel implementation methods for numerical integration algorithms.

4.1 Scalar Processor Performance Issues

A typical serial architecture consists of two main components. The central processing unit (CPU) performs the work, and the main memory (RAM) contains the data. The communication bandwidth between the CPU and RAM on the current generation of microprocessors is insufficient to sustain peak performance [Astfalk, 1990]. A significantly faster memory, or cache, provides a temporary storage location for data. Algorithms structured

to exploit cache memory experience significant performance improvements. Cache reuse is accomplished in linear algebra operations by forming block partitioned algorithms. The block algorithms treat the matrices in terms of two-dimensional submatrices instead of collections of row and column vectors. Block algorithms promote maximum cache reuse by minimizing data traffic between the cache and main memory [Dongarra, 1990].

The management of communication between RAM, cache and the CPU cannot generally be controlled from a high level language such as C or FORTRAN. High performance algorithms are realized through the use of highly optimized low level subroutines. The Basic Linear Algebra Subprograms (BLAS) comprise the computational primitives for linear algebra operations. Three different levels of the BLAS exist. The Level 1 BLAS are vector-vector operations which perform $\vartheta(n)$ operations on $\vartheta(n)$ data elements. The Level 2 BLAS are matrix-vector operations which perform $\vartheta(n^2)$ operations on $\vartheta(n^2)$ data elements. The Level 3 BLAS are matrix-matrix operations which perform $\vartheta(n^3)$ operations on $\vartheta(n^2)$ data elements. The Level 3 BLAS operations manipulate data in a manner which exploits cache memory.

```
BLAS Level One

xCOPY ( n,          x,  incx,  y,  incy  )
xSCAL ( n,  alpha,  x,  incx             )
xAXPY ( n,  alpha,  x,  incx,  y,  incy  )


BLAS Level Two

xGEMV  (       trans,        m, n,  alpha,  a,  lda,  x,  incx,  beta,  y,  incy  )
xSYMV  ( uplo,                  n,  alpha,  a,  lda,  x,  incx,  beta,  y,  incy  )
xTRMV  ( uplo, trans,  diag,    n,          a,  lda,  x,  incx                    )
xTRSV  ( uplo, trans,  diag,    n,          a,  lda,  x,  incx                    )
xGER   (                      m, n,  alpha,  x,  incx,  y,  incy,  a,  lda  )
xSYR   ( uplo,                  n,  alpha,  x,  incx,             a,  lda  )
xSYR2  ( uplo,                  n,  alpha,  x,  incx,  y,  incy,  a,  lda  )


BLAS Level Three

xGEMM  (             transa,  transb,        m, n,  k,  alpha,  a,  lda,   b,  ldb,   beta,  c,
        ldc     )
xSYMM  ( side,  uplo,              m,    n       alpha,      a, lda,  b, ldb,  beta,  c,  ldc )
xSYRK  (   uplo,  trans,                 n,  k,  alpha,      a, lda,          beta,  c,  ldc )
xSYR2K (   uplo,  trans,                 n,  k,  alpha,      a, lda,  b, ldb,  beta,  c,  ldc )
xTRMM  ( side,  uplo,  transa,       diag,  m,  n,       alpha,  a,  lda,  b,  ldb                )
xTRSM  ( side,  uplo,  transa,       diag,  m,  n,       alpha,  a,  lda,  b,  ldb                )
```

Figure 4 Basic Linear Algebra Subprograms for Real Valued Operations

4.1.1 Batch Filter Implementation

The following discussion illustrates the implementation of the BLAS within the precision orbit determination batch algorithm. A single iteration of the batch algorithm in a manner similar to Tapley is presented in Figure 5. A number of opportunities exists to exploit the BLAS primitives. The mapping of the partial vector, the accumulation of the partial information and the linear system solve all require dense linear algebra operations. In addition, implicit operations such as those found in the right hand side of the dynamic model may also incorporate dense linear algebra operations. As the number of

estimated parameters increases, the cost of accumulation and linear system solve dominate the total cost of the batch algorithm.

Initialize at t_0
While (observations remain and $t_i < t_f$)
    Read observation at t_i
        If all observations have been processed, then break
    Integrate reference trajectory to t_i
    Form $H_i = \tilde{H}_i \Phi(t_i, t_0)$
    Accumulate $\sum_i H_i^T R_i^{-1} H_i, \sum_i H_i^T R_i^{-1} y_i$
    Increment $t_i = t_i + 1$
End while
Solve normal equations

Figure 5 Tapley's Batch Least Squares Algorithm

4.1.1.1 Mapping of the Partial Vector

The mapping of the partial vector to the initial epoch requires the transformation of the partial vector at the current epoch. This operation consists of a matrix-vector multiply (**xGEMV**). The BLAS routine is written in terms of the operation $y \leftarrow \alpha \mathbf{A}x + \beta y$ where the vectors are considered column-oriented. Hence, the necessity of the transpose operation as presented in Figure 6.

4.1.1.2 Accumulation of the Normal Matrix

The accumulation of the normal equations requires the addition of information to the normal matrix and normal equation right-hand-side. The normal matrix is updated via a rank-1 update (**xGER**) of the partial vector. The

rank-1 update is a Level 2 BLAS operation. Since the normal matrix is symmetric, the symmetric variant of the rank-1 update (**xSYR**) which updates only the lower or upper portion of the matrix may be used. The normal equation right-hand-side is updated through the summation of a scaled vector (i.e., $y \leftarrow \alpha x + y$). The *axpy* operation (**xAXPY**) is a Level 1 BLAS operation. As mentioned previously, high performance on scalar processors requires the implementation of Level 3 BLAS operations. Therefore, the best performance would not be expected from the algorithm as stated.

The algorithm may be converted to Level 3 BLAS operations by forming batches of observations. The symmetric rank-1 operation becomes a Level 3 symmetric rank-k (**xSYRK**) operation. The axpy operation becomes a Level 2 matrix-vector multiply (**xGEMV**). The incorporation of these modifications into the batch accumulation algorithm requires only the addition of an **if** statement as shown in Figure 6.

4.1.1.3 Linear System Solve

The accumulation of the observation equations creates a large, dense linear system which must be solved to recover the state update. Assuming sufficient observability of the estimated parameters, the system may always be assumed positive definite [Golub, 1980]. The preferred method of solution requires the Cholesky factorization of the normal matrix and two triangular

system solves. The development of the Cholesky factorization is presented in Appendix F. The solution of the linear system may be easily accomplished once the matrix factorization is complete. The original system is grouped in a manner which leads to a sequence of Level 2 BLAS triangular system solves (**xTRSV**) as seen in Equation (41).

$$\begin{aligned} L\left(L^T x\right) &= y \\ Lz &= y \\ L^T x &= z \end{aligned} \tag{41}$$

The linear system solve algorithm may be extended to the inversion of the normal matrix for the purposes of obtaining the variance-covariance information. The goal of the inversion is the recovery of a matrix that when multiplied by the normal matrix yields the identity matrix. The search for this matrix leads to the construction of a linear system which consists of the identity matrix on the right-hand-side. The determination of the covariance matrix follows a sequence of Level 3 BLAS triangular system solves with multiple right-hand-sides (**xTRSM**) as seen in Equation (42).

$$\begin{aligned} L\left(L^T P\right) &= I \\ LZ &= I \\ L^T P &= Z \end{aligned} \tag{42}$$

Figure 6 presents the batch algorithm including the Level 3 BLAS.

```
Initialize at t_0

Do while (observations remain and t_i < t_f)
     Read observation at t_i
     Integrate reference trajectory to t_i

     !
     ! Accumulate a batch at a time
     !
     if (batch is full) then
          xSYRK ( "L", "N", n, m, 1.0e+0, H, ld_h, 1.0, HtH, ld_hth )
          xGEMV ( "N", m, n, 1.0, H, ld_h, y, 1, 1.0, Hty, 1 )
     end if

     !
     ! map partials to initial epoch
     !
     xGEMV ( "N", n, n, 1.0, PHI, ld_phi, Htilde, 1, 0.0, H(i,1), ld_h )

     Increment t_i = t_i+1

End while

!
! Accumulate incomplete batches
!
xSYRK ( "L", "N", n, i, 1.0e+0, H, ld_h, 1.0, HtH, ld_hth )
xGEMV ( "N", i, n, 1.0, H, ld_h, y, 1, 1.0, Hty, 1 )

!
! Solve normal equations
!
xCHOLESKY ( n, HtH, ld_hth )
xTRSV ( "L", "N", "N", n, HtH, ld_hth, Hty, 1 )
xTRSV ( "L", "T", "N", n, HtH, ld_hth, Hty, 1 )

!
! Invert normal matrix
!
set_identity ( n, P, ld_p )
xTRSM ( "L", "N", "N", "N", n, n, 1.0, HtH, ld_hth, P, ld_p )
xTRSM ( "L", "N", "T", "N", n, n, 1.0, HtH, ld_hth, P, ld_p )
```

Figure 6 Level 3 BLAS Modifications to Tapley's Batch Algorithm

4.2 Parallel Performance Issues

Parallel processing is the distribution of computational work among a collection of processors for the purpose of increased computational performance. Many types of parallel processors exist. Many users are familiar with the shared memory vector supercomputers such as the Cray Y-MP and Cray T90. These architectures consist of a relatively small number of very powerful processors which share a common memory unit. A second type of parallel architecture consists of a larger number of independent processor-memory subsystems connected by a high-speed network. Examples of distributed memory machines include the Cray T3E and the Intel Paragon.

The shared memory architectures offer a simple model for parallel algorithm design as data can be exchanged between processors quite easily through the common memory. Unfortunately, shared memory architectures also share a common communications bus. The capacity of the bus imposes a practical limit to the number of processors which exist on the system. As a result, improved aggregate performance on shared memory architectures depend primarily on performance gains on the individual processors [Astfalk, 1990].

Distributed memory architectures avoid memory access problems by physically distributing memory units to each of the processing elements. The

size of the network can increase without affecting individual processor performance and, theoretically, may consist of an arbitrarily large number of processors. The aggregate performance improves by simply adding more processors.

Communication between processors performed via common memory on shared memory architectures must now be performed over a network on the distributed memory architectures. The cost of communication between processors is significantly larger than the cost of moving data between main memory and the CPU. In addition, network conflicts also degrade performance. An effective algorithm design is required to organize and minimize inter-processor communication in the distributed memory environment.

From the point of view of the individual processor, the memory on remote processors may be treated as an additional level in its local memory hierarchy. The use of block algorithms would appear to be the natural method of minimizing data traffic and achieving high performance on distributed memory architectures. Block algorithms yield the desired result. However, the complexity of the parallel implementations increases greatly when moving from a serial to parallel block algorithm. The application programmer is responsible for the distribution and communication of data in the parallel

implementation. The management of the communication requires a complete understanding of the data movement and an effective programming paradigm.

Message passing is one of the more common programming paradigms on distributed memory architectures. Under message passing, communication between processors occurs in the form of messages. The source processor executes a send operation to initiate the communication, and the destination processor executes a receive operation to complete the transmission of data. The send and receive operations are not required to be synchronous, or blocking. Asynchronous, or non-blocking, message passing potentially hides the communication cost by overlapping the communication operation with useful computations. For example, a non-blocking receive may be posted prior to the corresponding send operation. The destination processor continues performing useful work until the communication completes.

Collective communication provides a higher level set of instructions to direct the movement of data. Parallel linear algebra algorithms in particular benefit from the use of collective communication operations.

Collective Operation	MPI Routine	Description
Barrier	MPI_Barrier	Blocks execution until all processors in the group have reached the barrier point
Broadcast	MPI_Bcast	Information residing on a single processor is copied to all processors
Gather	MPI_Gather MPI_Gatherv	Information distributed across all processors is copied to a single processor
Scatter	MPI_Scatter MPI_Scatterv	Information residing on a single processor is distributed across all processors
Collect	MPI_Allgather MPI_Allgatherv	Information distributed across all processors is copied to all processors
Reduce-to-One	MPI_Reduce	Element-wise reduce operation performed on information residing on all processors leaving the result on a single processor
Reduce-to-All	MPI_Allreduce	Element-wise reduce operation performed on information residing on all processors leaving the result on all processors
Distributed Reduce	MPI_Reduce_scatter	Element-wise reduce operation performed on information residing on all processors leaving the result distributed across all processors

Table 4 Collective Communication Operations

4.2.1 Parallel Linear Algebra

Dense linear algebra operations require an order of magnitude greater number of computations to number of data elements communicated. The situation presents a significant opportunity to exploit parallel resources. Two competing paradigms currently dominate implementations of parallel dense linear algebra. Block cyclic distributions provide the basis for implementation such as High Performance FORTRAN and ScaLAPACK. Physically Based Matrix Distribution (PBMD) forms the paradigm for the PLAPACK package.

Block cyclic distributions start with the assignment of the $\vartheta(n^2)$ data elements associated with the linear operator across a processor array. For scalability reasons, the processor array is organized in a two-dimensional grid layout [Dongarra, 1994]. The simplest method is a blocked distribution where the matrix is partitioned into r row blocks and c column blocks. The block (i,j) is assigned to processor (i,j). This distribution suffers from load balancing problems for triangular and banded matrices as some processors may not receive any data. A finer blocking of the matrix in row and column directions and a wrapping of the blocks around the processor grid provides an effective means of load balancing.

The block cyclic distribution possesses inherent difficulties interfacing with application software. The distribution of the domain and range spaces are imposed by the distribution of the linear operator. If the generation of the domain elements and/or the recovery of the range elements proceed more effectively with different distribution, a significant incompatibility is created between the application and the parallel linear algebra routine. Bridging this incompatibility may require a significant coding effort and significant degradation in performance due to data redistribution.

Block cyclic distributions are inherently complex to implement because of a lack of natural communication structures. Communication

operations involve the movement of matrix blocks between the rows and columns of processors. The developer must manage the message passing and maintain book-keeping of block indices and offsets. One parallel linear algebra library, ScaLAPACK, attempts to encapsulate communication details within the Basic Linear Algebra Communication Subprograms (BLACS). The block mapping and buffer space management remains exposed yielding a very complex and intricate piece of software.

Physically Based Matrix Distribution (PBMD) presents an alternative parallel linear algebra distribution paradigm which avoids many of the problems of block cyclic. PBMD begins by addressing the issue of compatibility between the application software and the linear algebra library. Under PBMD, the user distributes elements of the one-dimensional domain and range space over the two-dimensional array of processors to suit the application. The resulting vector distributions induce the distribution of the linear operator in a manner to yield a highly effective parallel linear algebra implementation.

The parallel implementation of the matrix-vector multiplication illustrates the principles of PBMD (See Figure 7). Each element of the domain space x multiplies a column of the operator A. Similarly, each element of the range space y is the result of a summation across the rows of

operator A. Therefore, a relationship exists between the columns of A and the element of x and the rows of A and the elements of y. Given a distribution of x and y on the processor array, A should be distributed in a manner which naturally brings together the columns of A with the appropriate elements of x and the rows of A with the appropriate elements of y. This distribution is identified by projecting the indices of the domain vector x in the column direction and the indices of the range vector y in the row directions. The projected indices specify the location of the matrix row or column block. The matrix-vector multiplication can be stated quite elegantly in terms of collective communication operations.

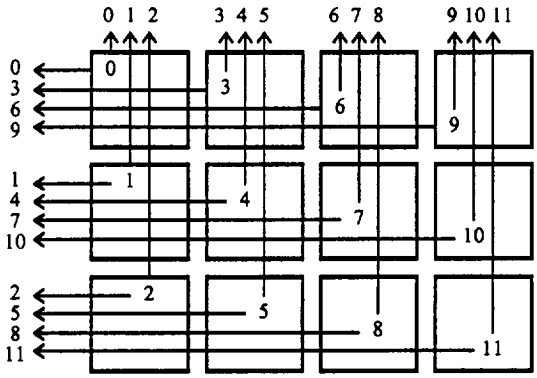
The matrix-vector multiplication $y \leftarrow Ax$ may be written as the summation,

$$y = A_0 x_0 + A_1 x_1 + \cdots + A_i x_i \tag{43}$$

where A_i is the i^{th} column of A. This form exposes the relationship between the columns of A and the elements of x. Likewise, the operation may be written as,

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_j \end{bmatrix} = \begin{bmatrix} A_{0,0}x_0 + A_{0,1}x_1 + \cdots + A_{0,i}x_i \\ A_{1,0}x_0 + A_{1,1}x_1 + \cdots + A_{1,i}x_i \\ \vdots \\ A_{j,0}x_0 + A_{j,1}x_1 + \cdots + A_{j,i}x_i \end{bmatrix} \tag{44}$$

This form exposes the relationship between the rows of A and the elements of y. Suppose that x and y have been divided into an equal number of partitions and those partitions have been distributed identically across a 3x4 array of processors (i.e., partition x_i is located on the same processor as y_i). The matrix distribution is the defined by the projection of the block indices in the row and column direction.



The algorithm for the parallel matrix-vector multiplication may be stated as,

Collect x in columns to $\tilde{x}$
Perform the local matrix-vector multiply $\tilde{y} \leftarrow A_{i,j}\tilde{x}$
Distributed reduce $\tilde{y}$ in rows to y

Figure 7 Matrix-Vector Multiplication Under PBMD

A significant characteristic of PBMD is the natural communication structure implicitly defined by the distributed vector. The communication structure consists of data mappings related by collective communication operations. The vector itself comprises one mapping. The projection of the vector onto a single row or column of processors comprises another mapping named a projected vector, or pvector. A duplicated projected vector, or dpvector, mapping is constructed by the simultaneous existence of pvectors on every row or column of processors. The relations between these fundamental objects are shown in Figure 8.

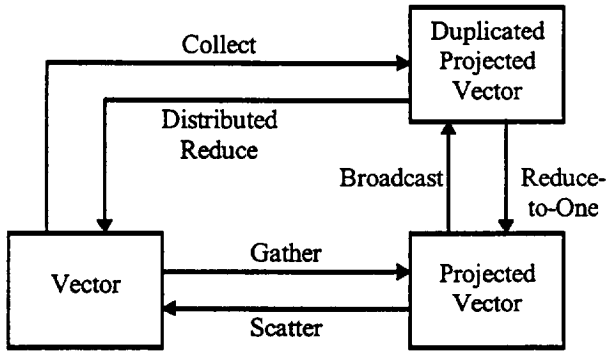


Figure 8 Communication Structure Between Vector Derived Objects

Classes of two-dimensional data objects may be developed if the vector object is permitted to possess a width greater than one. The multivector may be viewed as a collection of vector objects of equal length and distributed identically. The projection of the multivector yields a projected multivector.

Duplicated projected multivectors consists of projected multivectors simultaneously existing on all rows or columns of processors. Finally, the matrix distribution is determined by the projection of vector objects in the processor row and column directions as previously described.

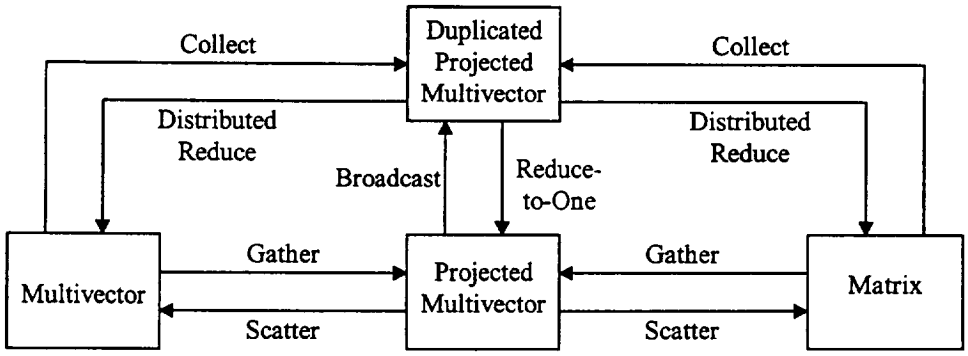


Figure 9 Communication Structure Between Multivector Derived Objects

4.2.2 PLAPACK Implementation

The PLAPACK parallel linear algebra library primarily consists of an object oriented infrastructure to describe and manage parallel linear algebra object distributions. PBMD describes the distribution of data. The Release 1.0 implementation possesses only a single inducing vector. The inducing vector consists of constant size vector partitions distributed across the array of processors in column-major order. Linear algebra objects encapsulate the details of the distribution. The object-oriented nature of the packages allows the development of high level routines which hide virtually

all the parallel implementation details. However, through the manipulation of different data objects and in-lining functionality, the experienced programmer may optimize performance to a considerable degree. The remainder of the section provides a summary of the PLAPACK package. van de Geijn [1997] describes the PLAPACK package in its entirety.

The typical PLAPACK user interacts with linear algebra objects through the manipulation of object views. An object view corresponds directly to the blocking specified in linear algebra block algorithms. As a result, the PLAPACK code exhibits a one-to-one correspondence to the natural expression of block algorithms as described in classroom setting. PLAPACK facilitates code development even further by incorporating view manipulation routines which discourage the use of explicit indexing. Users specify views according to partitions of larger linear algebra objects. The PLAPACK algorithms step through the entire object by sliding a view across the object or by recursively splitting a global view into even smaller partitions.

The different object types available in PLAPACK correspond to those described in the previous section. A single communication routine, **PLA_Copy**, performs all inter-processor data movement for PLAPACK. **PLA_Copy** identifies the correct collective operation based on the object types and

performs the optimum communication without any additional information required from the programmer.

4.2.3 Cholesky Factorization

The parallel Cholesky factorization implements the left-looking variant of the algorithm as presented in Appendix F. The following description of the parallel algorithm refers to Figure 10.

Lines 12-21	Creates the multiscalar scale parameters required by the BLAS function calls.
Line 22	Initialize the matrix views to **a_done** and **a_next**. At any point in the algorithm **a_done** consists of the matrix block $A_{1,0}$ and $A_{2,0}$. **a_next** consists of the matrix blocks $A_{1,1}$, $A_{1,2}$, $A_{2,1}$ and $A_{2,2}$.
Lines 25-30	Determine the size of matrix block $A_{1,1}$ which can completely exist on a single processor. If **size** is equal to zero, the factorization is complete. Otherwise, create views into the appropriate matrix blocks.
Lines 31-39	Perform the update of the current column panel based on the previous multiplication. The row panel $A_{1,0}$ which exists on a single row of processors must be copied to the other processor rows. The creation of a duplicated row projected multivector facilitates the copy. Likewise, the duplicated column projected multivector provide temporary storage of the multiplication result before reduction into the current column panel.
Lines 40-55	Optimize the matrix multiplication. On the Cray T3E, the no transpose, transpose GEMM operation is not well optimized. An explicit transpose operation enables the use of the well optimized no transpose, no transpose GEMM operation.
Line 56	Reduce the results of the matrix-matrix multiplication into the current column panel.
Line 57	Perform the single processor Cholesky factorization on the matrix block $A_{1,1}$.

