Parallel Implementation of Multiple Model Tracking Algorithms

appeared in IEEE Trans. on Parallel and Distributed Systems, Vol. 2, No. 2, pp. 242-252, April 1991

A. Averbuch
*S. Itzikowitz**
T. Kapon
The Moise and Frida Eskenasy Institute of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University
Tel Aviv 69978 , Israel
E-MAIL : amir@math.tau.ac.il

Abstract

Multifilter algorithms are usually applied in solving the maneuvering target tracking problem. However, in real applications, processing time may become impractical. Multiprocessors seem to offer the throughput needed for such applications. In this paper we investigate the performances of Viterbi algorithm and the Iterative Multiple Model algorithm, applied to the tracking problem, when implemented on a general purpose shared-memory & shared-bus MIMD multiprocessor. The computational complexity as well as the speedup and efficiencies are examined in detail. It is shown that the computational complexity of the parallel implementation of these algorithms is about the same in both memory space and processing time categories. Efficiency with P processors is about $1-\frac{1}{p}$ for small P and is expected to be relatively high also for large P, especially when many filters and large state and measurement vectors are considered.

Index Terms-Interacting multiple model algorithm, extended Kalman filter, maneuvering target, tracking, Viterfbi algorithm, performance analysis, shared-bus & shared-memory MIMD multiprocessor.

I. INTRODUCTION

In tracking applications there is a target which changes abruptly its speed or course, and our objective is to estimate its parameters from noisy radar measurements. The state estimation problem in such applications involves a dynamics equation and a measurement equation. The dynamics equation has the form:

$$\vec{x}(t+1) = A(t)\vec{x}(t) + \vec{U}(t) + \vec{v}(t) \quad t = 0, 1, 2, \ldots \tag{1}$$

where $\vec{x}(t)$ is a vector which describes the system state at time t, $A(t)$ is called the transition matrix, $\vec{U}(t)$ is a deterministic input, and $\vec{v}(t)$ is a white noise which is called the process noise. The measurement model is given by the equation:

$$\vec{z}(t) = H(\vec{x}(t)) + \vec{w}(t) \quad t = 0, 1, 2, \ldots \tag{2}$$

where $\vec{z}(t)$ is a measurement vector taken at time t, $H(\cdot)$ is nonlinear, and $\vec{w}(t)$ is a white noise which is called the measurement noise. The Extended Kalman Filter, which is a recursive algorithm, is usually used to give approximate conditional mean of the state vector $\vec{x}(t)$ given the measurement record $Z^t \stackrel{\text{def}}{=}$

*The work of this author was supported by research grant #0337 of the Israeli 'National Council for Research and Development

$\{\vec{z}(0), \vec{z}(1) \ldots \vec{z}(t)\}$. This conditional mean is denoted by $E(\vec{x}(t)|Z^t)$. When one or more of the parameters in Eqs. (1) - (2) are unknown we have to use a more elaborate model to describe the system.

In general, the techniques for detection and estimation of abrupt changes in dynamical systems can be divided into two main classes: the Residual Based approach[19], and the Multiple Model (or Multifilter) approach[19]. In the first case, there is a single filter driven by the measurements while in the second case we assume that the system obeys one of a finite number of models $\{M_i\}_{1 \leq i \leq N}$ at all times. Each one of the models has a structure that obeys Eqs. (1) - (2) and differ only by the values of one (or more) of the parameters. The sequence of models is governed by a finite state Markov [6] (or Semi-Markov [13]) process. The applications are based on a bank of filters which in principle run in parallel although the implementation may be serial. All filters are driven by the same measurements and each one may be viewed as testing another hypothesis about the true parameters or structure of the system. An additional mechanism uses techniques such as pruning or merging in order to keep the number of hypotheses constant (see Fig. 1).

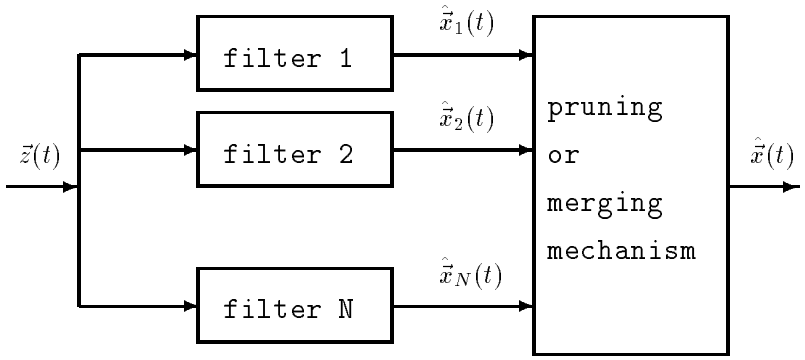

Figure 1: Multiple Model Algorithm: $\hat{\vec{x}}_i(t)$ denotes the estimation based on the i-th model.

Within the Multiple Model class of algorithms there are two primary approaches: the Bayesian approach and the Maximum A-posteriori Probability (MAP) approach.

The Bayesian Approach: The purpose in this approach is to approximate the system state vector $\vec{x}(t)$ by an approximation of its conditional mean $E(\vec{x}(t)|Z^t)$, where Z^t is the measurements sequence $\{\vec{z}(0), \vec{z}(1), \vec{z}(2), \ldots, \vec{z}(t)\}$.

Let us denote by $M_i(t)$ the event that the model M_i is valid at time t, and let $Hist_{\vec{k}}(t) \stackrel{\text{def}}{=} \{M_{k_0}(0), M_{k_1}(1), M_{k_2}(2), \ldots, M_{k_t}(t)\}$ where $\vec{k} = (k_0, k_1, k_2, \ldots, k_t)$, be a specific sequence of models from the space of all possible sequences. The conditional mean $E(\vec{x}(t)|Z^t)$ can be computed by the weighted sum of all the conditional means, given specific histories:

$$E(\vec{x}(t)|Z^t) = \sum_{\vec{k}} E(\vec{x}(t)|Hist_{\vec{k}}(t), Z^t) P(Hist_{\vec{k}}(t)|Z^t) \tag{3}$$

Each conditional mean on the right hand side of Eq. (3) can be approximated by using the corresponding Extended Kalman Filters, to yield an approximation for $E(\vec{x}|Z^t)$. However, as the number of such 'histories' is growing exponentially with time, the optimal solution is intractable and approximating techniques based on 'merging histories' are being used [6,17]. For example, the i-th order General Pseudo Bayesian (GPBi) multiple model algorithm considers only the current and the last $(i-1)$ sampling periods. In this way, at all times the number of terms in Eq. (3) remains constant (N^i for GPBi). However, the Performance to Computational Load Ratio (PCLR) in these models is unsatisfactory [5]. The Interacting Multiple Model (IMM) algorithm is another merging algorithm which conceptually is similar to GPB2, but reduces the computational load by suitable merging of previous estimates[5]. The IMM is considered the most cost-efficient in this class [3].

The MAP (Maximum A-posteriori Probability) Approach: In this approach we try to approximate the conditional mean of the system state vector based on a single 'most probable history'. In other words, we first find:

$$Hist^{\star}(t) = arg\{\max_{\vec{k}}(P(Hist_{\vec{k}}(t)|Z^t))\} \tag{4}$$

where $Hist^{\star}(t)$ is the 'most probable history', and then determine the state estimate as:

$$\hat{\vec{x}}(t) = E(\vec{x}(t)|Hist^{\star}(t), Z^t) \tag{5}$$

which is the conditional mean, given this history. This conditional mean is approximated by using the corresponding Extended Kalman Filters. Again, the optimal solution is impractical and we use suboptimal techniques [15,18].

