AN OPTIMIZATION-BASED PARALLEL PARTICLE FILTER FOR MULTITARGET TRACKING

BY

S. SUTHARSAN,
B.Sc.Eng. (UNIV. OF PERADENIYA, SRI LANKA)

An Optimization-Based Parallel Particle Filter for Multitarget Tracking

AN OPTIMIZATION-BASED PARALLEL PARTICLE FILTER FOR MULTITARGET TRACKING

By

S. SUTHARSAN, B.Sc. (Eng.)
University of Peradeniya, Sri Lanka, 2001

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Applied Science

McMaster University

MASTER OF APPLIED SCIENCE (2005) MCMASTER UNIVERSITY

(Electrical and Computer Engineering) Hamilton, Ontario, Canada

TITLE: **An Optimization-Based Parallel Particle Filter for Multitarget Tracking**

AUTHOR: S. Sutharsan

B.Sc. (Eng.)

University of Peradeniya

Sri Lanka, 2001

SUPERVISOR: Dr. T. Kirubarajan

NUMBER OF PAGES: ix, 57

Abstract

Particle filters are being used in a number of state estimation applications because of their capability to effectively solve nonlinear and non-Gaussian problems. However, they have high computational requirements and this becomes even more so in the case of multitarget tracking, where data association is the bottleneck. In order to perform data association and estimation jointly, typically an augmented state vector, whose dimensions depend on the number of targets, is used in particle filters. As the number of targets increases, the corresponding computational load increases exponentially. In this case, parallelization is a possibility for achieving real-time feasibility in large-scale multitarget tracking applications. In this paper, we present an optimization-based scheduling algorithm that minimizes the total computation time for the bus-connected heterogeneous primary-secondary architecture. This scheduler is capable of selecting the optimal number of processors from a large pool of secondary processors and mapping the particles among the selected ones. A new distributed resampling algorithm suitable for parallel computing is also proposed. Furthermore, a less communication intensive parallel implementation of the particle filter without sacrificing tracking accuracy using an efficient load balancing technique, in which optimal particle migration among secondary processors is ensured, is presented. Simulation results demonstrate the tracking effectiveness of the new parallel particle filter and the speedup achieved using parallelization.

To my mother, who sacrificed so much for my well-being,

and to my brother and sister

Acknowledgements

I would like to thank all my friends, family members and others who encouraged and supported me during the research leading to this thesis.

I would like to express my sincere gratitude to my supervisor, Dr. T. Kirubarajan, for giving me an opportunity to work on a very interesting topic and for providing continuous guidance, advice and support throughout the course of this research work. I also thank Dr. Abhijit Sinha for his valuable comments and suggestions.

I thank my fellow graduate students in the ECE department, in particular all my friends in the ETFLab, for their help and support. Further, my sincere thanks are due to the ECE department staff, especially Cheryl Gies and Helen Jachna.

Finally, I would like to thank my family for supporting and encouraging me to pursue this degree.

