FILE COPY

(1)

AD-A199 112

DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

UNCLASSIFIED

DTIC ELECTE SEP 2 0 1988 D Cs

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY
---

3 DISTRIBUTION / AVAILABILITY OF REPORT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
---

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
---

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
SYSTOLIC SYSTEMS, INC.

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
U.S. ARMY WHITE SANDS MISSILE RANGE

6c. ADDRESS (City, State, and ZIP Code)
2240 No. First Street
San Jose, CA 95131

7b. ADDRESS (City, State, and ZIP Code)
Commanding Officer, STEWS-ID-T
U.S. Army White Sands Missile Range
New Mexico, 88002-5143

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (If applicable)
---

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DAAD07-87-C-0077

8c. ADDRESS (City, State, and ZIP Code)
---

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
665502	1P6550 M40	----	----

11. TITLE (Include Security Classification)
COMPUTER ARCHITECTURE FOR KALMAN FILTERING

12. PERSONAL AUTHOR(S)
RICHARD H. TRAVASSOS

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical	FROM 10SEP87 TO 9MAR88	1988 MARCH 9	58

16. SUPPLEMENTARY NOTATION
---

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
----	----	----
----	----	----

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Parallel Kalman Filter, Systolic Architectures, Real-Time Target Tracking

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

Highly parallel architectures are developed to meet the increasing demanding throughput requirements of DoD signal processing applications. In particular, to perofrm multi-dimensional Kalman filtering at sample rates approaching 15 KHz, innovative techniques are presented to partition the problem for real-time implementation. A unique decoupling of the predictor and corrector equations in the Kalman filter is developed to speed up computations by two (2) orders of magnitude (100x). The proposed parallel algorithms and architectures are well suited for recursive filtering, image and signal processing for next generation instrument radar and telemetry equipment.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
FOO LAM	(505)678-3010	STEWS-ID-T

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

88 7 25 159

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

DTIC COPY INSPECTED

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

SECTION 1

INTRODUCTION

This project seeks to apply recently developed parallel Kalman filter (PKF) methods to actual flight test data obtainable from White Sands Missile Range (WSMR). In this Phase I feasibility study, we show that the PKF methods offer excellent speed up, reliable convergence characteristics and good accuracy compared with the standard Kalman filter (SKF). Parallel computing architectures are presented that enable the PKF methods to be implemented at 15 KHz sample rates at 64-bit floating-point precision.

This project is innovative in that the PKF architectures utilize both horizontal and vertical parallelism. A balance of nonlinear (e.g., trigonometric functions, exponentials, squares, square-roots) and linear (e.g., add, subtract, multiply, divide) computing resources rated at nearly 25 double-precision MFLOPs (million floating-point operations per second) each is recommended. Coupling this with an industry-standard, Vme bus chassis provides an open architecture to permit other WSMR contractors to add to the system.

1.1 BACKGROUND

To illustrate the need to develop Kalman filter parallel processing architectures, the total number of arithmetic operations that must be computed in the Kalman filter algorithm can be counted and multiplied by different multiplier and adder speeds. For the Kalman filter algorithm given in Table 1-1, the total number of multiplications, additions and divisions are given by: $(n^2 + 2n - 1)$ additions, $(2n^2 + 4n + 1)$ multiplications, and 1 division. Therefore, the overall execution time needed to update the Kalman filter algorithm in Table 1-1 is:

$$t = (n^2 + 2n - 1)\, t_a + (2n^2 + 4n + 1)\, t_m + t_d \qquad (1.1)$$

TABLE 1-1: STANDARD KALMAN FILTER ALGORITHM*

Kalman* Gain	$K_k = P_k(+)H_k^T(H_k^T P_k(+)H_k^T + 1)^{-1}$
Filter Update	$\hat{x}_k(+) = \hat{x}_k(-) + K_k(y_k - H_k\hat{x}_k(-))$ $= (I - K_k H_k)\,\hat{x}_k(-) + K_k y_k$
Covariance Update	$P_k(+) = (I - K_k H_k)\,P_k(-)$
Measurement Update	$y_{k+1} = H_{k+1}\,\hat{x}_k(+)$

* Note that when scalar measurements are processed, the inverse operation reduces to a division operation.

where t_a is the addition time, t_m is the multiplication time, and t_d is the division time. For example, if $t_a = 200$ nsec, $t_m = 200$ nsec and $t_d = 10$ usec, one pass through the Kalman filter with $n = 9$ states theoretically requires only 69 usec. This corresponds to 1/69 usec = 14.5 KHz update rate using state-of-the-art 32-bit floating-point VLSI chips. Since most systems deliver about 20% of theoretical peak performance, to keep up with real-time requirements the theoretical speed needs to be 1/20% = 5x faster than real time. Thus, the Kalman filter equations must be computed theoretically at 5 x 15 KHz = 75 KHz rate to deliver real-time performance at 15 KHz. Since it is well known that the Kalman filtering must be performed using floating-point arithmetic, the only viable method to increase the throughput of the Kalman filter by say 5 to 10 is with parallel processing. Optical processing is fast but optical fixed-point can cause stability problems with the Kalman filter. Nonlinear or extended Kalman filtering is even more computationally demanding. To perform nonlinear filtering in real time at 15 KHz, a 10 to 100 speed up is needed.

Note that although much progess has been made in floating-point adders and multipliers, the real problem is fast hardware divide (matrix inversion) is needed in the Kalman filter. Performing division in parallel on vector/matrix elements is, therefore, required to speed up Kalman filter computations.

1.2 RESULTS OF THE PHASE I WORK

Missile Math Model Based on Actual Flight Test Data

Unclassified flight test data was provided by White Sands Missile Range representative of modern radar tracking systems. The radar data provided measurements of six missile trajectories. Using the data, a math model could be developed taking into account the physics of the problem and parallel Kalman filter math model requirements. The math model form was defined in the Phase I study although the actual model parameters were not identified. Developing nonlinear models for the data and nonlinear (or extended) parallel Kalman filtering should be considered under Phase II. The state variables in the math model were range, range rate, altitude and elevation angle.

Parallel Kalman Filter Algorithm and Architecture Selection

For our Phase I feasibility study, a dual-processor (2) and quad-processor (4) PKF were used for the parallel architecture trade-off study on a nonlinear target tracking application. The parallel Kalman filter architecture trade-off parameters included integer versus floating-point arithmetic, memory sizing requirements, data bus speed and input/output subsystem requirements.

It was concluded that (1) at least 32-bit and preferalbe 64-bit floating-point arithmetic is needed for accurate tracking, (2) at least 36 (preferably 48) parallel processing elements capable of 25 MFLOP 32/64-bit linear processing speed and 24 MFLOPs at 64-bit nonlinear compute speed, (3) at least 2 Mbytes of fast RAM is needed to store the trajectory data with access times of less than 45 nsec (15 Mbytes of bulk memory with access speeds of 120 nsec was also recommended based on the PKF program code size) and (4) data bus speed of at least 20 Mbytes per second to move data in/out of the processors.

The basis for our conclusions is discussed later in this report.

Two-Processor Parallel Kalman Filter

The standard Kalman filter (SKF) and two-processor parallel Kalman filter have been coded and executed for a noisy, time-varying scalar example. Although this is a simple example, it illustrates that the PKF provides excellent estimates.

Nonlinear Extended Two/Four-Processor Parallel Kalman Filter

The nonlinear EKF and two- and four-processor EPKF have been coded for a realistic example involving gravity, drag and measurement noise. The two-processor and four-processor EPKF estimates agree quite well with the original EKF estimates. The EPKF utilized a parallel trapezoidal rule for integrating the state and covariance updates in the EPKF.

Two/Four-Processor Parallel Trapezoidal Rule

The two- and four-processor parallel trapezoidal rule, ode solver generally provided the same solution in a given number of iterations compared with the sequential trapezoidal-rule ode solver. Because each iteration of the parallel trapezoidal rule can run on two or four processors simultaneously, the time to solution was reduced almost linearly with the number of processors.

The floating-point primitives and macros for the PKF have been coded on the 481. Specifically, quad adds (i.e., four additions), multiplies, subtracts, divides, etc., have been written and put into PROMs on the 481 cards. The linking of these primitives into macros for say the Kalman gain update is feasible.

Accuracy, Speed and Implementation Considerations

The accuracy of the parallel Kalman filter degrades somewhat as more processors are utilized. It is recommended that no more than four to eight processors be used per target state but distribute the computations over each state variable. Specifically, a nine-state target model may use 36 processors (four per state variable) with excellent accuracy and speed up.

To meet the WSMR target data processing requirements, 25 double-precision MFLOPs speed are needed on linear (and nonlinear) computations. Thus, a balanced system architecture capable of 50 MFLOPs in total is recommended. A 20 slot Vme bus based tracking system based on a Motorola 68030 master, twelve Systolic-482 cards (using Motorola 68882), one array processor based on the new TI SN74ACT8847 and a SCSI disk/streaming tape subsystem is recommended.

Benefits Assessment

Our Phase I study determined that without parallel processing it is not possible to process radar data in real time with a Kalman filter based on today's data sample rates. Hence, the major benefit of parallel processing (more specifically parallel Kalman filtering) is that it enables real-time radar data processing that could not be performed otherwise. This translates to quick-look and improved down range safety during missile flight testing.

More specifically, the major benefits of the proposed Phase I and II research include:

- o A target tracking test facility that can be used in the lab or in the field for "quick-look" analysis of flight data improving safety and flight data quality
- o A parallel processing test facility that provides advanced state-of-the-art computing resources to solve WSMR target tracking problems that could not be solved otherwise.
- o An industry-standard parallel computing environment that can enable the validation of new parallel Kalman filter algorithms and architectures as they become available.

1.3 TECHNICAL OBJECTIVES

The major objectives of the proposed Phase I & II research include:

- o Ability to track multiple targets at sample rates approaching 15 KHz for real-time applications
- o Realize between two and three orders of magnitude improvement (100 to 1000x) in computational speed over existing Kalman-filter-based tracking systems
- o Design and optimize application-specific, parallel processing hardware for real-time target tracking
- o Fabricate, test and deliver a parallel computing system that is well suited for implementing parallel Kalman filter tracking algorithms, FFT computations and target motion resolution algorithms
- o Validate the proposed hardware and software by thoroughly exercising the system with actual missile flight test data available from WSMR
- o Deliver the parallel computing test facility and demonstrate the accuracy and speed benefits of the parallel Kalman filter methods by solving a realistic target tracking application of strategic interest to WSMR

1.4 SUMMARY

The remainder of this report is divided into four sections. Section 2 provides an overview of our technical approach including math modeling.and the parallel Kalman filter algorithms. Section 3 derives parallel computing architectures which are well suited for fast implementation of the PKF methods. Section 4 illustrates the performance of the PKF methods by solving two realistic test cases with the PKF methods. Conclusions and recommendations are given in Section 5.

SECTION 2

TECHNICAL APPROACH

2.1 OVERVIEW OF PHASE I & II TECHNICAL APPROACH

Our technical approach is shown in Figure 2-1. Our approach is based on (1) our in-depth understanding of the computational requirements associated with Kalman filtering and (2) two meetings with Foo Lam, Elwin Nunn and Bob Green to discuss WSMR missile data processing requirements. Our technical approach involves the design of new hardware as well as system integration of off-the-shelf components.

64-Bit Array Processor Design

The new design activity focusses mainly on next generation array processor technology for 64-bit floating-point FFTs (useful in precise TMR), ill-conditioned linear equation solvers and other linear algebra (matrix/vector operations) needed by the parallel Kalman filter. Our plan is to survey newly announced floating-point processors, fast memory and address generators from several vendors prior to design in. Since the project is three years in duration, the technology selected must provide significant performance gains to be state-of-the-art three years from now.

Off-the-Shelf Components

As indicated later in this report, twelve Systolic-481 parallel numeric processors are well matched for the WSMR target tracker. To ensure maximum performance the 16 MHz 68881s in the Systolic-481 shall be upgraded to 25 MHz 68882s to increase performance by nearly 10x (an order of magnitude) per 481 board. With twelve Systolic-481 cards in the test-bed, this upgrade represents 12 x 10 = 120x improvement in computational performance (i.e., over two orders of magnitude over a single 16 MHz Motorola 68020/68881 32-bit microprocessor pair. These off-the-shelf components shall be purchased along with fast memory cards, host interface, Vme bus chassis, etc. Our strategy shall be to delay the purchase

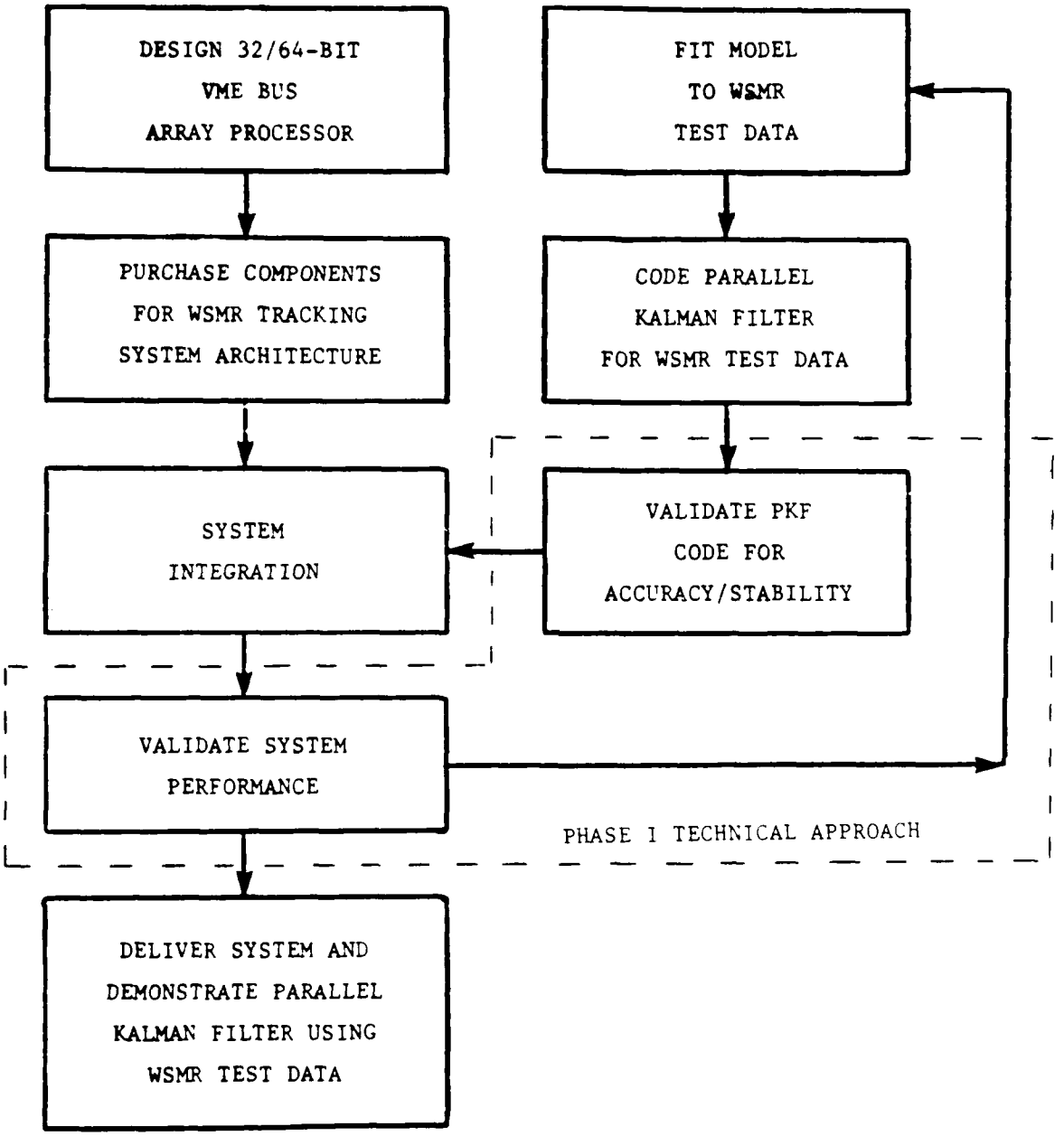


Figure 2-1: Phase II Technical Approach

of components as long as possible (perhaps until the end of the second year) to reduce cost and provide maximum performance within the budget. Hence, much of the first year effort will be on the 64-bit array processor design and fitting math models to the WSMR test data.