Lines 58-62	Update the remainder of the column panel. The duplicated multiscalar provides work space for the broadcast of $A_{1,1}$.
Lines 63-64	Adjust the views for the recursive application of the algorithm.

```
1   #include "PLA.h"
2   int PLA_Chol_l ( int int_uplo, PLA_Obj a )
3   {
4       int size, size_top, owner_top, size_left, owner_left, length;
5       PLA_Template template = NULL;
6       PLA_Obj    adone = NULL, anext = NULL,
7                  a10 = NULL, a11 = NULL, a12 = NULL,
8                  a10_dup = NULL, a11_dup = NULL,
9                  a20 = NULL, a21 = NULL, a21_dup = NULL,
10                 a_col = NULL, a_col_dup = NULL,
11                 min_one = NULL, zero = NULL, one = NULL;
12      PLA_Obj_template ( a, & template );
13      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
14                 PLA_ALL_COLS, 1, 1, template, & one );
15      PLA_Obj_set_to_one ( one );
16      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
17                 PLA_ALL_COLS, 1, 1, template, & zero );
18      PLA_Obj_set_to_zero ( zero );
19      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
20                 PLA_ALL_COLS, 1, 1, template, & min_one );
21      PLA_Obj_set_to_minus_one ( min_one );
22      PLA_Obj_vert_split_2 ( a, 0, & adone, & anext );
23      while ( TRUE )
24      {
25          PLA_Obj_split_size ( anext, PLA_SIDE_TOP, & size_top,
26                 & owner_top );
```

```
27          PLA_Obj_split_size ( anext, PLA_SIDE_LEFT, & size_left,
28                  & owner_left );

29          if ( ( size = min ( size_top, size_left ) ) == 0 ) break;

30          PLA_Obj_vert_split_2 ( anext, size, & a_col, & anext );

31          PLA_Obj_horz_split_2 ( a_col, size, & a11, & a21 );

32          PLA_Obj_horz_split_2 ( adone, size, & a10, & a20 );

33          PLA_Pmvector_create_conf_to ( adone, PLA_PROJ_ONTO_ROW,
34                  PLA_ALL_ROWS, size, & a10_dup );

35          PLA_Obj_set_orientation ( a10, PLA_PROJ_ONTO_ROW );

36          PLA_Copy ( a10, a10_dup );

37          PLA_Pmvector_create_conf_to ( adone, PLA_PROJ_ONTO_COL,
38                  PLA_ALL_COLS, size, & a_col_dup );

39          PLA_Obj_set_to_zero ( a_col_dup );

40  /*      PLA_Local_gemm ( PLA_NO_TRANS, PLA_TRANS, min_one, adone,
41                  a10_dup, zero, a_col_dup ); */

42          {

43              int loc_len, loc_wid;

44              PLA_Obj temp_mscalar = NULL;

45              double * buffer;

46              PLA_Obj_local_length ( a10_dup, & loc_len );

47              PLA_Obj_local_width ( a10_dup, & loc_wid );

48              PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
49                  PLA_ALL_COLS, loc_wid, loc_len, template, & temp_mscalar
50                  );

51              PLA_Obj_local_buffer ( temp_mscalar, ( void ** ) & buffer );

52              PLA_Obj_extract_transpose_contents ( a10_dup, loc_wid, 1,
53                  & loc_wid, & loc_len, buffer );
```

```
54              if ( loc_wid > 0 )
55                  PLA_Local_gemm ( PLA_NO_TRANS, PLA_NO_TRANS,
56                  min_one, adone, temp_mscalar, zero, a_col_dup );
57              PLA_Obj_free ( & temp_mscalar );
58          }
59          PLA_Reduce_x ( PLA_SHAPE_LOW_TRAP, a_col_dup, one, a_col );
60          PLA_Local_chol ( a11 );
61          PLA_Mscalar_create_conf_to ( a11, PLA_ALL_ROWS,
62                  PLA_INHERIT, & a11_dup );
63          PLA_Copy ( a11, a11_dup );
64          PLA_Local_trsm ( PLA_SIDE_RIGHT, PLA_LOW_TRIAN,
65                  PLA_TRANS, PLA_NONUNIT_DIAG, one, a11_dup, a21 );
66          PLA_Obj_view_shift ( anext, size, 0, 0, 0 );
67          PLA_Obj_view_shift ( adone, size, 0, size, 0 );
68      }
69      PLA_Obj_free ( & adone );
70      PLA_Obj_free ( & anext );
71      PLA_Obj_free ( & a10 );
72      PLA_Obj_free ( & a10_dup );
73      PLA_Obj_free ( & a11 );
74      PLA_Obj_free ( & a11_dup );
75      PLA_Obj_free ( & a12 );
76      PLA_Obj_free ( & a20 );
77      PLA_Obj_free ( & a21 );
78      PLA_Obj_free ( & a21_dup );
```

```
79      PLA_Obj_free ( & min_one );
80      PLA_Obj_free ( & zero );
81      PLA_Obj_free ( & one );
82      PLA_Obj_free ( & a_col );
83      PLA_Obj_free ( & a_col_dup );
84      return ( PLA_SUCCESS );
85  }
```

Figure 10 PLAPACK Cholesky - Parallel Implementation

4.2.4 Triangular Solve Multiple RHS

The parallel triangular solve with multiple right-hand-sides implements the algorithm presented in Appendix G. The following description of the LLNN variant refers to Figure 11. The LLTN variant possess a similar implementation.

Lines 10-15	Creates the multiscalar scale parameters required by the BLAS function calls.
Lines 16-17	Initializes the views into the matrix objects.
Line 18	Performs scaling of matrix B prior to TRSM algorithm.
Lines 21-25	Determine the size of matrix block $A_{0,0}$ which can completely exist on a single processor. If **size** is equal to zero, the operation is complete. Otherwise, create views into the appropriate matrix blocks.
Lines 26-30	Copy the current column of A into a duplicated column projected multivector. The update of the row panel B_0 requires copying data corresponding to matrix block $A_{0,0}$, and the update of B_1 requires copying data corresponding to matrix block $A_{1,0}$.
Lines 31-32	Update the row panel B_0.
Lines 33-35	Copy the current row of B into a duplicated row oriented projected multivector to set up the matrix multiplication.
Lines 36-37	Update the matrix B_1.
Line 38	Adjust the views for the recursive application of the algorithm.

```
1   int PLA_Trsm_lln ( int unit, PLA_Obj alpha, PLA_Obj a, PLA_Obj b)
2   {
3       int size, size_top, owner_top, size_left, owner_left;
4       PLA_Template  template = NULL;
5       PLA_Obj     a_cur = NULL, a_col = NULL, a_col_dup = NULL,
6                   a00_dup = NULL, a10_dup = NULL,
7                   b_cur = NULL, b_row = NULL, b_row_dup = NULL,
8                   minus_one = NULL, one = NULL;
9       PLA_Obj_template ( a, & template );
10      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
11                  PLA_ALL_COLS, 1, 1, template, & one );
12      PLA_Obj_set_to_one ( one );
13      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
14                  PLA_ALL_COLS, 1, 1, template, &minus_one );
15      PLA_Obj_set_to_minus_one ( minus_one );
16      PLA_Obj_view_all ( a, & a_cur );
17      PLA_Obj_view_all ( b, & b_cur );
18      PLA_Scal ( alpha, b_cur );
19      while ( TRUE )
20      {
21          PLA_Obj_split_size ( a_cur, PLA_SIDE_TOP,  & size_top,
22                  & owner_top );
23          PLA_Obj_split_size ( a_cur, PLA_SIDE_LEFT, & size_left,
24                  & owner_left );
25          if ( ( size = min ( top_size, left_size ) ) == 0 ) break;
```

```
26          PLA_Obj_vert_split_2 ( a_cur, size, & a_col, & a_cur );

27          PLA_Obj_horz_split_2 ( b_cur, size, & b_row, & b_cur );

28          PLA_Obj_set_orientation ( a_col, PLA_PROJ_ONTO_COL );

29          PLA_Pmvector_create_conf_to ( a_cur, PLA_PROJ_ONTO_COL,
30                  PLA_ALL_COLS, size, & a_col_dup );

31          PLA_Copy ( a_col, a_col_dup );

32          PLA_Obj_horz_split_2 ( a_col_dup, size, & a00_dup, & a10_dup );

33          PLA_Local_trsm ( PLA_SIDE_LEFT, PLA_LOW_TRIAN,
34                  PLA_NO_TRANS, unit, one, a00_dup, b_row )

35          PLA_Pmvector_create_conf_to ( b_cur, PLA_PROJ_ONTO_ROW,
36                  PLA_ALL_ROWS, size, & b_row_dup );

37          PLA_Copy ( b_row, b_row_dup );

38          PLA_Local_gemm ( PLA_NO_TRANS, PLA_NO_TRANS, minus_one,
39                  a10_dup, b_row_dup, one, b_cur );

40          PLA_Obj_view_shift ( a_cur, size, 0, 0, 0 );

41      }

42      PLA_Obj_free ( & a_cur );

43      PLA_Obj_free ( & a_col );

44      PLA_Obj_free ( & a_col_dup );

45      PLA_Obj_free ( & a00_dup );

46      PLA_Obj_free ( & a10_dup );

47      PLA_Obj_free ( & b_cur );

48      PLA_Obj_free ( & b_row );

49      PLA_Obj_free ( & b_row_dup );

50      PLA_Obj_free ( & minus_one );
```

```
51      PLA_Obj_free ( & one );
52  }
```

Figure 11 PLAPACK TRSM - LLNN Variant Parallel Implementation

4.3 Virtual Object Algorithms

The problems addressed in this research expand to a size which exceeds the memory capabilities of the architecture. This situation is not uncommon for parallel applications as the increase in performance permit solutions of large systems before run time becomes prohibitively large. Implementation of out-of-core (OOC) methods allow the examination of even larger problems by viewing the disk as an additional layer in the memory hierarchy. The problems associated with disk I/O traffic are analogous to the local processor memory traffic and inter-processor communication problems discussed previously. Not surprisingly, block algorithm techniques are the best approach to minimizing the communication between the processors and disk.

I/O specific issues which adversely effect performance must be considered in the implementation of OOC methods. The primary factor effecting I/O performance is the cost of accessing data located on disk. The cost of accessing the disk is very expensive as compared to other computational functions. Many networks and parallel machines are configured to share a single disk between processors. The operating system's ability to effectively manage the shared resource decreases with the number of

simultaneous requests. The difficulties are abated by minimizing the number of I/O requests in terms of both number of accesses by a single processor and the number of processors sharing the resource. Additional benefits are realized through the implementation of asynchronous I/O requests. On architectures which support non-blocking disk I/O, computations and interprocessor communications may proceed concurrently with the file system operations.

The data distribution abstractions in PLAPACK permit sharing of resources in a quite natural way. On an architecture which shares file resources, a single row or column of processors may be assigned to perform the I/O for the entire group. The data is read into/written from a projected multivector existing on the row or column of processors. The data may then be communicated to the rest of the processor through standard PLAPACK communication routines. Constraining I/O to occur on a single row or column of processors reduces the amount of data which may be brought into memory at any one time. However, larger objects may be filled by stepping though panels of the matrix.

The structure of OOC algorithms are bounded by two limiting cases. The first maximizes the overlap of computation and communication in an effort to hide the cost of file I/O. A complex algorithm is required to manage

the asynchronous file traffic between many different data buffers. Also, the allocation of memory to I/O operations limits the available memory for linear algebra operations thereby reducing performance. The second limiting case maximizes the amount of memory available for linear algebra operations in an effort to maximize the performance of the algorithm. No memory is available for overlapping with I/O leaving the disk access cost completely exposed.

As part of this research, PLAPACK was modified to support OOC operations. Block methods were used to develop the OOC algorithms necessary to invert a symmetric matrix. The routines include a OOC Cholesky factorization and two variants of the OOC TRSM.

4.3.1 PLAPACK Implementation

The extension of out-of-core functionality to PLAPACK requires the development of PLAPACK virtual objects. The virtual object manages views into objects whose data exists entirely within virtual data space, or more simply, on disk. Global data spaces, or parallel RAM, attach to virtual objects for the caching of data for parallel operations. The user manages the data movement between the virtual space and the global space through a set of virtual object I/O routines.

The PLAPACK object fully encapsulates the complexities of the out-of-core implementation. PLAPACK parallel functionality and virtual functionality exist within the same object structure. As a result, the assumptions which govern the development of PLAPACK parallel algorithms apply to the development of PLAPACK virtual object algorithms. For more information on the virtual object implementation and the usage of the new object methods, Appendix E describes the PLAPACK Virtual Object infrastructure.

4.3.1 Cholesky Factorization

The virtual object Cholesky factorization implements the left-looking variant of the algorithm as presented in Appendix F. The following description of the parallel algorithm refers to Figure 12.

Lines 13-19	Creates the multiscalar scale parameters required by the BLAS function calls.
Line 20	Initialize the matrix views to **a_matrix** and **c_matrix**. At any point in the algorithm **a_matrix** consists of the matrix block $A_{1,0}$ and $A_{2,0}$. **c_matrix** consists of the matrix blocks $A_{1,1}$, $A_{1,2}$, $A_{2,1}$ and $A_{2,2}$. At this point in the algorithm, the matrix resides entirely on disk.
Lines 23-31	Determine the size of matrix block C_{diag} which can exist complete in parallel memory. If **size** is equal to zero, the factorization is complete. Otherwise, create views into the appropriate matrix blocks.
Lines 29-32	Attach shadow space to the current diagonal block of matrix *C* and read the data into parallel memory.
Lines 33-44	Proceeding from left to right, read into memory the blocks of **a_panel** and update the current diagonal block of C.

Line 57	Perform the Cholesky factorization on the matrix block C_{diag} and write the results to disk.
Lines 49-82	Proceeding from top to bottom, update the subdiagonal blocks in the current column.
Lines 51-55	Determine the size of matrix block C_{panel} which can exist complete in parallel memory. If **size** is equal to zero, the update of the current column is complete. Otherwise, create views into the appropriate matrix blocks.
Lines 56-57	Attach shadow space to the current subdiagonal block of matrix *C* and read the data into parallel memory.
Lines 58-71	Proceeding from left to right, read into memory the blocks of **a_panel** and **a_matrix_next** to update the current subdiagonal block of C.
Lines 74-76	Attach shadow space to the current diagonal block of matrix C and read the data into parallel memory. Alternatively, the diagonal block data could have remained memory resident. A trade-off exists between I/O costs and memory usage.
Lines 77-79	Update the current subdiagonal column block and write the result to disk.
Lines 83	Adjust the views for the recursive application of the algorithm.

```
1   #include "PLA.h"
2   int PLA_vCHOL_l ( int uplo, PLA_Obj a )
3   {
4       int size, diag_size, top_size, left_size;
5       PLA_Template template = NULL;
6       PLA_Obj    one = NULL, minus_one = NULL,
7                  a_panel = NULL, a_panel_cur = NULL, a_panel_next =
8                      NULL,
9                  a_matrix = NULL, a_matrix_next = NULL,
10                 a_cur = NULL, a_next = NULL,
11                 c_diag = NULL, c_diag_cur = NULL,
12                 c_cur = NULL, c_panel = NULL, c_matrix = NULL;
13      PLA_Obj_template ( a, & template );
14      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
15                      PLA_ALL_COLS, 1, 1, template, & one );
16      PLA_Obj_set_to_one ( one );
17      PLA_Mscalar_create ( MPI_DOUBLE, PLA_ALL_ROWS,
18                      PLA_ALL_COLS, 1, 1, template, & minus_one );
19      PLA_Obj_set_to_minus_one ( minus_one );
20      PLA_Obj_vert_split_2 ( a, 0, & a_done, & c_matrix );
21      while ( TRUE )
22      {
23          PLA_vObj_split_size ( c_matrix, PLA_SIDE_LEFT, & left_size );
24          PLA_vObj_split_size ( c_matrix, PLA_SIDE_TOP, & top_size );
25          if ( ( diag_size = size = min ( left_size, top_size ) ) == 0 ) break;
```

```
26      PLA_Obj_horz_split_2 ( a_matrix, size, & a_panel, & a_matrix );

27      PLA_Obj_split_4 ( c_matrix, size, size, & c_diag,  PLA_DUMMY,
28              & c_panel, & c_matrix );

29      PLA_Obj_view_all ( c_diag, & c_diag_cur );

30      PLA_Obj_attach_shadow_conf_to_view ( c_diag_cur );

31      PLA_Read ( c_diag_cur );

32      PLA_Obj_view_all ( a_panel, & a_panel_next );

33      while ( TRUE )

34      {

35          PLA_vObj_split_size ( a_panel_next, PLA_SIDE_LEFT, & size );

36          if ( size == 0 ) break;

37          PLA_Obj_vert_split_2 ( a_panel_next, size, & a_panel_cur,
38                  & a_panel_next );

39          PLA_Obj_attach_shadow_conf_to_view ( a_panel_cur );

40          PLA_Read ( a_panel_cur );

41          PLA_Syrk ( PLA_LOW_TRIAN, PLA_NO_TRANS, minus_one,
42                  a_panel_cur, one, c_diag_cur );

43      }

44      PLA_Obj_free ( & a_panel_cur );

45      PLA_Chol ( PLA_LOW_TRIAN, c_diag_cur );

46      PLA_Write ( c_diag_cur );

47      PLA_Obj_free ( & c_diag_cur );

48      PLA_Obj_view_all ( a_matrix, & a_matrix_next );

49      while ( TRUE )

50      {
```

```
51          PLA_vObj_split_size ( c_panel, PLA_SIDE_TOP, & size );
52          if ( size == 0 ) break;
53          PLA_Obj_view_all ( a_panel, & a_panel_next );
54          PLA_Obj_horz_split_2 ( a_matrix_next, size, & a_next,
55                  & a_matrix_next );
56          PLA_Obj_horz_split_2 ( c_panel, size, & c_cur, & c_panel );
57          PLA_Obj_attach_shadow_conf_to_view ( c_cur );
58          PLA_Read ( c_cur );
59          while ( TRUE )
60          {
61              PLA_vObj_split_size ( a_panel_next, PLA_SIDE_LEFT,
62                  & size );
63              if ( size == 0 ) break;
64              PLA_Obj_vert_split_2 ( a_panel_next, size, & a_panel_cur,
65                  & a_panel_next );
66              PLA_Obj_vert_split_2 ( a_next, size, & a_cur, & a_next );
67              PLA_Obj_attach_shadow_conf_to_view ( a_panel_cur );
68              PLA_Obj_attach_shadow_conf_to_view ( a_cur );
69              PLA_Read ( a_panel_cur );
70              PLA_Read ( a_cur );
71              PLA_Gemm ( PLA_NO_TRANS, PLA_TRANS, minus_one,
72                  a_cur, a_panel_cur, one, c_cur );
73          }
74          PLA_Obj_free ( & a_panel_cur );
75          PLA_Obj_free ( & a_cur );
```

```
76              PLA_Obj_view_all ( c_diag, & c_diag_cur );
77              PLA_Obj_attach_shadow_conf_to_view ( c_diag_cur );
78              PLA_Read ( c_diag_cur );
79              PLA_Trsm ( PLA_SIDE_RIGHT, PLA_LOW_TRIAN,
80                      PLA_TRANS, PLA_NONUNIT_DIAG, one, c_diag_cur,
81                      c_cur );
82              PLA_Write ( c_cur );
83              PLA_Obj_free ( & c_diag_cur );
84              PLA_Obj_free ( & c_cur );
85          }
86          PLA_Obj_view_shift ( a_matrix, 0, 0, diag_size, 0 );
87      }
88      PLA_Obj_free ( & one );
89      PLA_Obj_free ( & minus_one );
90      PLA_Obj_free ( & a_panel );
91      PLA_Obj_free ( & a_panel_cur );
92      PLA_Obj_free ( & a_panel_next );
93      PLA_Obj_free ( & a_matrix );
94      PLA_Obj_free ( & a_matrix_next );
95      PLA_Obj_free ( & a_cur );
96      PLA_Obj_free ( & a_next );
97      PLA_Obj_free ( & c_diag );
98      PLA_Obj_free ( & c_diag_cur );
99      PLA_Obj_free ( & c_cur );
```

```
100     PLA_Obj_free ( & c_panel );
101     PLA_Obj_free ( & c_matrix );
102 }
```

Figure 12 PLAPACK Cholesky - Virtual Object Implementation

4.3.2 Triangular Solve Multiple RHS

The parallel triangular solve with multiple right-hand-sides implements a variation of the algorithm presented in Appendix G. The following description of the LLNN variant refers to Figure 13. The LLTN variant possess a similar implementation.

Lines 13-19	Creates the multiscalar scale parameters required by the BLAS function calls.
Lines 20-24	Initializes the views into the matrix objects. The algorithm proceeds by first computing the contribution to the row panel B_0 of the previously solved elements in the matrix above B_0. The update of B_0 based on the diagonal block A occurs at the end of the algorithm. The algorithm presented in the appendix computes the update first and then applies the contribution of the update to the remainder of the matrix.
Lines 27-34	Determine the length of matrix row panel A_{panel} which can completely exist in parallel memory. If **size** is equal to zero, the operation is complete. Otherwise, create views into the appropriate matrix blocks.
Lines 39-58	Compute the contribution of the previous solutions on the current row panel of B.
Lines 64-73	Update the current row panel of B using the current diagonal block of A.

```
1   int PLA_vTrsm_lln ( int diag, PLA_Obj alpha, PLA_Obj a, PLA_Obj b )
2   {
3       int top_size, left_size, size;
4       PLA_Obj a_matrix = NULL, a_panel = NULL, a_diag = NULL, a_cur =
5                   NULL,
6                   b_matrix = NULL, b_panel = NULL,
7                   b_done = NULL, b_done_matrix = NULL,
8                   b_panel_cur = NULL, b_panel_next = NULL,
9                   b_cur = NULL, b_next = NULL,
10                  one = NULL, minus_one_over_alpha = NULL;
11      PLA_Mscalar_create_conf_to ( alpha, PLA_ALL_ROWS,
12                  PLA_ALL_COLS, & one );
13      PLA_Obj_set_to_one ( one );
14      PLA_Mscalar_create_conf_to ( alpha, PLA_ALL_ROWS,
15                  PLA_ALL_COLS, & minus_one_over_alpha );
16      PLA_Obj_set_to_minus_one ( minus_one_over_alpha );
17      PLA_Inv_scal ( alpha, minus_one_over_alpha );
18      PLA_Obj_view ( a, PLA_DIM_ALL, 0, PLA_ALIGN_FIRST,
19                  PLA_ALIGN_FIRST, & a_matrix );
20      PLA_Obj_view ( b, 0, PLA_DIM_ALL, PLA_ALIGN_FIRST,
21                  PLA_ALIGN_FIRST, & b_done );
22      PLA_Obj_view_all ( b, & b_matrix );
23      while ( TRUE )
24      {
25          PLA_vObj_split_size ( a_matrix, PLA_SIDE_TOP, & size );
26          if ( size == 0 ) break;
```

```
27          PLA_Obj_view_shift ( a_matrix, 0, 0, size, 0 );
28          PLA_Obj_horz_split_2 ( a_matrix, size, & a_panel, & a_matrix );
29          PLA_Obj_horz_split_2 ( b_matrix, size, & b_panel, & b_matrix );
30          PLA_Obj_vert_split_2 ( a_panel, -size, & a_panel, & a_diag );
31          PLA_Obj_view_all ( b_done, & b_done_matrix );
32          PLA_Obj_view_shift ( b_done, 0, 0, 0, size );
33          /* while the view shift may seem out of place, it is necessary for the next
34                  iteration */
35          while ( TRUE )
36          {
37              PLA_vObj_split_size ( a_panel, PLA_SIDE_LEFT, & size );
38              if ( size == 0 ) break;
39              PLA_Obj_vert_split_2 ( a_panel, size, & a_cur, & a_panel );
40              PLA_Obj_horz_split_2 ( b_done_matrix, size, & b_next,
41                      & b_done_matrix );
42              PLA_Obj_attach_shadow_conf_to_view ( a_cur );
43              PLA_Read ( a_cur );
44              PLA_Obj_view_all ( b_panel, & b_panel_next );
45              while ( TRUE )
46                  PLA_vObj_split_size ( b_next, PLA_SIDE_LEFT, & size );
47                  PLA_Obj_vert_split_2 ( b_next, size, & b_cur, & b_next );
48                  PLA_Obj_vert_split_2 ( b_panel_next, size, & b_panel_cur,
49                      & b_panel_next );
50                  PLA_Obj_attach_shadow_conf_to_view ( b_cur );
51                  PLA_Obj_attach_shadow_conf_to_view ( b_panel_cur );
```

```
52                  PLA_Read ( b_cur );
53                  PLA_Read ( b_panel_cur );
54                  PLA_Gemm ( PLA_NO_TRANS, PLA_NO_TRANS,
55                      minus_one_over_alpha, a_cur, b_cur, one, b_panel_cur );
56                  PLA_Write ( b_panel_current );
57              }
58              PLA_Obj_free ( & b_cur );
59              PLA_Obj_free ( & b_panel_cur );
60              PLA_Obj_free ( & a_cur );
61          }
62          PLA_Obj_attach_shadow_conf_to_view ( a_diag );
63          PLA_Read ( a_diag );
64          while ( TRUE )
65              PLA_vObj_split_size ( b_panel, PLA_SIDE_LEFT, & size );
66              PLA_Obj_vert_split_2 ( b_panel, size, & b_cur, & b_panel );
67              PLA_Obj_attach_shadow_conf_to_view ( b_cur );
68              PLA_Read ( b_cur );
69              PLA_Trsm ( PLA_SIDE_LEFT, PLA_LOW_TRIAN,
70                      PLA_NO_TRANS, diag, alpha, a_diag, b_cur );
71              PLA_Write ( b_cur );
72          }
73          PLA_Obj_free ( & a_diag );
74          PLA_Obj_free ( & b_cur );
75      }
```

```
76      PLA_Obj_free ( & a_matrix );
77      PLA_Obj_free ( & a_panel );
78      PLA_Obj_free ( & a_diag );
79      PLA_Obj_free ( & a_cur );
80      PLA_Obj_free ( & b_matrix );
81      PLA_Obj_free ( & b_panel );
82      PLA_Obj_free ( & b_done );
83      PLA_Obj_free ( & b_done_matrix );
84      PLA_Obj_free ( & b_panel_cur );
85      PLA_Obj_free ( & b_panel_next );
86      PLA_Obj_free ( & b_cur );
87      PLA_Obj_free ( & b_next );
88      PLA_Obj_free ( & one );
89      PLA_Obj_free ( & minus_one_over_alpha );
90      return ( PLA_SUCCESS );
91  }
```

Figure 13 PLAPACK TRSM - LLTN Variant Virtual Object Implementation

4.4 Numerical Integration

The issues of numerical integration concern the relative cost between the physical system model and the accumulation operation. Allowing l to be the maximum degree and order of the geopotential expansion, the accumulation increases according to $\vartheta(l^4)$. (The symmetric rank-k operation requires mn^2 floating point operations where m corresponds to the number of observations and n corresponds to the number of gravity field parameters, or $(l+1)^2$.) The observation model costs increase at only $\vartheta(l^2)$ under the same conditions. (The force model requires the evaluation of the Legendre associated functions, a recursion over $\vartheta(l^2)$ elements, and the spherical harmonic evaluation, a summation over $\vartheta(l^2)$ terms. The number of first order differential equations to be integrated is $42+6(l+1)^2$, or $\vartheta(l^2)$.) On serial architectures the cost of the accumulation quickly dominates the cost of the algorithm [Bettadpur 1993].