An example of one way of reducing the number of histories is to throw away 'most unprobable' histories. Let us assume that there are N possible models, and until time t we kept track of the N histories with the maximal likelihood. At the next time step, we have N^2 possible extensions of the previous histories. Each such extension is treated as a new hypothesis and its likelihood is computed. At this stage, we choose only the N hypotheses with the maximal likelihood and delete the other $N^2 - N$ histories. Now, we can proceed to the next time step and repeat this procedure.

Another possible algorithm is Viterbi Algorithm which is a dynamic programming algorithm, extensively used in speech processing and communication applications [8,15]. The assumption underlying this algorithm is that the Probability Density Function of the measurements depends only on the current model which the system obeys i.e.:

$$P(\vec{z}(t)|Hist(t), Z^t) = P(\vec{z}(t)|M_i(t), Z^t) \tag{6}$$

where $Hist(t)$ is the history of models until time t and $M_i(t)$ is the event that model M_i is valid at time t. When this assumption is satisfied the Viterbi algorithm gives the optimal solution. In the tracking problem, however, this is not the case because the measurements depend on the target dynamics history. Thus, the VA does not give the optimal MAP history of models but some 'approximated' history. Let us note that the MAP approach is closely related to methods extensively used in the problem of multi-target tracking, but to the best of our knowledge has not been applied to a single target radar tracking problem. The Multiple Model approach provides a better accuracy on one hand, but has higher computational load on the other hand, as compared to the Residual Based approach [4,16]. However, the Multiple Model approach in tracking problems is very suitable to parallel processing as different models may be assigned to different processing elements and thereby reducing significantly the processing time. In this regard, the parallel Ada implementation of a specific multiple-model Kalman filter tracking systems was investigated in [12] from a software engineering point of view. In [11] a parallel implementation of the extended Kalman filter for target tracking systems is presented using a decoupling technique. The implementation of the extended Kalman filter for target tracking applications on the Warp computer(the Warp is systolic multiprocessor which is built of a linear array of identical cells connected to a general purpose host) is discussed in [14].

The purpose of this paper is, however, to investigate and compare the performance of various Multiple Model algorithms, applied to the maneuvering target radar tracking problem, when implemented on a general purpose shared-memory and shared-bus MIMD multiprocessor. Within the Multiple Model class of algorithms there are two primary approaches: the pseudo Bayesian approach and the Maximum A-posteriori Probability (MAP) approach. As representative algorithms we have chosen from the MAP approach the Viterbi Algorithm (VA) [8] and from the pseudo Bayesian approach we have chosen the Interacting Multiple Model (IMM) algorithm [5]. The computational complexity, speedup and efficiency of both algorithms is examined. The quality (accuracy) of the estimations is discussed in another paper[1].

The paper is organized as follows. Section II describes in detail the IMM model while section III describes the the Viterbi model. The Kinematic model of the problem is presented in section IV. Section V describes in detail the parallel implementation of the algorithm on a MIMD multiprocessor and numerical results are presented in Section VI. Finally, Section VII summarizes our main results.

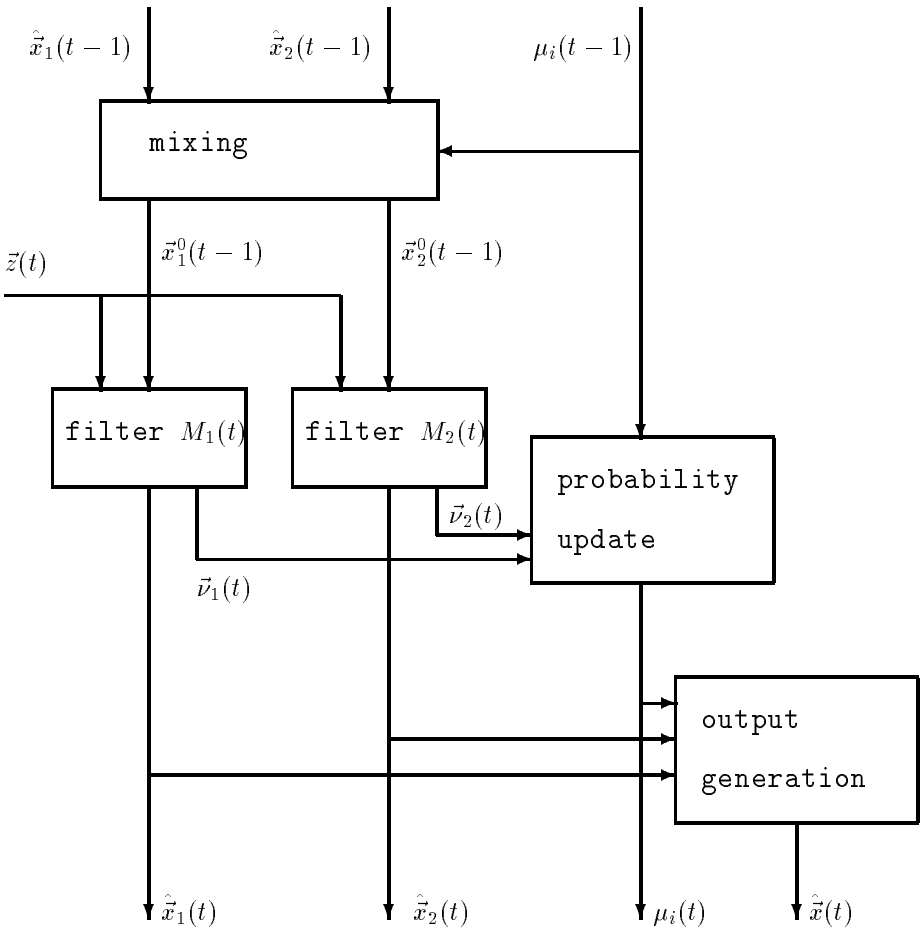

Figure 2: The Interacting Multiple Model algorithm for $N = 2$

II. THE INTERACTING MULTIPLE MODEL (IMM)

Given the forms of the dynamics equation (Eq. (1)) and of the measurement model (Eq. (2)), we wish to estimate $\vec{x}(t)$. The IMM algorithm is based on N Extended Kalman Filters, set up for N different models $M_i(t)\ \ 1 \leq i \leq N$ of Eqs. (1)-(2) and providing recursive N estimates $\hat{\vec{x}}_i(t)\ \ 1 \leq i \leq N$ of $\vec{x}(t)$. At time t, an initial input is imposed on each filter separately. This input consists of the current measurements vector $\vec{z}^t$, a combination of all filters previous estimates, $\hat{\vec{x}}_i(t-1)$, the corresponding covariances $C_i(t-1)\ \ 1 \leq i \leq N$, and the appropriate probability weights $\mu_i(t-1) \stackrel{\text{def}}{=} P(M_i(t-1)|Z^{t-1})$. The probability density is assumed to be normal i.e.

$$P(\vec{x}(t-1)|M_i(t-1), Z^{t-1}) \sim \mathbf{N}(\hat{\vec{x}}_i(t-1), C_i(t-1))$$

The outputs of these filters $\hat{\vec{x}}_i(t)$ and $C_i(t)$ are then used to update the probability weights and to produce the final estimations of the current cycle $\hat{\vec{x}}(t)$ and $C(t)$ on one hand, and to produce the input of each filter for the next cycle, on the other hand. Specifically, each cycle of this algorithm consists of 4 steps which are performed for each possible model $M_i(t) \quad 1 \leq i \leq N$ (see Fig. 2):

- *Input generation(mixing)* - The initial condition to the filter matched to $M_j(t)$ is generated by mixing the estimates of all filters in the previous cycle as follows:

$$\vec{x}_j^0(t-1) \stackrel{\text{def}}{=} E(\vec{x}(t-1)|M_j(t), Z^{t-1}) =$$

$$= \sum_{i=1}^{N} E(\vec{x}(t-1)|M_i(t-1), M_j(t), Z^{t-1}) P(M_i(t-1)|M_j(t), Z^{t-1})$$