Math Modeling

Parallel algorithms, such as the parallel maximum likelihood method, shall be investigated for fitting math models to the WSMR test data. Like the PKF methods utilized in Phase I, parallel trapezoidal rule can be incorporated into the parallel maximum likelihood method for missile model parameter identification. Code development for this activity can be initially done on a Systolic-481-based parallel computer. Later in the contract the 68881 chips can be replaced with fast 25 MHz 68882 chips prior to delivery.

Parallel Kalman Filtering

The parallel Kalman filter methods shall be employed to process actual missile flight test data under Phase II. The two- and four-processor parallel Kalman filter methods shall be used to estimate each missile state variable simultaneously in parallel. If a nine-state model is utilized, $9 \times 4 = 36$ processors shall be employed in the parallel Kalman filter.

The speed of the parallel Kalman filter formulation and parallel computing testbed shall be compared to a single M68020/68881 in a SUN 3/260 workstation, Micro VAX II and Intel 386/387 in an IBM PS/2 Model 80.

System Validation and Performance Enhancements

The integrity of the WSMR math model, the parallel Kalman filter algorithms, the parallel architectures, and the parallel computing testbed shall all be validated under the Phase II work plan. This can be accomplished by comparing the numerical results of the parallel formulation with the numerical results of the standard Kalman filter on a sequential computer. In the limit as the step size approaches zero, the parallel and sequential methods should give the same results. This shall be confirmed under Phase II.

2.2 MATH MODEL DEVELOPMENT FROM ACTUAL FLIGHT TEST DATA

2.2.1 Analysis of the Flight Data

Our objective is to determine a math model and its associated parameters from flgiht test data. The flight data provided by White Sands Missile Range (WSMR) was unclassified. The flight data covered a ten minute mission of six missiles launched in December 1987. Although over three Mbytes of data was provided, it is likely that only a small portion of the data is required for modeling purposes.

Detrending the Data

The mean (i.e., average) and covariance associated with the flight data shall be computed. The data can then be detrended by subtracting the mean from the data. The resulting data, although still noisy, can then be used for modeling.

Modeling Considerations

The data shall be divided into two parts - a training set and test set. The training set data can be selected by computing the information matrix (inverse covariance matrix) along the data. The objective is to find a data set that has maximum information (i.e., when the norm of the information matrix is maximum). The max norm data set is then used to compute a math model of the data. The test set (all data other than the training set) is used to test the "goodness of fit" of the model to the training set.

2.2.2 Math Modeling With Neural Networks

Neural-network-based modeling algorithms are self-organizing (or adaptive learning) methods, in that the "best" model is constructed from all possible combinations of sensor measurement data. Thus, no data event is overlooked. This feature of neural-network-type modeling algorithms along with the fact that all math models can be generated simultaneously (in parallel) makes neural-network-type algorithms particularly well suited for real-time missile modeling applications.

To illustrate how a parallel neural-network-type algorithm is used for math modeling, consider the following example:

In the parallel neural net algorithm, let

$$y_i = a_0 + a_1 x_{1_i} + a_2 x_{2_i} + \text{noise} \qquad i = 1, 2, \ldots, n \qquad (2.1)$$

where y_i are dependent sensor measurement values which depend on the independent state values x_{1_i} and x_{2_i}.

The objective is to find the "best" estimate of a_0, a_1 and a_2 in a least square sense. To estimate a_0, a_2 and a_3, the classical "normal" equations can be solved as follows:

$$\begin{bmatrix} N & \Sigma x_1 & \Sigma x_2 \\ \Sigma x_1 & \Sigma x_1^2 & \Sigma x_1 x_2 \\ \Sigma x_2 & \Sigma x_1 x_2 & \Sigma x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \Sigma y \\ \Sigma x_1 y \\ \Sigma x_2 y \end{bmatrix} \qquad (2.2)$$

Note that the terms in the normal equations can be solved rapidly using an array processor to compute the sum-of-products and sum-of-squares operations. In addition, due to the independence of the data, multiple array processors can be run in parallel to construct all possible model forms (see Figure 5-1). The resulting model forms can be compared and the "best" model selected based on prediction error of F-ratio statistics.

In Figure 2-2 suppose that

$$z_1 = a_0 + a_1 x_1 + a_2 x_2 \quad \text{and} \quad z_3 = c_0 + c_1 x_2 + c_3 x_3 \qquad (2.3)$$

Then it can be shown that

$$y = d_0 + d_1 z_1 + d_2 z_3 = A_1 x_1 + A_2 x_2 + A_3 x_3 \qquad (2.4)$$

where

$$A_0 = d_0 + a_0 d_1 + c_0 d_2 \qquad (2.5)$$

$$A_1 = a_1 d_1 \qquad (2.6)$$

$$A_2 = a_2 d_1 + c_1 d_2 \tag{2.7}$$

and

$$A_3 = c_3 d_3 \tag{2.8}$$

Thus, given the least square values of A_0, A_1, A_2, A_3, y can be estimated from the original independent variables x_1, x_2, and x_3. Note that using polynomials is ideal for constructing math models in terms of known physical variables. Also, note that the least squares curve fit can be performed simultaneously with multiple array processors. The availability of high performance board-level array processors makes large neural-net-based estimators feasible.

Note that the neural-network-based modeling algorithm is not limited to linear polynomials and in fact can be generalized for quadratic and cubic polynomials among others. Thus, the neural-net method can be used to construct nonlinear models from test data. These models, however, may not be related back to the original measurement variables as easily and, as such, may have less physical meaning.

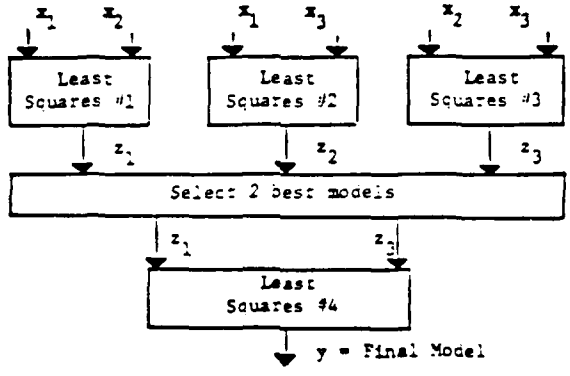


Figure 2-2: Neural Network for Least Squares Estimation of Model Parameters

The information flow in the generalized neural net modeling approach is illustrated in Figure 2-3.

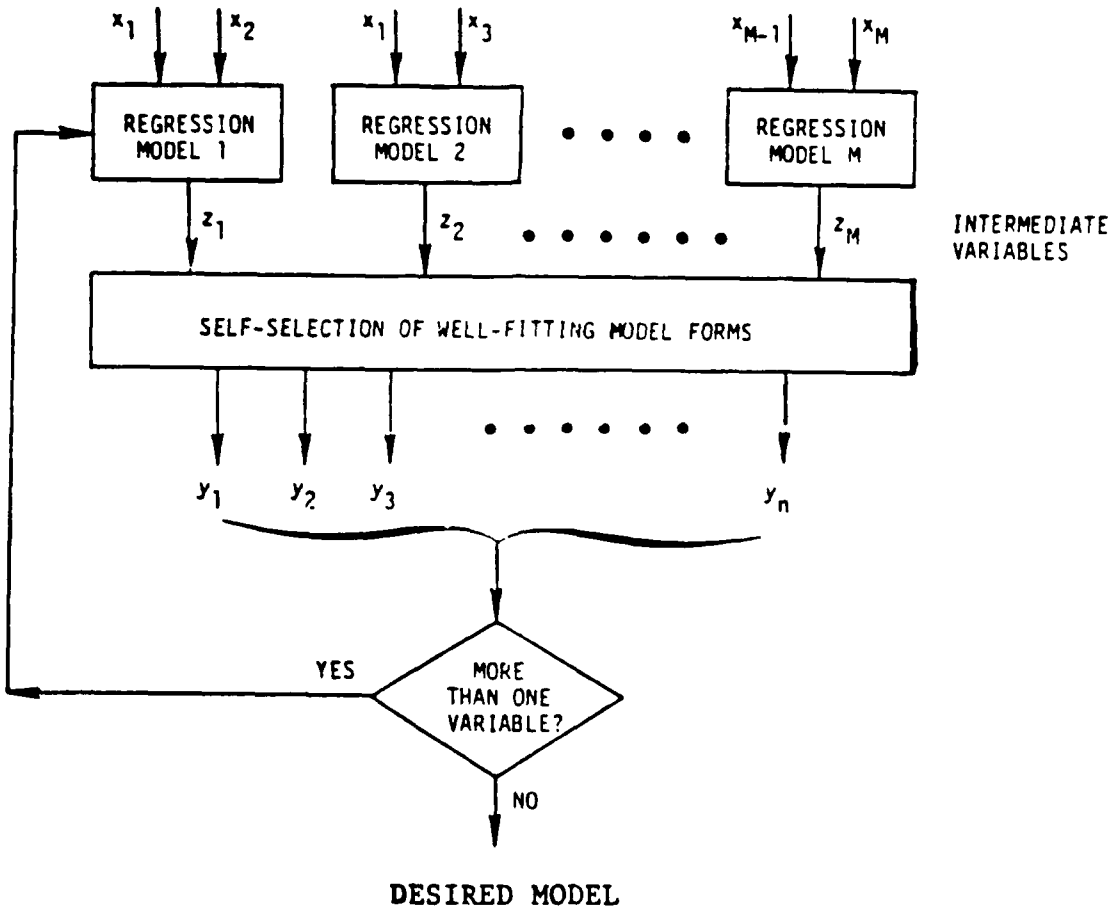


Figure 2-3: Generalized Neural Network for Model Determination

2.2.3 Real-Time Estimation of the Target Measurement Matrix Using Neural Nets

Now that the neural network technology has been discussed, we shall see how it can be applied to real-time estimation of the target measurement matrix, H. A bank of neural nets are employed to estimate several candidate values of H (the target measurement matrix) in Figure 2-4. Each neural net is given a precomputed trajectory corresponding to a candidate target type and measurement values of the actual target.

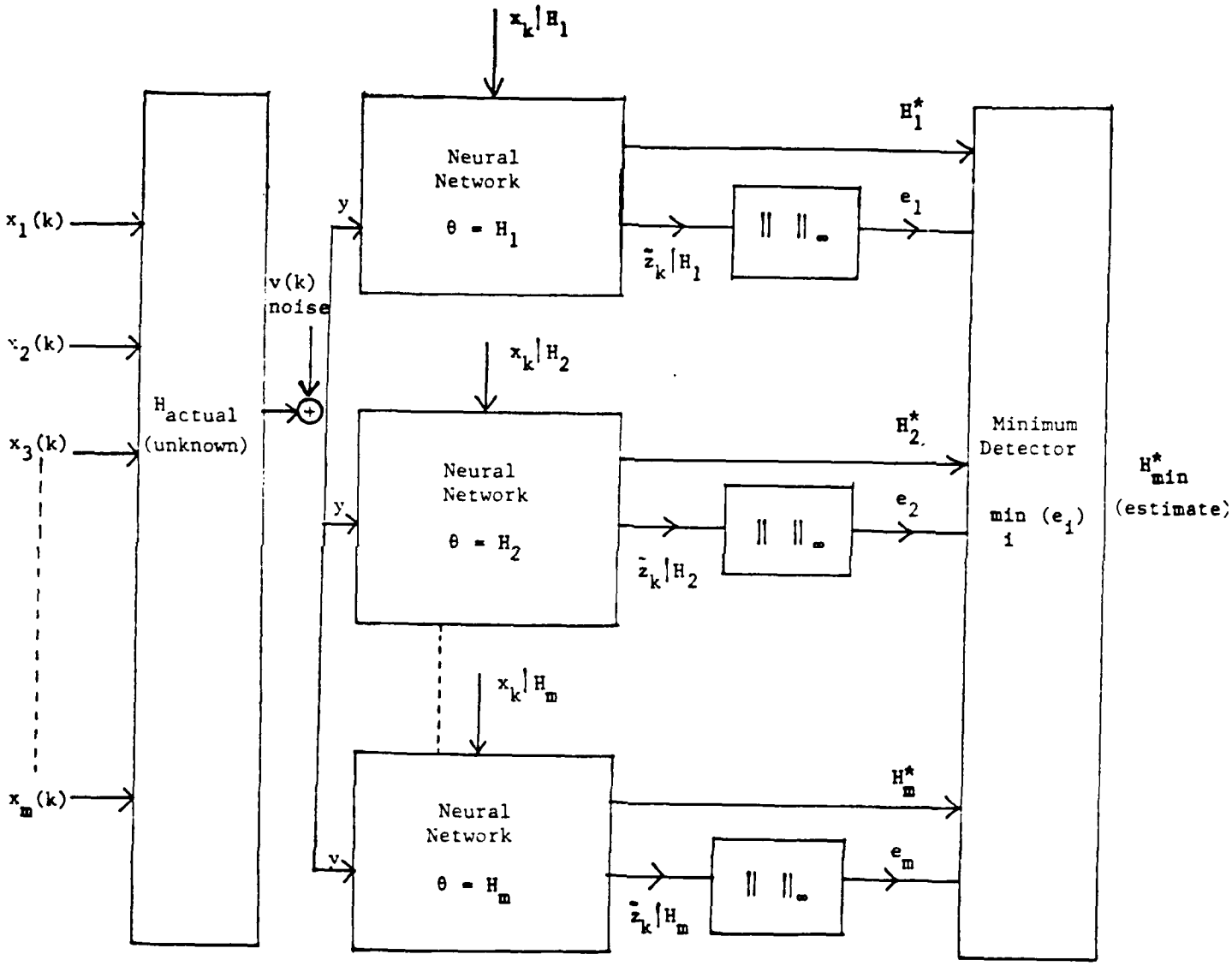


Figure 2-4: Parallel Neural Network Estimator to Find H* (y = Hx* + v)

Each neural net computes a least squares estimate for H.

For example, if H is a 2x1 matrix, then the elements of the H matrix can be found by solving the following set of "normal" equations:

$$\begin{bmatrix} \sum_k x_{1k}^2 & \sum_k x_{1k} x_{2k} \\ \sum_k x_{1k} x_{2k} & \sum_k x_{2k}^2 \end{bmatrix}^{2x2} \begin{bmatrix} h_{11} \\ h_{12} \end{bmatrix}^{2x1}_{i} = \begin{bmatrix} \sum_k x_{1k} y_{1k} \\ \sum_k x_{2k} y_{1k} \end{bmatrix}^{2x1} \tag{2.9}$$

Then the least squares estimate of H for the i^{th} neural net is given by:

$$H_i = (h_{11} \quad h_{12}) \tag{2.10}$$

This process is performed simultaneously (in parallel) by each neural net. The innovations for each neural network are:

$$\tilde{z}_k \, H_i = y_k - H_i \, x_k \tag{2.11}$$

Taking the norm of each innovation $e_i = ||z_k|H_i||_\infty$ provides a measure of the size of the accumulated estimation error for each neural net. The network with the smallest error, $e_{min} = \min_i e_i$, provides the best estimate of $H_i^* = H^*$. Once H_i is known, the class of target is classified immediately since it corresponds to one of the precomputed candidate target trajectories $x_k|H_i$. Now that H^* is known, the actual target trajectory can be determined by solving the following system of linear equations at every time step k.

$$(H^{*T}H^*) \, \hat{x}_k = H^{*T}H^* \, y_k \tag{2.12}$$

Estimated Target Trajector	Actual Target Measurements

Thus, the neural net learns H^* on-line and we use H^* to estimate the target trajectory, $\hat{x}_k$, for the missile.