The implementation on parallel architectures requires an examination of the distribution of each component's computations of over the processor array. For example, given a problem defined by l, the wall clock time required

to generate the observations and the wall clock time required to accumulate the observations would be expressed by Equation (45).

$$T_{obsgen} = \frac{\vartheta(l_{\max}^2)}{R_{obsgen} P_{obsgen}}$$
$$T_{accum} = \frac{\vartheta(l_{\max}^4)}{R_{accum} P_{accum}} \tag{45}$$

where R specifies the per processor speed of the operation and the P specifies the number of processors over which the operations are distributed. The percentage of time spent performing the observation generation is expressed in Equation (46).

$$F_{obsgen} = \frac{1}{\vartheta(l_{\max}^2)} \frac{R_{accum} P_{accum}}{R_{obsgen} P_{obsgen} + R_{accum} P_{accum}} \tag{46}$$

The fraction of the time spent performing observation equation evaluations decreases with $\vartheta(l^2)$ if the problem size is permitted to grow without any changes in the number of processors. However, if only the accumulation is spread over and increasing number of processors without any loss of efficiency, the fraction of the time spent performing observation evaluations increases to one. The observation generation functions must be effectively distributed to avoid placing a speed-up bound on the combined algorithm.

The investigation of parallelizing the numerical integration focused on two approaches. The first partitioned the system of equations at natural boundaries, and the second partitioned the interval of integration. Neither was found to be completely satisfactory for a costly observation model.

Natural partitions in the system of first order equations are identified according to the decoupling of equations above and below the partition point. Assuming perturbed two-body motion, one such break occurs between the equations of different satellites. The complete integration vector may be distributed across the processor array such that the partition boundaries correspond to the natural breaks in the differential equations. Since the equations are not coupled between breaks, no inter-processor communication is required to complete the integration. Most problems, however, do not examine the trajectories of an ever increasing number of satellites as the number of processors is increased. Once the number of processors exceeds the number of satellites, no additional parallelism may be obtained, and the algorithmic cost imbalance as described previously will occur.

A second method of decomposition occurs by partitioning along the interval of integration. Each processor is responsible for propagating the orbit and generating observation equations along the designated sub-interval. The load imbalance problems associated with the previous method do not occur.

However, an overhead cost is incurred from the propagation of the satellite from the global initial epoch to the local sub-interval epoch and the restart of the integration at the sub-interval epoch. This method of decomposition requires that enough memory exists on the local processor to perform the integration for all satellites in the physical system.

5. Software Design

Application development for this research requires the integration of serial and parallel components into a single software package. An improper interface between components creates bottlenecks in parallel performance. The implementation methods which lead to high performance algorithms result from experimentation and iteration of design. The examination of the gradiometer and GPS application requirements yields a generalized expression of the batch least squares algorithm for distributed memory parallel architectures.

5.1 Computational Constraints

The primary constraints on application performance involve memory capacity and processing speed. Each constraint affects the performance of each application scenario in a different manner. Since the computational cost of modeling the gradiometer is relatively small, the accumulation cost dominates the application. Wall-clock cost restrictions on the generation of gradiometer normal equations are effectively non-existent. Any gradiometer problem which fully resides in memory may be processed in a reasonable amount of time. The computational cost of processing GPS information is significantly larger and suffers from the inability to effectively parallelize

numerical integration operations. The amount of geopotential information recovered from GPS data is primarily constrained by the wall-clock cost of the operation.

The performance characteristics of a distributed memory architecture, both in terms of memory and speed, vary with the number of processors. The analysis should be scaled according to the available number of processors to most effectively utilize system resources. Table 5 presents the approximate memory usage per processor for the different application components. The memory usage of the accumulation grows with the forth power of the maximum gravity field degree and dominates the overall memory usage. The memory usage of the other application components grow with the square of the maximum gravity field degree.

Operation	Memory Usage per Processor (bytes)
Normal Equation Accumulation	$\frac{8}{p}\left[(l+1)^4 + m_{batch}(l+1)^2\right]$
Gravity Field (Potential, Force and Gradient)	$112(l+2)(l+1)$
ABFS Numerical Integration	$8n_{sat}\left[\left(42+6(l+1)^2\right)\left(n_{ord}+3\right)+10\right]$
Gradiometer Observation	$8(l+1)^2$
Satellite Tracking Observation	$32(l+1)^2$

Table 5 Memory Requirements for Application Algorithms

Table 6 presents the operation counts for the different application components. Dense matrix operations are $\vartheta(n^3)$ where n is the dimension of the matrix. Since $n=(l+1)^2$ for the gravity field problem (See Appendix D), the operations become $\vartheta(l^6)$ where l is the maximum degree of the spherical harmonic expansion. The observation and dynamic model algorithms consist primarily of summations over the gravity field coefficients requiring $\vartheta(l^2)$ operations. The processing efficiency varies between the different algorithms. The linear algebra operations execute optimized subprograms which approach the expected sustained performance of the architecture. The observation operations implement compiler optimized code which achieve only a fraction (5-10%) of the performance capability.

Operation	Number of Operations
Accumulation (SYRK)	$m(l+1)^4$
Cholesky	$\frac{1}{3}(l+1)^6$
Matrix Inversion (TRSM)	$2(l+1)^6$
Gradiometer Observation	$116l^2$
Satellite Tracking Observation	$12(l+1)^2$

Table 6 Computational Cost for Application Algorithms

The operation count for numerical integration was not calculated. Instead, empirical models based on the integration of the single satellite state and state transition matrix over an arc of one day were developed and the results are presented in Figure 14. A seventh order Adams-Bashforth-Moulton integration method with mesh sizes of 30 seconds, 45 seconds and 60 seconds (curves top to bottom) are shown. The starting cost associated with the multistep methods is included in the integration cost.

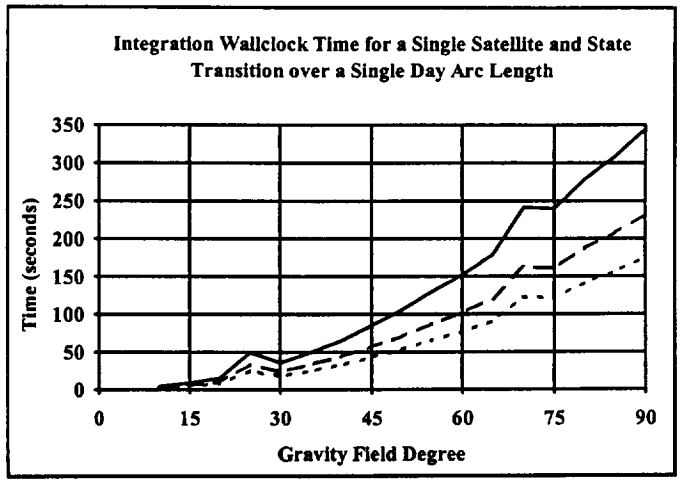


Figure 14 Computational Cost for Numerical Integration

5.2 Parallel Batch Estimation Algorithm

The serial batch estimation algorithm requires restructuring for use on distributed memory parallel architectures. For large numbers of estimated parameters, the normal matrix is too large to exist completely on a single

processor. The accumulation of information into the distributed matrix should exploit the performance capabilities of the multiple processors. Also, as mentioned in the previous chapter, the observation modeling process must effectively parallelize in order to fully realize the performance gain. Implementation of an effective batch estimator on a distributed memory architecture requires the development of a generalized parallel batch estimation algorithm. The following discussion assumes all system components are memory resident.

The batch algorithm may be partitioned according to the batch object methods and the physical system modeling methods. The batch object methods primarily consist of linear algebra operations and account for the vast majority of floating point operations. Parallel linear algebra is a well-understood process which can be implemented with a high degree of efficiency. Physical system models center on the implementation of the numerical integration process. The previous chapter described two approaches to parallelizing numerical integration. Of the two approaches, partitioning along the interval of integration promises the best scalability.

The parallel batch paradigm used the above partitioning to decompose functionality into serial and parallel executing components. The observations are generated in a data-parallel manner with each processor computing the

partials corresponding to a different interval of integration. The accumulation occurs in a work-parallel manner with all processor updating the same normal matrix.

Two variants were considered in the first implementations. An integrated method required each processor to participate in both the observation generation and the accumulation operations. A partitioned method grouped the processors according to function. Each methodology yielded advantages and disadvantages. The integrated method exhibited better load balancing characteristics due to the data parallel nature of the work sharing. The necessary storage of model simulation components limited the amount of memory which could be dedicated to the linear system operations. The partitioned method permitted an expansion of the problem size, but was prone to load imbalance. Incorrect sizing of the processor groups allows one group to complete sooner than the other. The integrated method was chosen due to its ability to provide reliable high performance

Parallel accumulation requires the caching of a number of partial arrays before performing an update of the normal matrix. To facilitate the caching, the partial arrays are copied into the columns of a distributed batch matrix The integrated approach requires each processor to simultaneously generate different partial arrays. The method by which partial arrays are

assigned to unique columns leads to two different expressions of the batch algorithm. The static assignment of partials allocates the partials to the columns, either explicitly or implicitly, at the beginning of each batch. The dynamic assignment of partials allocates columns to the partials on a first come, first serve basis though queries to an index server. The dynamic assignment method places no assumptions on the ordering of partials allowing operations such as on-the-fly observation editing.

The generalized data parallel batch algorithm with dynamic assignment is presented in Figure 15. The generalized data parallel batch algorithm with static assignment is presented in Figure 16. An explanation of each algorithm follows the Figures.

```
Initialize application components
While (global observations remain)
    While (local observations remain)
            Read next local observation(s) at epoch
            Integrate reference trajectory to epoch
            Form local observation equations at epoch
            While (observation(s) remain at epoch)
                    While (global partial batch is full) Accumulate
                    Send observation equation to global batch matrix
            End while
            Increment to next epoch
            if (local processing completed) Send completion signal to all processors
    End while
    if (local processing completed) Accumulate
End while
Solve normal equations
```

Figure 15 Data Parallel Batch Algorithm with Dynamic Assignment

The initialization process of the dynamic assignment algorithm creates the necessary physical and mathematical objects and distributes the observations to the processors. No assumptions are placed on the global distribution of the observations.

The outer loop continues while observations exist to be processed. Each processor tracks the availability of global observations through the index server. Once a processor has completed its assigned observations, the processor sends a signal to the other processors. Only after a completion signal has been received from all the other processors will the local processor be permitted to exit the outer loop.

The inner loop of the dynamic assignment algorithm is similar to the serial batch algorithm. The accumulation operation is replaced by a query to the index server and the communication of the local observation equation to the global batch matrix. The index server allocates the column indices on a first come, first serve basis. If the matrix is full, an accumulation operation occurs. To guarantee progress of the algorithm, multiple observations at a single epoch must be processed one at a time. Otherwise, invalid columns may be assigned resulting in either a core dump or application deadlock.

```
Initialize application components
While (global observations remain)
    Read batch of observations
    Initialize state at initial epoch
    While (local observations remain)
            Integrate reference trajectory to epoch
            Form local observation equations at epoch
            Send observation equation to global partial batch
            Increment to next local epoch
    End while
    Accumulate
End while
Solve normal equations
```

Figure 16 Data Parallel Batch Algorithm with Static Assignment

The initialization of the static assignment algorithm creates the necessary physical and mathematical objects. Distribution of the observations and column indices occurs at the top of the outer loop. The local processor exits the outer loop after all observations have been processed.

The inner loop of the static assignment algorithm is similar to the serial batch algorithm. The accumulation operation is replaced by the communication of the current observation equation to the global batch matrix. After all local observation in the current batch have been processed, an accumulation event occurs.

5.3 Batch Algorithm Implementations

Both variants of the data parallel batch algorithm produced successful gradiometer implementations. The static assignment algorithm performed

slightly better. However, neither variants of the algorithm produced a satisfactory GPS implementation. The dynamic assignment variant suffered from severe load imbalances. The index server was implemented using a master-slave approach directing all queries to a single processor. Individual processors incurred large idle time while waiting for responses from the index server. The server processor could not answer the index request until the completion of local physical system calculations. The static assignment variant fell prey to the inherent serialization of the numerical integration. Since a batch of observations is not widely separated in time, many processors shared the same integration sub-interval either during the processing of observations or in the initialization of its own sub-interval. The dynamic modeling proceeded in an essentially serial manner and produced bottleneck in the overall algorithm.

The GPS application was redesigned using the second parallelization strategy for numerical integration. A single GPS satellite and the gradiometer satellite was assigned to each processor, and a serial batch algorithm was used to accumulate the observations associated with the high-low satellite pair. The computations proceeded perfectly in parallel over each subarc with the exception of duplicated work in the integration of the gradiometer satellite. The processors were synchronized in between subarcs to merge subarc

information. The normal equations on each processor were merged at the completion of the global arc. The implementation of the serial batch algorithm restricted the problem size to that which could be contained on a single processor. While not desirable for general purpose use, the method produced results satisfactory for the completion of this research.

6. Error Analysis and Results

A gradiometer error analysis similar to that presented by Colombo will be conducted [Colombo, 1989]. The method requires the inversion of the normal matrix generated during a gradiometer mission simulation. The diagonal elements of the inverted normal matrix represent the *a posteriori* variance of the estimated potential coefficients. The analysis ignores the right-hand-side of the observation equations in order to reduce computational cost and avoid the use of real or simulated observations. In this respect, the analysis is not a true simulation study; however, the reduced cost permits the examination of a broader range of satellite and instrument configurations in the search of an optimal mission scenario. Analyses of this type are prevalent in the literature [Colombo, 1989; Schrama, 1991; Koop, 1993; Visser, 1994]. The work presented here contributes to previous research through the implementation of single point computations, and therefore added robustness, to the analysis.

The case studies demonstrate the effectiveness of parallel methods applied to the gravity field problem and the capability to perform a rigorous solution of the high resolution geopotential from satellite gradiometer and GPS observations. The information recovered from a satellite gradiometer mission requires the use of parallel methods in the formation of a batch least

squares estimate. The complexity of the GPS dynamic model warrants investigation of parallel numerical integration techniques. Besides illustrating the successful implementation of parallel methods, the science results verify previous error analyses which used approximate grid computation methods and provide insight to the capabilities of rigorous analysis for the high degree and order geopotential.

6.1 Observation Constraints

The uniform solution of the geopotential requires global coverage of the Earth's surface. Each coefficient in the geopotential expansion may be expressed as an integral of an observed gravity signal over the surface of the Earth [Colombo, 1981]. Incomplete coverage of the Earth's surface removes the contributions of omitted geographical areas and leads to errors in the coefficients. The only satellite trajectory which can sample the entirety of the Earth's surface is a polar orbit. Gradient observations collected along a polar orbit provides the best information about the geopotential.

The polar orbit, however, may not satisfy satellite specific mission requirements, specifically power subsystem requirements, due to its changing orientation with respect to the sun. Satellites which generate power from solar panels must spend a significant portion of the orbit exposed to the sun. A

common technique to maximize exposure of the satellite to the sun is to exploit the oblateness of the Earth in order to maintain a perpendicular orientation between the satellite orbit plane and the Earth-Sun radius. The resulting sun synchronous orbit possesses a non-polar inclination. Satellites located in non-polar orbits cannot collect observations within geographical regions centered at the poles. The size of the geographical region within the polar gaps, and therefore the errors introduced into the geopotential solution, depends on the inclination of the orbit.

The size of the orbit's semi-major axis results from a trade-off between the strength of the gravity signal and the surface drag effects on the satellite. The strength of the radial gravity gradient observable is proportional to the cube of the satellite radius and decreases rapidly with increasing altitude. The strongest gradient signals occur in the lowest orbits. However, drag effects dominate satellite motion in low altitude orbits. The drag effects limit the lifespan of the satellite and degrade the quality of the observation. An acceptable range of altitudes for satellite gradiometers is approximately 250 km to 300 km.

The spatial resolution of the observations also effects the quality of the geopotential estimate. The sampling theorem dictates the necessary observation spacing according to the spatial frequency of the gravity field

signal. The shortest spatial wavelength harmonic associated with a degree l coefficient is $\Delta^l = \frac{360}{l}$ degrees. According to the sampling theorem, the unique resolution of the observed signal requires sampling at twice its frequency or greater. For the gravity field harmonic of degree l, the observation spatial resolution in both longitude and latitude must be $\Delta^s = \frac{180}{l}$ degrees or less. A summary of sampling resolutions for various spherical harmonic expansion degrees is presented in Table 7.

Expansion Degree	Sampling Resolution (degrees)
2	90.
4	45.
12	15.
30	6.0
60	3.0
90	2.0
120	1.5
180	1.0

Table 7 Required Spatial Resolution for Gradiometer Observations

6.2 Orbit Design

The satellite gradiometer travels a trajectory with a repeating ground track. The repeating ground track orbit provides uniform coverage of the Earth's surface and allows satellites to visit the same geographical locations

once every repeat interval. Over time, the series of measurements provide the ability to track temporal changes at that location. The desired surface coverage resolution defines the repeat orbit parameters. The coverage resolution significantly affects the resolution of the recovered gravity field.

Repeat ground track orbits are developed according to the iterative solution to Lagrange's planetary equations. The mean perturbations of the non-spherical geopotential define the equations of the osculating orbit. Dr. Srinivas Bettadpur of the University of Texas Center for Space Research (UT/CSR) provided the orbit design tool used in this research. The different gradiometer mission scenarios required the development of two repeat ground track orbits. A polar orbit guaranteed global coverage for the mission. A sun synchronous orbit left polar gaps in the coverage. A coverage resolution capable of recovering a degree and order 360 gravity field defined the design criteria for the repeat orbit.

The sampling interval requires that the spacing between ascending ground tracks must be $\Delta^s = \frac{180}{360} = 0.5$ degrees or less to insure observability of a degree and order 360 gravity field. The spacing produces a repeat cycle which consists of 720 revolutions. At an initial altitude of 275 km in a circular orbit and a mean motion of $1.164x10^{-3}$ radians per second, a 45 day

repeat period is produced. Initial guesses for polar and sun synchronous orbits were input into the design tool described above. The converged orbits for the polar and sun synchronous orbits are presented in Table 8.

	Polar	Sun Synchronous
Altitude	249.71 km	261.98 km
Inclination	90.1 degrees	96.6 degrees
# revolutions	721	721

Table 8 Satellite Gradiometer Orbit Parameters

6.3 Problem Scaling

The resolution of the geopotential to be recovered depends on the performance characteristics of the computational platform. The available memory limits the linear system size and the number of gravity field coefficients which may be estimated. As discussed previously, observations with large dynamic model costs also pressure wall-clock constraints. Both the size of the gravity field and the number of observations may be reduced to bring the application wall-clock cost within specific constraints.

The limits imposed by the sampling theorem constrain the adjustment of the observation sampling interval. As with the longitudinal spacing, the

latitudinal spacing of the observations must be $\Delta^s = \frac{180}{l}$ degrees or less. For a satellite in a near-polar orbit, one observation must occur within every Δ^s degree interval of arc traveled. Assuming a constant angular velocity of $\omega = \frac{360}{TP}$ degrees per second, the minimum required sampling interval equates to $\delta t = \frac{\Delta^s}{\omega}$. Table 9 presents the sampling interval and the approximate number of observations for the satellite gradiometer and GPS observation datatype.

	Sampling Interval	Number of Observations
Gradiometer	5 seconds	777,600
GPS	15 seconds	1,400,000

Table 9 Summary of Sampling Quantities for Case Study

The Cray T3E distributed-memory parallel architecture located at the University of Texas at Austin provides the baseline for problem scaling. The machine possesses 44 DEC Alpha EV6 RISC processors of which a maximum of 32 processors execute during a single production run. Each processor contains 128 Mbytes of local memory and rates 600 Megaflops peak performance. Vendor optimized subprograms provide high single-processor performance. Using optimized BLAS, parallel accumulation and linear

system solve operations easily sustain 200 Megaflops per processor performance.

The available memory on 32 processors of the T3E determined the gradiometer application size. The two components using the largest amount of memory were the normal matrix and a batch matrix that cached observations prior to the accumulation. An empirical determination of the maximum problem size demonstrated a capability to process a geopotential model to degree and order 110. The normal matrix uses approximately 40 Mbytes per processor memory and the batch matrix uses approximately 20 Mbytes per processor memory. The dynamic model, temporary message buffers and other operating system functions use the remainder of memory.