Contents

1 INTRODUCTION **1**

1.1 Motivation and Contribution of the Thesis 1

1.2 Organization of the Thesis 4

1.3 Related Publications 5

2 OVERVIEW OF TARGET TRACKING **6**

2.1 Target Tracking 6

2.2 Multitarget Tracking 6

2.2.1 Data Association 9

2.3 Filtering Algorithms 10

2.3.1 Kalman Filter 12

2.3.2 Extended Kalman Filter 14

2.3.3 Grid Based Filters 14

2.3.4 Particle Filter 15

3 MULTITARGET PARTICLE FILTER (MPF) **17**

3.1 Sequential Importance Sampling 18

3.2 Degeneracy Problem 18

3.3 Resampling 19

3.4 MPF Algorithm 20

4 PRIMARY-SECONDARY MODEL FOR THE PARTICLE FILTER 22

4.1 Background 22

4.2 Optimality Condition 24

4.3 Approximation 25

5 SCHEDULING PROBLEM FORMULATION 26

6 DISTRIBUTED RESAMPLING 30

6.1 Algorithm 33

7 LOAD BALANCING 35

7.1 Processor Load Evaluation 36

7.2 Load Balancing Profitability Determination 36

7.3 Particle Migration 38

8 PERFORMANCE MEASURES 40

9 SIMULATION RESULTS 42

10 CONCLUSIONS 52

List of Tables

4.1 Notations 23

List of Figures

2.1 Basic elements of a conventional multitarget tracking system. 7

2.2 Validation regions of two well-separated targets. 8

2.3 Validation regions of two closely spaced targets and the measurements inside those validation region. 9

4.1 Primary-secondary architecture 22

4.2 Timing diagram for finely decomposable tasks 25

5.1 Timing diagram for a large number of processors 26

5.2 Timing diagram with optimal mapping 27

7.1 The load balancer 36

9.1 Communication characteristics between the primary and secondary nodes 45

9.2 Time comparison for a single target 46

9.3 Time comparison for two targets 47

9.4 Time comparison for three targets 48

9.5 Efficiency comparison of two parallel algorithms 49

9.6 RMSE comparison for a single target 50

Chapter 1

INTRODUCTION

1.1 Motivation and Contribution of the Thesis

Many real-world tracking problems, such as airborne surveillance, missile defence and maritime surveillance, require the tracking of a large number, possibly, hundreds or even thousands, of mobile objects (targets). For example, in ballistic missile tracking, the problem becomes complicated during the mid-course phase, in which spawning may create several hundreds of new targets from a single target [31]. In ground target tracking, hundreds of targets may be tracked using the Moving Target Indicator (MTI) measurements from airborne platforms [16]. Air traffic control is another tracking application where real-time capability for large-scale scenarios is needed [32]. In these cases, the computational requirement varies according to the number of targets in the surveillance region. Furthermore, with nonlinear target dynamics, nonlinear measurements, and/or non-Gaussian noise, in which case the computationally intensive Particle Filter (PF) [2] [12] is the common choice, the computational burden becomes significant. This limits the deployment of the particle filter, which is also called the Sequential Monte Carlo (SMC) method, in large-scale real-world applications. This is because the high computational time and memory requirements needed

to yield real-time feasible implementations cannot generally be met by uniprocessor systems. In these cases, parallelization is a possibility for a real-time feasible implementation.

Particle filter parallelization has rarely been addressed in the literature and this becomes the primary focus of this thesis. In the classic multitarget particle filter, an augmented state vector of target dynamics, whose dimension depends on the number of targets being tracked, is used. This is in contrast to traditional tracking algorithms like the Kalman Filter (KF), in which case different instances of the filter are run (with the same or different filter parameters) for different targets in a sequential or parallel manner. Kalman filter based techniques consist of a data association phase followed by estimation [3]. In the multitarget particle filter, data association and estimation are done jointly. Thus, multitarget particle filter parallelization is very different from Kalman filter or Interacting Multiple Model (IMM) estimator [4] based parallel multitarget implementations [29] [28], where an Auction based assignment is commonly used to handle measurement-to-track association. For nonlinear systems, existing association techniques like assignment are not directly applicable in conjunction with particle filter based techniques [15]. Instead, augmented state multitarget tracking replaces measurement-to-track association in this case. The Gibbs' sampler based multitarget data association technique, which uses the augmented target states, was introduced in [15]. In [26], a method called Couple Partition (CP), which makes it possible to use the same number of particles even though the number of targets increases, was introduced. In [18] [17] an independent partition technique was introduced together with augmented state. This also handles the tracking when the targets undergo challenging transitions like crossings and convoy movements.

In our current work, the classical multitarget particle filter with augmented state of target dynamics is considered for parallelization in order to achieve real-time feasibility when many targets are present in the scenario. Using a multiprocessor architecture or by connecting several personal computers (i.e., network of workstations) the high computational requirement for problems like

multitarget tracking can be overcome. We consider the problem in a master-slave (or, more appropriately, primary-secondary) topology, which is suitable for both multiprocessor architectures and network of workstations. Previously, scheduling algorithms for multitarget tracking were developed [29] [27] within the IMM-Assignment framework. In this thesis, we are concerned with the mapping of a multitarget particle filter onto a set of single instruction, multiple data stream (SIMD) multiprocessors, wherein the processors may be homogeneous or heterogeneous.

Because of the time-varying nature of multitarget scenarios, we consider the development of a dynamic scheduling algorithm for parallelization. Dynamic scheduling algorithms are computationally costlier than a static one. However, the real-time mapping of particles is required in order to utilize the resources efficiently and to handle the problem satisfactorily when critical situations such as processor failures and changes in the performance of processors occur. Multitarget particle filter is similar to running a single target particle filter except that the state space dimension will vary, which in turn will cause computational and communication loads for particles to vary, with the number of targets. Furthermore, most parallel processor systems are multiuser environments and the performance of each node may vary with users' access to system resources. Thus, a load balancer capable of monitoring the performance of each node and reacting accordingly in real-time is needed.