2.2.4 Parallel Maximum Likelihood Parameter Identification for Nonlinear Missile Math Models

The least square parameter estimates obtained with the parallel neural-net method are biased estimates of the true missile parameters. To remove the bias and ensure that the parameters are consistent over the full dynamic range of the missile model, the parallel maximum likelihood (PML) method and/or the parallel maximum a priori (PMAP) method can be used. The refinement of the nonlinear model parameters can be handled directly or cast as a nonlinear two-point boundary value problem (NTPBVP). The NTPBVP can be solved using the parallel shooting method. In addition, any unknown states of the missile model can be reconstructed simultaneously with the aerodynamic parameters using the PML and/or PMAP method. In the remainder of this section, the parallel maximum likelihood (PML) and parallel maximum a posteriori (PMAP) methods are described in detail.

Parallel Maximum Likelihood Method

The maximum likelihood method of parameter identification has been proven to be highly successful over the years in aerospace application. Parallel processing technology can be embedded into the standard maximum likelihood method to speed up computations. In particular, the parallel maximum likelihood (PML) method uses (1) parallel numerical methods for integrating the missile's equations of motion, (2) a parallel Kalman filter to construct the innovations and (3) parallel optimization methods to select the best parameter values which minimize the ML performance index. Mathematically, the maximum likelihood method for missile parameter identification can be stated as follows:

Consider the following nonlinear missile state and measurement model:

$$\dot{x}(t) = f(x(t),\theta(t),t) + \theta w(t) \tag{2.13}$$

$$z(t) = h(z(t),\theta(t),t) + v(t) \tag{2.14}$$

where $x(t)$ is an n-dimensional state vector, $\theta(t)$ is an r-dimensional vector of unknown parameters, $w(t)$ is a p-dimensional process noise, $z(t)$ is an m-dimensional measurement vector which has been corrupted by the measurement noise, $v(t)$.

The maximum likelihood parameter estimates are obtained by minimizing the following negative log likelihood function:

$$J(\theta) = \tfrac{1}{2}\sum_{i=1}^{N} \left\{ \nu^T B^{-1} \nu + \log |B| \right\} \tag{2.15}$$

where

$$\nu = z - h\,[x(t),\ \theta(t),\ t] \tag{2.16}$$

and

$B = E[\nu\nu^T]$ is the covariance of the innovations ν .

The innovations (ν) and the covariance of the innovations (B) are obtained from an extended Kalman filter. The Kalman filter equations can be reformulated for parallel processing and the parameter estimates can then be updated using the following rule:

$$\theta_{k+1} = \theta_k - \lambda_k R_k g_k \tag{2.17}$$

where λ_k is a scalar stepsize parameter chosen to ensure that $J(\theta_{k+1}) < J(\theta_k)$, g_k is a vector of gradients of the negative log likelihood function

$$g_k = \left.\frac{\partial J}{\partial \theta}\right|_{\theta = \theta_k} \tag{2.18}$$

and R_k is an approximation to the Hessian of J

$$\left.\frac{\partial^2 J}{\partial \theta^2}\right|_{\theta=\theta_k} . \tag{2.19}$$

Furthermore, it can be shown that

$$g_k = \sum_{i=1}^{N} \left\{ \nu^T B^{-1} \frac{\partial \nu}{\partial \theta_k} - \tfrac{1}{2}\, \nu^T B^{-1} \frac{\partial B}{\partial \theta_k}\, B^{-1}\nu + \tfrac{1}{2}\, \mathrm{tr}\left(B^{-1} \frac{\partial B}{\partial \theta_k} \right) \right\} \tag{2.20}$$

The major computational savings due to parallel processing is in evaluating g_k, ν and B in the Kalman filter. Clearly, the complexity of this technique requires parallel processing for real-time missile applications. The stability of the Kalman filter computations also needs to be closely monitored (say via an expert system) so that the innovations are proper.

An alternative to the parallel maximum likelihood (PML) method is the parallel maximum-a-posteriori (PMAP) method. The PMAP method is based on solving nonlinear two-point boundary value problems via the parallel shooting method. Although this method is batch-processing oriented (the PML method is recursive and as such, is well suited for real-time application), the PMAP method can be useful in flight if the batch processing can be performed within the sample time of the measurements. The PMAP method, however, will provide superior smoothed parameter estimates compared with the PML method. Thus, it may be appropriate to consider the PMAP approach for off-line, ground-based missile data reduction.

Parallel Maximum-a Posteriori Method

Consider the nonlinear aircraft state and measurement model represented by:

$$\dot{x}(t) = f(x(t),t) + \Gamma(x(t),t)\, w(t) \tag{2.21}$$

$$z(t) = h(x(t),t) + v(t) \tag{2.22}$$

where $x(t)$ is an n-dimensional augmented state vector which includes the unknown parameters, $w(t)$ is a p-dimensional process noise, and $z(t)$ is the m-dimensional measurement vector which has been corrupted by the measurement noise $v(t)$.

Now let m_{x_0} = the mean of $x(t_0)$ and $E(x(t_0)\, x^T(t_0)) = P_{x_0}$ = the covariance of $x(t_0)$. Similarly, let $E(ww^T) = Q$ = the process noise covariant and $e(vv^T) = R$= the measurement noise covariance. In addition, suppose z and v have a zero mean in this analysis (if not, a bias term can be identified).

Now let $Z_t = (z(\tau)\ t_0 \leq \tau \leq t)$ define the accumulated noisy state measurements up to and including time t. The problem is to obtain an

estimate of the augmented state vector $x(t)$ at time t on the basis of the observations represented by Z_s. Our interest will be restricted to the case when $s > t$ in which case $\hat{x}_{t|s}$ is referred to as a "smoothed" estimate of $x(t)$.

By defining $p(x;t|Z_t)$ as the a posteriori probability that the state vector assumes the value x at time t conditioned upon the measurement data represented by Z_t, the maximum-a-posteriori (MAP) estimate of $\hat{x}_{t|s}$ (denoted as $\hat{x}^{MAP}_{t|s}$) is defined by

$$P(\hat{x}^{MAP}_{t|s};t|Z_t) = \max_{x\epsilon R^n} p(x;t|Z_t) \qquad t_0 \leq t < s \leq t_f \tag{2.23}$$

It has been shown that the maximization indicated in Eq. (5.24) is equivalent to finding the deterministic signal, $\hat{w}(t)$ $t\epsilon(t_0,t_f)$ which minimizes the functional

$$J = \tfrac{1}{2}||\hat{x}(t_0) - m_{x_0}||^2_{P_{x_0}} + \tfrac{1}{2}\int_{t_0}^{t_f}(||z(t) - h(\hat{x}(t),t)||^2_{R^{-1}(t)} + ||\hat{w}(t)||^2_{Q^{-1}(t)})\,dt \tag{2.24}$$

subject to the dynamic-equality constraint given by

$$\dot{\hat{x}}(t) = f(\hat{x}(t),t) + \Gamma(\hat{x}(t),t)\,\hat{w}(t) \qquad \forall\ t\ \epsilon\ (t_0,t_f)\ . \tag{2.25}$$

Note that the SAP estimation problem has been converted to a deterministic optimization problem and that once $\hat{w}(t)$ is found such that equation (2.24) is minimized, equation (2.25) can be integrated to obtain the MAP estimate of $\hat{x}(t)$ provided $\hat{x}(t_0)$ is known.

To find $w(t)$ using the calculus of variations, let the Hamiltonian be defined as

$$H = \tfrac{1}{2}(||z(t) - h(\hat{x}(t),t)||^2_{R^{-1}(t)} + ||\hat{w}(t)||^2_{Q^{-1}(t)}) + \lambda^T(t)(f(\hat{x}(t),t) + \Gamma(\hat{x}(t),t)\,\hat{w}(t)) \qquad \forall\ t\ \epsilon\ (t_0,t_f) \tag{2.26}$$

The most direct method for solving this problem would be to initially set $\hat{x}(t_0)$ to m_{x_0}, integrate Eq. (2.25) forward in time over the interval (t_0,t_f) and evaluate the performance index (2.24). By considering changes in the performance index due to changes in $\hat{x}(t_0)$, one may use this information to decide if this procedure should be repeated. Specifically, if the change in J is sufficiently small and the gradient of J equals zero, then $\hat{x}(t_0)$ is accepted. Otherwise, the value of $\hat{x}(t_0)$ should be selected such that the performance index is minimized. To speed computations, parallel integration methods can be used to integrated Eq. (2.25), while the selection of the next value of $x(t_0)$ can be made using an optimization method.

2.3 PARALLEL KALMAN FILTERING BASED ON THE DECOUPLING PRINCIPLE

To speed up Kalman filter computations, several methods have been considered during the past decade. For linear filtering, the Kalman filter equations have been coded on an array processor. This method works well for large models but not for relatively small models (models with less than 20 state variables). Small models frequently occur in target tracking, navigation, guidance and control. Hence, there is a need to develop fast Kalman filter methods for small models. In particular, in the context of target tracking, each target can be represented by a nine-state model. The problem, however, is to estimate the trajectory of multiple targets in real time. Although many target trajectories can be computed in parallel, ultimate tracker performance is dependent on the speed in which each target trajectory can be predicted with the Kalman filter.

To provide a more accurate solution, the Parallel Kalman Filter (PKF) is based on the trapezoidal rule of numerical integration rather than Euler integration which has been used to date. The Parallel Kalman Filter update is then accurate to $0(h^2)$ rather than $0(h)$. The additional accuracy is important because it is anticipated that the integration step size, h, will be large due to the computational complexity of the Kalman filter equations. For example, with a sample rate of 100 Hz, the Euler-integration-based PKF would be accurate to $0(h) = 0.01$, while the trapezoidal-rule-based PKF would be accurate to $0(h^2) = 0.0001$ (i.e., as accurate as the 12-bit sensor data).

2.3.1 Nonlinear Extended Parallel Kalman Filter Based on the Trapezoidal Rule

The trapezoidal rule for integrating a set of ordinary differential equations is given by:

$$x_{k+1} = x_k + \tfrac{1}{2} h \, (f(x_k,t_k) + f(x_{k+1},t_{k+1})) \tag{2.27}$$

where x is the solution of the ode, $f(x,t)$ is the right-hand-side (RhS) of the initial value problem and $h = t_{k+1} - t_k$ is the integration step size. The trapezoidal rule is an implicit method since x_{k+1} appears implicitly on the RHS of Eq. (2.27). Note that $f(x,t)$ can be, in general, a nonlinear function or linear such as $f(x,t) = Fx(t)$. To solve Eq. (2.27), a predictor is needed of the form below to estimate x_{k+1}:

$$x_{k+1} = x_k + hf(x_k,t_k) \tag{2.28}$$

Hence, by combining (2.27) and (2.28) we obtain a predictor-corrector method based on the trapezoidal rule:

$$\text{Predictor:} \quad x^p_{k+1} = x^c_k + hf(x^c_k,t_k) \tag{2.29}$$

$$\text{Corrector:} \quad x^c_{k+1} = x^c_k + \tfrac{1}{2} h \, (f(x^c_k,t_k) + f(x^p_{k+1},t_{k+1}) \tag{2.30}$$

Note that the predictor must be evaluated <u>before</u> the corrector equation can be computed. A parallel predictor-corrector (PPC) method allows the predictor and corrector to be evaluated simulataneously on two processors as follows:

2.3.2 Parallel Trapezoidal Rule (Two Processors)

$$\text{Predictor:} \quad x^p_{k+1} = x^c_{k-1} + 2 h f^p_k \tag{2.31}$$

$$\text{Corrector:} \quad x^c_k = x^c_{k-1} + \tfrac{1}{2} h \, (f^p_k + f^c_{k-1}) \tag{2.32}$$

where $f^p_k = f(x^p_k,k)$ and $f^c_{k-1} = f(x^c_{k-1},k-1)$.

In the special case when f^p_k is the RHS of the Kalman filter state update before a measurement and G^p_k is the RHS of the covariance update before a measurement then the two-processor extended parallel Kalman filter is given by:

Nonlinear Extended Parallel Kalman Filter Based on Trapezoidal Rule (Two Processors):

Time Update:

$$\text{Predictor:}\quad \hat{x}_{k+1}(-) = \hat{x}_{k-1}(+) + 2h\, f(\hat{x}_k(-)) \tag{2.33}$$

$$P_{k+1}(-) = P_{k-1}(+) + 2h\, G_k^p \tag{2.34}$$

$$\text{where}\quad G_k^p = F(\hat{x}_{k-1}(-))P_{k-1}(-) + P_{k-1}(-)F^T(\hat{x}_{k-1}(-)) + Q_{k-1} \tag{2.35}$$

$$\text{Corrector:}\quad \hat{x}_k(-) = \hat{x}_{k-1}(+) + \tfrac{1}{2} h\, (f(\hat{x}_k(-)) + f(\hat{x}_{k-1}(+))) \tag{2.36}$$

$$P_k(-) = P_{k-1}(+) + \tfrac{1}{2} h\, (G_k^p + G_{k-1}^c) \tag{2.37}$$

$$\text{where}\quad G_{k-1}^c = F(\hat{x}_{k-1}(+))P_{k-1}(+) + P_{k-1}(+)F(\hat{x}_{k-1}(+)) + Q_{k-1} \tag{2.38}$$

Measurement Update:

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k(z_k - h_k(\hat{x}_k(-))) \tag{2.39}$$

$$P_k(+) = (I - K_k H_k(\hat{x}_k(-)))\, P_k(-) \tag{2.40}$$

$$K_k = P_k(-)H_k^T(\hat{x}_k(-)) * (H_k(\hat{x}_k(-))P_k(-)H_k^T(\hat{x}_k(-)) + R_k)^{-1} \tag{2.41}$$

where

$$F(\hat{x}_{k-1}(-)) = \left.\frac{\partial f(x(t_k))}{\partial x(t_k)}\right|_{x(t) = \hat{x}_{k-1}(-)}$$

$$H_k(\hat{x}_k(-)) = \left.\frac{\partial h(x(t_k))}{\partial x(t_k)}\right|_{x(t_k) = \hat{x}_k(-)}$$

In the above PKF, the (-) notation represents a value before a measurement update and the (+) notation is a value after a measurement update. Similarly, the p for predictor corresponds to the (-) notation and the c for corrector value corresponds to the (+) notation.