The available memory on a single processor of the T3E determined the GPS application size. As with the gradiometer, the two components using the largest amount of memory are the normal matrix and batch matrix. The normal matrix corresponding to a degree and order 50 geopotential model requires 54 Mbytes of memory.

6.4 Case Studies

Three parallel applications collectively perform the geopotential error analysis. The processing of the polar and sun synchronous orbits proceed similarly.

The first application, **gradio**, generates the normal equations for the satellite gradiometer observable. Batch queue time restrictions require the partitioning of the 45 day arc into three 15 day arcs. The *a priori* variance on the radial gravity gradients is 10^{-3} Eotvos (10^{-12} s^{-2}). Each job executes on 32 processors. The normal matrix accumulation is entirely memory resident. A single normal matrix file structure is created for each 15 day integration interval. PLAPACK virtual object I/O routines facilitate the copying of normal matrix data to disk.

Table 10 and Table 11 present the **gradio** application performance for the polar and sun synchronous orbit solutions. The figures reflect the performance of the accumulation and the overall application. The wall clock time was computed by summing the execution time for all three 15 day arcs. The timings illustrate the dominance of the accumulation costs as a fraction of the total cost of the algorithm. The timings also illustrate the ability to achieve high performance for a real POD application.

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Application	21,684.7	170.1
Accumulation	18,239.8	202.2

Table 10 gradio Application Performance for Polar Orbit

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Application	21,696.1	170.0
Accumulation	18,255.6	202.1

Table 11 gradio Application Performance for Sun Synchronous Orbit

The second application, **gps**, generates the normal equations for the GPS tracking observable. Batch queue time restrictions required the partitioning of the 45 day arc into three 15 day arcs. The *a priori* variance on the range observable is 5 mm. Each job executes 25 processors with one high-low satellite pair allocated to each processor. The processing of the normal matrix is entirely memory resident. A single normal matrix disk file is created for each 15 day integration interval. Standard C I/O routines copy the normal matrix to disk. Utility routines redistribute the data into a form suitable for PLAPACK virtual objects.

Table 12 and Table 13 present the **gps** application performance for the polar and sun synchronous orbit solutions. The figures reflect the

performance of the accumulation and the overall application. The wall clock time was computed by summing the execution time for all three 15 day arcs. The timings illustrate the dominance of the dynamic modeling cost relative to the accumulation for an the GPS problem.

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Application	18,008.4	21.9
Accumulation	4,942.6	79.8

Table 12 gps Application Performance for Polar Orbit

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Application	18,055.3	21.9
Accumulation	4,990.2	79.1

Table 13 gps Application Performance for Sun Synchronous Orbit

The last application, **xena**, implements a general out-of-core linear system solver using PLAPACK virtual objects. The application merges an arbitrary number of matrices and performs the matrix inversion. **xena**'s input consists of a list of filenames and corresponding matrix dimensions. The input data file must exist in a PLAPACK I/O format. For this case study, **xena** summed and inverted the normal matrices corresponding to the gradiometer only, GPS only and the combination solution.

Table 14 and Table 15 present the **xena** application performance for the polar and sun synchronous orbit solutions. The figures reflect the performance of the gradiometer only solution, but the combination solution performance is similar. The timings illustrate the good performance of the PLAPACK virtual object methods.

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Total	1784	76.4
Cholesky	219.7	88.7
Inversion	911.9	128.2

Table 14 xena Application Performance for Polar Orbit

	Wall Clock Time (seconds)	Efficiency (Mflops/PE)
Total	1759	77.5
Cholesky	218.2	89.3
Inversion	930.7	125.6

Table 15 xena Application Performance for Sun Synchronous Orbit

6.4.1 Polar Orbit Solutions

Figure 17 and Figure 18 presents the error degree variance for the polar orbit solutions. The gradiometer only solution possesses greater uncertainty for the lower degree coefficients. The GPS only solution possesses greater uncertainty for the higher degree coefficients. The combination solution receives the best characteristics of both datatypes. The combination solution, however, does not remain completely under the GPS solution at the low degrees. Also, a discontinuity exists on the combination solution variance triangle plot in Figure 18. Optimal weighting strategies for combining the two datatypes may provide a possible solution for both anomalies.

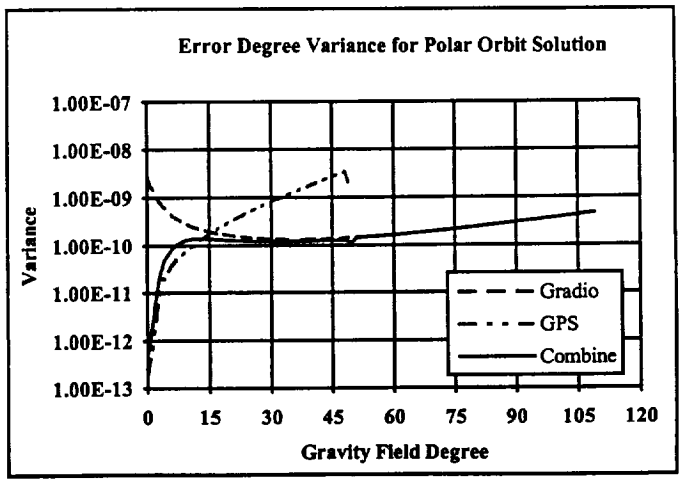


Figure 17 Error Degree Variance for Polar Orbit Solution

Figure 19 presents the geoid height variance for the polar orbit solution. The gradiometer only solution produces a mean geoid height error of 25 millimeters. The GPS only solution produces a larger mean geoid height error of 45 millimeters. The combination solution produces a significantly improved mean geoid height error of 13 millimeters due to the improvement in the lower order coefficient uncertainty. The cause of the spots in the GPS geoid height variance plot is unknown.

Figure 18. Coefficient Variances for Polar Orbit Solution

(see color plate on following page)

Formal Variances of Normalized Geopotential Coefficients
Polar Orbit Solutions

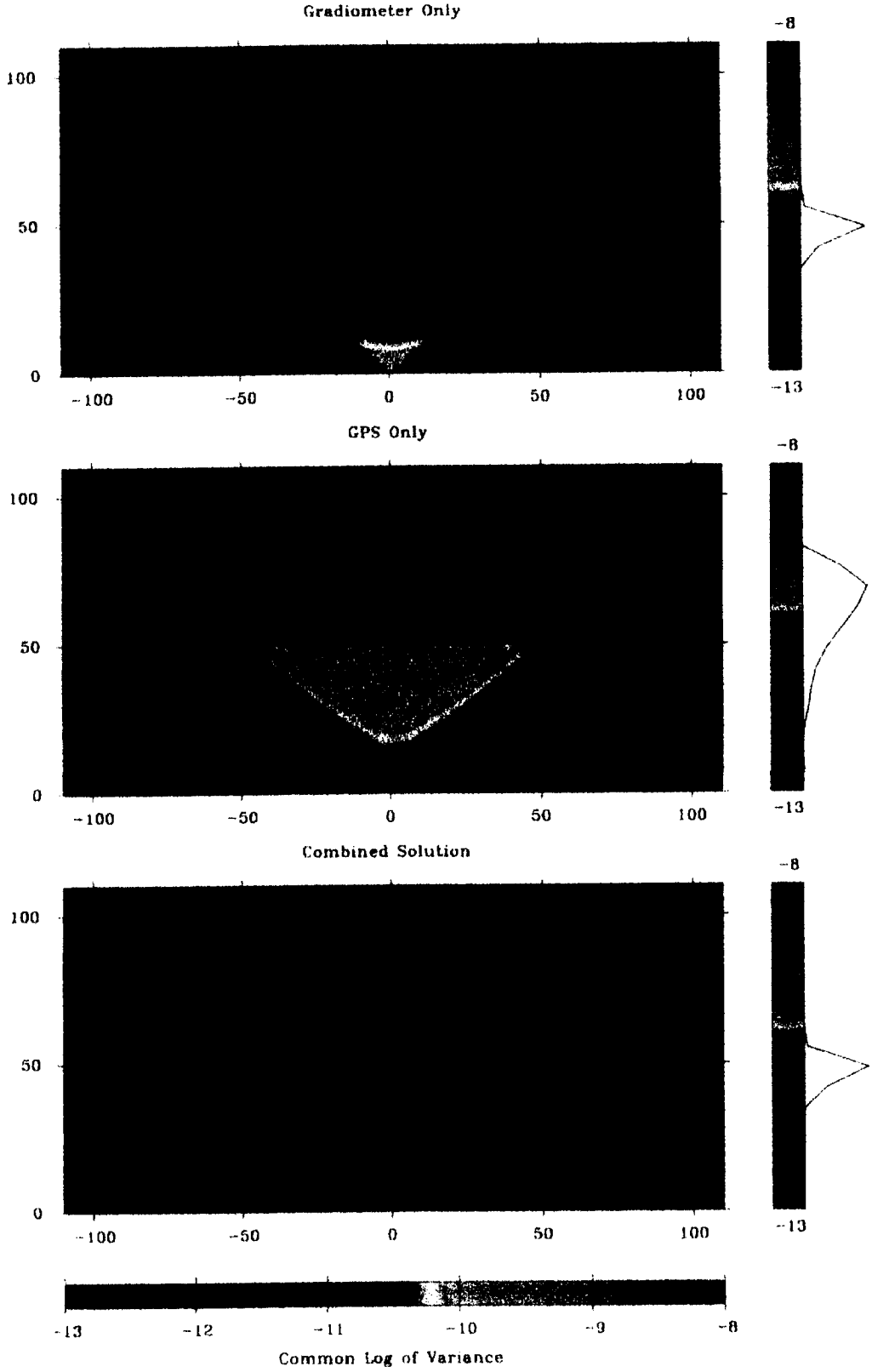
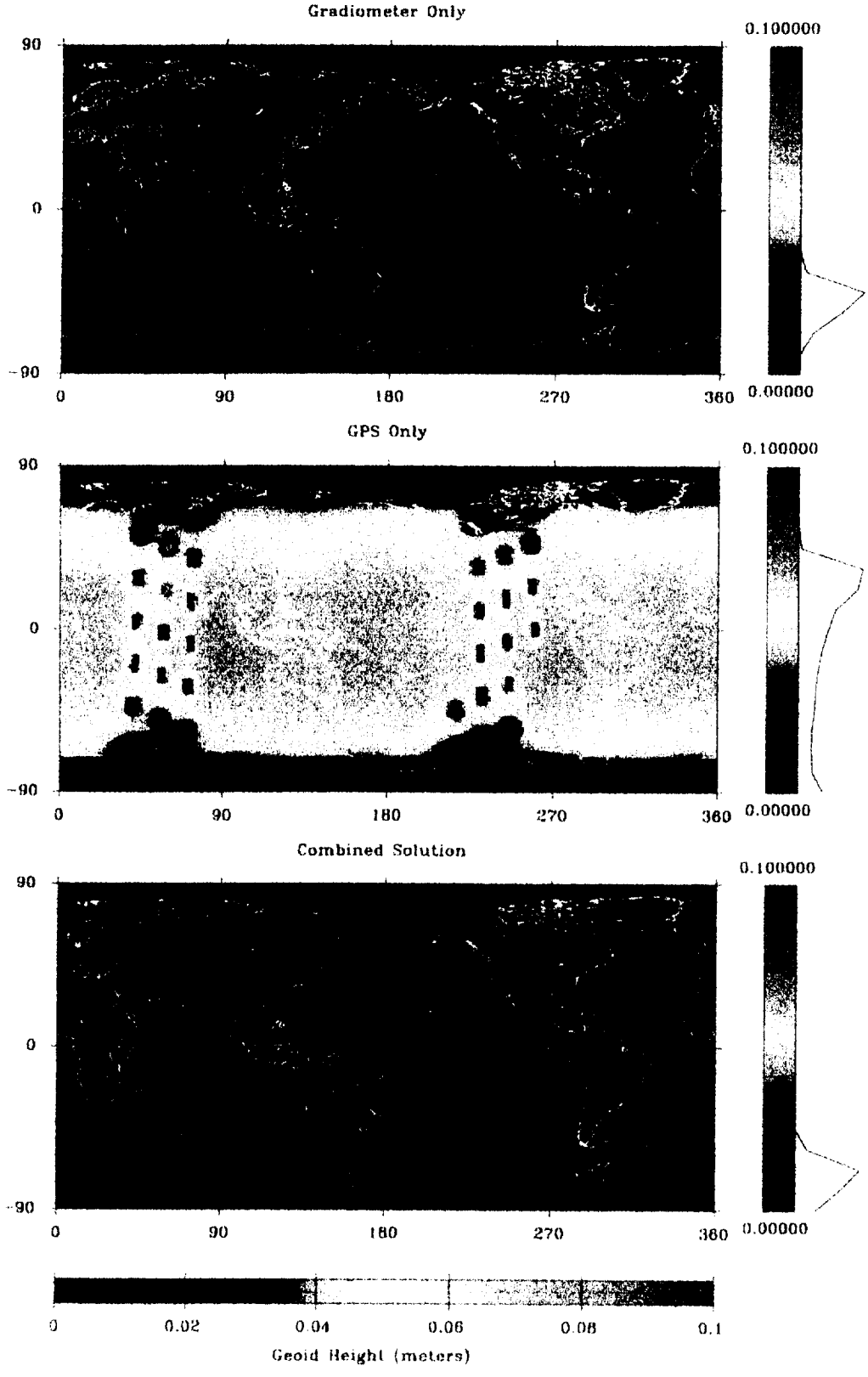


Figure 19. Geoid Height Variance for Polar Orbit Solution

(see color plate on following page)

Geoid Height Variances
Polar Orbit Solutions



6.4.2 Sun Synchronous Orbit Solutions

Figure 20 and Figure 21 present the error degree variance for the sun synchronous orbit solutions. The gradiometer only solution possesses greater uncertainty across the range of coefficients due to the loss of information at the poles. The high variance along the zonal coefficients as seen in Figure 21 further illustrates the deficiency. The GPS only solution is similar to the polar orbit solution. The combination solution illustrates the ability to incorporate the GPS observability of the zonal coefficients into the gradiometer solutions. The error degree variance of the combination solution is almost identical to the polar solution. However, deficiencies still exist in the zonal coefficients as seen in the variance triangle plot.

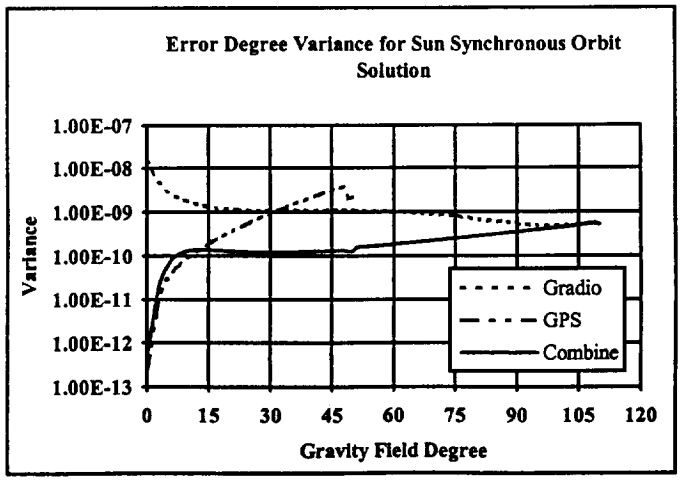


Figure 20 Error Degree Variance for Sun Synchronous Orbit Solution

Figure 22 presents the geoid height variance for the sun synchronous orbit solution. The gradiometer only solution's inability to resolve the zonal coefficients produces an extreme mean geoid height error of 224 millimeters. The GPS only solution produces a larger mean geoid height error of 51 millimeters. The combination solution produces a significantly improved mean geoid height error of 17 millimeters further illustrating the necessity to incorporate GPS tracking information into the sun synchronous gradiometer solution. The cause of the streak in the GPS geoid height variance plot is unknown.

Figure 21. Coefficient Variances for Sun Synchronous Orbit Solution

(see color plate on following page)

Formal Variances of Normalized Geopotential Coefficients
Sun Synchronous Orbit Solutions

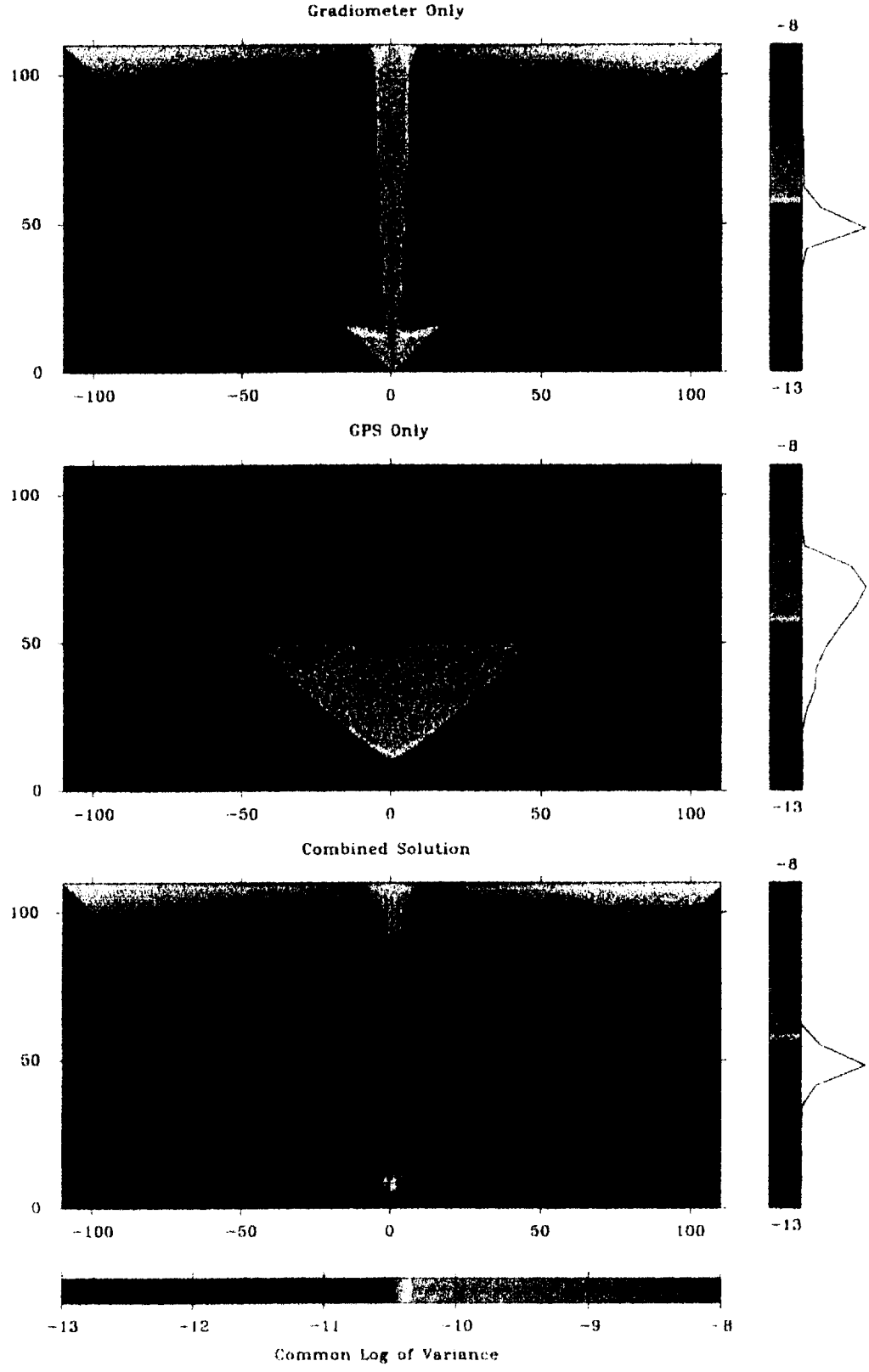
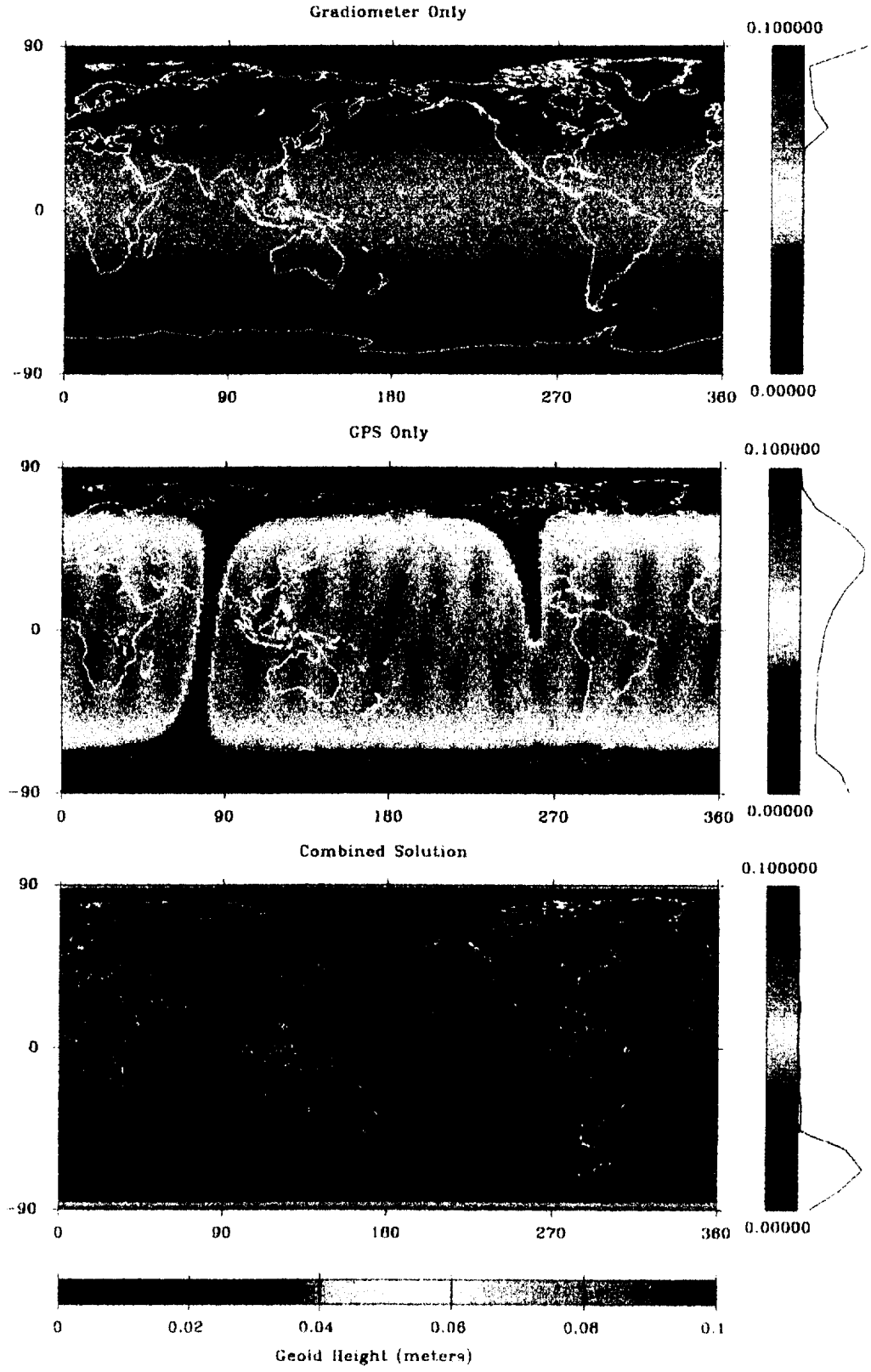


Figure 22. Geoid Height Variance for Sun Synchronous Orbit Solution

(see color plate on following page)

Figure 22 Geoid Height Variance for Polar Orbit Solution

Geoid Height Variances
Sun Synchronous Orbit Solutions



7. Conclusions

The examination of computational aspects associated with the global geopotential recovery problem constitutes the theme of this work. The primary subject matter addresses specifically the use of satellite gradiometer and unbiased GPS slant range observations to form and invert the normal matrix associated with a large degree and order geopotential solution. Memory resident and out-of-core parallel linear algebra techniques along with data parallel batch algorithms form the foundation of the least squares application structure. A secondary topic includes the adoption of object oriented programming techniques to enhance the modularity and reusability of code. Applications implementing the parallel and object oriented methods successfully calculate the degree variances for a degree and order 110 geopotential solution on 32 processors of the Cray T3E.