If $M_i(t-1)$ is known then $\vec{x}(t-1)$ is independent of $M_j(t)$ and we get:

$$\vec{x}_j^0(t-1) = \sum_{i=1}^{N} E(\vec{x}(t-1)|M_i(t-1), Z^{t-1}) P(M_i(t-1)|M_j(t), Z^{t-1})$$

But $\hat{\vec{x}}_i(t-1) \stackrel{\text{def}}{=} E(\vec{x}(t-1)|M_i(t-1), Z^{t-1})$ and therefore

$$\vec{x}_j^0(t-1) = \sum_{i=1}^{N} \hat{\vec{x}}_i(t-1) P(M_i(t-1)|M_j(t), Z^{t-1}) \tag{7}$$

where:

$$P(M_i(t-1)|M_j(t), Z^{t-1}) = \frac{p_{ij}\mu_i(t-1)}{c} \tag{8a}$$

in which

$$p_{ij} = P(M_j(t)|M_i(t-1)) \tag{8b}$$

and

$$c = \sum_{i=1}^{N} p_{ij}\mu_i(t-1). \tag{8c}$$

The corresponding covariances are computed in a similar way and thus we get:

$$C_j^0(t-1) = \sum_{i=1}^{N} P(M_i(t-1)|M_j(t), Z^{t-1})[C_i(t-1) + (\hat{\vec{x}}_i(t-1) - x_i^0(t-1))((\hat{\vec{x}}_i(t-1) - \vec{x}_i^0(t-1))^T]$$

- *Kalman step* - Each of the pairs $(\vec{x}_j^0(t-1), C_j^0(t-1))$ is now used as an input to the Extended Kalman Filter matched to the j-th model. The output is denoted by $(\hat{\vec{x}}_j(t), C_j(t))$.

- *Probability update* - The weights $\{\mu_j(t)\}$ $1 \leq j \leq N$, are now updated using the innovations of the filters:

$$\mu_j(t) = P(M_j(t)|Z^t) = \frac{P(\vec{z}(t)|M_j(t), Z^{t-1}) \cdot c}{c_1} \tag{9a}$$

where c is given in Eq. (8c) and

$$c_1 = P(\vec{z}(t)|Z^{t-1}) \tag{9b}$$

is a normalization constant which ensures that $\sum_{j=1}^{N} P(M_j(t)|Z^t) = 1$. In Eq. (9a) $P(\vec{z}(t)|M_j(t), Z^{t-1})$ is the likelihood function obtained from the innovation of the j-th Kalman filter $\vec{\nu}_j(t)$:

$$P(\vec{z}(t)|M_j(t), Z^{t-1}) = \frac{1}{\sqrt{(2\pi)^n |S_j(t)|}} \exp(-\frac{1}{2}\vec{\nu}_j^T(t) S_j(t) \vec{\nu}_j(t)) \tag{10}$$

where $\vec{\nu}_j(t) = \vec{z}(t) - H(\hat{\vec{x}}_j(t))$, $S_j(t)$ is the corresponding covariance and n is the state vector dimension.

- *Output generation* - The final estimation for output purposes is:

$$\hat{\vec{x}}(t) = \sum_{j=1}^{N} \hat{\vec{x}}_j(t)\mu_j(t) \tag{11}$$

with the corresponding covariance:

$$C(t) = \sum_{j=1}^{N} \mu_j(t)[C_j(t) + (\hat{\vec{x}}_j(t) - \hat{\vec{x}}(t))(\hat{\vec{x}}_j(t) - \hat{\vec{x}}(t))^T] \tag{12}$$

III. VITERBI ALGORITHM

As mentioned earlier, in this approach we search for an approximation to the conditional mean of the system state vector given the history of models with Maximal A-posteriori Probability (see Fig. 3). Specifically, we first find:

$$Hist^{\star}(t) = arg\{\max_{\vec{k}}(P(Hist_{\vec{k}}(t)|Z^t)\} \tag{13}$$

where $Hist_{\vec{k}}(t) = \{M_{k_0}, M_{k_1}, \ldots, M_{k_t}\}$, $\vec{k} = (k_0, k_1, \ldots, k_t)$ and $P(Hist_{\vec{k}}(t)|Z^t)$ is computed recursively. Then we apply a Kalman filter on $Hist^{\star}(t)$ to get:

$$\hat{\vec{x}}(t) = E(\vec{x}(t)|Hist^{\star}(t), Z^t) \tag{14}$$

To implement Eq. (13) we need a growing number of Kalman filters (N^t) in order to compute $P(Hist_{\vec{k}}(t)|Z^t)$ for every possible $\vec{k}$. As this optimal solution is impractical, some hypotheses reduction technique is needed as in the Bayesian approach. In order to keep constant the number of histories we keep track of, we have to decide at every stage which of the histories that 'survived' until this time should be ignored and which should be passed on to the next step.

The Viterbi Algorithm (VA) is a dynamic programming algorithm, designed for a system which obeys one of N models with Markovian transition process, where the measurement depends probabilistically only on the current model. Hence,

$$P(\vec{z}(t)|Hist_{\vec{k}}(t), Z^{t-1}) = P(\vec{z}(t)|M_j(t), Z^{t-1}) \tag{15}$$

where $M_j(t)$ is the **last** model in the sequence $Hist_{\vec{k}}(t)$. In other words, the Probability Density Function (PDF) of the measurement depends only on the current model regardless when and how this model was entered. Under this assumption, Viterbi Algorithm finds the optimal MAP history of models. In our case however, as

$$\vec{z}(t) = H(\vec{x}(t)) + \vec{w}(t)$$

and

$$P(\vec{x}(t)|Hist_{\vec{k}}(t), Z^{t-1}) \neq P(\vec{x}(t)|M_j(t), Z^{t-1})$$

we get:

$$P(\vec{z}(t)|Hist_{\vec{k}}(t), Z^{t-1}) \neq P(\vec{z}(t)|M_j(t), Z^{t-1}).$$

Thus, the PDF of the measurement depends on the entire history and not only on the current model. As the assumption in Eq. (15) is not valid, VA does not give the optimal MAP history of models but some 'approximated' history. In our VA algorithm, the estimate of $\vec{x}(t)$ is determined by the outputs of the 'most probable' models, assuming Markovian transition process.

Specifically, the algorithm at time slice t consists of the following stages:

1. *probability calculations* - For each possible model $M_j(t)$ do:

   (a) For each possible previous model $M_i(t-1)$ compute the probability weight of the history $\{Hist_i(t-1), M_j(t)\}$, where $Hist_i(t-1)$ is the MAP history until time $t-1$ which ends in model M_i. This is given by:

$$\begin{aligned} \mu_{ij}(t) &= \mu_i(t-1)p_{ij}P(z(t)|M_j(t), M_i(t-1), Z^{t-1}) = \\ &= \mu_i(t-1)p_{ij}P(z(t)|\hat{x}_i(t-1), C_i(t-1), M_j(t)) \end{aligned} \tag{16a}$$

   The term $P(\vec{z}(t)|\hat{\vec{x}}_i(t-1), C_i(t-1), M_j(t))$ is evaluated from the innovation of the Extended Kalman Filter matched to $M_j(t)$ with $(\hat{\vec{x}}_i(t-1), C_i(t-1))$ as input:

$$P(z(t)|\hat{\vec{x}}_i(t-1), C_i(t-1), M_j(t)) = \mathbf{N}(\nu_{ij}(t); 0, S_{ij}(t)) \tag{16b}$$

   where $\nu_{ij}(t)$ and $S_{ij}(t)$ are the corresponding innovation and covariance.