Nonlinear Extended PKF Based on the Trapezoidal Rule (Four Processors)

With four processors, the parallel trapezoidal rule is given by:

2.3.3 Parallel Trapezoidal Rule (Four Processors)

$$\text{Predictor:}\quad x_{2k+2}^p = x_{2k-2}^c + 4h\, f_{2k}^p \tag{2.42}$$

$$x_{2k+1}^p = x_{2k-2}^c + \frac{3}{2} h\, (f_{2k}^p + f_{2k-1}^p) \tag{2.43}$$

$$\text{Corrector:}\quad x_{2k}^c = x_{2k-3}^c - \frac{h}{2} (3f_{2k}^p - 9f_{2k-1}^p) \tag{2.44}$$

$$x_{2k-1}^c = x_{2k-3}^c + 2h\, f_{2k-2}^c \tag{2.45}$$

$$\text{where}\quad f_{2k}^p = f(x_{2k}^p, 2k),\quad f_{2k-1}^p = f(x_{2k-1}^p, 2k-1)\quad \text{and}$$

$$f_{2k-2}^c = f(x_{2k-2}^c, 2k-2)$$

For the nonlinear EPKF, the time update is given by:

Time Update:

$$\text{Predictor:}\quad \hat{x}_{2k+2}(-) = \hat{x}_{2k-2}(+) + 4\,h\,f^{p}_{2k} \tag{2.46}$$

$$P_{2k+2}(-) = P_{2k-2}(+) + 4\,h\,G^{p}_{2k} \tag{2.47}$$

$$\hat{x}_{2k+1}(-) = \hat{x}_{2k-2}(+) + \frac{3}{2}\,h\,(f^{p}_{2k} + f^{p}_{2k-1}) \tag{2.48}$$

$$P_{2k+1}(-) = P_{2k-2}(+) + \frac{3}{2}\,h\,(G^{p}_{2k} + G^{p}_{2k-1}) \tag{2.49}$$

$$\text{Corrector:}\quad \hat{x}_{2k}(-) = \hat{x}_{2k-3}(-) - \frac{h}{2}\,(3f^{p}_{2k} - 9f^{p}_{2k-1}) \tag{2.50}$$

$$P_{2k}(-) = P_{2k-3}(-) - \frac{h}{2}\,(3G^{p}_{2k} - 9G^{p}_{2k-1}) \tag{2.51}$$

$$\hat{x}_{2k-1}(-) = \hat{x}_{2k-3}(-) + 2\,h\,f^{c}_{2k-2} \tag{2.52}$$

$$P_{2k-1}(-) = P_{2k-3}(-) + 2\,h\,G^{c}_{2k-2} \tag{2.53}$$

$$\text{where}\quad f^{p}_{2k} = f(\hat{x}_{2k}(-)) \tag{2.54}$$

$$f^{p}_{2k-1} = f(\hat{x}_{2k-1}(-)) \tag{2.55}$$

$$f^{c}_{2k-2} = f(\hat{x}_{2k-2}(+)) \tag{2.56}$$

$$G^{p}_{2k} = F(\hat{x}_{2k}(-))P_{2k}(-) + P_{2k}(-)F(\hat{x}_{2k}(-)) + Q_{2k} \tag{2.57}$$

$$G^{p}_{2k-1} = F(\hat{x}_{2k-1}(-))P_{2k-1}(-) + P_{2k-1}(-)F(\hat{x}_{2k-1}(-)) + Q_{2k-1} \tag{2.58}$$

$$G^{c}_{2k-2} = F(\hat{x}_{2k-2}(+))P_{2k-2}(+) + P_{2k-2}(+)F(\hat{x}_{2k-2}(+)) + Q_{2k-2} \tag{2.59}$$

Measurement Update:

$$\hat{x}_{2k-1}(+) = \hat{x}_{2k-1}(-) + K_{2k-1}(z_{2k-1} - h_{2k-1}(\hat{x}_{2k-1}(-))) \tag{2.60}$$

$$P_{2k-1}(+) = (I - K_{2k-1}H_{2k-1}(\hat{x}_{2k-1}(-)))P_{2k-1}(-) \tag{2.61}$$

$$K_{2k-1} = P_{2k-1}(-)H^{T}_{2k-1}(\hat{x}_{2k-1}(-))*(H_{2k-1}(\hat{x}_{2k-1}(-))P_{2k-1}(-)H^{T}_{2k-1}(\hat{x}_{2k-1}(-)) + R_{2k-1})^{-1} \tag{2.62}$$

where

$$F(\hat{x}_{2k-1}(-)) = \left.\frac{f(x(t_{2k-1}))}{x(t_{2k-1})}\right|_{x(t)\,=\,\hat{x}_{2k-1}(-)}$$

$$H_{2k-1}(\hat{x}_{2k-1}(-)) = \left.\frac{h(x(t_{2k-1}))}{x(t_{2k-1})}\right|_{x(t_{2k-1})\,=\,\hat{x}_{2k-1}(-)}$$

This section discussed the parallel estimation methods that are needed for real-time target tracking. These methods are well suited for implementation on several Systolic-481 parallel numeric processors due to the high degree of parallelism associated with each method. The next section develops parallel architectures that are well matched to the parallel Kalman filter algorithms to achieve further reductions in computation time.

SECTION 3

THE WSMR PARALLEL COMPUTING ARCHITECTURE FOR KALMAN FILTERING

3.1 PARALLEL KALMAN FILTER ARCHITECTURES BASED ON THE DECOUPLING PRINCIPLE

Now that the systolic computational primitive architectures have been summarized, we shall develop parallel architectures for the PKF methods. The computational primitive architectures can be applied to the PKF when evaluating the RHS functions (e.g., $f(x,t) = Fx(t)$, etc.). Because, in general, $f(x)$ can be linear or nonlinear, generalized or systolic-like architectures must be considered. Hence, the computational primitive architectures for linear algebra must be expanded to consider nonlinearities. With this in mind, generalized systolic-like architectures are presented in the remainder of this section for implementing the trapezoidal-rule-based PKF.

3.1.1 PKF Architectures Based on the Trapezoidal Rule (Two Processors)

In the last section, the dual-processor PKF was developed based on the trapezoidal rule. In particular, Eqs. (2.33) to (2.41) define the method. The systolic architecture for the PKF predictor equations (2.33) and (2.36) can be most easily derived from the signal flow graph (SFG) of the PKF equations. The SFG of this method is shown in Figure 3-1. Note that the computations behind the computational wavefront can proceed in parallel on separate processors by forcing the corrector to lag the predictor by one time step. As it turns out, this is fundamental to all the PKF methods based on the decoupling principle. In this diagram, h (the integration step size) is related to the data sample rate in the filter. For example, if the sample rate is 100 Hz then the sample period is $1/100 = 0.01 = h =$ integration step size. Because the structure of the PKF state update (predictor and corrector) are essentially the same as the covariance update (predictor and corrector) without loss of generality the SFG and systolic-like architectures for the state update are discussed here.

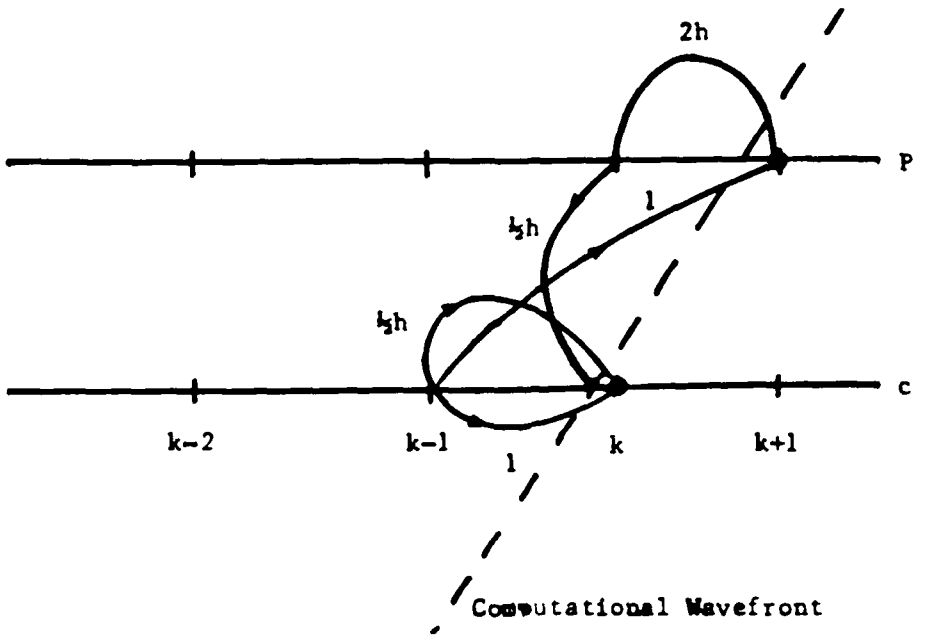


Figure 3-1: SFG of PKF Based on Trapezoidal Rule

The generalized systolic architecture for implementing the PKF based on the trapezoidal rule can be derived from the SFG of Figure 3-1 (see Figure 3-2). Figure 3-2 shows that the PKF is decoupled into two parts: the predictor processor architecture and corrector processor architecture. Note that both architectures utilize the traditional inner-product cell and a set of registers to hold intermediate values of x and f. Note also in the corrector that the output of one inner-product cell maps directly into the input of the next inner-product cell. The value $h/2$ also flows through from cell to cell.

3.1.2 PKF Architectures Based on the Trapezoidal Rule (Four Processors)

As in the previous section, the systolic-like architecture for the four-processor PKF method can be derived from its signal flow graph (SFG). The SFG for the method (Eqs. (2.46) to (2.59)) is shown in Figure 3-3. Once again Figure 3-3 illustrates that the computations can proceed in parallel becuase the computations ahead of the computational wavefront depend only on data behind the front. The four processors consist of two processors for the predictor (A and B) and two for the corrector (A and B).

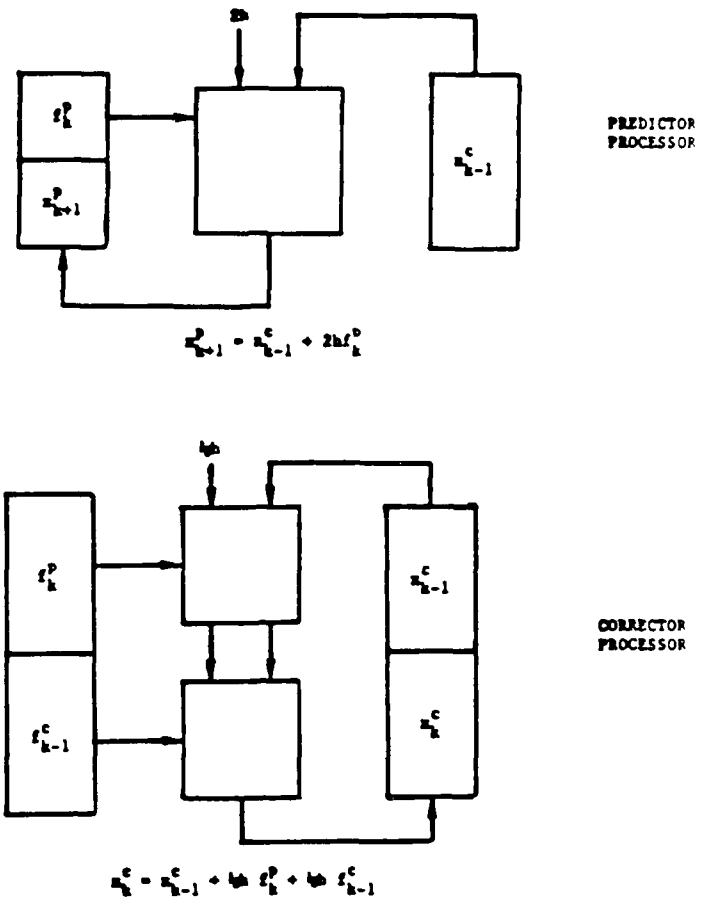


Figure 3-2: Dual-Processor PKF Architecture Based on the Decoupling Principle

The gains associated with each processor are dependent on the accuracy requirements and sampling rate through the step size parameter, h.

The PKF architecture for four processors shown in Figure 3-4a and b is derived from the SFG in Figure 3-3. These parallel architectures can run simultaneously providing a computational speed advantage of 4x. The structure of these PKF architectures are not fundamentally much different than the two-processor case in that inner-product cells and registers are employed. The stack size and number of inner-product cells has increased however. The data flow with the cells is identical to the two-processor case.

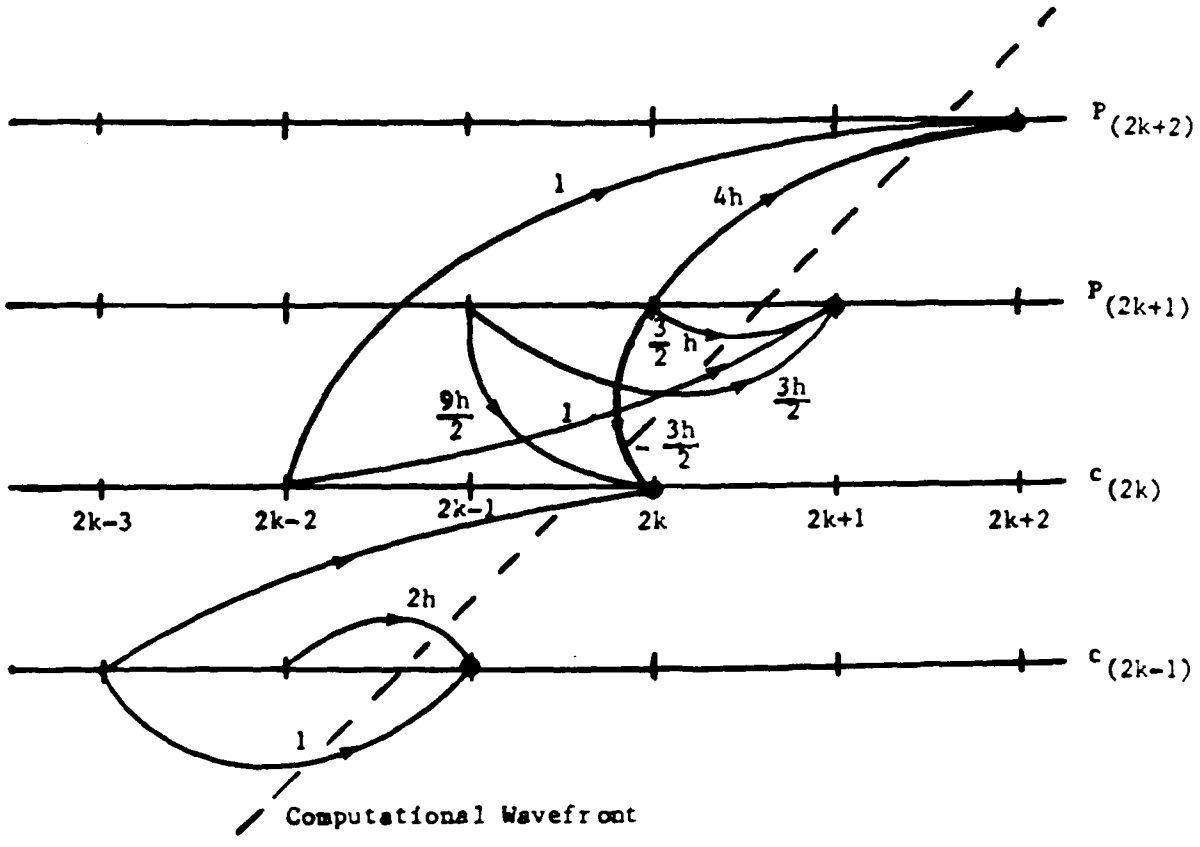
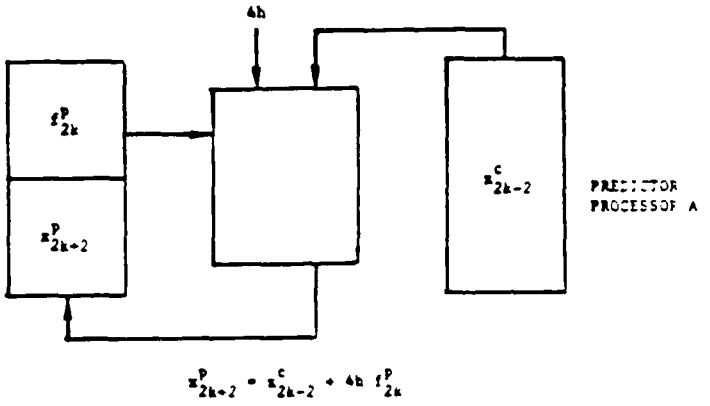


Figure 3-3: SFG of PKF Based on Trapezoidal Rule



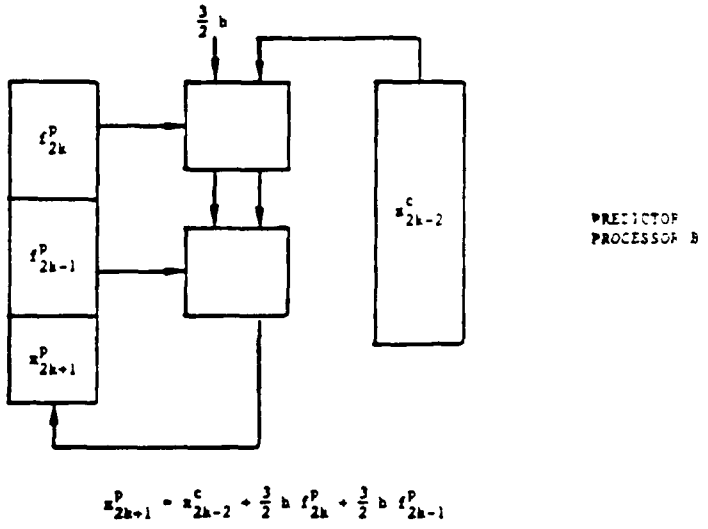


Figure 3-4a: Four-Processor PKF Predictor Architecture Based on the Decoupling Principle