7.1 Object Oriented Programming

Application software developed as part of this research utilizes the design philosophy of object oriented programming. Precision orbit determination applications require two general classes of objects. The physical class abstracts the physical objects that comprise the satellite environment. The mathematical class abstracts the mathematical techniques

used in the simulation analysis. The prototype satellite application library, OOPOD, demonstrates how OOP effectively manages complexity and contributes to the development of modular and highly useable library components. A series of examples in the text and the appendicies illustrate the facility of the library in developing common orbit determination applications.

7.2 Parallel Processing

The performance capabilities of parallel processing enable the rigorous examination of the high resolution geopotential problem. Physically Based Matrix Distribution distributes of parallel linear algebra operations according to the data mapping of the linear system domain and range spaces. Communications between different linear algebra objects proceed using well-defined collective communication operations. Parallel BLAS Level 3 operations necessary for accumulation and matrix inversion build upon the PLAPACK library infrastructure. In addition, modifications to the PLAPACK infrastructure extend the capability of PLAPACK to out-of-core methods. Virtual object methods perform the linear system solve operations required for the geopotential recovery.

7.3 Algorithm and Software Development

Three different pieces of application software provide the functionality required for the completion of this research. Two data parallel applications generate the normal equations for satellite gradiometer and GPS observations. Each application implements a different approach to data parallelism depending on the complexities of the dynamic model. A single linear system solve application completes the matrix inversion. The case studies verify the software functionality through the computation of gravity field degree variances for a simulated mission scenario.

7.4 Conclusions

7.4.1 Object Oriented Programming

Object oriented programming techniques effectively manage complexity in precision orbit determination software. The association of data and algorithms into objects removes unnecessary complexity from the scope of the application developer. Object encapsulation permits the addition of new functionality or changes in implementation without modification to existing software.

The OOPOD library illustrates the modular nature of objects and demonstrates the facility of such a library. A well-designed library could

support many different applications sharing similar physical and mathematical methods.

Objects designed according to the physical and mathematical entities in the POD problem induce computational algorithms which mirror the natural description of the solution process. For example, POD applications manipulate satellite objects in terms of the forces governing the satellite motion. The similarity between algorithm description and software implementation enhances the readability and maintainability of code.

The initial development cost of an object oriented design slightly exceeds structured programming techniques due to the additional work of developing the object infrastructure. Reuse of software through the addition of new functionality (which is a much simpler process when working in terms of objects) and incorporation into libraries offsets the development costs.

7.4.2 Parallel Processing

The PLAPACK library infrastructure simplifies the development of high performance parallel linear algebra functions. Algorithms developed using the PLAPACK style of coding mirrors the natural description of linear algebra block algorithms.

The use of vendor optimized Level 3 BLAS maximizes the computational performance of each processor. PLAPACK communication based on PBMD and MPI efficiently moves data between processors.

Virtual object performance depends on the available I/O hardware and the method of data communication between disk and memory. Effective performance requires the movement in data in contiguous blocks. Algorithms that minimize I/O requests achieve good performance on systems with shared file systems such as the University of Texas Cray T3E.

PLAPACK objects permit the integration of parallel memory resident and out-of-core linear algebra operations into a single general implementation. The complexity of an architecture independent I/O infrastructure exists completely within the object. Linear algebra object views continue to provide the interface with user applications. The addition of object methods which mirror standard C I/O functions provide the functionality to transfer data between memory and disk.

7.4.3 Algorithm and Application Design

The complexity of dynamic and observation models, the amount of memory required for linear algebra operations and the existence of inherently serial computations all influence the parallel performance of POD

applications. In particular, the propagation of satellite trajectories governed by complex dynamic models through numerical integration techniques significantly inhibit the parallelization of POD code.

Many design options exist for the parallelization of POD applications. The combination data-parallel and work-parallel design attains good performance for the satellite gradiometer application. The numerical integration creates a bottleneck in the GPS application due to the increase in the number of satellites and the complexity of the dynamic model.

The three software applications, **gradio**, **gps** and **xena**, illustrate the successful merging of parallel processing and POD applications. The results of the case studies confirm the software design approach by verifying previous results presented by Schrama [1991], Koop [1993] and Visser [1994].

7.5 Recommendations

7.5.1 Object Oriented Programming

The OOPOD infrastructure prototypes only a subset of the components required for generalized POD applications. A complete implementation requires additional force models (e.g., drag, third-body) , additional observation models (e.g., range-rate, GPS double difference), more complete satellite models (e.g., satellite orientation) and additional estimation and

integration techniques (e.g., orthogonal transforms, runge-kutta, second-order methods). A complete implementation would also include libraries of derived objects corresponding to specific satellite systems (e.g., TOPEX/Poseidon, GPS).

7.5.2 Parallel Processing

The numerical integration of complex dynamic models constitutes the primary obstruction to generalized parallel POD applications. Futher progress in the development of generalized applications requires an examination of parallel numerical integration algorithms.

The current PLAPACK I/O design exposes the file structure and requires the application developer to manage the memory to disk communications. All I/O operations would become transparent by extending the view functionality to control data caching within the memory resident portion of the virtual object.

The MPI-2 specification includes an architecture independent parallel I/O interface. Adherence to the MPI-2 specification could simplify the PLAPACK I/O implementation and increase the portability of the PLAPACK library. The potential benefits warrant the close examination of MPI-2.

7.5.3 Algorithm and Application Design

An increase in the number of processors used in the analysis would permit an increase in the geopotential resolutions. Similar case studies performed on larger architectures would verify the scalability and provide information concerning the stability of the least squares solution at larger problem sizes.

In the sun synchronous case study, the inclusion of satellite tracking information causes a significant improvement in the geopotential degree variances. The result does not necessarily imply a corresponding increase in estimated coefficient accuracy. A rigorous simulation study would clarify the relationship between degree variances and coefficient accuracy for high resolution geopotential models.

Appendix A CRF to TRF Transformations

The first order transformation from the CRF to the TRF is accomplished via nine plane transformations. The first six rotations correspond to external torque effects due to general precession and nutation. The last three rotations correspond to torque free motion of the pole due to time varying rotation and polar motion

$$\begin{aligned}\bar{r}_{TRF} = {} & R_2(-x_p)R_1(-y_p)R_3(\alpha_{gast}) \\ & R_1(-\varepsilon-\Delta\varepsilon)R_3(\Delta\psi)R_1(\varepsilon)R_3(-z_A)R_2(\theta_A)R_3(\zeta_a)\bar{r}_{CRF}\end{aligned} \tag{47}$$

The evaluation of the terms found in Equation (47) require the definition of several time systems. The time systems may be grouped into three categories: dynamic time, atomic time, and sidereal time. Dynamic time is the independent variable in the equations of motion. Atomic time is a uniformly running time scale used for basic time keeping purposes. Sidereal time is a measure of the Earth rotation. The primary rotation angle between the *CRF* and the *TRF* is Greenwich Apparent Sidereal Time (*GAST*). Variations in Earth rotation are usually expressed as differences between *UT1* and *UTC*. The different time systems are related to *GAST* via the following relationship.

$$\alpha_{GAST} = \alpha_{GMST_0} + \underset{Eqn.E}{\frac{d(\alpha_{GMST})}{dt}}\left[TDT - (TDT - TAI) - (TAI - UTC) - (UTC - UT1)\right] + \quad (48)$$

International Atomic Time (*TAI*) is related to Terrestrial Dynamic Time (*TDT*) by the relation $TDT = TAI + 32.184$ seconds. The fundamental unit of *TAI* is one SI second which is equivalent to 9,192,631,770 periods of transition between two hyperfine levels of the ground state of Cs-133 [Bock, 1996]. The SI day is defined as 86400 seconds, and the Julian century as 36525 days.

The Julian date is defined as the number of days from 12^h UT Jan 1, 4713 BCE. The standard epoch is taken as J2000 = JD 2451545.0 = 2000 Jan $1^d.5$ UT. Time indexes for the rotational quantities are usually expressed in Julian centuries since J2000.

$$T = \frac{JD - 2451545.0}{36525} \quad (49)$$

For civil time keeping purposes, Universal Coordinated Time (*UTC*) is an atomic time-keeping standard to which leap seconds are added and subtracted such that $|DUT1| = |UTC - UT1| < 0.9$ seconds.

Greenwich Apparent Sidereal Time (*GAST*) is the local hour angle between the Greenwich meridian and the true vernal equinox (i.e., corrected

for precession and nutation). Green Mean Sidereal Time (*GMST*) is not corrected for nutation effects. The two angles are related by the equation of equinoxes where ε is the mean obliquity, Δε is the nutation in obliquity, and Δψ is the nutation in longitude.

$$Eq.E = \alpha_{GAST} - \alpha_{GMST} = \Delta\psi \cos(\varepsilon + \Delta\varepsilon) \tag{50}$$

The apparent motion of the sun about the Earth is non-uniform due to the eccentricity of the Earth's orbit. Universal Time (*UT*) is defined as the hour angle of a fictitious mean sun which moves with constant velocity along the equator. *UT1* is universal time corrected for polar motion. The relationship between *UT1* and *GMST* is given where T_U expresses the fraction of Julian century since J2000 UT.

$$\begin{aligned}
\alpha_{GMST} &= UT1 + 6^h 41^m 50.^s 54841 + \\
&\quad 8640184.^s 812866 T_U + 0.^s 093104 T_U^2 + 6.^s 2x10^{-6} T_U^3 \\
\frac{d(\alpha_{GMST})}{dt} &= 1^s.002737909350795 + 5^s.9005x10^{-11} T_U - 5^s.9x10^{-15} T_U^2 \\
\alpha_{GMST} &= \alpha_{GMST_0} + \frac{d(\alpha_{GMST})}{dt} UT1
\end{aligned} \tag{51}$$

The main motion of the Earth's rotation axis is due to luni-solar attraction on the Earth's equatorial bulge. The precession period is 25,800 years with an amplitude of 23º.5. The combination of this effect with the precession caused by other planetary bodies is termed general precession. The

nutation of the Earth's rotation axis is a shorter period oscillation with periods of 1 day to 18.6 years.

The CRF is defined as a geocentric, equatorial frame with the mean equinox and equatorial of J2000 according to the 1976 IAU convention. The definition is supplemented by the 1980 nutation series for an Earth with a liquid core and elastic mantle [Wahr, 1979]. The precession and nutation quantities are given in terms of fraction of Julian century since J2000 and time from specified epoch.

$$\begin{aligned}
\zeta_A &= (2306''.2181 + 1''.39656T - 0''.000139T^2)t + \\
&\quad (0''.30188 - 0''.000344T)t^2 + 0''.017998t^3 \\
z_Z &= (2306''.2181 + 1''.39656T - 0''.000139T^2)t + \\
&\quad (1''.09468 - 0''.000066T)t^2 + 0''.018203t^3 \\
\theta_A &= (2004''.3109 - 0''.85330T - 0''.000217T^2)t - \\
&\quad (0''.42665 - 0''.000217T)t^2 - 0''.041833t^3
\end{aligned} \tag{52}$$

$$\begin{aligned}
\varepsilon = {} & (84381''.448 - 46''.8150T + 0''.00059T^2 + 0''.001813T^3) + \\
& (-46''.815 - 0''.00177T + 0''.005439T^2)t + \\
& (-0''.00059 + 0''.005439T)t^2 + 0''.00181t^3
\end{aligned} \tag{53}$$

The nutation in obliquity and nutation in longitude are defined in terms of 5 fundamental arguments of the sun and moon: mean anomaly of moon (l), mean anomaly of sun (l'), mean argument of latitude of moon (F), mean

elongation of moon from the sun (D) and mean longitude of ascending lunar node (Ω).

$$\begin{aligned}\Delta\psi &= \sum_{j=1}^{N}\left[\left(A_{0j}+A_{1j}T\right)\sin\left(\sum_{i=1}^{5}k_{ji}\alpha_i(T)\right)\right]\\ \Delta\varepsilon &= \sum_{j=1}^{N}\left[\left(B_{0j}+B_{1j}T\right)\cos\left(\sum_{i=1}^{5}k_{ji}\alpha_i(T)\right)\right]\end{aligned} \tag{54}$$

$$\begin{aligned}\alpha_1 &= l = 485866''.733 + (1325^r + 715922''.633)T + 31''.310T^2 + 0''.064T^3\\ \alpha_2 &= l' = 1287009''.804 + (99^r + 1292581''.224)T - 0''.577T^2 - 0''.012T^3\\ \alpha_3 &= F = 335778''.877 + (1342^r + 295263''.137)T - 13''.257T^2 + 0''.011T^3\\ \alpha_4 &= D = 1072261''.307 + (1236^r + 1105601''.328)T - 6''.891T^2 + 0''.019T^3\\ \alpha_5 &= \Omega = 450160''.280 - (5^r + 482890''.539)T + 7''.455T^2 + 0''.008T^3\end{aligned} \tag{55}$$

Movement of the rotation axis is also influenced by elastic properties of the Earth and the exchange of angular momentum between the solid Earth, oceans and atmosphere. The polar motion of the true celestial pole as defined by precession and nutation corrections contains a free component with a period of about 430 days (Chandler period), and a forced components with dominant terms at the diurnal (tidal forces) and annual (atmosphere excitations) periods. The polar motion parameters and difference *UTC-UT1* are tabulated from observational values.

Appendix B Conversion of Gravity Gradients from Body Fixed Spherical to Topographic Coordinates

The expressions for the gravitational accelerations and gradients expressed in topographic coordinates were restated by Bettadpur [1992].

$$\begin{aligned} U_e &= \frac{1}{r\cos\phi} U_\lambda \\ U_n &= \frac{1}{r} U_\phi \\ U_u &= U_r \end{aligned} \tag{56}$$

$$\begin{aligned} U_{ee} &= \frac{1}{r^2\cos^2\phi} U_{\lambda\lambda} - \frac{\sin\phi}{r^2\cos\phi} U_\phi + \frac{1}{r} U_r \\ U_{en} &= \frac{1}{r^2\cos\phi} U_{\lambda\phi} + \frac{\sin\phi}{r^2\cos^2\phi} U_\lambda \\ U_{eu} &= \frac{1}{r\cos\phi} U_{\lambda r} - \frac{1}{r^2\cos\phi} U_\lambda \\ U_{nn} &= \frac{1}{r^2} U_{\phi\phi} + \frac{1}{r} U_r \\ U_{nu} &= \frac{1}{r} U_{\phi r} - \frac{1}{r^2} U_\phi \\ U_{uu} &= U_{rr} \end{aligned} \tag{57}$$

The following is C-like pseudocode of an efficient algorithm to make the conversion. The partials with respect to the gravity field coefficients are converted using Level 1 BLAS routines.

```
/* Input */
```

```
    a = [ U_r U_λ U_φ ]
    g = [ U_rr U_rλ U_rφ U_λλ U_λφ U_φφ ]
    ap = [ dU_r dα dU_λ dα dU_φ dα ]
    gp = [ dU_rr dα dU_rλ dα dU_rφ dα dU_λλ dα dU_λφ dα dU_φφ dα ]

/* accelerations */
    a[1] *= 1 / ( r * cos φ );
    a[2] *= 1 / r;
    tmp = a[0] ; a[0] = a[1] ; a[1] = a[2] ; a[2] = tmp;

/* gradients */
    g[1] *= 1 / ( r * cos φ );          g[1] += -a[0] / r;
    g[2] *= 1 / r;                      g[2] += - a[1] / r;
    g[3] *= 1 / ( r^2 * cos^2 φ );      g[3] += a[2] / r - ( a[1] * sin φ ) / ( r * cos φ );
    g[4] *= 1 / ( r^2 * cos φ );        g[4] += ( a[0] * sin φ ) / ( r * cos φ );
    g[5] *= 1 / r^2;                    g[5] += a[2] / r;
    tmp = g[0] ; g[0] = g[3] ; g[3] = g[5] ; g[5] = tmp;
    tmp = g[1] ; g[1] = g[4] ; g[4] = g[2] ; g[2] = tmp;

/* acceleration partials (length == number of partials) */
    xSCAL ( length, 1 / ( r * cos φ ), ap [ 1 * length ], 1 );
    xSCAL ( length, 1 / r, ap [ 2 * length ], 1 );
    xSWAP ( length, ap [ 0 * length ], 1, ap [ 1 * length ], 1 );
    xSWAP ( length, ap [ 1 * length ], 1, ap [ 2 * length ], 1 );

/* gradient partials */
    xSCAL ( length, 1 / ( r * cos φ ), gp [ 1 * length ], 1 );
    xSCAL ( length, 1 / r, gp [ 2 * length ], 1 );
    xSCAL ( length, 1 / ( r^2 * cos^2 φ ), gp [ 3 * length ], 1 );
    xSCAL ( length, 1 / ( r^2 * cos φ ), gp [ 4 * length ], 1 );
    xSCAL ( length, 1 / r^2,  gp [ 5 * length ], 1 );
    xAXPY ( length, 1 / r, ap [ 0 * length ], 1, gp [ 1 * length ], 1 );
```

```
    xAXPY ( length, 1 / r, ap [ 1 * length ], 1, gp [ 2 * length ], 1 );
    xAXPY ( length, 1 / r, ap [ 2 * length ], 1, gp [ 3 * length ], 1 );
    xAXPY ( length, -sin φ / ( r * cos φ ), ap [ 1 * length ], 1, gp [ 3 * length ], 1 );
    xAXPY ( length, sin φ / ( r * cos φ ), ap [ 0 * length ], 1, gp [ 4 * length ], 1 );
    xAXPY ( length, 1 / r, ap [ 2 * length ], 1, gp [ 5 * length ], 1 );
    xSWAP ( length, gp [ 0 * length ], 1, gp [ 3 * length ], 1 );
    xSWAP ( length, gp [ 3 * length ], 1, gp [ 5 * length ], 1 );
    xSWAP ( length, gp [ 1 * length ], 1, gp [ 4 * length ], 1 );
    xSWAP ( length, gp [ 4 * length ], 1, gp [ 2 * length ], 1 );

/* Output */
    a = [ U_e U_n U_u ]
    g = [ U_ee U_en U_eu U_nn U_nu U_uu ]
    ap = [ dU_e dα dU_n dα dU_u dα ]
    gp = [ dU_ee dα dU_en dα dU_eu dα dU_nn dα dU_nu dα dU_uu dα ]
```

Appendix C OOPOD Object Methods

C.1 Generic Physical Object

C.1.1 Properties

observable	Specifies the ability to use the object as an observable quantity
dynamic	Specifies the requirement of numerical integration to propagate the object's state
force	Specifies the ability to use the object as a dynamic force effecting the motion of a satellite object

C.1.2 Interface

```
int32 PhysObj_create          ( PhysObj *    object );
int32 PhysObj_free            ( PhysObj *    object );
int32 PhysObj_isobservable    ( PhysObj *    object );
int32 PhysObj_isdynamic       ( PhysObj *    object );
int32 PhysObj_isforce         ( PhysObj *    object );
```

C.2 Generic Mathematical Object

C.2.1 Properties

realized	Specifies the realization of the mathematical object via association with physical objects

C.2.2 Interface

```
int32 MathObj_create          ( MathObj *    object );
int32 MathObj_free            ( MathObj *    object );
int32 MathObj_isrealized      ( MathObj *    object );
```

C.3 Earth Orientation Parameters Table

C.3.1 Properties

number_points	Number of points in the table
interval	Interval between points in the table
first_ut	Epoch of first entry in the table
last_ut	Epoch of last entry in the table
ut	Epoch of current table interpolation
xp	Polar motion of current table interpolation
yp	Polar motion of current table interpolation
ut1_tai	Time difference of current table interpolation
dutdtai	Variation in time difference of current table interpolation
et_ut1	Time difference of current table interpolation
ut_table	Array of table entries
xp_table	Array of table entries
yp_table	Array of table entries
ut1_tai_table	Array of table entries

C.3.2 Interface

```
int32 EopTable_create          ( char *          filename,
                                 int32           file_format,
                                 EopTable *      table );
int32 EopTable_free            ( EopTable *      table );
int32 EopTable_calculate       ( EopTable        table,
                                 double          julian_et );
int32 EopTable_ut              ( EopTable        table );
int32 EopTable_xp              ( EopTable        table );
int32 EopTable_yp              ( EopTable        table );
int32 EopTable_ut1_minus_tai   ( EopTable        table );
int32 EopTable_dutdtai         ( EopTable        table );
int32 EopTable_et_minus_ut1    ( EopTable        table );
```

C.4 Reference Frame

C.4.1 Properties

julian_et	Epoch of current frame evaluation
days_from_epoch	Days from true-of-date reference epoch at evaluation
days_from_epoch_ut1	Time difference at evaluation
greenwich_mean	Hour angle at evaluation
greenwich_true	Hour angle at evaluation
et_ut1	Time difference at evaluation
dutdta	Variation in time difference at evaluation
dghadt	Variation in hour angle at evaluation
xp	Polar motion angle at evaluation
yp	Polar motion angle at evaluation
nutation_longitude	Nutation angle at reference epoch
nutation_obliquity	Nutation angle at reference epoch
nutation_right_ascension	Nutation angle at reference epoch
zeta	Precession angle at reference epoch
z	Precession angle at reference epoch
theta	Precession angle at reference epoch
mean_obliquity	Defining frame angle at reference epoch
true_obliquity	Defining frame angle at reference epoch
epoch2meanofdate	Rotation matrix
meanofdate2trueofdate	Rotation matrix
table	Earth orientation parameter table
object	Physical object properties