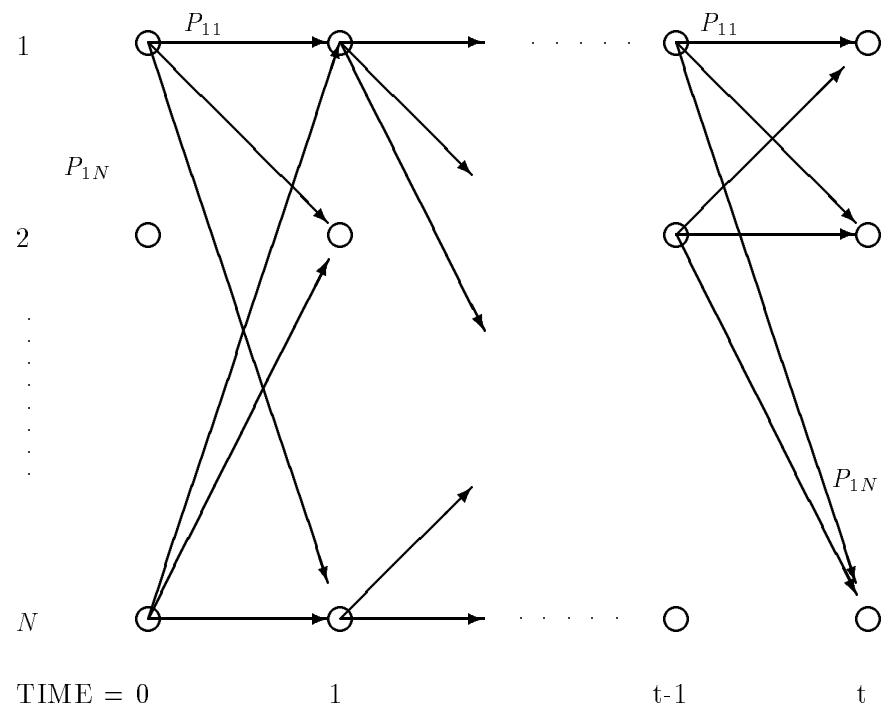

Figure 3: The MAP approach

(b) Choose the surviving history by:

$$\begin{aligned} i_{max}(j) &= arg\{\max_{1\le i\le N}(\mu_{ij}(t))\} \\ Hist_j(t) &= \{Hist_{i_{max}(j)}(t-1), M_j(t)\} \\ \mu_j(t) &= \mu_{i_{max}(j)j}(t) \end{aligned}$$

2. *Output generation* - The output may be generated in at least two ways. If we follow the original algorithm then:

$$j_{max} = arg\{\max_{1\le j\le N}\mu_j(t)\} \quad \text{and} \quad \hat{\vec{x}}(t) = \hat{\vec{x}}_{j_{max}}(t) \tag{17}$$

Another possibility is to merge histories which gives:

$$\hat{\vec{x}}(t) = \sum_{j=1}^{N} \hat{\vec{x}}_j(t)\mu_j(t) \tag{18}$$

IV. THE KINEMATIC MODEL

In solving the maneuvering target problem, we first have to choose a mathematical model of the target dynamics. If we want to apply the Multiple Model approach, in most cases we then have to select one of the unknown parameters in the model, describe it by a finite set of values, and assign each such value to each of the filter models. We consider a motion in the $x-y$ plane and we use a model in which the acceleration during the maneuver is assumed to be almost constant, and its mean is taken from a finite set of accelerations. The perturbation upon this constant acceleration is modeled as white zero-mean Gaussian noise. Thus the continuous time dynamics equation in a generic coordinate system is:

$$\ddot{\vec{r}}(\tau) = \vec{a}_i + \vec{V}(\tau) \quad 1 \le i \le N \tag{19}$$

where $\vec{a}_i$ is a discrete acceleration level and $\vec{V}(\tau)$ is a continuous time zero-mean Gaussian white noise. The corresponding state-space equation is:

$$\dot{\vec{x}}_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vec{x}_0 + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \vec{a}_i + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \vec{V}(\tau) \tag{20}$$

where $\vec{x}_0 = \begin{pmatrix} \vec{r} \\ \dot{\vec{r}} \end{pmatrix}$ is the 4×1 state vector.

Preliminary tests showed that both algorithms, which concern our study (IMM and Viterbi), suffer from a severe bias problem when the true acceleration of the target does not match one of the discrete levels of the model. Our solution was to augment an additional acceleration component to the state vector. This component represents the bias from the given discrete acceleration. After this augmentation, the discretized state-space equation becomes:

$$\vec{x}(t+1) = A\vec{x}(t) + \vec{U}(t) + \vec{v}(t) \tag{21}$$

in which

$$\vec{x} = \begin{pmatrix} \vec{r} \\ \dot{\vec{r}} \\ \ddot{\vec{r}} \end{pmatrix}, \qquad \vec{U}(t) = B\vec{a}_i, \qquad \vec{v}(t) = C\vec{V}(t) \tag{22}$$

$$A = \begin{pmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} \frac{(\Delta t)^2}{2} & 0 \\ 0 & \frac{(\Delta t)^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and } C = \begin{pmatrix} \frac{(\Delta t)^2}{2} & 0 \\ 0 & \frac{(\Delta t)^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{23}$$

The transitions between models (accelerations) are governed by a Markov process i.e.:

$$P(M_i(t)|Hist(t-1)) = P(M_i(t)|M_j(t-1)) \tag{24}$$

where $M_k(t)$ denotes the event that model M_k is valid at time t, and $Hist(t-1)$ is the complete history of models until time $t-1$. We adopted the approach of Gholson and Moose in [10] that suggested choosing the probabilities as follows:

$$P(M_i(t)|M_j(t-1)) = \begin{cases} p & i = j \\ (1-p)/(N-1) & i \neq j \end{cases} \tag{25}$$

where $p \sim 1$ and N is the number of models. The covariance of the process noise was recomputed according to the procedure described in [2] for the constant acceleration model, and is given by:

$$Q = E(\vec{v}(t)\vec{v}^T(t+k)) = CC^T q\delta(k) \tag{26}$$

V. IMPLEMENTATION ON THE PARALLEL PROCESSOR

A. The MMX Parallel Processor

The Multiprocessor Multitasking eXecutive (MMX)[9] parallel processor on which we have implemented our IMM and VA algorithms,consists of 4 processor boards and 3 MB of shared memory connected by MULTIBUS I. The boards are National Semiconductor's DB32332 boards. The processor boards are connected to a host computer (SUN 3/180) by serial links (see Fig. 4). MMX is written mainly in C, as are the user programs. Table 1 describes the MMX services used in our application.