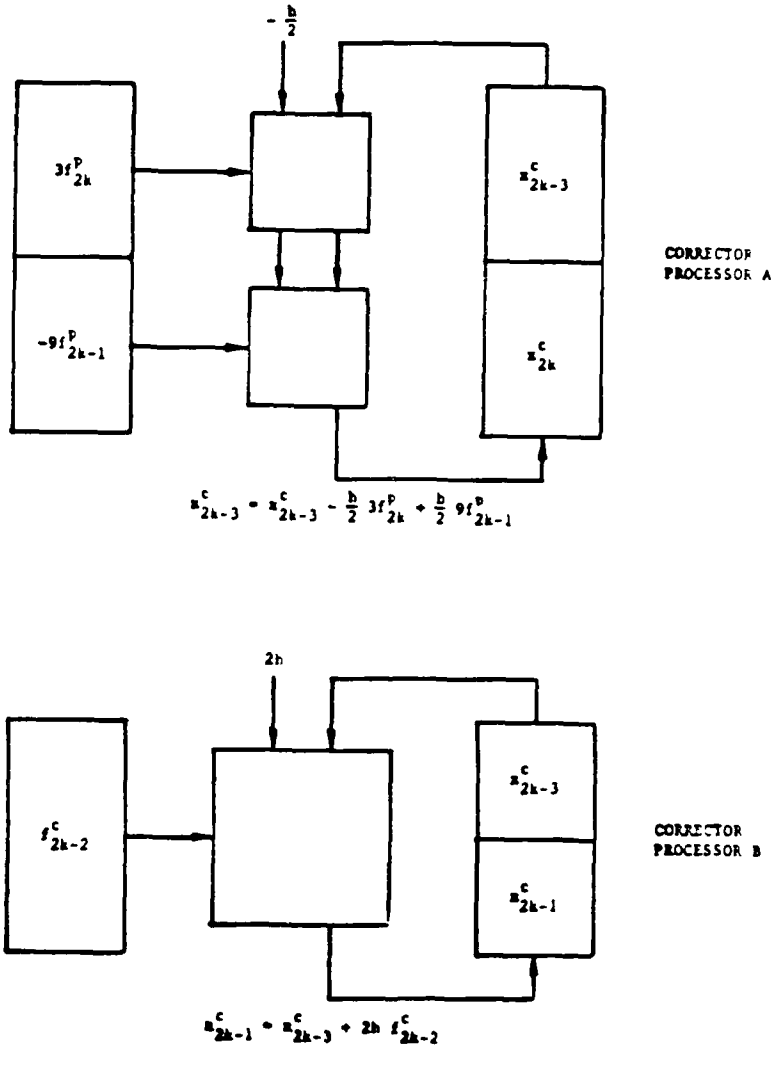


Figure 3-4b: Four-Processor PKF Corrector Architecture Based on the Decoupling Principle

3.2 THE WSMR PARALLEL COMPUTING HARDWARE ARCHITECTURE

The WSMR parallel computing architecture is based on industry-standard components. It utilizes an open architecture centered around the Vme bus so that the government can add special function cards to the testbed as desired (see Figure 3-5). Twelve Systolic-482 boards provide 48 parallel processing elements and three Mbytes of fast statis RAM as local memory.

Local processor-to-processor communications can be achieved over the Vme local bus. On-board parallelism can be configured by each Systolic-482's software controlled state machine. Global communications between bulk memory, the twelve Systolic-482 parallel processing cards and the master CPU are carried out over the Vme bus. Sixteen Mbytes of global memory is provided for main program memory storage. The master CPU contains one Motorola 68030, one 68882 math coprocessor and eight Mbytes of RAM. The master CPU is the program task scheduler and manages the PKF algorithm computations. A 25 MFLOP (64-bit array processor) is also proposed for accurate FFT's, solving ill-conditioned linear systems of equations and the linear algebra in the Parallel Kalman Filter.

3.2.1 64-bit Floating-Point Array Processor Hardware

The linear algebra requirements of the WSMR testbed can be most effectively met with a single-board Vme bus array processor. At 32-bit precision, 20 MFLOP array processor boards are commonly available for the Vme bus. At 64-bit precision, however, very few off-the-shelf products exist. The VORTEX from Sky Computers for example performs 64-bit floating-point operations at an eight MFLOP rate. To meet WSMR throughput requirements, at least three (perhaps four) of these cards are needed.

As an alternative to the above, we recommend that a next generation 64-bit Vme bus array processor be designed by Systolic Systems' technical staff. New chips from Weitek, AMD, TI, IDT, and BIT should all be evaluated for potential design into the WSMR testbed. We should design in the fastest technology available to ensure an extended life time for the testbed.

The high-speed linear equation solver hardware could be fabricated as a two-board set (see Figure 3-6). Board #1 supports the transfer of data to and from the host and a high-speed memory array. The memory array design is multi-ported to allow access by multiple computational boards (e.g., Systolic-482 cards via the Vmx port). The mathematical operation sequences are under

HARDWARE SPECIFICATIONS

20 SLOT VME SYSTEM
24 DMFLOP (NONLINEAR)
25 DMFLOP (LINEAR)
4 MBYTES SRAM
16 MBYTES DRAM
120 MBYTES STREAMING TAPE

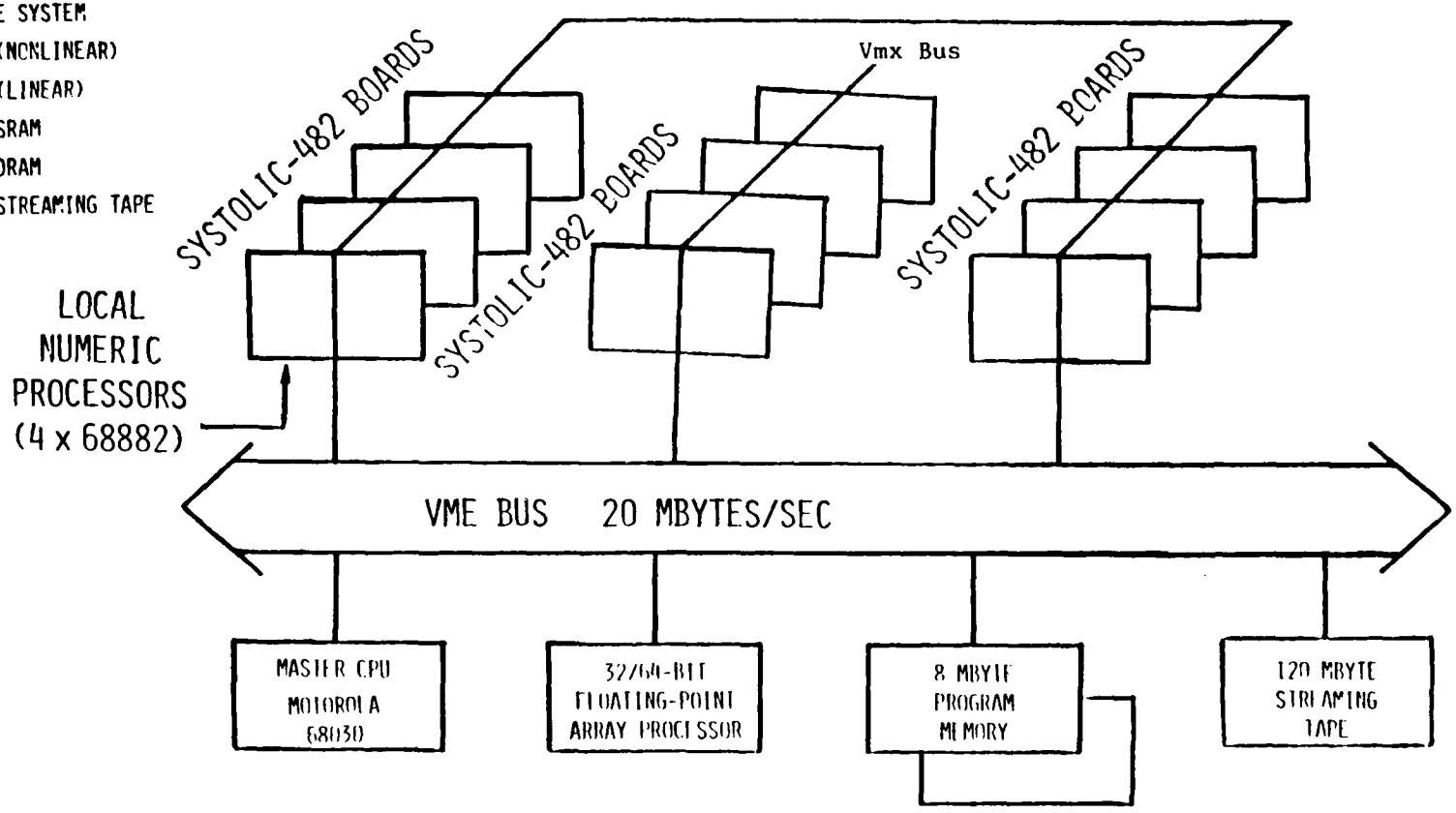


Figure 3-5: Recommended WSMR Parallel Computing Architecture

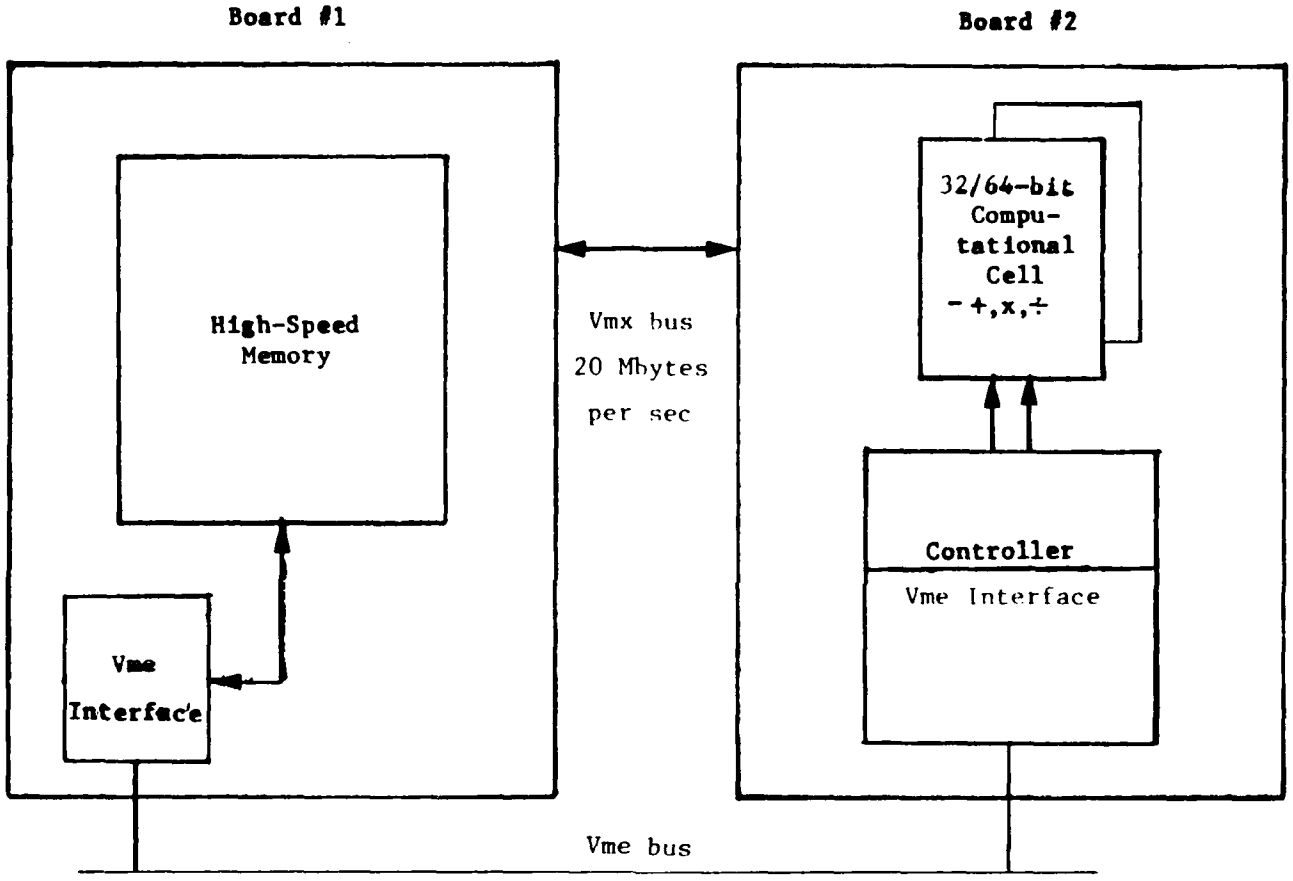


Figure 3-6: 32/64-bit Linear Algebra/ FFT Array Processor

firmware control by board #2. Each board has 64-bit chips, one multiplier and one adder. Two real-time assembly language codes shall be written to handle data communications between the host and the Systolic array processor/FFT cards.

3.2.2 Memory Considerations

The parallel Kalman filter algorithms and architectures derived earlier use decoupling to permit the predictor and corrector equations to be computed on separate processors. At any given time step k, the state, covariance, and measurements must be stored, as well as intermediate values associated with the linear PKF's matrix/vector calculations. For a typical nine-state filter, the operation count given earlier indicates that the number of operations in the standard Kalman filter is

additions:	$n \times n + 2n - 1 = 98$
multiplications:	$2n \times n + 4n + 1 = 199$
divisions:	1

when $n = 9$. The data storage requirement is on the order of 8 bytes x (98 + 199 + 1) = 2384 bytes of 2 Kbytes for 64-bit precision. If the linear two-processor PKF is used, it must be initialized by running the standard SKF for the first two tune steps. Then the dual processor PKF can be run at step 3 (i.e., at k = 3). Hence, the following values of x, ϕ, P, z, K, H and R must be stored in memory.

State Vector Values:	$\hat{x}_0(+)$, $\hat{x}_1(+)$, $\hat{x}_0(-)$, $\hat{x}_2(-)$
State Transition Matrix Values:	ϕ_0, ϕ_1, ϕ_2
Covariance Gain Values:	$P_0(+)$, $P_1(+)$, $P_1(-)$, $P_2(-)$
Kalman Gain Values:	K_1, K_2
Others:	H_1, H_2, R_1, R_2

Once the linear processor PKF is initialized, memory is needed to store the updated values of x, ϕ, P, z, H and R. Hence, in general, the overall memory requirement is:

Memory = (np +1) x (storage requirement of the standard Kalman filter)
where np = the number of parallel processing elements.

Thus, (np +1) x (2 Kbytes) are needed to store the data in the parallel filter. With np = 2, this corresponds to 6 Kbytes. With np = 48, this corresponds to approximately 98 Kbytes of RAM.

Note that the above memory sizing is for data only. The PKF program memory has not been sized. Because the two-processor PKF algorithm can be coded with less than 1000 lines of code and the compiled version of a 1000 line program requires about 128 Kbytes of RAM to store, a reasonable estimate of the storage requirement for the linear PKF program would be:

PKF Program Memory = np x 128 Kbytes

where np = the number of parallel processing elements. With 48 processors, the program memory is, therefore, estimated to be 48 x 128 Kbytes = 6 Mbytes. The four-processor PKF would need 2 x 6 Mbytes = 12 Mbytes of RAM.

Hense, a parallel processor with 6 Mbytes of bulk memory (i.e., relatively slow DRAM) and 98 bytes of fast RAM (i.e., cache memory) should be capable of implementing a 48 processor linear PKF.

Because nonlinear function evaluation generally results in more intermediate values than linear matrix/vector operations, the amount of local data storage might be increased by a factor of 2.5x. Hence, 2.5 x 98K = 256K of fast RAM is recommended for nonlinear extended parallel Kalman filter data storage. Twelve to sixteen Mbytes of program memory should be sufficient, however, for the nonlinear PKF.