C.4.2 Interface

```
int32 RefFrame_create    ( char *       filename,
                           int32        file_format,
                           RefFrame *   frame );
int32 RefFrame_free      ( RefFrame *   frame );
```

```
int32 RefFrame_calculate                    ( int32      set_tod,
                                              float64    epoch,
                                              RefFrame   frame );
int32 RefFrame_J2000_to_meanofdate          ( RefFrame   frame,
                                              float64 *  a );
int32 RefFrame_meanofdate_to_trueofdate     ( RefFrame   frame,
                                              float64 *  a );
int32 RefFrame_trueofdate_to_geocentric     ( RefFrame   frame,
                                              float64 *  a,
                                              float64 *  b );
int32 RefFrame_geocentric_to_topographic    ( float64    radius,
                                              float64    lambda,
                                              float64    phi,
                                              float64 *  a );
int32 RefFrame_geocentric_to_rtn            ( float64 *  pos,
                                              float64 *  vel,
                                              float64 *  a );
int32 RefFrame_general                      ( float64    psi,
                                              float64    theta,
                                              float64    phi,
                                              float64 *  a );
int32 RefFrame_cartesian_to_spherical       ( float64 *  x,
                                              float64 *  radius,
                                              float64 *  lambda,
                                              float64 *  phi );
int32 RefFrame_spherical_to_cartesian       ( float64 *  x,
                                              float64    radius,
                                              float64    lambda,
                                              float64    phi );
int32 RefFrame_merge                        ( int32      side,
                                              int32      trans,
                                              float64 *  a,
                                              float64 *  b );
int32 RefFrame_vector                       ( int32      trans,
                                              float64 *  a,
                                              float64 *  x );
int32 RefFrame_tensor                       ( int32      trans,
                                              float64 *  a,
                                              float64 *  g );
int32 RefFrame_vector_partials              ( int32      trans,
                                              float64 *  a,
                                              float64 *  xp,
                                              int32      length );
int32 RefFrame_tensor_partials              ( int32      trans,
                                              float64 *  a,
                                              float64 *  gp,
                                              int32      length );
```

```
int32 RefFrame_set_global     ( RefFrame     frame );
int32 RefFrame_get_global     ( RefFrame *   frame );
int32 RefFrame_isobservable   ( RefFrame     frame );
int32 RefFrame_isdynamic      ( RefFrame     frame );
int32 RefFrame_isforce        ( RefFrame     frame );
```

C.5 Legendre Associated Functions

C.5.1 Properties

shape	Shape of geopotential model
degree	Maximum degree of expansion
order	Maximum order of expansion
ldim	Leading dimension of all matrix data
deriv	Specifies calculation of which derivatives
u	Argument of function
anm	Recursion coefficients
bnm	Recursion coefficients
fnm	Recursion coefficients
pnm	Function evaluations
dpnm	Function evaluations
ddpnm	Function evaluations
object	Mathematical object properties

C.5.2 Interface

```
int32 Legendre_create     ( int32       shape,
                            int32       degree,
                            int32       order,
                            int32       deriv,
                            Legendre *  legendre );
int32 Legendre_free       ( Legendre *  legendre );
int32 Legendre_calculate  ( Legendre    legendre,
                            float64     u );
int32 Legendre_index      ( Legendre    legendre,
                            int32       n,
                            int32       m );
```

int32 Legendre_index	(Legendre int32	legendre, n,
	int32	m);
int32 Legendre_shape	(Legendre	legendre);
int32 Legendre_degree	(Legendre	legendre);
int32 Legendre_order	(Legendre	legendre);
int32 Legendre_ldim	(Legendre	legendre);
int32 Legendre_deriv	(Legendre	legendre);
float64 Legendre_u	(Legendre	legendre);
float64 * Legendre_anm	(Legendre	legendre);
float64 * Legendre_bnm	(Legendre	legendre);
float64 * Legendre_fnm	(Legendre	legendre);
float64 * Legendre_pnm	(Legendre	legendre);
float64 * Legendre_dpnm	(Legendre	legendre);
float64 * Legendre_ddpnm	(Legendre	legendre);
int32 Legendre_isrealized	(Legendre *	legendre);

C.6 Gravity Field

C.6.1 Properties

measurement	Desired geopotential evaluation
coordinates	Desired coordination system
shape	Specifies geopotential model representation
radius	Function argument for current evaluation
lambda	Function argument for current evaluation
phi	Function argument for current evaluation
ae	Geophysical constant
mu	Geophysical constant
potential	Function evaluation
acceleration0	Function evaluation in desired coordinates
acceleration1	Function evaluation in desired coordinates
acceleration2	Function evaluation in desired coordinates
gradient0	Function evaluation in desired coordinates
gradient1	Function evaluation in desired coordinates
gradient2	Function evaluation in desired coordinates
gradient3	Function evaluation in desired coordinates

gradient4	Function evaluation in desired coordinates
gradient5	Function evaluation in desired coordinates
cnm	Summation coefficients
snm	Summation coefficients
dpotdcnm	Function partials in desired coordinates
dpotdsnm	Function partials in desired coordinates
da0dcnm	Function partials in desired coordinates
da0dsnm	Function partials in desired coordinates
da1dcnm	Function partials in desired coordinates
da1dsnm	Function partials in desired coordinates
da2dcnm	Function partials in desired coordinates
da2dsnm	Function partials in desired coordinates
dg0dcnm	Function partials in desired coordinates
dg0dsnm	Function partials in desired coordinates
dg1dcnm	Function partials in desired coordinates
dg1dsnm	Function partials in desired coordinates
dg2dcnm	Function partials in desired coordinates
dg2dsnm	Function partials in desired coordinates
dg3dcnm	Function partials in desired coordinates
dg3dsnm	Function partials in desired coordinates
dg4dcnm	Function partials in desired coordinates
dg4dsnm	Function partials in desired coordinates
dg5dcnm	Function partials in desired coordinates
dg5dsnm	Function partials in desired coordinates
legendre	Basis function for expansion
object	Physical object properties

C.6.2 Interface

int32 GravityField_create	**(**	**int32**	**shape,**
		int32	**degree,**
		int32	**order,**
		int32	**measurement,**

```
                                    int32           coordinates,
                                    char *          filename,
                                    int32           file_format,
                                    GravityField *  gfield );
int32 GravityField_calculate     (  GravityField    gfield,
                                    float64         radius,
                                    float64         lambda,
                                    float64         phi );
int32 GravityField_free          (  GravityField *  gfield );
int32 GravityField_partial_length (  GravityField   gfield,
                                    int32           estim_param,
                                    int32 *         length );
int32 GravityField_extract_partials (  GravityField  gfield,
                                    int32           which_set,
                                    int32           estim_param,
                                    float64 *       partials );
float64 GravityField_potential   (  GravityField    gfield );
float64 GravityField_acceleration0 (  GravityField  gfield );
float64 GravityField_acceleration1 (  GravityField  gfield );
float64 GravityField_acceleration2 (  GravityField  gfield );
float64 GravityField_gradient0   (  GravityField    gfield );
float64 GravityField_gradient1   (  GravityField    gfield );
float64 GravityField_gradient2   (  GravityField    gfield );
float64 GravityField_gradient3   (  GravityField    gfield );
float64 GravityField_gradient4   (  GravityField    gfield );
float64 GravityField_gradient5   (  GravityField    gfield );
float64 GravityField_acceleration (  GravityField gfield,
                                    float64 *       acceleration );
float64 GravityField_gradient    (  GravityField gfield,
                                    float64 *       gradient );
int32 GravityField_index         (  GravityField    gfield,
                                    int32           n,
                                    int32           m );
int32 GravityField_shape         (  GravityField    gfield );
int32 GravityField_degree        (  GravityField    gfield );
int32 GravityField_order         (  GravityField    gfield );
int32 GravityField_ldim          (  GravityField    gfield );
int32 GravityField_isobservable  (  GravityField *  gfield );
int32 GravityField_isdynamic     (  GravityField *  gfield );
int32 GravityField_isforce       (  GravityField *  gfield );
```

C.7 Satellite

C.7.1 Properties

epoch	Reference epoch
epoch_state	Satellite state at reference epoch
arc_length	Seconds since reference epoch
state	Satellite state at seconds since reference
number_forces	Number of forces acting on satellite
force_tags	List of forces acting on satellite
forces	Force objects acting on satellite
object	Physical object properties

C.7.2 Interface

```
int32 Satellite_create          ( float64      epoch,
                                  int32        coordinates,
                                  float64 *    state,
                                  Satellite *  satellite,
                                  int32        number_forces,
                                               /* vargs */ );
int32 Satellite_free            ( Satellite *  satellite );
int32 Satellite_epoch           ( Satellite    satellite,
                                  float64 *    epoch );
int32 Satellite_state           ( Satellite    satellite,
                                  int32        coordinates,
                                  float64 *    state );
int32 Satellite_force           ( Satellite    satellite,
                                  int32        which_force,
                                  int32 *      force_tag,
                                  PhysObj *    force );
float64 Satellite_epoch         ( Satellite    satellite );
float64 Satellite_arc_length    ( Satellite    satellite );
int32 Satellite_number_forces   ( Satellite    satellite );
```

C.8 Satellite Gradiometer Observable

C.8.1 Properties

psi	Rotation angle between satellite and gradiometer frames
theta	Rotation angle between satellite and gradiometer frames
phi	Rotation angle between satellite and gradiometer frames
a	Gradiometer measured accelerations
g	Gradiometer measured gradients
sat2xg	Rotation matrix
topographic2xg	Rotation matrix
partials	Gradient partial derivatives
partial_length	Number of gradient partial derivatives
gravity_field	Gravity field influencing gradiometer
satellite	Satellite containing the gradiometer
object	Physical object properties

C.8.2 Interface

```
int32 Gradiometer_create                ( GravityField      gravity_field,
                                          Satellite         satellite,
                                          float64           psi,
                                          float64           theta,
                                          float64           phi,
                                          Gradiometer *     gradio );
int32 Gradiometer_free                  ( Gradiometer *     gradio );
int32 Gradiometer_calculate             ( Gradiometer       gradio );
int32 Gradiometer_extract_partials      ( Gradiometer       gradio,
                                          float64           which_set,
                                          float64           estim_param,
                                          float64 *         partials );
int32 Gradiometer_gradient              ( Gradiometer       gradio,
                                          float64 **        gradient );
float64 Gradiometer_psi                 ( Gradiometer       gradio );
float64 Gradiometer_theta               ( Gradiometer       gradio );
float64 Gradiometer_phi                 ( Gradiometer       gradio );
float64 Gradiometer_g                   ( Gradiometer       gradio );
GravityField Gradiometer_gravity_field  ( Gradiometer       gradio );
```

C.9 Satellite-to-Satellite Ranging Observable

C.9.1 Properties

elevation_mask	Elevation mask for observation
below_mask	Specifies whether current observation is below elevation mask
range	Current observation
drangedxhigh	Partials with respect to high satellite position
partials	Observation partials
integ_high	Integration object
integ_low	Integration object
object	Physical object properties

C.9.2 Interface

```
int32 SatTrack_create                ( float64          elevation_mask,
                                       MsilibABFS       integ_high,
                                       MsilibABFS       integ_low,
                                       SatTrack *       sst );
int32 SatTrack_free                  ( SatTrack *       sst );
int32 SatTrack_calculate             ( SatTrack         sst );
int32 SatTrack_extract_partials      ( SatTrack         sst,
                                       float64          estim_param,
                                       float64 *        partials );
int32 SatTrack_number_partials       ( SatTrack         sst );
int32 SatTrack_below_mask            ( SatTrack         sst );
float64 SatTrack_range               ( SatTrack         sst );
```

C.10 Batch Estimator

C.10.1 Properties

which_observation	Specifies type of observable
sigma	Observation variance
batch_count	Number observation in current batch
observation_count	Total number of observations

batch_size	Size of accumulation batch
number_equations	Number equations in linear system
number_parameter_types	Number of classes of estimated parameters
parameter_type	List of parameter types
number_parameters	Number equations per parameter type
batch_matrix	Matrix of partial arrays in current batch
normal_matrix	Linear system operator
normal_array	Linear system right-hand-side
covariance	Inverted linear system operator
subarc_matrix	Linear system operator for subarc parameters
correl_matrix	Correlation matrix between subarc parameters and global parameters
object	Mathematical object properties

C.10.2 Interface

```
int32 Batch_create                  ( float64          batch_size,
                                      Batch *          estimator );
int32 Batch_free                    ( Batch *          estimator );
int32 Batch_initialize_parameters   ( Batch            estimator,
                                      float64          n_parm_types,
                                                       /* vargs */ );
int32 Batch_realize                 ( Batch            estimator,
                                      float64          which_obs,
                                      PhysObj          observation,
                                      float64          sigma );
int32 Batch_extract_observation     ( Batch            estimator,
                                      PhysObj          observation );
int32 Batch_increment_subarc        ( Batch            estimator );
int32 Batch_accumulate              ( Batch            estimator );
int32 Batch_solve                   ( Batch            estimator );
int32 Batch_extract_variance        ( Batch            estimator,
                                      float64 *        variances );
int32 Batch_observation_count       ( Batch            estimator );
int32 Batch_number_equations        ( Batch            estimator );
float64 * Batch_covariance_matrix   ( Batch            estimator );
float64 * Batch_normal_matrix       ( Batch            estimator );
int32 Batch_isfull                  ( Batch            estimator );
```

C.11 Adams-Bashforth Numerical Integrator

C.11.1 Properties

state_transition	Specifies integration of state transition information
number_equations	Number equations in integration vector
number_parameter_types	Number of classes of dynamic parameters
parameter_type	List of parameter types
number_parameters	Number equations per parameter type
t	Current epoch
state	Integration state at current epoch
satellite	Physical object associated with integration
object	Mathematical object properties

C.11.2 Interface

```
int32 MsilibABFS_create                ( float64          state_transition,
                                         float64          nord,
                                         float64          nloop,
                                         float64          alim,
                                         float64          meshsize,
                                         MsilibABFS *     integrator );
int32 MsilibABFS_free                  ( MsilibABFS *     integrator );
int32 MsilibABFS_initialize_parameters ( MsilibABFS       integrator,
                                         float64          n_parm_types,
                                                          /* vargs */ );
int32 MsilibABFS_realize               ( MsilibABFS       integrator,
                                         Satellite        satellite );
int32 MsilibABFS_propagate             ( MsilibABFS       integrator,
                                         float64          tout );
int32 MsilibABFS_restart               ( MsilibABFS       integrator );
int32 MsilibABFS_parameter_types       ( MsilibABFS       integrator,
                                         float64          which_param,
                                         float64 *        parameter_tag,
                                         float64 *        num_param );
int32 MsilibABFS_num_param_types       ( MsilibABFS       integrator );
Satellite MsilibABFS_satellite         ( MsilibABFS       integrator );
```

C.12 Gradiometer Application

```
1   #include "oopod.h"
2
3   int main ( int argc, char ** argv )
4   {
5       int32           i, restart_index = 1,
6                       satellite_gravity_field_degree = 3,
7                       gradiometer_gravity_field_degree = 20,
8                       batch_size = 5000;
9       float64         sigma = 1.0e-12,
10                      t = 0.0e+0,
11                      arc_length = 45.0e+0 * DAY2SECOND,
12                      interval = 60.0e+0,
13                      restart_interval = 0.5 * DAY2SECOND,
14                      epoch = 2447527.500650278e+0,
15                      state [ 6 ] = { 6637163.04066062e+0, 0.0e+0, 0.0e+0,
16                      0.0e+0, -13.516035414281e+0, 7749.946235174633e+0 };
17      RefFrame        reference_frame = NULL;
18      GravityField    satellite_gravity_field = NULL,
19                      gradiometer_gravity_field = NULL;
20      Satellite       satellite = NULL;
21      MsilibABFS      integrator = NULL;
22      Gradiometer     gradiometer = NULL;
23      Batch           estimator = NULL;
24      RefFrame_create ( "/work/utexas/csr/byab323/EOPDAT.BIN",
25                      EOP_FILE_FORMAT, & reference_frame );
26      RefFrame_calculate ( SET_TOD, epoch, reference_frame );
27      RefFrame_set_global ( reference_frame );
28      GravityField_create ( TRIANGULAR, satellite_gravity_field_degree,
29                      atellite_gravity_field_degree, ALL_MEASUREMENTS,
30                      TOPOGRAPHIC,
```

```
                    "/work/utexas/csr/byab323/GEO.BIN.180x180",
                    GEO_FILE_FORMAT, & satellite_gravity_field );

    GravityField_create ( TRIANGULAR, gradiometer_gravity_field_degree,
                    gradiometer_gravity_field_degree,
                    ALL_MEASUREMENTS, TOPOGRAPHIC,
                    "/work/utexas/csr/byab323/GEO.BIN.180x180",
                    GEO_FILE_FORMAT, & gradiometer_gravity_field );

    Satellite_create ( epoch, GEOCENTRIC, state, & satellite, 1,
                    GEOPOTENTIAL, satellite_gravity_field );

    Gradiometer_create ( gradiometer_gravity_field, satellite,  0.0e+0, 0.0e+0,
                    00.0e+0 & gradiometer );

    MsilibABFS_create ( NO_STATE_TRANSITION, 7, 10, 1.0e-6, 30.0e+0, &
                    integrator );

    MsilibABFS_realize ( integrator, satellite );

    Batch_create ( batch_size, & estimator );

    Batch_initialize_parameters ( estimator, 1, GRAV_COEF_ALL );

    Batch_realize ( estimator, GRADIOMETER, gradiometer, sigma );

    while ( t <= arc_length )
    {

        if ( t > restart_index * restart_interval )
        {

            MsilibABFS_restart ( integrator );

            restart_index++;
        }

        MsilibABFS_propagate ( integrator, t );

        Gradiometer_calculate ( gradiometer );

        if ( Batch_isfull ( estimator ) ) Batch_accumulate ( estimator );

        Batch_extract_observation ( estimator, gradiometer );

        t += interval;
    }
```

```
61      Batch_accumulate ( estimator );
62      Batch_solve ( estimator );
63      Batch_free ( & estimator );
64      Gradiometer_free ( & gradiometer );
65      MsilibABFS_free ( & integrator );
66      Satellite_free ( & satellite );
67      GravityField_free ( & gradiometer_gravity_field );
68      GravityField_free ( & satellite_gravity_field );
69      RefFrame_free ( & reference_frame );
70  }
```

C.13 GPS Application

```
1   #include "oopod.h"
2   int main ( int argc, char ** argv )
3   {
4       int32           i, number_gps_satellites = 25,
5                       restart_index = 1,
6                       satellite_gravity_field_degree = 3,
7                       batch_size = 5000;
8       float64         sigma = 1.0e-3,
9                       elevation_mask = 10.0e+0 * DEGREE2RADIAN,
10                      t = 0.0e+0,
11                      arc_length = 12.0e+0 * DAY2SECOND,
12                      interval = 60.0e+0,
13                      restart_interval = 0.5 * DAY2SECOND,
14                      epoch = 2447527.500650278e+0,
15                      state_gps [ 25 * 6 ] = { /* GPS states may be hardcoded
16                      here */ },
17                      state_low [ 6 ] = {  6633085.575,-405.372, 232450.642,
18                      -271.802957, 13.505910, 7745.171574 },
```

```
19      RefFrame        reference_frame = NULL;
20      GravityField    satellite_gravity_field = NULL;
21      Satellite       * satellite_gps = NULL,
22                      satellite_low = NULL;
23      MsilibABFS      * integrator_gps = NULL,
24                      integrator_low = NULL;
25      SatTrack        sst = NULL;
26      Batch estimator = NULL;
27      RefFrame_create ( "/work/utexas/csr/byab323/EOPDAT.BIN",
28                      EOP_FILE_FORMAT,  & reference_frame );
29      RefFrame_calculate ( SET_TOD, epoch, reference_frame );
30      RefFrame_set_global ( reference_frame );
31      GravityField_create ( TRIANGULAR, satellite_gravity_field_degree,
32                      satellite_gravity_field_degree, ALL_MEASUREMENTS,
33                      TOPOGRAPHIC,
34                      "/work/utexas/csr/byab323/GEO.BIN.180x180",
35                      GEO_FILE_FORMAT, & satellite_gravity_field );
36      Satellite_create ( epoch, GEOCENTRIC, state_low, & satellite_low, 1,
37                      GEOPOTENTIAL, satellite_gravity_field );
38      MsilibABFS_create ( STATE_TRANSITION, 7, 10, 1.0e-6, 30.0e+0, &
39                      integrator_low );
40      MsilibABFS_initialize_parameters ( integrator_low, 2, INITIAL_STATE,
41                      GRAV_COEF_ALL );
42      MsilibABFS_realize ( integrator_low, satellite_low );
43      satellite_gps = calloc ( number_gps_satellites, sizeof ( Satellite ) );
44      integrator_gps = calloc ( number_gps_satellites, sizeof ( MsilibABFS ) );
45      for ( i = 0 ; i < number_gps_satellites ; i++ )
46      {
```

```
47          Satellite_create ( epoch, GEOCENTRIC, & state_gps [ 6 * i ], &
48                  satellite_gps [ i ], 1, GEOPOTENTIAL,
49                  satellite_gravity_field );

50          MsilibABFS_create ( STATE_TRANSITION, 7, 10, 1.0e-6, 30.0e+0, &
51                  integrator_gps [ i ] );

52          MsilibABFS_initialize_parameters ( integrator_gps [ i ], 2,
53                  INITIAL_STATE, GRAV_COEF_ALL );

54          MsilibABFS_realize ( integrator_gps [ i ], satellite_gps [ i ] );

55      }

56      SatTrack_create ( elevation_mask, integrator_gps [ 0 ], integrator_low, &
57              sst );

58      Batch_create ( batch_size, & estimator );

59      Batch_initialize_parameters ( estimator, 2, INITIAL_STATE, SUBARC,
60              GRAV_COEF_ALL, GLOBAL );

61      Batch_realize ( estimator, SST, sst, sigma );

62      while ( t <= arc_length )

63      {

64          if ( t > restart_index * restart_interval )

65          {

66              MsilibABFS_restart ( integrator_low );

67              for ( i = 0 ; i < number_gps_satellites ; i++ )

68                  MsilibABFS_restart ( integrator_gps [ i ] );

69              Batch_increment_subarc ( estimator );

70              restart_index++;

71          }

72          MsilibABFS_propagate ( integrator_low, t );
```

```
73          for ( i = 0 ; i < number_gps_satellites ; i++ )
74          {
75              MsilibABFS_propagate ( integrator_gps [ i ], t );
76              SatTrack_calculate ( integrator_gps [ i ], integrator_low, sst );
77              if ( ! SatTrack_below_mask ( sst ) )
78              {
79                  if ( Batch_isfull ( estimator ) )
80                      Batch_accumulate ( estimator );
81                  Batch_extract_observation ( estimator, sst );
82              }
83          }
84          t += interval;
85      }
86      Batch_accumulate ( estimator );
87      Batch_solve ( estimator );
88      Batch_free ( & estimator );
89      MsilibABFS_free ( & integrator_low );
90      Satellite_free ( & satellite_low );
91      for ( i = 0 ; i < number_gps_satellites ; i++ )
92      {
93          MsilibABFS_free ( & integrator_gps [ i ] );
94          Satellite_free ( & satellite_gps [ i ] );
95      }
```

```
96          SatTrack_free ( & sst );
97      free ( integrator_gps );
98      free ( satellite_gps );
99      free ( sst );
100     GravityField_free ( & satellite_gravity_field );
101     RefFrame_free ( & reference_frame );
102 }
```

Appendix D Gravity Field Indexing Techniques

D.1 Number of Coefficients in a Degree and Order l Expansion

The coefficients in a spherical harmonic expansion may be organized in terms of two lower triangular matrices. The first matrix stores the coefficients of the cosine terms, and the second stores the coefficients of the sine terms. While the order zero sine terms are undefined, it is convenient use matrices of equal dimensions with the first column of the sine coefficient matrix set to zero.