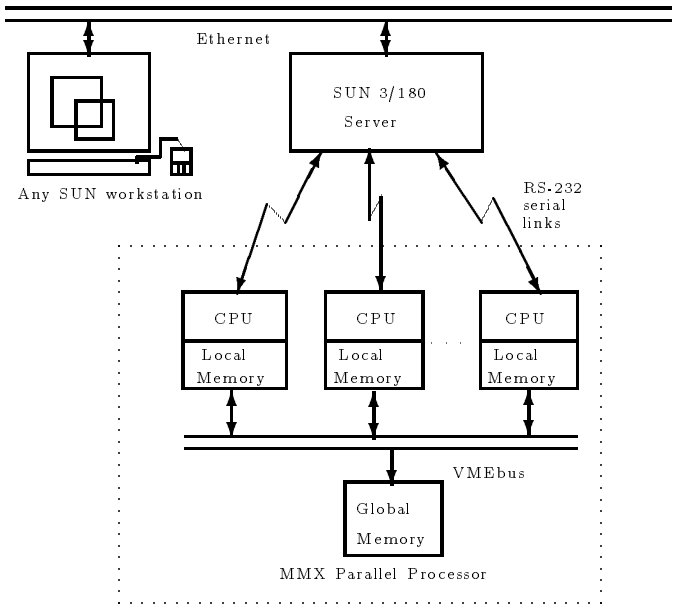

Figure 4: MMX Parallel Machine

Table 1: MMX services used in our application

Service	**Description**
Task Management:	
spawn(routine,stack-sz,arg,proc#)	Spawn a new task to execute *routine* with argument *arg* on processor *proc* and stack size *stack-sz*
Synchronization:	
s=screate(val)	Create a new semaphore with initial value *val*
request(s)	Request a semaphore *s*
release(s)	Release a semaphore *s*
releasen(s,n)	Perform *release(s)* *n* times
sreset(s,v)	Reset semaphore *s* count to *v*

B. Work Distribution

Both the IMM and VA algorithms basically consist of a group of filters which run in parallel, and a global computation process which collects the results of the filters and produces the output estimation. Thus, the implementation of both algorithms on the parallel processor is based on the concept of one processor acting as a manager which performs the global computations, and all other processors (in our case only 1,2, or 3) acting as workers. Both the manager and the workers are implemented as local tasks. The assignments of the manager during each time slice are:

- Getting new measurements.
- Generating the output based on results computed by the other processors.
- Synchronizing between processors.
- Updating some variables which are part of the Extended Kalman Filter(EKF) and are relevant to all processors.

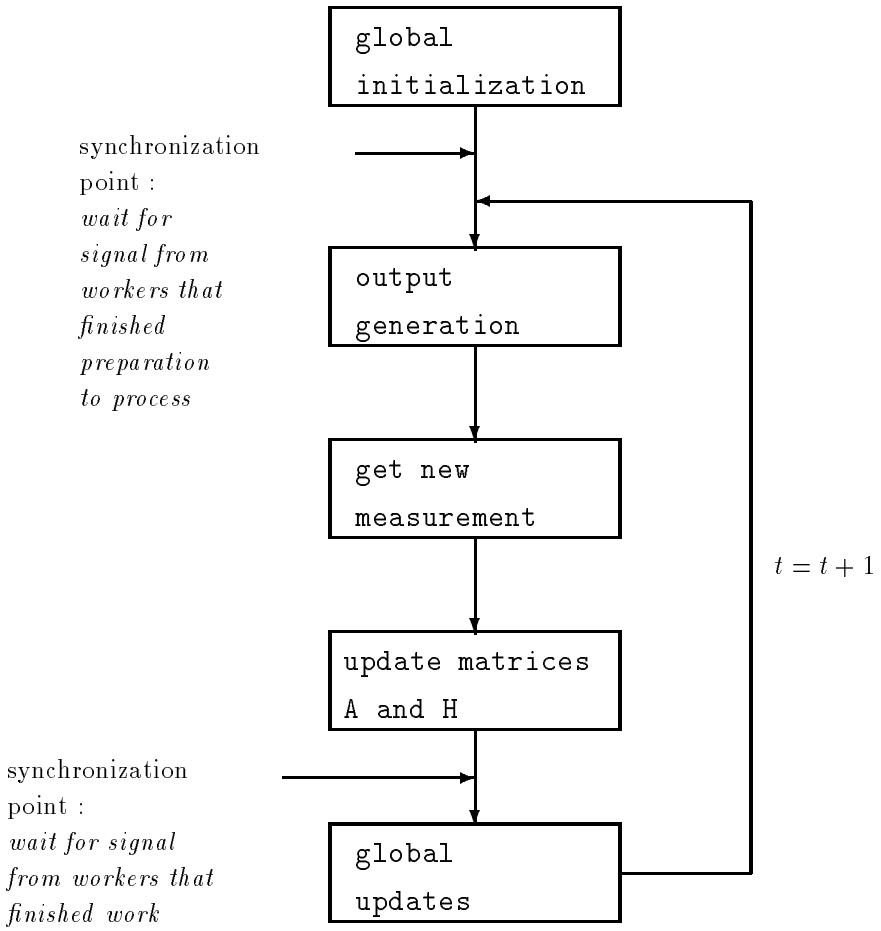

Figure 5: The manager's activities in IMM and Viterbi algorithms

Fig. 5 illustrates the sequence of the manager's activities. In both IMM and VA each time slice consists of a large number of computations, most of which are an Extended Kalman Filter step performed independently for each possible current model $\{M_j(t), 1 \leq j \leq N\}$. The input of each such filter is an estimation of the system state vector along with its covariance for the previous time step and a new measurement. The output is an updated estimation. This computational load can be divided among the processors almost without additional overhead by assigning each processor a subgroup of the models. Figures 6 and 7 illustrate the activities of the worker processor in IMM and VA.

C. Synchronization and Read/Write Conflicts

The regular sequence of events during each time step of IMM and VA on the parallel machine can be described roughly by:

- Obtaining measurement - done by the manager.
- Main processing: Kalman filter operations and probability calculations. This is done by the workers.
- Additional processing and output generation: The updates of the measurement error covariance matrix and the linearized measurement matrix which are common to all models are done by the manager. the manager also produces the final output estimation either by a weighted average over the model conditioned estimations or by choosing the estimation with maximal probability.