3.2.3 Parallel Processor Selection

One method of estimating the computational requirements for the parallel Kalman filter is to total the number of additions, multiplications and divisions needed to complete one cycle of the Kalman filter algorithm. For example, the simple Kalman filter algorithm defined in Section 1 requires only 98 additions, 199 multiplications and 1 division per cycle for a nine-state filter. Hence, at 100 cycles per second (i.e., 100 Hz sample rate) the number of arithmetic operations is given by 100 x 298 = 29,800 operations per second. Ideally, a microprocessor capable of 33.1 usec per operation is all that is needed to implement a nine-state filter. Hence, a single Motorola 68020/68881 pair can easily handle the computational requirements of the Kalman filter assuming 100% efficiency. Note that 33.6 usec per pass through the filter corresponds to an update rate of 39,800 samples per second.

Typically, however, only 10 to 30% of peak performance is achieved in practice due to data bus and memory access time restrictions. Therefore, one target may be updated at a 4000 updates-per-second rate. For nonlinear filtering typical of target tracking problems, 64-bit precision and the need to compute trigonometric functions for coordinate transformations can slow computations down by one or perhaps two orders of magnitude (10x to 100x).

Since it is well known that Kalman filtering must be performed using floating-point arithmetic to avoid stability problems, the only viable method to gain back the throughput for nonlinear filtering problems using an extended Kalman filter is with parallel processing. Therefore, to rapidly implement the parallel Kalman filter with 32/64-bit floating-point precision, a parallel processing system of 48 processors needs a computation rate of 2.52 MFLOPs per processor to perform the necessary computations. Using four 25 MHz Motorola 68882 math coprocessors per board, 2.52 MFLOP performance is readily achievable. Hence, with twelve boards it is feasible to track one target in real time.

The general-purpose nature of the Motorola 68020/68882 processors is well suited for nonlinear, as well as linear, Kalman filtering. In particular because trigonometric functions (sin, cos, tan, etc.) and square-roots commonly occur in coordinate transformations associated with lead-angle prediction, high speed, general-purpose hardware (such as the Systolic-481 parallel numeric processor) is needed to handle the throughput requirements.

3.3 THE WSMR PARALLEL COMPUTING SOFTWARE ARCHITECTURE

The testbed software architecture can be divided into three major areas: (1) host-to-testbed communication software, (2) Master Processor embedded software and (3) Systolic-482 parallel numeric processor software. This section describes our system integration efforts related to implementing the PKF equations on the testbed.

3.3.1 Host-to-Testbed Communication Software

The testbed is a "compute engine" for rapidly evaluating nonlinear functions for the PKF. The testbed does not have its own compiler or linker. Software development for the testbed is performed on a "host-computer" which is familiar to the user. Since the testbed utilizes the 32-bit Motorola 68030

microprocessor, a Motorola development system or host such as a SUN 3/260 workstation or MacIntosh 11 (which both use the 68020) provides a direct path for software development for the testbed. If a VAX or IBM computer is used as a development system for the testbed, a cross compiler is needed to create native code for the 68020 in the testbed. Since Systolic Systems has a SUN 3 and MacIntosh II, these systems have been used for software development for the testbed.

Once created, an application such as the PKF code must be downloaded to the testbed, executed and the results uploaded back to the host. The performance of this procedure, commonly employed in the industry, depends on the data transfter link between the host/testbed. An RS-232C serial link using Motorola S-record format has been created and tested. A high-speed RS-422 link is being developed. These two approaches provide standard interfaces to the testbed and are quite useful, even if an ethernet link is developed between the host/testbed.

3.3.2 Testbed Master Processor Software

The real-time software in the master processor is responsible for overall system operation and managing the interactions with the host/user. From a parallel computing point of view, the master is a task scheduler. It schedules tasks (i.e., floating-pomt operations, subroutines) to execute on one or more Systolic-482 cards. As discussed in the next section, tasks tend to involve quad (four) simultaneous floating-point operations, such as ADD, SUBTRACT, MULTIPLY, DIVIDE, TRIG, SQUARE-ROOT. Also, linking of tasks into a string and routing these tasks to available 482 cards is the responsibility of the master processor.

3.3.3 Systolic-482 Parallel Numeric Processor Software

The assembly language software for the 482 involves self-test diagnostics, parallel floating-point operations, error detection/correction, and status information. All communications with the Master and other 482 cards is based on a "mailbox" structure. Messages are passed to the 482s mailbox from the master and read by the 482 to determine which tasks to execute on what data.

3.3.4 Self-Test Diagnostics and Exception Tests

The self-test diagnostics for the Systolic-482 include a memory test, LED status indicators, bus integrity test, floating-point integrity test and mail-box activity test. The 482 memory test includes a walking ones and zeroes test through all 256 Kbytes of its local memory. A mix of instructions with known answers is used to test the 68020 microprocessor. Light Emitting Diodes (LEDs) are used for status indicators. The LEDs light each time the 482 executes a major piece of software correctly indicating the 482 is healthy. Known data is also passed from EPROM to RAM and compared to verify that the 32-bit-wide data bus on the 482 is operationally sound. A known set of floating-point numbers is also used to verify that the four 68882 math coprocessors can accurately perform floating-point operations reliably. Additionally, known messages are passed through the 482's mailbox to ensure its integrity. Comprehensive diagnostic test routines were written to exercise all the above back-to-back and display the pass/fail results for several thousand passes. This diagnostic self-test software can be involved by the user to "check" the testbed at any time.

3.3.5 Floating-Point Operations

Although the 482 can operate in 32-bit, 64-bit or 80-bit extended precision, the floating-point test code was written for 64-bit floating-point operation. The 64-bit test numbers can be entered using the VFILL ("vector file") routine to allow the user to pick which numbers to test. Since the answer is known by the user, he can type the answer in and the software can subtract the numbers and display the error (if any). Thus, this test can be tailored by the user.

Floating-point operations on the 482 can be computed as singles (on one 68882 math coprocessor) or quads (on four 68882s simultaneously). Singles are useful for truly "scalar" processing. Quads are desireable for vector operations that are "chained together" four elements at a time. Long vectors (say greater than 32 elements) can be computed on a single 482 card or eight to twelve cards depending on task loading.

The quad (four) parallel floating-point primitives in the Systolic-482 include:

ADD, SUB, MULT, DIV, NOP, TAN, COS, SIN, SQRT, ..., etc.

3.3.6 Error Detection Code

The 482 has on-board error detection code for the following error conditions: bus error, address error, illegal instructions, division by zero, and floating-point coprocessor overflow/underflow detection. There error codes are accessible through the mailbox to the master in event of a fault condition. Hence, the master can take appropriate action (e.g., inform user by illuminating a special combination of LEDs).

3.3.7 Mailbox Error Codes

The 482's mailbox scheme has its own set of status conditions. These include: mailbox empty, mailbox not empty, no error present in mailbox, and mailbox unknown (i.e., confused). The status of the mailbox is available to the testbed's master and each 482 card in the testbed. The issue of multi-access to the mailbox, deadlock (i.e., system fault when multi-access is attempted) and task priority are topics still under consideration.

SECTION 4

PARALLEL KALMAN FILTER ALGORITHM VALIDATION

To show the effectiveness/payoff of the parallel Kalman filter (PKF) testbed research, it is important to consider a meaningful target tracking problem. The test problem should be representative of typical missile applications and serve as a baseline to measure the benefits/accuracy of the PKF technology. With this in mind, two test problems are considered. The first problem (test case #1) is very simple, utilizes time-varying parameters, and illustrates the speed/accuracy of the PKF method. Test case #2 is more realistic and more challenging since it involves nonlinearities and exponentials.

4.1 TEST CASE #1

Test case #1 is based on the following state and measurement model:

$$x(k+1) = \phi(k+1,k)\, x(k) + \Gamma(k+1,k)\, w(k) \tag{4.1}$$

$$z(k+1) = H(k+1)\, x(k+1) + v(k+1) \tag{4.2}$$

where x is the state of the target, ϕ is the state transition matrix, z are measurements and w and v are noise terms.

For the purpose of test case #1, the model parameters were selected as follows:

$$\phi(k+1,k) = \exp(-k*0.001), \quad \Gamma(k+1,k) = 1, \quad H(k+1) = 1 \tag{4.3}$$

The noise terms

$$w(k) \sim N(0,Q(k)) \quad \text{and} \quad v(k+1) \sim N(0,R(k+1)) \tag{4.4}$$

had zero mean and covariance $Q(k)$ and $R(k+1)$, respectively.

In this case, the PKF equations are given by:

```
procedure new PKF (i:integer);
begin

  H [i+1]: = 1;
  A [i+1,i]: = exp (-i*0.001);
  A [i+2,i+1]: = exp (-(i+1)*0.001);
  B [i+1,i]: = 1;
  B [i+2,i+1]: = 1;

     { predictor }
  X [i+2,i+1]: = A [i+2,i+1] * A [i+1,i] * X [i,i];
  P [i+2,i+1]: = A [i+2,i+1] * A [i+1,i] * P [i,i] * A [i+1,i] * A [i+2,i+1]
                 + A [i+2,i+1] * B [i+1,i] * Q [i] * B [i+1,i] * A [i+2,i+1]
                 + B [i+2,i+1] * Q [i+1] * B [i+2,i+1];

     { Kalman gain }
  K [i+1]:      = (P [i+1,i] * H [i+1]) / ((H [i+1] * P [i+1,i] * H [i+1]) + R
[i+1]);

     { corrector }
  X [i+1,i+1]: = X [i+1,i] + K [i+1] * (Z [i+1] - (H [i+1] * X [i+1,i]));
  P [i+1,i+1]: = (1 - (K [i+1] * H [i+1])) * P [i+1,i];
end;  { new PKF }
```

Similarly, the SKF equations, when coded, appear as follows:

```
procedure SKF (i:  integer);
begin

  A [i+1,i]:    = exp (-i*0.001);
  B [i+1,i]:    = 1;
  H [i+1]   :   = 1
  X [i+1,i]:    = A [i+1,i] * X [ i,i];
  P [i+1,i]:    = A [i+1,i] * P [i,i] * A [i+1,i] + B [i+1,i] * Q [i] *
B [i+1,i];

  K [i+1]:      = P (- [i+1,i]* H [i+1])/ ((H[i+1] * P  [i+1,i] * H
[i+1])+ R [i+1]);

  X [i+1,i+1]:= X [i+1,i] + K [i+1] * (Z [i+1] - (H [i+1] * A [i+1,i] *
X [i+1,i]));
  P [i+1,i+1]:= (1 - (K [i+1] * H [i+1])) * P [i+1,i];
end;  { SKF }
```

The results for this time-varying case are shown in Table 4-1. The results indicate that the PKF equations are well behaved and exhibit similar convergence characteristics as the SKF. The optimal solution can be obtained via one pass through the SKF equations once the PKF converges.

Since the computer simulations on these test casees were encouraging, the four-processor PKF equations were coded (see below) with similar results.

TABLE 4-1 COMPARISON OF THE SKF AND PKF ESTIMATES OF AN EXPONENTIALLY DECAYING TARGET TRAJECTORY

Sequential Kalman Filter

P[0,0]:	10.000000	X[0,0]:	1.000000	Z[0]:	1.000000
P[1,1]:	0.099020	X[1,1]:	1.000001	Z[1]:	1.000001
P[2,2]:	0.066535	X[2,2]:	0.999666	Z[2]:	0.999002
P[3,3]:	0.062444	X[3,3]:	0.998501	Z[3]:	0.997007
P[4,4]:	0.061842	X[4,4]:	0.996434	Z[4]:	0.994022
P[5,5]:	0.061737	X[5,5]:	0.993419	Z[5]:	0.990055
P[6,6]:	0.061704	X[6,6]:	0.989442	Z[6]:	0.985118
P[7,7]:	0.061681	X[7,7]:	0.984501	Z[7]:	0.979226
P[8,8]:	0.061660	X[8,8]:	0.978609	Z[8]:	0.972396
P[9,9]:	0.061639	X[9,9]:	0.971781	Z[9]:	0.964649
P[10,10]:	0.061618	X[10,10]:	0.964037	Z[10]:	0.956007
P[11,11]:	0.061597	X[11,11]:	0.955398	Z[11]:	0.946496
P[12,12]:	0.061576	X[12,12]:	0.945891	Z[12]:	0.936142
P[13,13]:	0.061556	X[13,13]:	0.935542	Z[13]:	0.924977
P[14,14]:	0.061535	X[14,14]:	0.924381	Z[14]:	0.913031
P[15,15]:	0.061514	X[15,15]:	0.912441	Z[15]:	0.900339
P[16,16]:	0.061493	X[16,16]:	0.899755	Z[16]:	0.986935
P[17,17]:	0.061473	X[17,17]:	0.886358	Z[17]:	0.872858
P[18,18]:	0.061452	X[18,18]:	0.872289	Z[18]:	0.858146
P[19,19]:	0.061432	X[19,19]:	0.857584	Z[19]:	0.842839
P[20,20]:	0.061411	X[20,20]:	0.842285	Z[20]:	0.826977
P[21,21]:	0.061390	X[21,21]:	0.826432	Z[21]:	0.810603
P[22,22]:	0.061370	X[22,22]:	0.810067	Z[22]:	0.793759
P[23,23]:	0.061349	X[23,23]:	0.793232	Z[23]:	0.776488
P[24,24]:	0.061329	X[24,24]:	0.775971	Z[24]:	0.758833
P[25,25]:	0.061308	X[25,25]:	0.758326	Z[25]:	0.740839
P[26,26]:	0.061288	X[26,26]:	0.740342	Z[26]:	0.722549
P[27,27]:	0.061268	X[27,27]:	0.722063	Z[27]:	0.704005
P[28,28]:	0.061247	X[28,28]:	0.703530	Z[28]:	0.685253
P[29,29]:	0.061227	X[29,29]:	0.684789	Z[29]:	0.666333
P[30,30]:	0.061207	X[30,30]:	0.665880	Z[30]:	0.647287

Parallel Kalman Filter

[0,0]:	10.000000	X[0,0]:	1.000000	Z[0]:	1.000000
P[1,1]:	0.099010	X[1,1]:	1.000002	Z[1]:	1.000002
P[2,2]:	0.099027	X[2,2]:	0.999003	Z[2]:	0.999003
P[3,3]:	0.074876	X[3,3]:	0.997008	Z[3]:	0.997008
P[4,4]:	0.074839	X[4,4]:	0.994022	Z[4]:	0.994023
P[5,5]:	0.073193	X[5,5]:	0.990055	Z[5]:	0.990056
P[6,6]:	0.073155	X[6,6]:	0.985118	Z[6]:	0.985119
P[7,7]:	0.073003	X[7,7]:	0.979226	Z[7]:	0.979227
P[8,8]:	0.072965	X[8,8]:	0.972396	Z[8]:	0.972397
P[9,9]:	0.072919	X[9,9]:	0.964649	Z[9]:	0.964650
P[10,10]:	0.072881	X[10,10]:	0.956007	Z[10]:	0.956008
P[11,11]:	0.072842	X[11,11]:	0.946496	Z[11]:	0.956497
P[12,12]:	0.072804	X[12,12]:	0.936143	Z[12]:	0.936143
P[13,13]:	0.072767	X[13,13]:	0.924977	Z[13]:	0.924978
P[14,14]:	0.072729	X[14,14]:	0.913031	Z[14]:	0.913032
P[15,15]:	0.072691	X[15,15]:	0.900339	Z[15]:	0.900340
P[16,16]:	0.072653	X[16,16]:	0.886935	Z[16]:	0.986935
P[17,17]:	0.072616	X[17,17]:	0.872858	Z[17]:	0.872859
P[18,18]:	0.072578	X[18,18]:	0.858146	Z[18]:	0.858147
P[19,19]:	0.072540	X[19,19]:	0.842839	Z[19]:	0.842840
P[20,20]:	0.072503	X[20,20]:	0.826977	Z[20]:	0.826978
P[21,21]:	0.072465	X[21,21]:	0.810603	Z[21]:	0.810504
P[22,22]:	0.072428	X[22,22]:	0.793759	Z[22]:	0.793760
P[23,23]:	0.072390	X[23,23]:	0.775488	Z[23]:	0.776489
P[24,24]:	0.072353	X[24,24]:	0.758833	Z[24]:	0.758834
P[25,25]:	0.072315	X[25,25]:	0.740839	Z[25]:	0.740840
P[26,26]:	0.072278	X[26,26]:	0.722549	Z[26]:	0.722550
P[27,27]:	0.072240	X[27,27]:	0.704006	Z[27]:	0.704006
P[28,28]:	0.072203	X[28,28]:	0.685253	Z[28]:	0.685254
P[29,29]:	0.072166	X[29,29]:	0.666333	Z[29]:	0.666334
P[30,30]:	0.072128	X[30,30]:	0.647288	Z[30]:	0.647289

4.2 TEST CASE #2

Consider the problem of estimating the position of an object (missile) from angle-only (or range) measuremements. The geometry of this target tracking application is illustrated in Figure 4-1. Typical parameter values for this exercise are given in Table 4-2.