The task mapping problem is usually NP-hard, i.e., the computational requirement of an optimal solution increases exponentially with the number of tasks and the number of processors. In problems like particle filtering, each particle can be considered as a separate task. The number of particles is typically very high and thus optimal real-time scheduling is not possible. For example, standard mapping technique based on dynamic programming [11] cannot be used in real-time — there exists a need for efficient, possibly approximate or sub-optimal, task mapping solution.

Even though a number of particle filters can be run independently on several processors, there are many issues that need to be addressed for parallelization. In particle filtering methods, the

particles interact due to the resampling step [2] [12] and, thus, are statistically dependent. Consequently, with parallelization, the particles have to combined at the primary node in order to perform resampling. This results in a large amount of data being transmitted between the primary and secondary node processors. That is, resampling creates significant amount of communication at every time step of filtering and it prevents the particle filter from being parallelized efficiently.

Some work has already been done on improving resampling for the parallelization of the particle filter [13] [14] [7]. Even though they have improved the resampling process, they are suitable only for hardware implementations. In the message passing parallel multiprocessor computing interface, it is particularly desirable to reduce the communication overhead.

In [5], another method called the Compressed Distributed Particle Filter (CDPF), which enables significantly less data exchange between the primary and secondary nodes than direct parallelization. The idea is to avoid sending duplicate particles that are generated when resampling. However, there is no guarantee that particle duplication will occur at every time step. In such situations, CDPF will be almost as complex as direct parallelization.

In this paper, we introduce a new Distributed Resampling Particle Filter (DRPF), which requires significantly less communication among the processors while maintaining the estimation performance of the filter at the same level. The optimization based DRPF is complemented with an efficient load balancing algorithm, which is needed in view of the fact that the DRPF may not always generate the optimal number of particles to be scheduled at all secondary nodes.

1.2 Organization of the Thesis

This thesis is organized as follows. In Section 4, we describe the primary-secondary node model used in our formulation together with an approximation that is suitable for multitarget particle filter parallelization. In Section 5, we present the mathematical formulation for the particle filter

mapping problem. The same algorithm can be used for our DRPF mapping as well. Section 6 describes the DRPF method, which minimizes the amount of data being transmitted between the primary and the secondary nodes. The load balancing technique that helps the DRPF operate efficiently is discussed in Section 7. The performance measures used to demonstrate the effectiveness of parallelization are discussed in Section 8. Finally, in Section 9 we present simulation results.

1.3 Related Publications

1. S. Sutharsan, A. Sinha, T. Kirubarajan and M. Farooq, "An Optimization-Based Parallel Particle Filter for Multitarget Tracking", *Proceedings of the SPIE Conference on Signal and Data Processing of Small Targets*, San Diego, CA, August 2005.

2. S. Sutharsan, A. Sinha and T. Kirubarajan, "An Optimization-Based Parallel Particle Filter for Multitarget Tracking", Submitted to *IEEE Transaction on Aerospace and Electronic Systems*, August 2005.

Chapter 2

OVERVIEW OF TARGET TRACKING

2.1 Target Tracking

The process of inferring the value of a quantity of interest from indirect, inaccurate and uncertain observations is called estimation. The tracking process can be described as the task of estimating the state of a target at the current time and at any point in the future. The estimation of the current state of a dynamic system from noisy data is called filtering and estimating the future state is called prediction [4]. In addition to the estimates, the tracking system should produce some measure of the accuracy of the state estimates.

2.2 Multitarget Tracking

A track is a symbolic representation of a target moving through an area of interest. Internally, in the tracking system, a track is represented by a filter state that gets updated on each new measurement.