The number of coefficients is simply the count of the elements contained within the two lower triangular matrices minus the number of elements in the first column of the sine coefficient matrix. The number of elements in a lower triangular matrix is $\frac{n(n+1)}{2}$ where n is the dimension of the matrix. An expression for the number of coefficients in terms of the maximum degree and order l may be developed.

$$n = 2\left(\frac{(l+1)(l+2)}{2}\right) - (l+1) \tag{58}$$

This expression reduces a very simple form.

$$n = (l+1)^2 \tag{59}$$

D.2 Lower Triangular Matrix Mapped to a Linear Array

Reducing memory usage may be accomplished by packing the storage of lower triangular matrix in to linear array. A useful expression may be developed which relates the row and column indices of the matrix to the index of the linear array. The ordering of the matrix elements could be in row-major or column-major order. Four expressions will be developed. The first two expressions use indices which start a zero as is common in the C programming language and in the indexing of gravity field coefficients. The second two expressions use indices which start at one as is the convention in the FORTRAN programming language. The development of the expressions follow the same general formulas.

Column-major ordering

Index = Linear Array Dimension - Number of elements in the smallest lower triangular matrix containing the target element + Number of elements below the diagonal term + Array index starting bias

Row-major ordering

Index = Number of elements in the largest lower triangular matrix above the target element + Number of elements from the first column + Array index starting bias

The expressions for indices starting at zero will use l and m to represent the row and column indices of the lower triangular matrix. The parameter D is defined as the maximum l or m permitted. The starting bias B specifies the linear array index of the *(0,0)* element. Therefore, $B=0$ if starting at the beginning of the array.

Column-major ordering

	0	1	2	3	4	m
0	0					
1	1	5				
2	2	6	9			
3	3	7	10	12		
4	4	8	11	13	14	
l						

(60)

$$I = \frac{(D+1)(D+2)}{2} - \frac{(D-m+1)(D-m+2)}{2} + (l-m) + B \tag{61}$$

$$I = \frac{m(2D+1-m)}{2} + l + B \tag{62}$$

Row-major ordering

	0	1	2	3	4	m
0	0					
1	1	2				
2	3	4	5			
3	6	7	8	9		
4	10	11	12	13	14	
l						

(63)

$$I = \frac{l(l+1)}{2} + m + B \tag{64}$$

The expressions for indices starting at one will use i and j to represent the row and column indices of the lower triangular matrix. The parameter D is defined as the maximum i or j permitted. The starting bias B specifies the linear array index of the $(1,1)$ element. Therefore, $B=1$ if starting at the beginning of the array.

Column-major ordering

	1	2	3	4	5	j
1	1					
2	2	6				
3	3	7	10			
4	4	8	11	13		
5	5	9	12	14	15	
i						

(65)

$$I = \frac{D(D+1)}{2} - \frac{(D-j+1)(D-j+2)}{2} + (i-j) + B \tag{66}$$

$$I = \frac{j(2D+1-j)}{2} - D + (i-1) + B \tag{67}$$

Row-major ordering

	1	2	3	4	5	j
1	1					
2	2	3				
3	4	5	6			
4	7	8	9	10		
5	11	12	13	14	15	
i						

(68)

$$I = \frac{(i-1)i}{2} + (j-1) + B \tag{69}$$

D.3 Mapping to Degree-major Coefficient Storage

The triangular storage patterns are useful when working separately with the cosine and sine terms of the spherical harmonic expansion. However, when working with the entirety of the coefficients, it is often convenient to group the terms such that terms of the same degree appear contiguous in a linear array. An example is the formation of the partial derivatives with respect to the gravity field coefficients. Different observation datatypes may calculate partials to different degree and orders. By using a degree-major storage pattern, contiguous arrays aligned at the *(0,0)* coefficient will map correctly into the information matrix.

The following is an example of the linear array indices of the degree-major storage for a set of coefficients to degree and order 4.

$$\begin{array}{lllllll} \overline{C}_{lm} & 0 & & & & \overline{S}_{lm} & - \\ & 1 & 2 & & & & - & 3 \\ & 4 & 5 & 6 & & & - & 7 & 8 \\ & 9 & 10 & 11 & 12 & & - & 13 & 14 & 15 \\ & 16 & 17 & 18 & 19 & 20 & - & 21 & 22 & 23 & 24 \end{array} \tag{70}$$

The key to indexing degree-major storage is to recognize that the number of elements per degree is the sequence of odd integers. The linear array index for the first element of degree l may be determined from the prefix sum of the odd integers.

$$\sum_{i=1}^{n}(2i-1)=n^2 \tag{71}$$

Therefore the first element of degree l appears at linear array index l^2.

The indexes for general l and m appear below.

$$\begin{aligned} I_{\bar{C}_{lm}} &= l^2+m \\ I_{\bar{S}_{lm}} &= l^2+l+m, \quad m\neq 0 \end{aligned} \tag{72}$$

Expressions for indexing starting with one may be developed similarly.

Appendix E PLAPACK Virtual Objects

The integration of virtual memory functionality into the PLAPACK library provides the capability to manipulate data located on disk storage devices. At the application level, the new functionality requires only a few new PLAPACK methods. The implementation of the data mapping routines changes significantly. The organization of the PLAPACK object parameters reflect the natural partitions of disk and parallel memory. The changes in the infrastructure maintain the abstraction of the linear algebra object at the application level.

E.1 PLAPACK Object Spaces : View, Global and Virtual

The memory hierarchy in a distributed memory environment consists of many different levels from the disk through register memory. The PBMD philosophy addresses inter-processor communication and relies on vendor routines to optimize single processor operations. The PLAPACK object only maintains mappings from the abstract linear algebra entity (i.e., a matrix) to the slower memory areas: local memory, parallel memory and disk.

The PLAPACK object consists of different sets of parameters to describe the data mappings. Header information specifies the linear algebra object type and ownership of the object. Virtual space parameters describe the

disk resident data mapping of a linear algebra object. Global space describe the memory resident data mapping of a linear algebra object. View space parameters describe the portion of the data available to the application. The typical user of PLAPACK will only work in terms of view space parameters. A list of a sampling of PLAPACK object members and their associated space is presented in Table 16.

Member	Description	Space
Object Type	Type of PLAPACK object	Header
Template	Structure defining distribution	Header
Datatype	Type of data	Header
Virtual Length	Row dimension of the virtual space	Virtual
Virtual Width	Column dimension of the virtual space	Virtual
Virtual Align Row	Row alignment of the virtual space	Virtual
Virtual Align Column	Column alignment of the virtual space	Virtual
PLAPACK File Structure	PLAPACK structure containing the machine specific information required to access data located on disk	Virtual
Views	Counter tracking the number of objects referencing the same global space	Global
Master Length	Row dimension of the global space	Global
Master Width	Column dimension of the global space	Global
Master Align Row	Row alignment of the global space	Global
Master Align Column	Column alignment of the global space	Global
Master Buffer	Address of the beginning of the local global data buffer	Global
Local Leading Dimension	Leading dimension of the local global data buffer	Global
Global Length	Row dimension of the view space	View
Global Width	Column dimension of the view space	View
Global Align Row	Row alignment of the view space	View
Global Align Column	Column alignment of the view space	View
Buffer	Address of the beginning of the local view data buffer	View
Local Length	Row dimension of the local view data buffer	View
Local Width	Column dimension of the local view data buffer	View

Table 16 Sample of PLAPACK Object Members

For solely memory resident objects, global space parameters specify all memory mapping information. For virtual objects, global space behaves as a layer of cache memory between the disk and the application.

All parameters depend on the type of linear algebra object. Object creation routines set header and virtual space parameters according to the object type. Likewise, library routines call object specific methods to set global space and view space parameters.

Manipulations of view space parameters are restricted by the boundaries of virtual and global spaces. The view of a memory resident object cannot exceed the boundaries of global space. The view of a virtual object cannot exceed the bounds of virtual space. In addition, if global space has been attached to a virtual object, and attempt to create a view outside the global space boundaries will cause the current global space to be released and a new global space equivalent to the size of the view will be attached.

Spaces may not be attached to the PLAPACK object in an arbitrary order. Table 17 presents the required ordering when attaching spaces to an object.

Object Status	Permitted Actions
Empty Object	Define Header Free Object
Header Defined	Attach Virtual Space Attach Global Space Free Object
Header Defined Virtual Space Attached	Attach Global Space Attach View Space Remove Virtual Space
Header Defined Global Space Attached	Attach View Space Remove Global Space
Header Defined Virtual Space Attached Global Space Attached	Attach View Space Remove Global Space
Header Defined Virtual Space Attached View Space Attached	Attach Global Space Remove View Space
Header Defined Global Space Attached View Space Attached	Remove View Space
Header Defined Virtual Space Attached Global Space Attached View Space Attached	Remove View Space

Table 17 Ordering Restrictions on Layering of Object Spaces

E.2 Utility Functions for Space Management

The following low level infrastructure routines facilitate the attaching and detaching of object spaces. Routines specified in all capitals are macros.

pla_object_create	Creates an empty PLAPACK object structure.
pla_object_free	Frees an empty PLAPACK object structure.
pla_initialize_virtual	Initializes virtual space parameters to default values.
pla_initialize_global	Initializes global space parameters to default values.
pla_initialize_view	Initializes view space parameters to default values.

pla_set_<object>_virtual	Sets virtual space parameters according to specified dimensions and alignments. A static PLA_File structure is allocated for each virtual space.
pla_set_<object>_global	Sets global space parameters according to virtual space parameters and specified global space dimensions and alignments.
pla_set_<object>_view	Sets view space parameters according to global space parameters and specified view space dimensions and alignments.
pla_remove_virtual	Removes virtual space references from the object and re-initializes header parameters. An empty object is returned.
pla_remove_global	Removes global space references from the object. If object is memory resident, header parameters are re-initialized and an empty object is returned.
pla_remove_view	Removes view space references from the object.
pla_duplicate_object	Duplicates header information from one object to another.
pla_duplicate_virtual	Duplicates virtual space information from one object to another. Global space and view space is left undefined.
pla_duplicate_global	Duplicates virtual and global space information from one object to another. View space is left undefined.
pla_duplicate_view	Duplicates virtual, global and view space information from one object to another.
PLA_IS_VIRTUAL_SPACE	Returns TRUE if virtual space is attached.
PLA_IS_GLOBAL_SPACE	Returns TRUE if global space is attached.
PLA_IS_VIEW_SPACE	Returns TRUE if view space is attached.
PLA_IS_SAME_VIRTUAL_SPACE	Determines if two objects share identical virtual space. Always false for memory resident objects.
PLA_IS_SAME_GLOBAL_SPACE	Determines if two objects share identical global space.
PLA_IS_SAME_VIEW_SPACE	Determines if two objects share identical view space.
pla_is_view_inside_global	Determines if specified view occurs within current global space.

pla_is_view_inside_virtual	Determines if specified view occurs within current virtual space. Always false for memory resident objects.
pla_set_view	Low level view management routine which calls appropriate object space management routines. Inputs are dimensions and alignments respective to the template.

E.3 File I/O Implementation

PLAPACK implements block partitioned algorithms to achieve high performance. Virtual object operations require movement of the linear algebra blocks between disk and parallel memory. The structure of blocks depends on the data access pattern required by the linear algebra operation. Two primary types of blocking should be considered. The first blocking attempts to maximize the memory resident performance by transferring large, square blocks. The second blocking attempts to most effectively overlap computation and communication by transferring rectangular panels.

To illustrate the different access patterns, allow nb_{matrix} to be the dimension of the largest square block and nb_{panel} to be the lesser dimension of the rectangular panel. The different blocking sizes may be defined as nb_{matrix} x nb_{panel}, nb_{panel} x nb_{matrix} or nb_{matrix} x nb_{matrix}. A virtual GEMM routine using a rank-k update requires the first blocking pattern for matrix A, the second blocking pattern for matrix B and the third blocking pattern for matrix C.

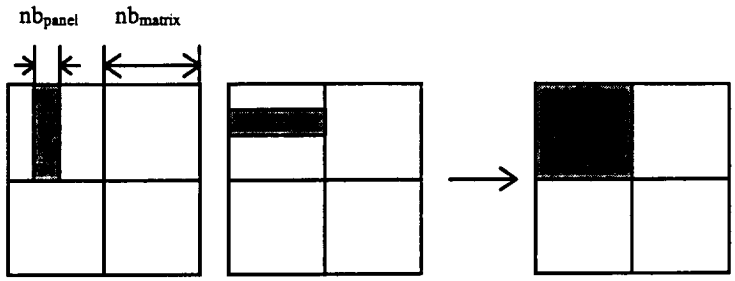


Figure 23 Out-of-Core Blocking Patterns for GEMM

The most efficient file I/O operations occur when the transferred data is organized into contiguous memory segments. The blocking requirements of the algorithm dictate which mapping should be used. While satisfactory for the GEMM operation, many algorithms access data from objects differently at different stages of the computation. For example, the Cholesky factorization requires all three blocking patterns.

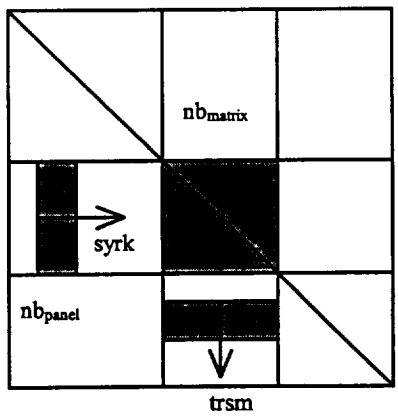


Figure 24 Out-of-Core Blocking Patterns for Cholesky

To simplify the implementation, only a single blocking size, $nb_{virtual}$ is implemented. All linear algebra operations are applied to square blocks which optimize the global BLAS operations. However, a greater amount of memory will be necessary for double buffering in asynchronous I/O operations.

E.1.1 Miscellaneous Implementation Issues

The first issue addresses the relationship between the virtual object view and the I/O block boundaries. Performing I/O on views which do not occur at block boundaries becomes difficult. I/O operations must include the entire block. Transferring the data associated with views which do not occur on block boundaries may require multiple I/O requests. Multiple requests creates problems for asynchronous operation since multiple buffers and multiple request objects are required.

For synchronous I/O operations, the object view expands to the nearest I/O block boundaries which encompass the original view. Recursive I/O requests on the larger view and memory-to-memory copies achieve the desired result. Asynchronous I/O operations for views which do not occur on block boundaries are not support due to the requirements of multiple concurrent I/O operations and multiple request objects.

The second issue addresses the structure of the disk files. Each processor assigned ownership of the linear algebra object possesses ownership of a file which contains the object data. The file consists of constant length records with the records corresponding to the column-major ordering of the linear algebra blocks. The record length is determined by calculating the memory usage on processor (0,0) of an $nb_{virtual}$ x $nb_{virtual}$ matrix with row and column alignments of zero. Each processor responsible for I/O transfers **local_length** x **local_width** elements to/from the local file block. If the block is not full, the data is packed into the first **local_length** x **local_width** elements of the record.

E.4 Application Routines

The PLAPACK I/O interface routines are intended to be general enough to support a wide range of functionality. However, I/O operations, especially those associated with asynchronous operations, vary widely between architectures. The current specification defines different I/O types which are either standardized or may be associated with a particular architecture. Each I/O type may or may not permit asynchronous operations. A list of supported I/O types are presented in Table 18.

I/O Type	Architecture	Async Allowed
PLA_ANSI_C	All	Never
PLA_UNIX_IO	All UNIX	Cray, IBM

Table 18 PLAPACK I/O Types

PLAPACK I/O routines consists of two groups analogous to global and local BLAS operations. Global I/O operations are collective and manage view boundary conditions and asynchronous requests. Local I/O operations are local to the individual processor and manage the architecture dependencies of the operation. Calling the local I/O operation directly is not recommended. All I/O routines operate only on virtual objects and may require different parameter lists for each I/O type.

```
int PLA_Open          ( PLA_Obj      object,
                        int          iotype,
                        ...          /* vargs */ );
int PLA_Open          ( PLA_Obj      object,
                                     PLA_ANSI_C,
                        char *       filename,
                        char *       mode );
int PLA_Open          ( PLA_Obj      object,
                                     PLA_UNIX_IO,
                        char *       filename,
                        int          mode );
```

int PLA_Local_open	(PLA_Obj	object,
	int	iotype,
	...	/* vargs */);
int PLA_Local_open	(PLA_Obj	object,
		PLA_ANSI_C,
	char *	filename,
	char *	mode);
int PLA_Local_open	(PLA_Obj	object,
		PLA_UNIX_IO,
	char *	filename,
	int	mode);
int PLA_Close	(PLA_Obj	object);
int PLA_Local_close	(PLA_Obj	object);

The **PLA_Open** method initializes a virtual object for file I/O operations according to the information contained in the PLAPACK object and the I/O type specified in the parameter list. Each processor responsible for I/O opens a file **filename_<row index>,<column index>** according to the permissions specified in the parameter list. The **PLA_Close** method finalizes the I/O operations and closes the file.

int PLA_Read	(PLA_Obj	object);
int PLA_Local_read	(PLA_Obj	object);
int PLA_Write	(PLA_Obj	object);
int PLA_Local_write	(PLA_Obj	object);

The **PLA_Read** and **PLA_Write** methods perform the synchronous I/O operations according to the current view of the object and the I/O type specified in the **PLA_Open** method.

```
int PLA_Iread                ( PLA_Obj       object,
                               PLA_Request * request );
int PLA_Local_iread          ( PLA_Obj       object );
int PLA_Iwrite               ( PLA_Obj       object,
                               PLA_Request * request );
int PLA_Local_iwrite         ( PLA_Obj       object );
```

The **PLA_Iread** and **PLA_Iwrite** methods perform the synchronous I/O operations according to the current view of the object and the I/O type specified in the **PLA_Open** method. The **PLA_Request** and **PLA_Status** objects are used to identify and return information concerning posted asynchronous operations. Asynchronous I/O operations are only supported for views which require transfer of a single disk record.

```
int PLA_Wait ( PLA_Request * request, PLA_Status * status );
int PLA_Test ( PLA_Request * request, PLA_Status * status );

int PLA_Local_wait ( PLA_Request * request, PLA_Status * status );
int PLA_Local_test ( PLA_Request * request, PLA_Status * status );
```

The **PLA_Wait** and **PLA_Test** methods are, respectively, blocking and non-blocking test for completion operations.

Appendix F Cholesky Factorization

F.1 Level 2 BLAS, Right Looking

The Cholesky factorization may be developed in terms of a block partitioned algorithm. The derivation begins by observing that the Cholesky factorization of A is a lower triangular matrix L such that $A = LL^T$. The matrices are partitioned into quadrants such that the top left corner consists of a single scalar value. The asterisk represents the transposed portions of the matrix.

$$\begin{bmatrix} a_{00} & * \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} l_{00} & 0 \\ L_{10} & L_{11} \end{bmatrix} \begin{bmatrix} l_{00} & L_{10}^T \\ 0 & L_{11}^T \end{bmatrix} \tag{73}$$

The relationship between the factored matrix and the original matrix may be recovered by explicitly performing the matrix multiplication.

$$\begin{aligned} a_{00} &= l_{00}^2 \\ A_{10} &= L_{10} l_{00} \\ A_{11} &= L_{10} L_{10}^T + L_{11} L_{11}^T \end{aligned} \tag{74}$$

The operations in Equation (74) may be rearranged to form the factorization of A which overwrites the previous values in memory. The algorithm computes the square root of the top left element. The elements of the column vector are scaled by the reciprocal of the result. The remainder of

the matrix is updated via a rank-1 update. The factorization of the remaining submatrix proceeds recursively.

$$\begin{aligned} a_{00} &\leftarrow \sqrt{a_{00}} \\ A_{10} &\leftarrow \frac{1}{l_{00}} A_{10} \\ A_{11} &\leftarrow A_{11} - A_{10} A_{10}^T \end{aligned} \tag{75}$$

The algorithm expressed in Equation (75) is the Level 2 right-looking Cholesky factorization. The algorithm is classified as Level 2 BLAS since the majority of operations occurs during the rank-1 matrix update. The factorization is termed right-looking because all operations occur on or to the right of the current matrix column. Figure 25 presents the FORTRAN code fragment to perform the decomposition of lower triangular matrix A of dimension N with leading dimension LDA.

```
DO I=1,N
    A(I,I) = SQRT ( A(I,I) )
    DSCAL ( N-I, 1.0d+0 / A(I,I), A(I+1,I), LDA )
    DSYR ( "LOWER", N-I, -1.0d+0, A(I+1,I), 1, A(I+1,I+1), LDA )
ENDDO
```

Figure 25 Level 2 Right-Looking Cholesky Factorization

F.2 Level 2 BLAS, Right Looking

The extension of the Cholesky algorithm to Level 3 BLAS operations begins by allowing the top left partition of A to consist of an nb x nb

submatrix. The remainder of the derivation is analogous to the Level 2 BLAS factorization. The matrices are again partitioned into quadrants. However, the top left corner is now a matrix block. The asterisk represents the transposed portions of the matrix.

$$\begin{bmatrix} A_{00} & * \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 \\ L_{10} & L_{11} \end{bmatrix} \begin{bmatrix} L_{00}^T & L_{10}^T \\ 0 & L_{11}^T \end{bmatrix} \tag{76}$$

The relationship between the factored matrix and the original matrix may be recovered by explicitly performing the matrix multiplication.

$$\begin{aligned} A_{00} &= L_{00} L_{00}^T \\ A_{10} &= L_{10} L_{00}^T \\ A_{11} &= L_{10} L_{10}^T + L_{11} L_{11}^T \end{aligned} \tag{77}$$

The operations in Equation (77) are again rearranged to form the factorization of A which overwrites the previous values in memory. The algorithm computes the Level 2 factorization of the top left submatrix A_{00}. The column panel A_{10} is updated via a Level 3 triangular solve with multiple right-hand-sides. The remainder of the matrix is updated via a Level 3 symmetric rank-k operation. The factorization of the remaining submatrix proceeds recursively.

$$
\begin{aligned}
A_{00} &\leftarrow BLAS2\ Cholesky\left(A_{00}\right) \\
A_{10} &\leftarrow A_{10}A_{00}^{-T} \\
A_{11} &\leftarrow A_{11} - A_{10}A_{10}^{T}
\end{aligned}
\tag{78}
$$

The Level 3 right-looking Cholesky algorithm may also be coded in a straight forward manner. Figure 26 presents the FORTRAN code fragment to perform the decomposition of lower triangular matrix A of dimension N with leading dimension LDA.

```
DO I=1,N,NB
    NBT=MIN(NB,N-I+1)
    BLAS2_CHOL ( NBT, A(I,I), LDA )
    DTRSM( "RIGHT", "LOWER", "TRANS", "NONUNIT",
           N-I+1-NBT, NBT,
           1.0d+0, A(I,I), LDA, A(I+NBT,I), LDA )
    DSYRK ( "LOWER", "NOTRANS", N-I+1-NBT, NBT,
           -1.0d+0, A(I+NBT,I), LDA,
           1.0d+0, A(I+NBT,I+NBT), LDA )
ENDDO
```

Figure 26 Level 3 Right-Looking Cholesky Factorization

F.3 Level 3 BLAS, Left Looking

A left-looking variant to the Cholesky factorization may also be developed. The derivation of the Level 3 left-looking algorithm begins with the partitioning presented in Equation (79) such that block (1,1) is *nb* x *nb*. The asterisks represent the transposed portions of the matrix.

$$\begin{bmatrix} A_{00} & * & * \\ A_{10} & A_{11} & * \\ A_{20} & A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 & 0 \\ L_{10} & L_{11} & 0 \\ L_{20} & L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{00}^T & L_{10}^T & L_{20}^T \\ 0 & L_{11}^T & L_{21}^T \\ 0 & 0 & L_{22}^T \end{bmatrix} \tag{79}$$

Assuming that the factorization of the first column partition has been accomplished, the relationship between block (1,1) and (2,1) of the factored matrix and the original matrix may be recovered by explicitly performing the matrix multiplication.

$$\begin{aligned} A_{11} &= L_{10}L_{10}^T + L_{11}L_{11}^T \\ A_{21} &= L_{20}L_{10}^T + L_{21}L_{11}^T \end{aligned} \tag{80}$$

The operations in Equation (80) are rearranged to form the factorization of the second column panel of A which overwrites the previous values. The algorithm begins by updating the current column based on previous factorizations. Submatrix $A_{1,1}$ is updated via a Level 3 BLAS symmetric rank-k update. Submatrix $A_{2,1}$ is updated via a Level 3 BLAS matrix-matrix multiplication. The factorization of the column panel proceeds similarly to the left-looking algorithm.

$$\begin{aligned} A_{11} &\leftarrow A_{11} - A_{10}A_{10}^T \\ A_{21} &\leftarrow A_{21} - A_{20}A_{10}^T \\ A_{00} &\leftarrow \textit{BLAS2 Cholesky}(A_{00}) \\ A_{10} &\leftarrow A_{10}A_{00}^{-T} \end{aligned} \tag{81}$$

The BLAS Level 3, left-looking Cholesky algorithm may also be coded in a straight forward manner. Figure 27 presents the FORTRAN code fragment to perform the decomposition of lower triangular matrix *A* of dimension *N* with leading dimension *LDA*.

```
DO I=1,N,NB
    NBT=MIN(NB,N-I+1)
    DSYRK ( "LOWER", "NOTRANS", NBT, I-1,
                    -1.0d+0, A(I,1), LDA, 1.0d+0, A(I,I), LDA )
    DGEMM ( "NOTRANS", "TRANS", N-I+1-NBT, NBT, I-1,
                    -1.0d+0, A(I+NBT,1), LDA, A(I,1), LDA,
                    1.0d+0, A(I+NBT,I), LDA )
    BLAS2_CHOL ( NBT, A(I,I), LDA )
    DTRSM( "RIGHT", "LOWER", "TRANS", "NONUNIT",
                     N-I+1-NBT, NBT,
                    1.0d+0, A(I,I), LDA, A(I+NBT,I), LDA )
ENDDO
```

Figure 27 Level 3 Left-Looking Cholesky Factorization

Appendix G Triangular Solve Multiple RHS

The parallel implementation of TRSM requires the statement of the algorithm in terms of matrix blocks. Two variants of the Level 3 BLAS triangular solve multiple right-hand-sides apply to the matrix inversion as implemented in this research. The BLAS parameters LEFT, LOWER, NOTRANS and NONUNIT specify the first case. The BLAS parameters LEFT, LOWER, TRANS and NONUNIT specify the second case. For brevity, the first case will be referred to as the LLNN case, and the second as the LLTN case.