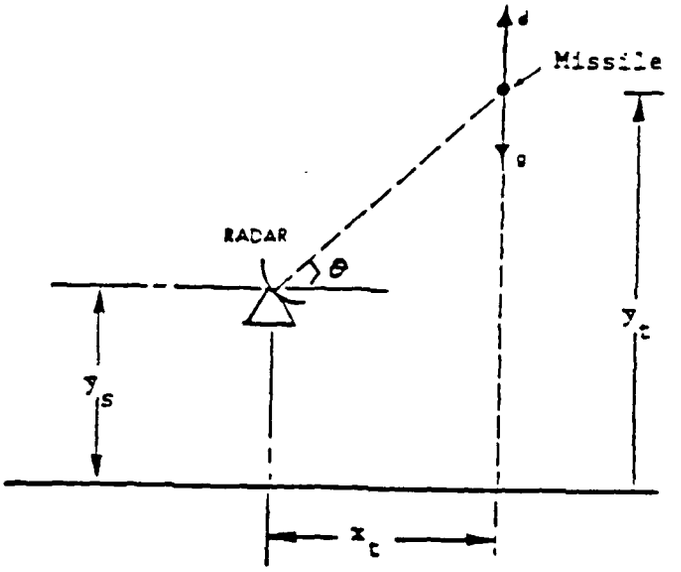


Figure 4-1 Geometry of Test Case #1

In Figure 4-1, x_t, y_t represent the target position in X-Y coordinates and y_s represents the sensor position.

The target range is given by

$$z_t = (x_t^2 + (y_t - y_s)^2)^{\frac{1}{2}} \tag{4.5}$$

With angle-only measurements the line-of-sight angle, θ, can be estimated as follows:

$$\theta = \tan^{-1}\left(\frac{y_t - y_s}{x_t}\right) \tag{4.6}$$

TABLE 4-2

PARAMETER VALUES FOR TEST CASE #1

$\rho_o = 3.4 \times 10^{-3}$ lb sec^2/ft^4

$g = 32.2$ ft/sec^2

$k_\rho = 22{,}000$ ft

$\beta \sim N(2000 \text{ lb/ft}^2,\ 2.5 \times 10^5 \text{ lb}^2/\text{ft}^4)$

$\frac{\rho_o g}{2} = 0.05$ lb/ft^3

$p_{11_o} = 500$ ft^2

$p_{22_o} = 2 \times 10^4$ ft^2/sec^2

$p_{33_o} = 2.5 \times 10^5$ lb^2/ft^4

$p_{44_o} = 50$ ft^2

$p_{55_o} = 50$ ft^2

$x_1(0) = 3 \times 10^5$ ft

$x_2(0) = 2 \times 10^4$ ft

$x_3(0) = 1/2 \times 10^{-2}$ ft^2/lb

$x_4(0) = 5 \times 10^3$ ft

$x_5(0) = 2 \times 10^3$ ft

The target's motion is modeled as a falling body in state variable form as:

$$x_1 = y_t, \quad x_2 = -y_t, \quad x_3 = 1/\beta, \quad \dot{x}_3 = 0, \quad x_4 = x_t,$$

$$\dot{x}_4 = 0, \quad x_5 = y_s, \quad \dot{x}_5 = 0 \tag{4.7}$$

where β is the so-called ballistic coefficient of the target and y_t is the target's height above the earth.

The equations of motion for the target are:

$$\underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{bmatrix}}_{\dot{\underline{x}}} = \underbrace{\begin{bmatrix} -x_2 \\ g - d \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\underline{f}(x)} \qquad d = \frac{\rho x_2^2}{2x_3} \tag{4.8}$$

$$\rho = \rho_o e^{-x_1/k_\rho} \tag{4.9}$$

where d is drage deceleration, g is acceleration of gravity, ρ is atmospheric density (with ρ_o the atmospheric density at sea level) and k_ρ is a decay constant. The differential equation of velocity, x_2, is nonlinear through the dependence of drag on velocity, air density and the ballistic coefficient, β.

Initial values of the state variables are assumed to have covariance matrix of the from

$$P_o = \text{diag}\,(P_{11_o}, P_{22_o}, P_{33_o}, P_{44_o}, P_{55_o}) \tag{4.10}$$

Estimating all the state variables may be solved using the following extended Kalman filter:

Predictor:

$$\dot{\hat{x}}(t) = f(\hat{x}(t)) \qquad \hat{x}(t_0) = x_0 \tag{4.11}$$

$$\dot{P}(t) = F(\hat{x}(t))P(t) + P(t)F^T(\hat{x}(t)) + Q(t), \qquad P(t_0) = P_0 \tag{4.12}$$

Corrector:

$$\hat{x}_k(+) = \hat{x}(-) + K_k(z_k - h_k(\hat{x}_k(-))) \tag{4.13}$$

$$P_k(+) = (I - K_k H_k(\hat{x}_k(-)))P_k(-) \tag{4.14}$$

$$K_k = P_k(-)H_k^T(\hat{x}_k(-)) * (H_k(\hat{x}_k(-))P_k(-)\ H_k^T(\hat{x}_k(-)) + R_k)^{-1} \tag{4.15}$$

where

$$F(\hat{x}(t)) = \left.\frac{\partial f(x(t))}{\partial x(t)}\right|_{x(t) = \hat{x}(t)}$$

$$H_k(\hat{x}(-)) = \left.\frac{\partial h(x(t_k))}{\partial x(t_k)}\right|_{x(t_k) = \hat{x}(-)}$$

The nonlinear differential equations were integrated using the trapezoidal rule. The partial derivatives were computed using a finite difference method. The model uncertainty matrix, Q, and the measurement noise covariance, R, were given by:

$$Q = \text{diag}\ (0.2,\ 0.2,\ 0.2,\ 0.2,\ 0.2) \tag{4.16}$$

$$R = \text{diag}\ (0.2,\ 0.2) \tag{4.17}$$

4.3 EVALUATING THE PERFORMANCE OF THE PARALLEL KALMAN FILTER BASED ON THE PARALLEL TRAPEZOIDAL RULE

After careful inspection of the nonlinear extended parallel Kalman filter (EPKF) formulation, it is clear that the performance of the filter is closely related to the accuracy and speed of the numerical method used for propagating the state and covariance time updates. The trapezoidal rule discussed in the previous section is accurate to $0(h^2)$. For $h = 0.01$ four-digit accuracy is attainable. With $h = 0.001$, six-digit accuracy results. Automatic step size (h) selection based on a prespecified accuracy requirement (e.g., norm of the local truncation less than 0.000001) can be achieved using a variable step size integration scheme. Time to solution can also be used a a criteria for selecting h. Since it is anticipated that computation time is key, the parallel methods are evaluated with a relatively large step size ($h = 0.01$ for each trapezoidal rule being evaluated).

4.3.1 Parallel Solution of Test Case #1

The target's equations of motion were used to demonstrate the numerical properties of the parallel trapezoidal rule. The initial conditions and physical problem parameters were described. The results of propagating the target altitude (x1) and target velocity (x2) with the parallel trapezoidal methods are shown in Tables 4-3 and 4-4.

The results in Tables 4-3 and 4-4 are based on a fixed step size of h = 0.01. The data indicates that the absolute error between the sequential trapezoidal rule and the parallel trapezoidal rule is relatively small (0.1%). However, the absolute error for the target velocity is growing with time. Although at first glance this may appear unacceptable, a more detailed examination indicates that the change in the percent error normalized by the change in the variable being integrated is only 1.15×10^{-7}% per ft for the target altitude and 4.4×10^{-5}% per ft/sec for the target velocity. Since the flight time for this application is only 15 to 20 seconds, the maximum absolute error on impact for the target altitude is 0.11% and for the target velocity 0.55%. If not to improve accuracy, a smaller step size (say h = 0.001) is worth evaluating.

The results in Tables 4-5 and 4-6 are based on a fixed step size of h = 0.001. Note that in each case the parallel trapezoidal methods are as accurate as the standard sequential trapezoidal rule. Note that the worst-case maximum error is much less than 0.01%. Thus, with a small step size, the parallel methods are equally good with the sequential methods only 400% faster to compute per state equation. A trade off between speed and ultimate accuracy must be made.

4.3.2 Summary

The results of this section indicate that as more processing elements are used to speed up the PKF state (and covariance) propagation, the accuracy of the parallel solution can degrade compared with satndard sequential methods. Thus, because the EPKF utilizes the parallel trapezoidal rule to integrate the state and covariance equations, the EPKF solution should be less accurate compared with the EKF. The execution speed of the EPKF should improve linearly with the number of processors.

Table 4-3 State Equation Integration Comparing the Trapezoidal Rule and the 2&4 Processor Parallel Trapezoidal Rule (h=0.01)

	Trapezoidal Rule	Parallel Trapezoidal Rule		
	1 Processor	2 Processors	4 Processors	
Time (sec)	Target Altitude (ft)	Target Altitude (ft)	Target Altitude (ft)	Absolute Maximum Error
0.500	290001.923	290001.923	290001.923	0.07%
1.000	280012.071	280012.070	280211.725	0.07%
1.500	270039.727	270039.726	270238.905	0.07%
2.000	260099.339	260099.339	260297.685	0.07%
2.500	250213.191	250213.192	250410.158	0.08%
3.000	240415.165	240415.165	240609.927	0.08%
3.500	230755.644	230755.636	230946.984	0.08%
4.000	221307.016	221306.981	221493.216	0.08%
4.500	212168.020	212167.913	212346.788	0.08%
5.000	203463.303	203463.031	203631.846	0.08%

Table 4-4 State Equation Integration Comparing the Trapezoidal Rule and the 2&4 Processor Parallel Trapezoidal Rule (h=0.01)

	Trapezoidal Rule	Parallel Trapezoidal Rule		
	1 Processor	2 Processors	4 Processors	
Time (sec)	Target Velocity (ft/sec)	Target Velocity (ft/sec)	Target Velocity (ft/sec)	Absolute Maximum Error
0.500	19990.367	19990.369	19990.690	0.0016%
1.000	19966.004	19966.006	19966.692	0.0034%
1.500	19918.640	19918.642	19919.896	0.0063%
2.000	19835.603	19835.606	19837.737	0.0107%
2.500	19697.851	19697.859	19701.325	0.0176%
3.000	19477.635	19477.654	19483.112	0.0281%
3.500	19136.462	19136.509	19144.840	0.0437%
4.000	18625.119	18625.229	18637.499	0.0665%
4.500	17888.965	17889.210	17906.486	0.0979%
5.000	16882.258	16882.759	16905.719	0.1389%

Table 4-5 State Equation Integration Comparing the Trapezoidal Rule and the 2&4 Processor Parallel Trapezoidal Rule (h=0.001)

	Trapezoidal Rule	Parallel Trapezoidal Rule	
	1 Processor	2 Processors	4 Processors
Time (sec)	Target Altitude (ft)	Target Altitude (ft)	Target Altitude (ft)
0.500	290001.924	290001.924	290021.914
1.000	280012.072	280012.072	280032.038
1.500	270039.728	270039.728	270059.647
2.000	260099.341	260099.341	260119.176
2.500	250213.192	250213.192	250232.890
3.000	240415.163	240415.163	240434.641
3.500	230755.630	230755.630	230774.766
4.000	221306.970	221306.970	221325.595
4.500	212167.895	212167.895	212185.784
5.000	203463.006	203463.006	203479.889

Table 4-6 State Equation Integration Comparing the Trapezoidal Rule and the 2&4 Processor Parallel Trapezoidal (h=0.001)

	Trapezoidal Rule	Parallel Trapezoidal Rule	
	1 Processor	2 Processors	4 Processors
Time (sec)	Target Velocity (ft/sec)	Target Velocity (ft/sec)	Target Velocity (ft/sec)
0.500	19990.367	19990.367	19990.399
1.000	19966.005	19966.005	19966.073
1.500	19918.641	19918.642	19918.767
2.000	19835.606	19835.606	19835.819
2.500	19697.860	19697.860	19698.207
3.000	19477.656	19477.656	19478.203
3.500	19136.512	19136.512	19137.347
4.000	18625.234	18625.234	18626.463
4.500	17889.215	17889.215	17890.946
5.000	16882.764	16882.764	16885.064

Table 4-7 2 Processor Parallel Trapezoidal Rule

```
double x1p[5100] , x2p[5100] , x1c[5100]
, x2c[5100] , rhop[5100] , rhoc[5100] ;
double rho, beta, x3, g, x4, x5, d, h ;
double x1 , x2 , drag , x ,krho ,rho0 , stime ;
int i , nstep , nprint ;

# include(math.h)
# include(stdio.h)

main()
{

FILE *tout1 ;
tout1 = fopen( "tdemo1.out" , "w" ) ;


/*** x1p denotes x1 pred. x1c denotes x1 corrector ***/

x1p[0] = x1 ; x1c[0] = x1 ; x2c[0] = x2 ; x2p[0] = x2 ;
rhop[0] = rho0*exp( -x1p[0]/krho) ; rhoc[0] = rho0*exp( -x1c[0]/krho ) ;
x1p[1] = x1 ; rhop[1] = rho0*exp( -x1p[1]/krho ) ;
x2p[1] = x2 ;

for( i =1 ; i<=nstep ; ++i ){

/*** predictor equations ***/

rhop[i] = rho0*exp( -x1p[i]/krho ) ;
rhoc[i-1] = rho0*exp( -x1c[i-1]/krho ) ;
drag = rhop[i]*x2p[i]*x2p[i]/2/x3 ;

x1p[i+1] = x1c[i-1] + 2.*h*( -x2p[i] ) ;
x2p[i+1] = x2c[i-1] + 2.*h*( g - rhop[i]*x2p[i]*x2p[i]/2/x3 ) ;

/*** corrector equations ***/

x1c[i] = x1c[i-1] + .5*h*( -x2p[i] - x2c[i-1] ) ;
x2c[i] = x2c[i-1] + h*g - (h/4.)*( rhop[i]*x2p[i]*x2p[i]/x3
  + rhoc[i-1]*x2c[i-1]*x2c[i-1]/x3 ) ;



stime = i*h ;

nprint = nstep/10 ;

if( (i/nprint)*nprint == i ){
fprintf(tout1, "\n\n time = %7.3f sec x1 = %7.3f ft x2 = %7.3f ft/sec "
,stime , x1c[i] , x2c[i] ) ;

printf("\n\n time = %7.3f sec x1 = %7.3f ft x2 = %7.3f ft/sec "
, stime , x1c[i] , x2c[i] ) ;
}


}
fclose(tout1) ;

}
```