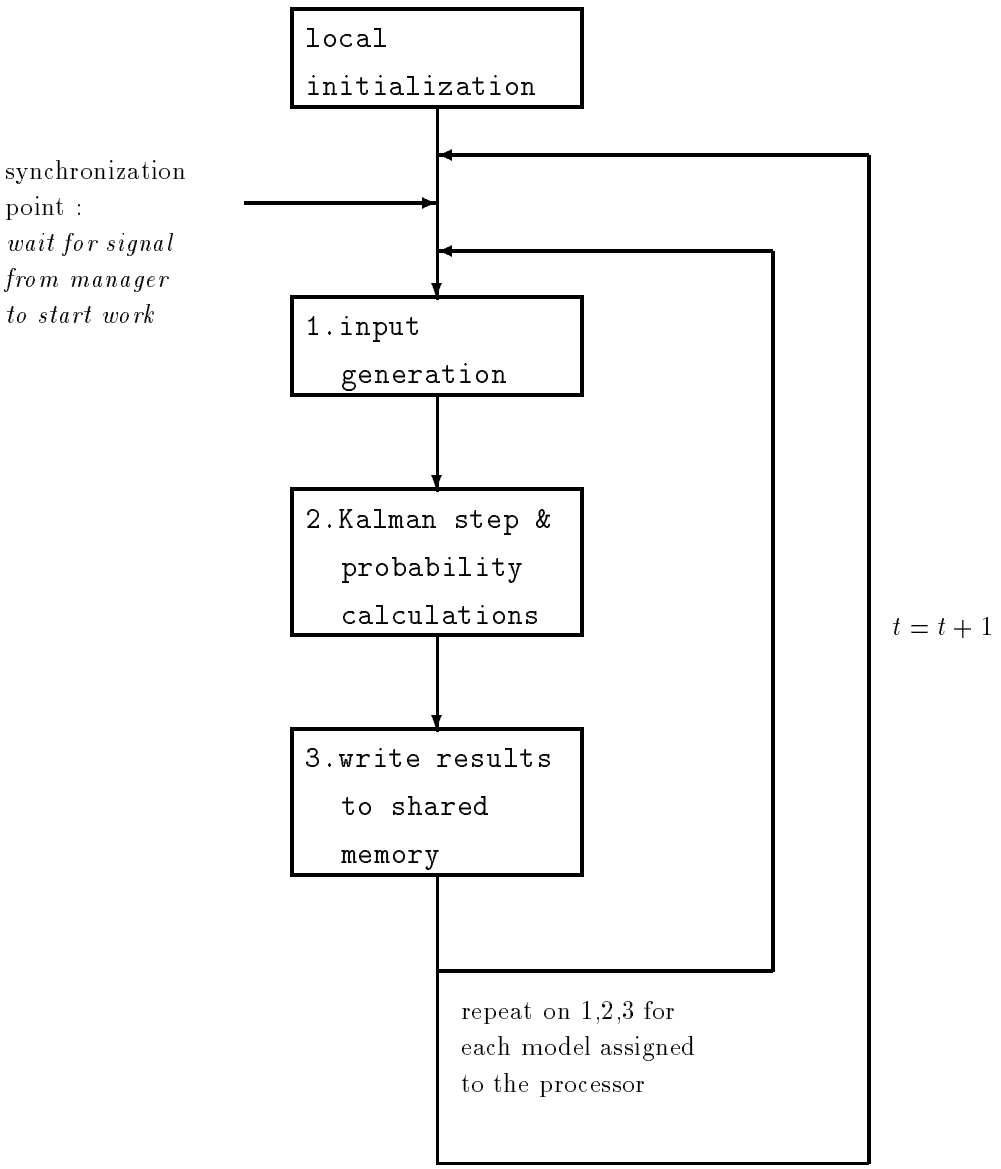

Figure 6: The worker's activities in IMM

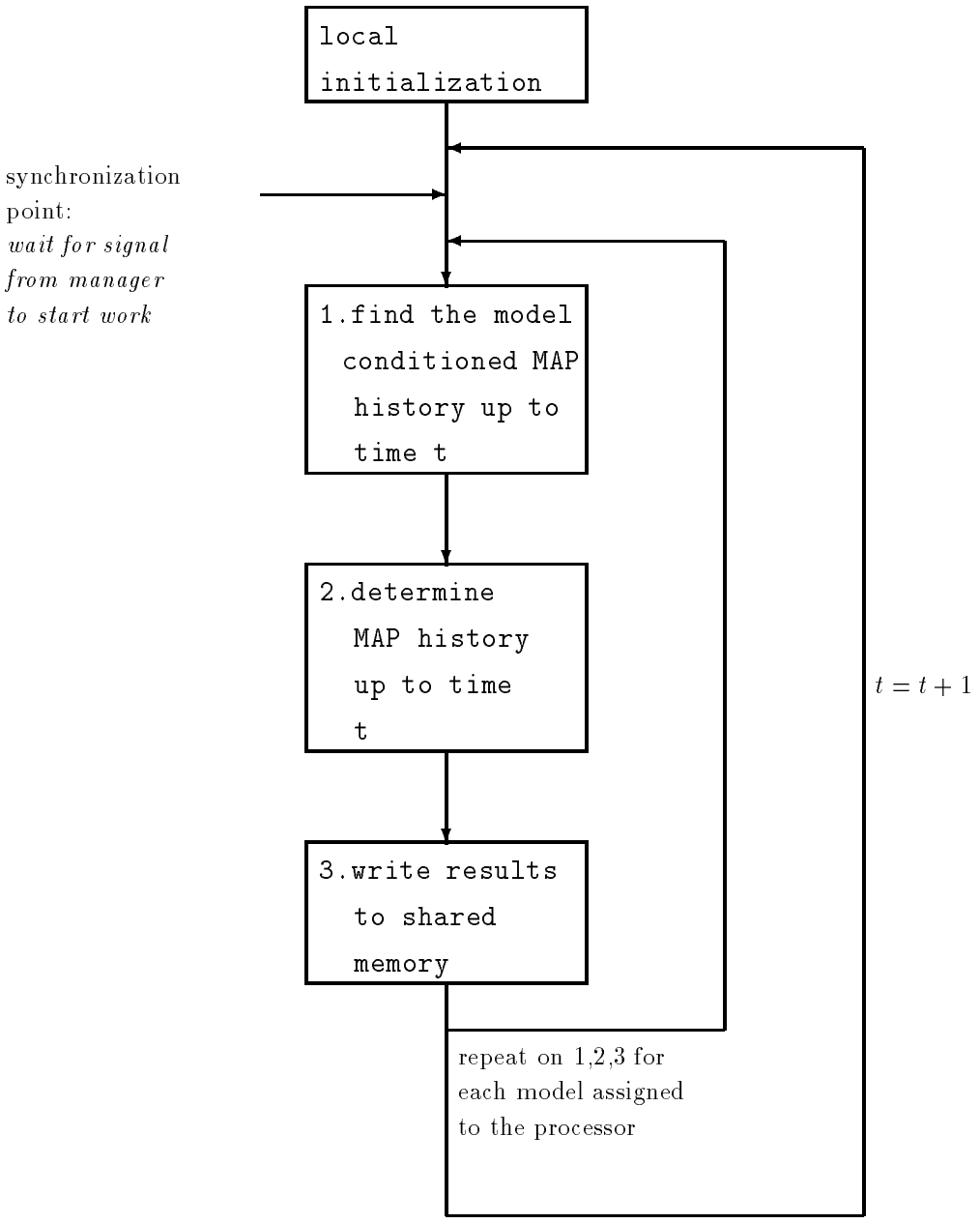

Figure 7: The worker's activities in Viterbi algorithm

Thus, the implementation has to guarantee:

- *Synchronization* - ensure that all processors process the same measurement at the same time.
- *Avoid Read/Write Conflicts* - prevent cases of contradicting accesses to the shared memory by the processors (for example: simultaneous write accesses by two processors or write access of one processor while another processor is trying to read)

1) Synchronization

To synchronize among the processors we use two semaphores named by r and s. These semaphores are used by the manager to signal the workers that a new measurement has arrived and that they may start processing it. They are also used by the workers to notify the manager that they have finished processing the measurement and are ready to start processing a new one. The following tables describe how the synchronization is done using the semaphores. The left column lists the synchronization activities of the manager, and the right column lists the activities of each worker. P is defined as the number of active processors (4 in our case).

Initialization

Table 2 describes the synchronization activities since the startup of the user program (which is actually the start of the manager task) until the main process begins.

Table 2: Manager and worker activities during initialization

	Manager Activities	Worker Activities
1.	*r=screate(0)* Create semaphore s with initial value 0 *s=screate(2 − P)* Create semaphore s with initial value $(2-P)$	Not started yet
2.	Spawn the workers tasks	Local initialization process
3.	*request(s) ; sreset(s,2 − P)* Wait for signal from workers that they finished local initialization process. Reset the value of s to $2-P$ to make it ready for the first time step	*release(s)* Notify manager that local initialization process has been completed

Main Process

Table 3 describes the synchronization activities after the initialization has been completed.

2) Avoiding read/write conflicts

All the shared memory variables used in our algorithms are accessed for read and write purposes. For example, the manager updates at each time step the measurement variable z. At the same time the other processors need the previous value of z (i.e. the previous measurement) as input. The nature of the algorithms guarantees that there are no write conflicts because each sharable variable is updated only by one processor. However, read/write conflicts between manager and workers may occur. In order to avoid such conflicts we use two copies of each sharable variable. Let us assume that at time t copy #1 is used for write purposes only and copy #2 is used for read purposes only. At the next time step we exchange the roles. Copy #1

Table 3: Manager and worker activities during the main process

	Manager Activities	Worker Activities
1.	*releasen(r,P − 1)* Signal all workers to start work	*request(r)* Wait for signal from manager to start work
2.	Working...	Working...
3.	*request(s) ; sreset(s,2 − P)* Wait for signal from workers that they have finished their work. Reset the value of s to $2 - P$ to prepare it for the next time step	*release(s)* Signal manager that work on current measurement has been completed
4.	Update the normalization constant and switch INPUT/OUTPUT buffers. (see next section for explanation)	IDLE
5.	Return to 1	Return to 1

is used for input and copy #2 is used for output purposes only. All we have to do now in order to avoid read/write conflicts is to ensure that all processors know which copy is used for input and which is used for output at a specific time. This scheme for the shared memory variable of the measurement z is illustrated in Fig. 8. The contents of the frames titled copy #1 and copy #2 represents the contents of the two copies of the shared memory variable z.