Figure 2.1 illustrates the basic elements of a conventional Multiple Target Tracking (MTT) system. A signal processing unit converts the signals from the sensor to measurements, which become

the input data to the MTT system. The incoming measurements are used for track maintenance. Track maintenance refers to the functions of track initiation, confirmation, and deletion [3]. Observations not assigned to existing tracks can initiate new tentative tracks. A tentative track becomes confirmed when the number and the quality of the measurements included in that track satisfy a certain confirmation criteria. Similarly, a track that is not updated becomes degraded, and it must be deleted if not updated within some reasonable interval. Gating tests evaluate which possible measurements-to-track pairings are reasonable and a more detailed association technique is used to determine final pairings. After inclusion of new observations, tracks are predicted ahead to the arrival time for the next set of observations. Gates are placed around these predicted positions and the processing cycle repeats.

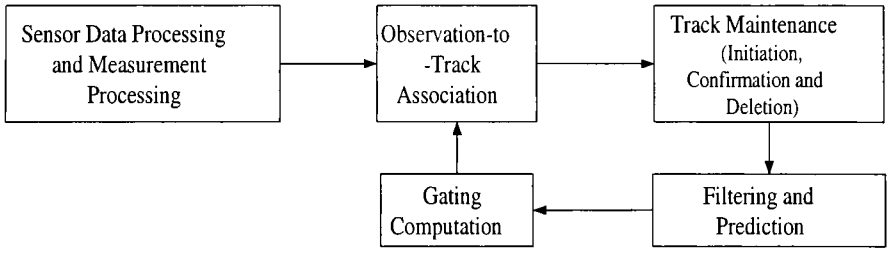

Figure 2.1: Basic elements of a conventional multitarget tracking system.

If the true measurement conditioned on the past is normally (Gaussian) distributed with its probability density function (PDF) given by

$$p(z_{k+1}|Z_k) = \mathcal{N}\left[z_{k+1}; \hat{z}_{k+1|k}, S(k+1)\right] \tag{2.1}$$

where z_{k+1} is the measurement at time $k+1$, $Z_k = [z_1, z_2, \ldots, z_k]$, $\hat{z}_{k+1|k}$ is the predicted (mean) measurement at time $k+1$ and $S(k+1)$ is the measurement prediction covariance, then the true measurement will be in the following region

$$\mathcal{V}(k+1,\gamma) = \{z : [z-\hat{z}_{k+1|k}]' S(k+1)^{-1} [z-\hat{z}_{k+1|k}] < \gamma\} \tag{2.2}$$

with the probability determined by the gate threshold γ.

The region defined by (2.2) is called the gate or validation region ($\mathcal{V}$) or association region. It is also known as the ellipse (or ellipsoid) of probability concentration: the region of minimum volume that contains a given probability mass.

The validation procedure limits the region in the measurement space where the information processor looks to find the measurement from the target of interest. Measurements outside the validation region can be ignored, since they are too far from the predicted location and very unlikely to have originated from the target of interest. It can so happen that more than one measurement is found in the validation region.

Figures 2.2 and 2.3 illustrate the gating for two well-separated and closely-spaced targets, respectively. In the figures, • indicates the expected measurement and the $*$ indicates the received measurement.

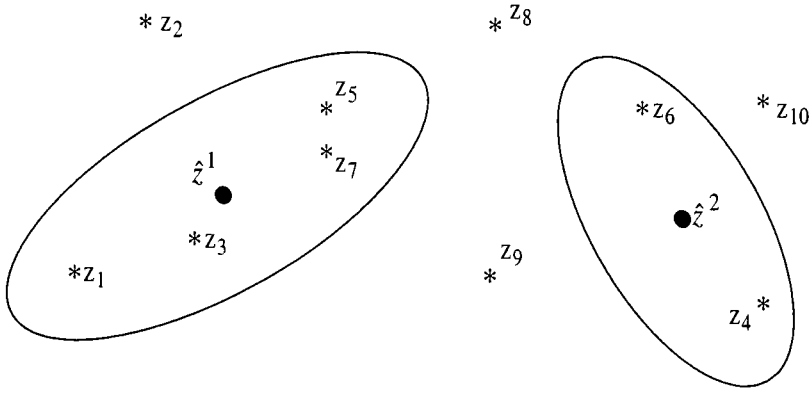

Figure 2.2: Validation regions of two well-separated targets.