G.1 LLNN Case

The derivation of the LLNN case of TRSM begins by observing the operation produces the matrix X such that $AX=B$ where A is a lower triangular matrix with a non-unit diagonal. Matrix A is partitioned into quadrants such that $A_{0,0}$ is an nb x nb submatrix.

$$\begin{bmatrix} A_{0,0} & 0 \\ A_{1,0} & A_{1,1} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} \tag{82}$$

The relationship between the unknown matrix X and the known matrices A and B may be recovered by explicitly performing the matrix multiplication.

$$\begin{aligned} A_{0,0}X_0 &= B_0 \\ A_{1,0}X_0 + A_{1,1}X_1 &= B_1 \end{aligned} \tag{83}$$

The operation may be rewritten such that the result overwrites matrix B.

$$\begin{aligned} X_0 &\leftarrow A_{0,0}^{-1}X_0 \\ X_1 &\leftarrow X_1 - A_{1,0}X_0 \end{aligned} \tag{84}$$

G.2 LLTN Case

The derivation of the LLTN case of TRSM begins by observing the operation produces the matrix X such that $A^T X = B$ where A is a lower triangular matrix with a non-unit diagonal. Matrix A is partitioned into quadrants such that $A_{0,0}^T$ is an *nb* x *nb* submatrix.

$$\begin{bmatrix} A_{0,0}^T & A_{1,0}^T \\ 0 & A_{1,1}^T \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} \tag{85}$$

The relationship between the unknown matrix X and the known matrices A and B may be recovered by explicitly performing the matrix multiplication.

$$\begin{aligned} A_{0,0}^T X_0 + A_{1,0}^T X_1 &= B_0 \\ A_{1,1}^T X_1 &= B_1 \end{aligned} \tag{86}$$

The operation may be rewritten such that the result overwrites matrix B.

$$\begin{aligned} X_1 &\leftarrow A_{1,1}^{-T} X_1 \\ X_0 &\leftarrow X_0 - A_{1,0}^{T} X_1 \end{aligned} \tag{87}$$

Appendix H Memory Efficient Subarc Update Algorithm

In the satellite tracking observable, the initial conditions of the satellite and the errors in the geopotential perturb the range measurements. The effect of estimating satellite initial conditions as well as other subarc parameters contribute directly to the geopotential covariance. The following derivation demonstrates that the subarc contributions may be applied to the geopotential covariance through a series of updates which may or may not include the explicit estimation of the subarc parameters. In addition, the method conserves memory by permitting the discard of previous subarc information.

Equation (88) presents the relationship between the information matrix and the covariance matrix for the batch estimate. The upper diagonal block of each matrix corresponds to the subarc parameters, and the lower diagonal block corresponds to the geopotential coefficients. The off-diagonal blocks correspond to the correlation information between the subarc parameters and the geopotential coefficients. The estimation of multiple subarcs will create a block diagonal structure for submatrix $N_{0,0}$.

$$\begin{bmatrix} N_{0,0} & N_{1,0}^T \\ N_{1,0} & N_{1,1} \end{bmatrix} \begin{bmatrix} P_{0,0} & P_{1,0}^T \\ P_{1,0} & P_{1,1} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{88}$$

The matrix multiplication leads to the following system of matrix equations.

$$\begin{aligned} N_{0,0}P_{0,0} + N_{1,0}^{T}P_{1,0} &= I \\ N_{1,0}P_{0,0} + N_{1,1}P_{1,0} &= 0 \\ N_{0,0}P_{1,0}^{T} + N_{1,0}^{T}P_{1,1} &= 0 \\ N_{1,0}P_{1,0}^{T} + N_{1,1}P_{1,1} &= I \end{aligned} \tag{89}$$

The rearrangement of the third and fourth expressions in Equation (89) lead to the expressions in Equation (90).

$$\begin{aligned} P_{1,1} &= N_{1,1}^{-1}\left(I - N_{1,0}P_{1,0}^{T}\right) \\ P_{1,0}^{T} &= -N_{0,0}^{-1}N_{1,0}^{T}P_{1,1} \end{aligned} \tag{90}$$

Combining the equations and solving for $P_{1,1}$.

$$\begin{aligned} P_{1,1} &= N_{1,1}^{-1}\left(I - N_{1,0}N_{0,0}^{-1}N_{1,0}^{T}P_{1,1}\right) \\ P_{1,1} - N_{1,1}^{-1}N_{1,0}N_{0,0}^{-1}N_{1,0}^{T}P_{1,1} &= N_{1,1}^{-1} \\ P_{1,1} &= \left(I - N_{1,1}^{-1}N_{1,0}N_{0,0}^{-1}N_{1,0}^{T}\right)^{-1}N_{1,1}^{-1} \end{aligned} \tag{91}$$

The inversion of the final expression in Equation (91) yields an expression for the inverse geopotential covariance matrix in terms of the geopotential information matrix, the subarc information matrix and the correlation information.

$$P_{1,1}^{-1} = N_{1,1} - N_{1,0}N_{0,0}^{-1}N_{1,0}^{T} \tag{92}$$

The expression in Equation (92) represents an update to the geopotential information matrix. Due to the block diagonal nature of $N_{0,0}$, Equation (92) may be expressed as a series of updates with each update corresponding to a k^{th} subarc.

$$P_{1,1}^{-1} = N_{1,1} - \left[N_{1,0}\big|_0 \quad \cdots \quad N_{1,0}\big|_k \right] \begin{bmatrix} N_{0,0}^{-1}\big|_0 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & N_{0,0}^{-1}\big|_k \end{bmatrix} \begin{bmatrix} N_{1,0}^T\big|_0 \\ \vdots \\ N_{1,0}^T\big|_k \end{bmatrix} \tag{93}$$

$$P_{1,1}^{-1} = N_{1,1} - \sum_k \left(N_{1,0} N_{0,0}^{-1} N_{1,0}^T \right)\Big|_k \tag{94}$$

The algorithm implementing Equation (94) proceeds in a straight forward manner. For each subarc, the accumulation of information matrices $N_{0,0}$, $N_{1,0}$ and $N_{1,1}$ occurs in the conventional manner. At the completion of the subarc, the information matrix $N_{1,1}$ is updated using Equation (94). The memory used for storing the subarc information may be reused.

Bibliography

Astfalk, Greg. *Fundamentals and practicalities of MPP.* **The Leading Edge.** August, September, and October 1993.

Bate, R. R., D. D. Mueller and J. E. White. **Fundamentals of Astrodynamics.** New York: Dover Publications, 1971.

Beck, Kent, and Ward Cunningham. *A Laboratory for Teaching Object-Oriented Thinking.* Proceedings of OOPSLA '89. October 1989.

Bertiger, W. I., et al. *GPS precise tracking of TOPEX/POSEIDON: Results and implications.* **Journal of Geophysical Research,** Vol. 99, No. C12. December 1994.

Bettadpur, Srinivas V. **A Simulation Study of High Degree and Order Geopotential Determination Using Satellite Gravity Gradiometry.** Doctoral Dissertation. The University of Texas at Austin, May 1993.

Bettadpur, Srinivas V., Bob E. Schutz, and John B. Lundberg. *Spherical harmonic synthesis and least squares estimation in satellite gravity gradiometry.* **Bulletin Geodesique,** Vol 66. 1992.

Bock, Yehuda. *Reference Systems.* **GPS for Geodesy,** Alfred Kleusberg and Peter J. G. Teunissen, ed. Berlin: Springer-Verlag, 1996.

Chtchelkanova, Almadena, John Gunnels, Greg Marrow, James Overfelt, and Robert van de Geijn. *Parallel Implementation of BLAS: General Techniques for Level 3 BLAS.* **Concurrency: Practice and Experience,** 1996.

Choi, J., et al. *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms.* UT CS-95-292, LAPACK Working Note 100. May 1995

Choi, J., et al. *ScaLAPACK: A Protable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance.* UT CS-95-283, LAPACK Working Note 95. March 1995.

Choi, Jaeyoung, et al. *The Design and Implementation of the SCALAPACK LU, QR and Cholesky Factorization Rotuines.* ORNL/TM-12470, Oak Ridge National Laboratory. September 1994.

Choi, Jaeyoung, Jack J. Dongarra and David W. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. ORNL/TM-12252, Oak Ridge National Laboratory. May 1993.

Coffey, Shannon L., Edna Jenkins, Harold L. Neal and Herbert Reynods. *Parallel Processing of Uncorrelated Observations into Satellite Orbits.* Proceedings of AAS/AIAA Astrodynamics Specialist Conference. February 1996.

Columbo, O. L. *Notes on the Mapping of the Gravity Field Using Satellite Data.* **Mathematical and Numerical Techniques in Physical Geodesy**, Hans Sunkel, ed. Springer-Verlag: Berlin, 1986.

Colombo, Oscar L. *Precise GPS Orbits for Geodesy.* **Advances in Space Research**, Vol. 14, No. 5. 1994.

Columbo, Oscar L. and Michael M. Watkins. *Satellite Positioning.* **Review of Geophysics**, Supplement, U.S. National Report to I.U.G.G. April 1991.

Colombo, Oscar L. *Advanced Techniques for High-Resolution Mapping of the Gravitational Field.* **Theory of Satellite Geodesy and Gravity Field Determination**. F. Sanso, R. Rummel, eds. Berlin: Springer-Verlag, 1989.

Colombo, Oscar L. *Numerical Methods for Harmonic Analysis on the Sphere.* Reports of the Department of Geodetic Science: Report No. 310. Ohio State University, March 1981.

D'Azevedo, E. F., and Jack Dongarra. *The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR and Cholesky Factorization Routines.* UT CS-97-347, LAPACK Working Note 118. University of Tennessee, January 1997.

Danby, J. M. A. **Fundamentals of Celestial Mechanics**. Richmond: William-Bell, 1988.

Demmel, James W., Michael T. Heath, and Henk A van der Vorst. *Parallel numerical linear algebra*. UT CS-93-192, LAPACK Working Note 60. University of Tennessee, 1993.

Dongarra, Jack, Jeremy Du Croz, Iain Duff, and Sven Hammarling. *A Set of Level 3 Basic Linear Algebra Subprograms*. **TOMS**, Vol. 16, No. 1. March 1990.

Dongarra, Jack and David Walker. *The Design of Linear Algebra Libraries for High Performance Computers*. UT CS-93-188, LAPACK Working Note 58. University of Tennessee, June 1993.

Dongarra, Jack J., Robert van de Geijn, and David W. Walker. *Scalability Issues Affecting the Design of a Dense Linear Algebra Library*. **Journal of Parallel and Distributed Computing**, Vol. 22, No. 3. September 1994..

Dongarra, Jack, J., Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. *A Extended Set of Fortran Basic Linear Algebra Subprograms*. Argonne National Laboratory Mathematics and Computer Science Division, Technical Memorandum No. 41. September 1986.

Edwards, Harold Carter. **MMPI: Asynchronous Message Management for the Message-Passing Interface**. TICAM Report 96-44. Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin. October, 1996.

Edwards, Carter, Po Geng, Abani Patra, and Robert van de Geijn. *Parallel Matrix Distributions: Have we been doing it all wrong?* Department of Computer Science TR-95-40, The University of Texas at Austin. October 1995.

Gelb, Arthur. **Applied Optimal Estimation**. Cambridge: MIT Press, 1974.

Greenburg, Michael D. **Foundation of Applied Mathematics**. Englewood Cliffs: Prentice-Hall, 1978.

Gropp, William and Ewing Lusk. *User's Guide for mpich, a Portable Implementation of MPI.* ANL/MCS-TM-ANL-96/6. Argonne National Laboratory, 1996.

Golub, G. H. and G. F. van Loan. **Matrix Computations**. John Hopkins Press, 1989.

Gustafson, John L. *Compute-Intensive Applications on Advanced Computer Architectures*. **Parallel Computing '91**, D. J. Evans, G. R. Joubert and H. Liddell, ed. North-Holland: Amsterdam, 1992.

Gustafson, John L., Gary R. Montry, and Robert E. Benner. *Development of Parallel Methods for a 1024-Processor Hypercube*. **SIAM Journal of Scientific and Statistical Computing**. Vol. 9, No. 4. July 1988.

Haagmans, R. H. N., and M. van Gelderen. *Error Variances-Covariances of GEM-T1: Their Characteristics and Implications in Geoid Computations*. **Journal of Geophysical Research**, Vol. 96, No. B12. November 1991.

Ja Ja, Joseph. **An Introduction to Parallel Algorithms**. Reading: Addison-Wesley, 1992.

Kernighan, Brian W. and Dennis M. Ritchie. **The C Programming Language**. Englewood Cliffs: Prentice-Hall, 1978.

Klimkowski, Kenneth and Robert van de Geijn. *Anatomy of a Parallel Out-of-Core Dense Linear Solver.* Procedings of the International Conference on Parallel Processing, Volume III, 1995.

Konopliv, Alex. *High Resolution Gravity Modeling Using Parallel Supercomputers.* JPL Interoffice Memorandum 312.D-95-103. October 1995.

Koop, Rayboud. **Global Gravity Field Modeling Using Satellite Gravity Gradiometry**. Netherlands: Delft, 1993.

Lundberg, John B. **Computational Errors and Their Control in the Determination of Satellite Orbits**. Doctoral Dissertation. The University of Texas at Austin, 1985.

Lundberg, John B. **Multistep Integration Formulas for the Numerical Integration of the Satellite Problem.** Masters Thesis. The University of Texas at Austin, 1981.

Maybeck, Peter S. **Stochastic Models, Estimation and Control, Vol. 1**. New York: Academic Press, 1979.

McNutt, Marcia. *If Only We Had Better Gravity Data...* **Geodesy in the Year 2000.** National Research Council, Committee on Geodesy. Washington D.C.: National Academy Press, 1990.

Melbourne, William G., E.S. Davis, Thomas P. Yunck, and Byron D. Tapley. *The GPS flight experiment on TOPEX/Poseidon.* **Geophysical Research Letters**, Vol. 21, No. 19. September 1994.

Moyer, Theodore D. *Mathematical Formulation of the Double-Precision Orbit Determination Program (DPODP).* JPL Technical Report 32-1527. May 1971.

National Aeronautic and Space Administration. *New Missions Selected to Study Earth's Forest and Gravity Field Variability.* NASA Press Release 97-46. March 1997.

Nerem, R. S., et al. *Gravity Model Development for TOPEX/POSEIDON: Joint Gravity Models 1 and 2.* **Journal of Geophysical Research**, Vol. 99, No C12. December 1994.

Nerem, R. S., C. Jekeli and W. M. Kaula. *Gravity field determination and characteristics: Retrospective and prospective.* **Journal of Geophysical Research**, Vol. 100, No B8. August 1995.

Nyhoff, Larry and Sanford Leestma. **FORTRAN 77 for Engineers and Scientists**. New York: Macmillan, 1985.

Paik, Ho Jung et al. *Mission Concepts for a Superconducting Gravity Gradiometer* Earth *Survey to Establish a Baseline for Global Change.* Proceedings of the 1996 Spring Meeting of the American Geophysical Union. May 1996.

Pavlis, Erricos C. *Gravity Field Estimation from Future Space Missions: TOPEX/Poseidon, Gravity Probe B, and ARISTOTELES.* **From Mars to Greenland: Charting Gravity with Space and Airborne Instruments**. O. L. Colombo, ed. Berlin: Springer-Verlag, 1992.

Press, William H. et al. **Numerical Recipes: The Art of Scientific Computing (FORTRAN Version).** Cambridge: Cambridge Uni versity Press, 1989.

Rapp, R. H. *Global Geopotential Solutions.* **Mathematical and Numerical Techniques in Physical Geodesy,** Hans Sunkel, ed. Springer-Verlag: Berlin, 1986.

Rapp, Richard H., Nikolaos K. Pavlis and Yan Ming Wang. *High Resolution Gravity Models Combining Terrestrial and Satellite Data.* **From Mars to Greenland: Charting Gravity with Space and Airborne Instruments**. O. L. Colombo, ed. Berlin: Springer-Verlag, 1992.

Rapp, Richard H. *Combination of Satellite, Altimetric and Terrestrial Gravity Data.* **Theory of Satellite Geodesy and Gravity Field Determination**, F. Sanso, R. Rummel, eds. Berlin: Springer-Verlag, 1989.

Reigber, Christoph. *Gravity Field Recovery From Satellite Tracking Data.* **Theory of Satellite Geodesy and Gravity Field Determination**, F. Sanso, R. Rummel, eds. Berlin: Springer-Verlag, 1989.

Roy, A. E. **Orbital Motion**. Bristol: Adam Hilger, 1988.

Rummel, R. **Spherical Harmonic Analysis of Satellite Gradiometry**. Netherlands: Delft, 1993.

Rummel, R. *Satellite Gradiometry.* **Mathematical and Numerical Techniques in Physical Geodesy**, Hans Sunkel, ed. Springer-Verlag: Berlin, 1986.

Schildt, Herbert. **C: The Pocket Reference, Second Edition**. Berkeley: Osborne McGraw-Hill, 1991.

Schrama, Ernst J O. *Gravity Field Error Analysis: Applications of Global Positioning System Receivers and Gradiometers on Low Orbiting Platforms*. **Journal of Geophysical Research**, Vol. 96, No. B12. November 1991.

Schuh, W.-D., H. Sunkel, W. Hausleitner and E. Hock. *Refinement of Iterative Procedures for the Reduction of Spaceborne Gravimetry Data*. ESA CIGAR IV, Final Report. June 1996.

Schutz, Bob E. and Gregory A. Baker. *Application of Massively Parallel Processing Techniques to Least Squares Problems in Satellite Gravity Gradiometry*. FY95 Annual Report, NASA Earth and Space Science Project, 1996.

Schutz, B. E., et al. *Dynamic orbit detemination using GPS measurements from TOPEX/Poseidon*. **Geophysical Research Letters**, Vol. 21, No. 19. September 1994.

Scott, Steven L. *Synchronization and Communication in the T3E Multiprocessor*. Proceedings of ASPLOS-VII. October 1996.

Sneeuw, Nico. *Covariance Propagation of Block-diagonal Covaraince Matrices from Error Simulations*. June 1995.

Snir, Marc, et al. **MPI: The Complete Reference**. Cambridge: MIT, 1996.

Sidani, Majed and Bill Harrod. *Parallel Matrix Distributions: Have we been doing it all right?* LAPACK Working Note ??. University of Tennessee, 1996.

Szabo, Bela. *The Estimation of the Earth's Gravity Field*. Reports of the Department of Geodetic Science: Report No. 369. Ohio State University, June 1986.

Tapley, B. D., B. E. Schutz, R. J. Eanes and M. M. Watkins. *Lageos Laser Ranging Contributions to Geodynamics, Geodesy, and Orbit Dynamics*. Contributions of Space Geodesy to Geodynamics: Earth Dynamics and Geodynamics. c1993. American Geophysical Union.

Tapley, B. D., B. E. Schutz and R. J. Eanes. *Station Coordinates, Baselines, and Earth Rotation From LAGEOS Laser Ranging: 1976-1984.* **Journal of Geophysical Research**, Vol. 90, No. B11. September 1985.

Tapley, Byron D., et al. *Precision orbit determination for TOPEX/POSEIDON.* **Journal of Geophysical Research**, Vol. 99, No. C12. December 1994.

Tapley, Byron D. *ASE 381P Statistical Estimation Theory, Class Notes.* The University of Texas at Austin. Spring Semester, 1991.

van de Geijn, Robert A. *CS 391T Parallel Methods, Class Notes.* The University of Texas at Austin. Spring Semester, 1996.

van de Geijn, Robert A. **Using PLAPACK: Parallel Linear Algebra Package**. Cambridge: MIT Press, 1997.

van de Geijn, Robert A. and Jerrell Watts. *SUMMA: Scalable Universal Matrix Multiplication Algorithm.* UT CS-95-286, LAPACK Working Note 96. April 1995.

van Gelderen, M. and R. Koop. *The use of degree variances in satellite gradiometry.* **Journal of Geodesy**, No. 71. 1997.

Vanicek, Petr and Edward Krakiwsky. **Geodesy: The Concepts**. Amsterdam: Elsevier, 1986.

Visser, P. N. A. M. et al. *Global gravity field recovery from the ARISTOTELES satellite mission.* **Journal of Geophysical Research**, Vol. 99, No. B2. February 1994.

Wallace, Scott T., P. J. Cefola and R. J. Prouix. *Parallel Orbit Propagation and the Analysis of Satellite Constellations.* Proceedings of AAS/AIAA Astrodynamics Specialist Conference. August 1995.

Wells, David E., et al. **Guide to GPS Positioning**. Fredericton, Canadian GPS Associates, 1986.

Wells, William C., ed. **Spaceborne Gravity Gradiometers**. NASA Conference Publication 2305, 1983.

Zlotnicki, Victor. *Common Interests in Geodetic and Oceanographic Research*. **Geodesy in the Year 2000**. National Research Council, Committee on Geodesy. Washington D.C.: National Academy Press, 1990.