Table 4-8 4 Processor Parallel Trapezoidal Rule

```
  double rho,beta,x3,g,x4,x5,c,h ;
  double x1 , x2 , drag ;
  double x ,  krho ,  rho0 , stime ;
  double x1p[5100] , x2p[5100] , x1c[5100]
  , x2c[5100] , rhop[5100] , rhoc[5100] ;

  int i , l , m , nstep , nprint ;

#include(math.h)
#include(stdio.h)

main()
{


FILE *tout2 ;
tout2 = fopen( "tdemo2.out" , "w" ) ;


  /*** x1p denotes x1 pred. x1c denotes x1 corrector ***/

  /*** initial conditions need to be specified are:
  x1p[ 0 - 4 ]  x2p[ 0 - 4 ]  x1c[ 0 - 2 ]  x2c[ 0 - 2 ]  ***/

  x1p[0] = x1 ; x1c[0] = x1 ; x2c[0] = x2 ; x2p[0] = x2 ; x2p[1] = x2 ;
  x1p[1] = x1 ; x1p[2] = x1 ; x2p[2] = x2 ; x2p[2] = x2 ;
  x1p[3] = x1 ; x1p[4] = x1 ; x2p[3] = x2 ; x2p[4] = x2 ;
  x1c[1] = x1 ; x1c[2] = x1 ; x2c[1] = x2 ; x2c[2] = x2 ;
  rhop[0] = rho0*exp( -x1p[0]/krho) ; rhoc[0] = rho0*exp( -x1c[0]/krho ) ;
  rhop[1] = rho0*exp( -x1p[1]/krho) ; rhoc[1] = rho0*exp( -x1c[1]/krho ) ;
  rhop[2] = rho0*exp( -x1p[2]/krho) ; rhoc[2] = rho0*exp( -x1c[2]/krho ) ;
  rhop[3] = rho0*exp( -x1p[3]/krho) ;
  rhop[4] = rho0*exp( -x1p[4]/krho) ;
  rhop[2] = rho0*exp( -x1p[2]/krho ) ;
  rhoc[1] = rho0*exp( -x1c[1]/krho ) ;

  /*** run only predictor equations for i = 1 to obtain better initial
  estimate for x1p[3] , x1p[4] , x2p[3] , x2p[4] ***/

  x1p[4] = x1c[0] + 4.*h*( - x2p[2] ) ;
  x2p[4] = x2c[0] + 4.*h*( g - rhop[2]*x2p[2]*x2p[2]/2/x3 ) ;

  x1p[3] = x1c[0] + 1.5*h*( -x2p[2] - x2p[1] ) ;
  x2p[3] = x2c[0] + 1.5*h*(  g
  -rhop[2]*x2p[2]*x2p[2]/2/x3 + g -
   rhop[1]*x2p[1]*x2p[1]/2/x3 ) ;

  rhop[3] = rho0*exp(-x1p[3]/krho ) ;
  rhop[4] = rho0*exp(-x1p[4]/krho ) ;


  for( i = 2 ; i<=nstep ; ++i ){
```

Table 4-8 4 Processor Parallel Trapezoidal Rule (continued)

```
/*** predictor equations ***/

rhop[2*i] = rho0*exp( -x1p[2*i]/krho ) ;
rhop[2*i-1] = rho0*exp( -x1p[2*i-1]/krho ) ;
rhoc[2*i-2] = rho0*exp( -x1c[2*i-2]/krho ) ;
rhoc[2*i-3] = rho0*exp( -x1c[2*i-3]/krho ) ;


x1p[2*i+2] = x1c[2*i-2] + 4.*h*( -x2p[2*i] ) ;
x2p[2*i+2] = x2c[2*i-2] + 4.*h*( g - rhop[2*i]*x2p[2*i]*x2p[2*i]/2/x3 ) ;

x1p[2*i+1] = x1c[2*i-2] + 1.5*h*( -x2p[2*i] - x2p[2*i-1]) ;
x2p[2*i+1] = x2p[2*i-2] + 1.5*h*(
g - rhop[2*i]*x2p[2*i]*x2p[2*i]/2/x3 +
g - rhop[2*i-1]*x2p[2*i-1]*x2p[2*i-1]/2/x3 ) ;


/*** corrector equations ***/

x1c[2*i] = x1c[2*i-3] + .5*h*( 3.*x2p[2*i] - 9*x2p[2*i-1] ) ;
x2c[2*i] = x2c[2*i-3] - .5*h*( 3.*( g - rhop[2*i]*
x2p[2*i]*x2p[2*i]/2./x3) - 9.*( g - rhop[2*i-1]*x2p[2*i-1]*
x2p[2*i-1]/2./x3 )) ;

x1c[2*i-1] = x1c[2*i-3] - 2.*h*x2c[2*i-2] ;
x2c[2*i-1] = x2c[2*i-3] + 2.*h*( g -
rhoc[2*i-2]*x2c[2*i-2]*x2c[2*i-2]/2./x3 ) ;
}

for( l = 1 ; l <= 10 ; ++l ){ m = nprint*l ;
stime = m*h ;
fprintf(tout2 ,"\n\n time = %7.3f sec x1c = %7.3f ft x2c = %7.3f ft/sec "
, stime , x1c[m] , x2c[m] ) ;

printf("\n\n time = %7.3f sec x1c = %7.3f ft x2c = %7.3f ft/sec "
, stime , x1c[m] , x2c[m] ) ;

}
fclose(tout2) ;

}
```

Target Height Estimation

Figure 4-2 shows the noisy target height measurements and the EKF estimate of the target's height above the earth. Note that the EKF estimates filter (or reduce) the measurement noise considerably. Also, note that the EKF tracks the data quite well as the target descends toward earth.

Target Velocity Estimation

The target's velocity is illustrated in Figure 4-3. Note that the EKF converges to the actual velocity within four seconds. The target's velocity is essentially constant at this altitude making target prediction easier once the EKF has converged. This knowledge can be built into the knowledge base for the expert system that manages the EKF.

Line-of-Sight (LOS) Angle Estimation

Figure 4-4 shows the LOS angle estimate from the EKF for test case #1. The LOS angle is relatively steep (87° on average) and decreasing as the target height declines. This follows because the observer is stationary while the target is moving. Because the tartet is 250,000 feet above sea level, it follows that the LOS angle be large (approaching 90°). Based on the change in the LOS angle estimate per unit time ($2^{\circ}/4$ seconds = 0.5° per second), it would take 176 seconds to reduce the LOS angle to zero.

Target Range Estimation

The target range estimate from the EKF is given in Figure 4-5. Note that the target is getting closer to the observer at a rate of 250,000 ft/sec. Hence, within ten seconds the target will hit the observer unless some action is taken. Thus, the EKF must be capable of rapidly updating the range estimates (in under one second) to be effective for WSMR applications.

Summary

This test case illustrates the class of computations required for WSMR target tracking. Squares, divides, square roots, exponentials and

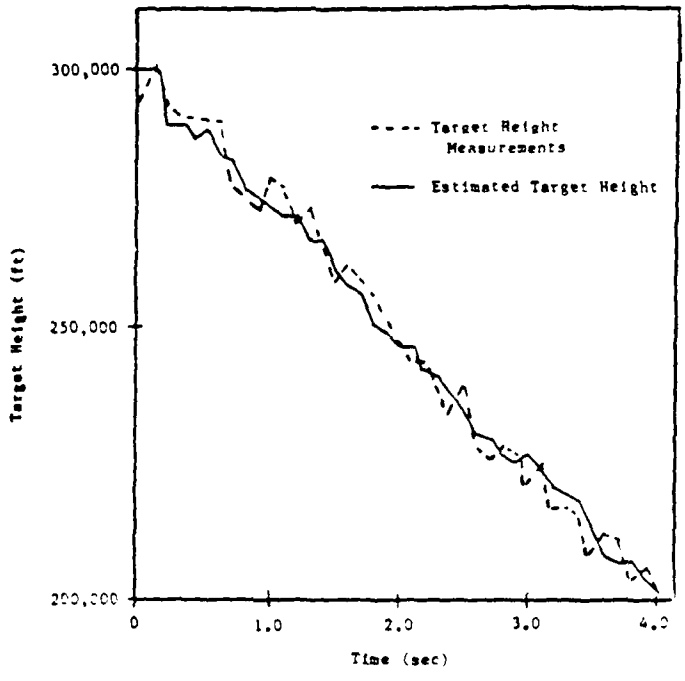


Figure 4-2 Target Measurements and Estimated Target Height Above the Earth From EKF

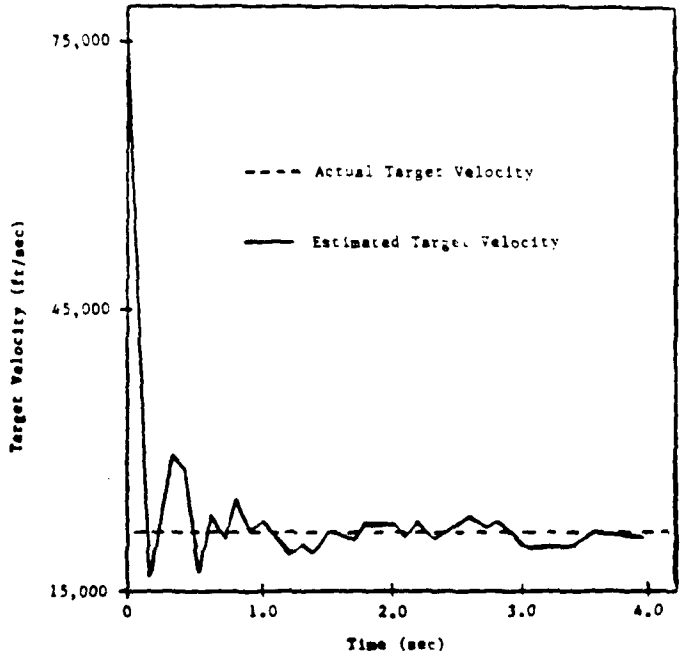


Figure 4-3 **Actual and Estimated Target Velocity From EKF**

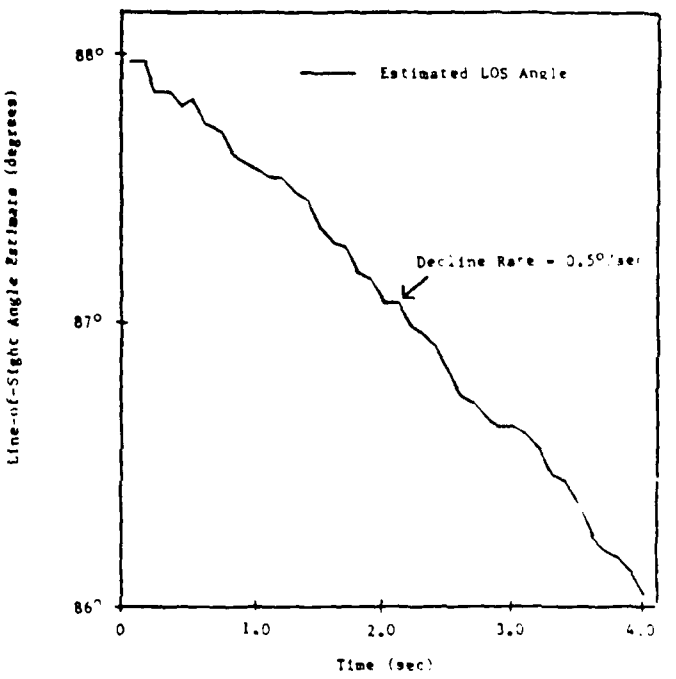


Figure 4-4 Estimated Line-of-Sight (LOS) Angle From EKF

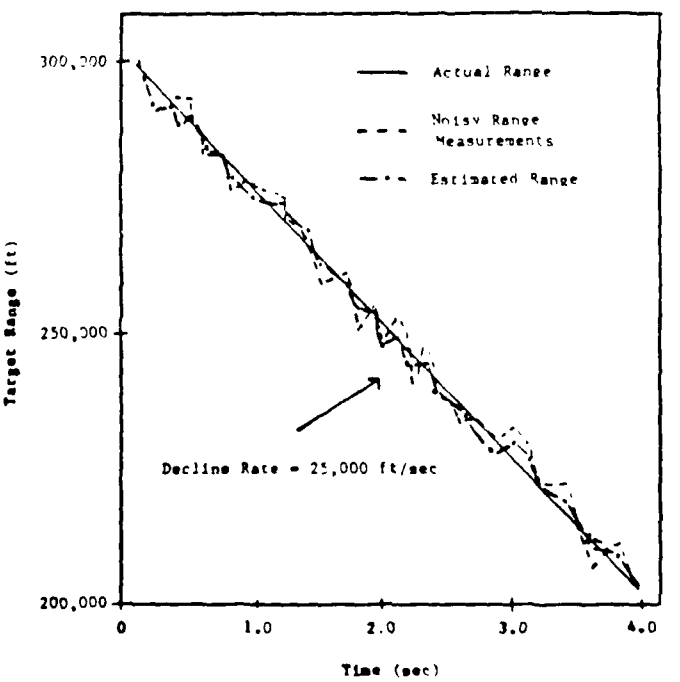


Figure 4-5 **Estimated Target Range From EKF**

trigonometric functions are needed. In addition, the extended Kalman filter requires the solution of nonlinear ordinary differential equations. Thus, high-speed nonlinear function evaluation is required to solve target tracking problems on a timely basis. This test case, although simple, can be solved relatively easily to provide a known solution to verify the parallel Kalman filter algorithms and architectures. This test case can be expanded to three dimensions using angle-only measurements of the target. In this case, the target tracking problem becomes more nonlinear and involves additional trig functions to be computed during coordinate transformations.

SECTION 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

Based on the results of our Phase I study, the following conclusions can be drawsn:

- It is technically feasible to decouple the predictor and corrector equations in a standard Kalman filter for parallel processing on multiple processors.
- The decoupling principle allows the parallel Kalman filter's predictor and corrector equations to be computed on separate processors improving computational speed directly proportional to the number of available processing elements.
- It is feasible to extend the linear PKF theory to nonlinear target tracking and estimation problems allowing an extended Kalman filter to run on multiple parallel processors.

Because both linear and nonlinear filtering can benefit from the decoupling principle, this research activity appears well suited for transition to the Phase II stage of the SBIR program.

5.2 RECOMMENDATIONS

Based on the conclusions derived from our Phase I results, the following recommendations are presented:

- Continue coding and evaluating the PKF algorithms on a parallel computer whose architecture can be reconfigured to validate newly developed target tracking algorithms and architectures. Many issues regarding the implementation of the parallel Kalman filter can be learned by coding the PKF algorithms and architectures. For example, timing, synchronization, drift, potential divergence of the error covariance update, and model sensitivities could have a major impact on the ultimate application of the PKF. Hence, it is recommended that an expert system be developed to manage PKF computations.

- o Create a parallel computing testbed facility (based on industry-standard hardware and software). A flexible/reconfigurable parallel computing testbed is recommended to rapidly test and evaluate the performance of newly developed parallel processing algorithms and architectures. Because WSMR target tracking applications tend to be nonlinear, a scalable architecture (i.e., expandable based on problem size) for nonlinear function evaluation is recommended. General-purpose microprocessor/coprocessor technology augmented by a 64-bit floating-point 25 MFLOP array processor board set is recommended to accommodate a wide-class of parallel algorithms. Four processors per card are recommended to simultaneously compute the equations in the decoupled PKF (i.e., two processors for the predictor and two processors for the corrector per card). Multiple cards (say twelve) can be installed in the testbed to provide 25 MFLOPs of 64-bit nonlinear function evaluation power. An industry-standard Vme bus is also recommended for several reasons: (1) Vme is a high-performance bus, (2) Vme is supported by several major companies allowing the government to add "special function" cards to the system, and (3) Vme is also standard in high-rel, milspec and ruggedized systems for actual field test of our PKF technology.

- o Use actual flight test data to show the benefits of the PKF technology. Because of the complexity of realistic target tracking applications, it is anticipated that even today's supercomputer architectures will not be capable of solving these problems in real time. Due to the unique matching of the PKF algorithms and architectures, it is anticipated that problems that could not be s lved otherwise in a reasonable time (at a reasonable cost) can be solved on the proposed testbed. Thus, it is recommended that a target tracking problem based on actual flight test data be solved and benchmark performance documented so that future designs can be compared. Due to the "special" architecture of the recommended parallel computing system, it is anticipated that it can be the standard to improve upon for the next five years.

5.3 SUMMARY

Based on the results in this report, it is clear that the PKF theory is well developed, mature and ready to proceed to full-scale validation on actual flight test data on a parallel processing testbed. Because the PKF technology has been needed to solve several applications in the DoD for more than a decade, it is anticipated that once fully developed this technology can benefit several sectors of the DoD. This is possible because the necessary technology (e.g., 25 MFLOP 64-bit floating-point adders/multipliers/dividers and 25 MHz 68030/68882 general-purpose microprocessors) has only recently been available to transition the PKF theory into practice. Hence, Systolic Systems would be pleased to continue this program under Phase II of the SBIR program.