D. Computational Complexity

1) Categories of Evaluation

We have checked both the time and space(memory) complexity of IMM and VA. All the values in the following tables are related to our specific implementation. However, it is clear that there is quite a large possible tradeoff between space and execution time. For example, as shared memory accesses are slower than local memory accesses we try to make more computations in the local memory. Copying a large part of the shared memory to local memory of each processor at each time step saved a large number of shared memory accesses. However, if the local memory is not sufficient, the data can remain in the shared memory. Another example is the way we avoid read/write conflicts. In the previous section we described a solution which duplicates the amount of shared memory space we use. An alternative solution which is cheaper in space but more expensive in time is to use a more complex synchronization scheme which prevents such conflicts.

2) Time Complexity

The time complexity is evaluated using 4 categories:

- Elementary arithmetic operations $(+, -, \cdot, /)$.
- Complex arithmetic operations $(\exp(), \arctan())$.
- Accesses to local memory
- Accesses to shared memory

Let us denote:

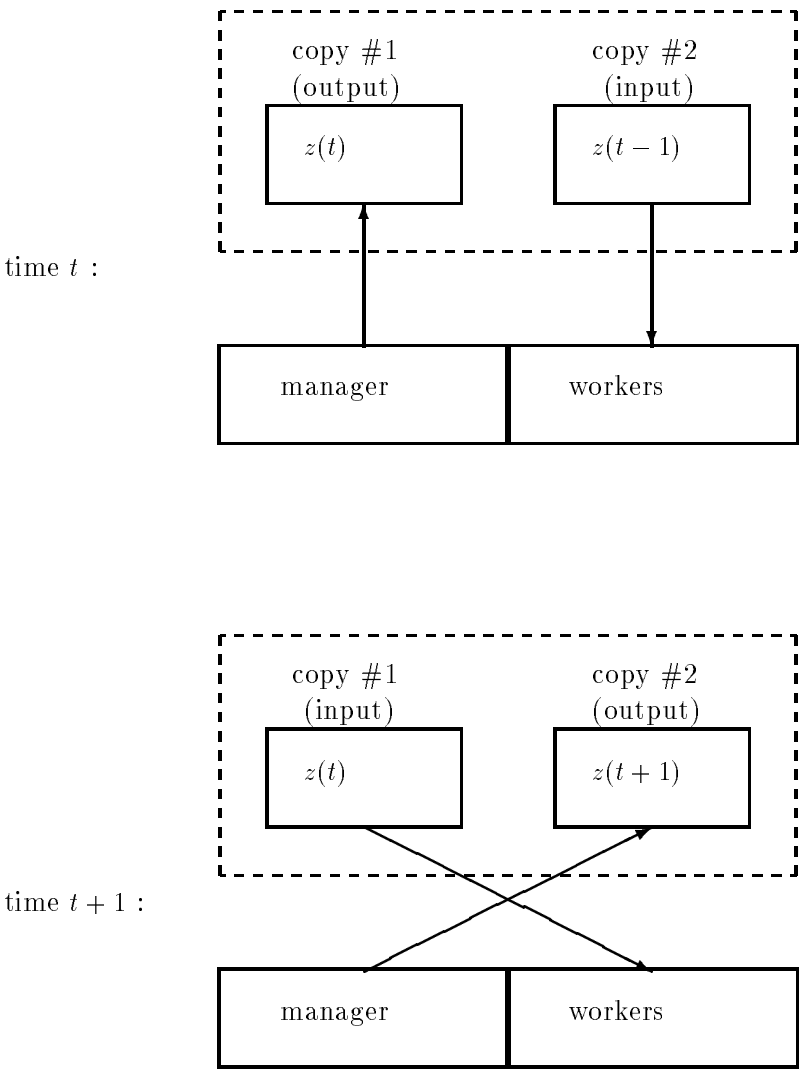

Figure 8: Avoiding read/write conflicts

n - The dimension of the state vector (in our case - 6)
m - The dimension of the measurement vector (in our case - 2).
N - The total number of models.
N_p - The number of models assigned to each worker.

Tables 4 and 5 present the computational load on a worker processor for the IMM and VA algorithms respectively. Notice that the computational load on the manager is negligible compared to a worker and the overall runtime depends only on the computational load on the workers. All the values in these Tables reflect only the order of number of operations and we omitted elements which are not dominant as well as the multiplying constants.

Table 4: Time complexity of the IMM algorithm:

	Shared Memory Accesses	**Local Memory Accesses**	**Elementary Operations**	**Complex Operations**
Copy from Shared to Local Memory	Nn^2	Nn^2	-	-
Input Generation	$N \cdot N_p n^2$	$N \cdot N_p n^2$	$N \cdot N_p n^2$	-
Time Update	-	$N_p n^3$	$N_p n^3$	-
Measurement Update & Probability Calculations	$N_p m^4$	$N_p m^4 + N_p n^3$	$N_p m^4 + N_p n^3$	N_p
Write to Shared memory	$N_p n^2$	$N_p n^2$	-	-
Dominant	$N \cdot N_p n^2$	$N_p n^3 + N_p \cdot Nn^2$	$N_p n^3 + N_p \cdot Nn^2$	N_p

Table 5: Time complexity of the Viterbi algorithm

	Shared Memory Accesses	**Local Memory Accesses**	**Elementary Operations**	**Complex Operations**
Copy from Shared to Local Memory	Nn^2	Nn^2	-	-
Time Update of the State Vector	-	$N \cdot N_p n^2$	$N \cdot N_p n^2$	-
Probability Calculations	$N \cdot N_p m^4$	$N \cdot N_p m^4$	$N \cdot N_p m^4$	$N \cdot N_p$
Measurement Update & Covariance Time Update	-	$N_p n^3$	$N_p n^3$	-
Write to Shared memory	$N_p n^2$	$N_p n^2$	-	-
Dominant	$N \cdot N_p m^4$	$N_p n^3 + N_p \cdot Nn^2$	$N_p n^3 + N_p \cdot Nn^2$	$N \cdot N_p$

3) Space Complexity

Table 6 shows the space complexity in two categories: shared memory and local memory.

The basic unit in our calculations is the space needed to store a float variable which is 4 bytes. Note again that the values in Table 6 reflect the order of the amount of space needed rather than the accurate numbers.

Table 6: Space complexity of IMM and VA algorithms

	Shared Memory	Local Memory
Viterbi	$2Nn^2$	$2Nn^2$
IMM	$2Nn^2$	Nn^2

The difference between IMM and VA in space complexity is in the amount of local memory. However, the double space in VA is not essential and is used mainly for the sake of simplicity and clarity of the program.

4) Conclusions

- Generally, the computational complexity of IMM and VA is similar in both the memory space and time categories. The dominant values in time complexity are identical, except for shared memory access and complex operations. The difference between IMM and VA in space complexity is in the amount of local memory. However, the double space in VA is not essential and is used mainly for the sake of simplicity and clarity of the program.
- Increasing the number of models by a factor of A, when the number of processors remains constant will increase the computational load on each worker (and thus the run time) by a factor of A^2 because of the $N \cdot N_p n^2$ elementary operations during each time step. Even if we increase the number of processors by the same factor A the computational load on each processor will increase by factor of A.

VI. SIMULATION RESULTS

We have selected the use of 12 models which cover an accelerations range of about $0-4$g ($0-40$ meter/sec^2) , and used Eq. (11) for IMM algorithm and Eq. (18) for VA. The allowed rate of change of acceleration within a specific model ($\sqrt{q\Delta t}$) was chosen to be of the order of 1 m/sec^2. The probability for staying in the same model, $P(M_i(t)|M_i(t-1))$, was chosen to be 97%.

The measurements are taken in polar coordinates by range and azimuth. We assume that both range and azimuth errors have normal distribution and are independent of each other. The standard deviation of the range error is given by:

$$\sigma_r = \begin{cases} 50 \text{ m} & \text{if } r > 5000 \text{ m} \\ r/100 & \text{if } r \leq 5000 \text{ m} \end{cases} \tag{27}$$

where r is the range. The standard deviation of the azimuth error σ_{θ_A} is 2mrad(mili radians). Thus, the matrix equation for the measurement can be written:

$$z(t) = \begin{pmatrix} r \\ \theta_A \end{pmatrix} = H(\vec{x}(t)) + \vec{w}(t) \tag{28a}$$

in which $H(\vec{x}(t))$ is the non-linear function:

$$H(\vec{x}(t)) = \begin{pmatrix} \sqrt{x^2(t) + y^2(t)} \\ \arctan(y(t)/x(t)) \end{pmatrix} \tag{28b}$$

and $\vec{w}(t)$ is the measurement noise with the covariance:

$$R(t) = E(w(t)w^T(t)) = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_{\theta_A}^2 \end{pmatrix} \tag{28c}$$

As the minimal number of processors that our implementation needs is two (manager + worker) we examined the run time when 2,3 and 4 processors are active. We also inspected the speed-up and efficiency values which are defined as:

$$\text{speed–up} = \frac{\text{serial–time}}{\text{parallel–time}} \qquad \text{efficiency} = \frac{\text{speed–up}}{\text{P}} \tag{29}$$

where P is the number of active processors. The results for both algorithms are shown in Table 7. The indicated values are the run times in seconds for 100 iterations, the speed-up and the efficiency for each run. If we assume that the speed-up can be approximated by a quadratic function and use our results to determine this function we can predict the run-time for more than 4 processors (see the last column).

Table 7: Dependence of run time (in seconds) on the number of processors for 100 iterations

	Serial	**Manager + worker**	**Manager + 2 workers**	**Manager + 3 workers**	**Prediction for manager + 6 workers**
IMM:					
Run time	66.54	64.77	32.73	22.17	11.61
Speed-up		1.03	2.03	3.00	5.73
Efficiency		0.51	0.67	0.75	0.81
VA:					
Run time	57.71	55.79	28.69	19.75	11.03
Speed-up		1.03	2.01	2.92	5.23
Efficiency		0.51	0.67	0.73	0.75

As observed from Table 7, for a small number of processors, manager's run time as well as communication and sysnchronization overhead are negligible compared to worker's run time. Therefore, the speedup with P processors is about $P-1$ and the efficiency is about $1-\frac{1}{P}$. Obviously, for large number of processors, efficiency would decrease since the relative effect of communication and synchronization overhead as well as of the manager's run time would increase. Nevertheless, the above algorithms are expected to provide relatively high speedup and efficiency, even for large number of processors, especially when many filters and large state and measurement vectors are considered.

VII. SUMMARY

We have investigated the parallel implementation of multiple model algorithms in tracking applications. Specifically, we examined the implementations of Viterbi Algorithm (VA) and the Interactive Multiple Model (IMM) algorithm on a shared bus&shared memory MIMD multiprocessor. These algorithms were proven to be very suitable to parallel processing. The implementation is almost straight-forward and is based on one manager and few workers. The computational complexity in the parallel implementation of the algorithms is about the same in both memory space and processing time categories. Increasing the number of models by a factor of A will increase each worker processing time by a factor of A^2. Efficiency with P processors when P is relatively small, is about $1-\frac{1}{P}$, for both algorithms since communication and synchronization overhead as well as manager's run time are negligible compared to workers run time. It is expected to obtain relatively high efficiency, even for large P, especially when considering many filters and large state and measurement vectors.

Acknowledgment:

The authors wish to thank Dr S. Rogers and Dr I. Hammerman from Elta Corp. for introducing the problem and continuing support during all stages of this research.

References

1. Averbuch A., Itzikowitz S. and Kapon S., "Radar Target Tracking - Viterbi vs. IMM", IEEE Trans. on Aerospace and Electronic Systems, Vol. 27, No. 3, pp. 550-563, May 1991.
2. Bar-Shalom Y. and Fortman T.E., "it Tracking and data association", New York: Academic Press, 1986.
3. Blom H.A.P. "An efficient filter for abruptly changing systems", Proc. of the 23rd Conf. on Decision and Control, Las Vegas NV, Dec 1984.
4. Blom H.A.P. "Overlooked potential of systems with Markovian coefficients", Proc. of the 25th Conf. on Decision and Control, Athens Greece, Dec 1986.
5. Blom H.A.P and Bar-Shalom Y., "The interacting multiple model algorithm with Markovian switching coefficients", IEEE Tran. on Automatic Control, Vol AC-33, No 8,pp 780-783, Aug 1988.
6. Chang C.B. and Athans M., "State estimation for discrete systems with switching parameters", IEEE Tran. on Aerospace and Electronic Systems, Vol AES-14, No 3, pp.418-424, May 1978.
7. Chang C.B. and Dunn K.P., "On GLR detection estimation scheme of unexpected inputs in linear systems", IEEE Tran. on Automatic Control, Vol AC-24, No 3, pp.499-501 , 1979.
8. Forney G.D.Jr., "The Viterbi algorithm", Proc. of the IEEE, Vol 61, No. 3 March 73.
9. Gabber E., "Parallel programming using the MMX operating system and its processor", Technical Report 107/88,Tel Aviv University, May 1988.
10. Gholson N.H. and Moose R.L., "Maneuvering target tracking using adaptive state estimation" , IEEE Tran. on Aerospace and Electronic Systems, Vol AES-13, No 3 , pp.310-317 , May 1977.
11. Lee E.K.B. and Haykin S., "Parallel implementation of the tracking Kalman filter", in Proc. ICASSP 88, pp. 2092-2094.
12. Lemanski W.J., "Parallel Ada implementation of a multiple model Kalman Filter tracking system: a software engineering approach", Master Thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 1989.
13. Mookerjee P., Campo L. ,and Bar-Shalom Y., "Estimation in systems with semi-markov switching model" , Proc. of the 26th Conf. on Decision and Control, Los Angeles CA, 1987.
14. O'Hallarin D.R. and R.S. Baheti, "Parallel Implementation of a Kalman Filter on the Warp Computer", in Proc. ICPP 88, Vol. 3, pp. 108-111.
15. Rabiner L.R and Juang B.H, "An introduction to hidden Markov models", IEEE ASSP magazine, pp. 4-16, Jan 1986.
16. Schnepper K., "A comparison of GLR and multiple model filters for target tracking problem", Proc. of the 25th Conf. on Decision and Control, Athens, Dec 1986.
17. Tugnait J.K., "Detection and estimation for abruptly changing systems", Automatica, Vol 18, No 5, pp.607-615, 1982.
18. Tugnait J.K. and Haddad A.H., "A detection estimation scheme for state estimation in switching environments", Automatica , Vol 15, pp.477-481, 1979.
19. Willsky A.S., "Detection of abrupt changes in dynamic systems" in *Detection of abrupt changes in signals and dynamical systems (Edt.)*, Bassenville and Beneviste, Heidelberg : Springer-Verlag, 1985.