Any measurement falling inside the validation region is called a validated measurement. Since

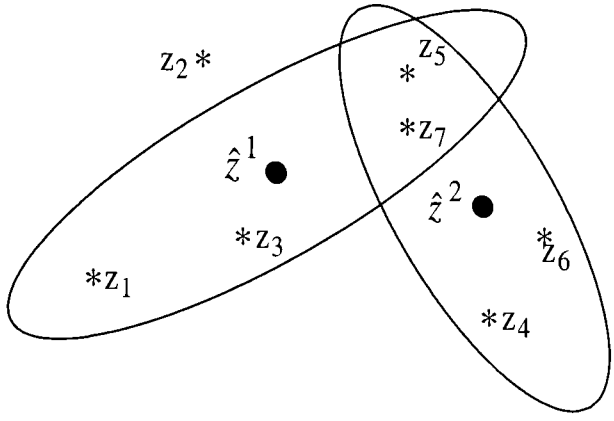

Figure 2.3: Validation regions of two closely spaced targets and the measurements inside those validation region.

more than one measurement is validated in Figure 2.2, there is an association uncertainty. That is, there is ambiguity as to which, if any, of the validated measurements is target originated, assuming that at most one measurements is target generated.

In tracking multiple targets, the problem is basically the same as single target tracking provided the targets are well-separated (Figure 2.2). However, if the targets are closely spaced, which causes the corresponding validation regions to overlap (Figure 2.3), the validation region for a particular target may also contain detections from nearby targets as well as clutter detections. Hence, there is a need for a data association technique in order to resolve the measurement origin uncertainty.

2.2.1 Data Association

The problem of tracking multiple targets in clutter considers the situation where there are possibly several measurements in the validation region of each target. The set of validated measurements consists of:

- the correct measurement (if detected and falls within the gate)
- the undesired measurements: false alarms

Then the problem, which is called as data association, is that of associating the measurements in each validation region with the corresponding track (target). The simplest possible approach, called Nearest Neighbor (NN), is to use the measurement nearest to the predicted measurement as if it were the correct one. An alternative approach, called Strongest Neighbor (SN), is to select the strongest measurement among the validated ones [3].

Since any of the validated measurements could have originated from the target, this suggests that all the measurements from the validation region should be used in some fashion. A Bayesian approach, called Probabilistic Data Association (PDA), associates probabilistically all the "neighbors" to the target of interest [3]. This is a standard technique used for data association in conjunction with the Kalman filter or the extended Kalman filter. The Kalman filter can be applied only if the models are linear and measurement and process noises are independent and white Gaussian. In this thesis a bearing-only tracking scenario is used as the test case. Since this is a nonlinear filtering problem, a particle filter is used to track each target [12]. For bearing-only tracking in a cluttered environment the expected likelihood particle filter has been shown to outperform the gated probabilistic data association (PDA) Extended Kalman Filter (EKF) [23].

2.3 Filtering Algorithms

In order to analyze and make inference about a dynamic system, at least two models are required: First, a model describing the evolution of the state with time (the system model)

$$x_{k+1} = f_k(x_k) + v_k \tag{2.3}$$

and, second, a model relating the noisy measurements to the state (the measurement model)

$$z_k = h_k(x_k) + \omega_k \tag{2.4}$$

where x_k is the state of the target and z_k is the measurement vector at revisit time k. We will assume that these models are available. The probabilistic state-space formulation and the requirement for the updating of information on receipt of new measurements are ideally suited for the Bayesian approach. This provides a rigorous general framework for dynamic state estimation problems.

In the Bayesian approach to dynamic state estimation, one attempts to construct the posterior probability density function of the state based on all available information, including the set of received measurements. Since this PDF embodies all available statistical information, it may be said to be the complete solution to the estimation problem. In principle, an optimal estimate of the state may be obtained from the PDF. A recursive filtering approach means that the received data can be processed sequentially rather than as a batch so that it is not necessary to store the complete data set nor to reprocess existing data if a new measurement becomes available. Such a filter consists of essentially two stages: prediction and update. The prediction stage uses the system model to predict the state PDF forward from one revisit time to the next. Since the state is usually subject to unknown disturbances, prediction generally translates, deforms and spreads the state PDF.

Suppose that the required PDF $p(x_k|Z_k)$ at revisit time k is available, where $Z_k = [z_1, z_2, \ldots, z_k]$. The prediction stage involves using the system model (2.3) to obtain the prior PDF of the state at time $k+1$ and given by

$$p(x_{k+1}|Z_k) = \int p(x_{k+1}|x_k)p(x_k|Z_k)dx_k \tag{2.5}$$

The update operation uses the latest measurement to modify the prediction PDF. At revisit time

k, a measurement z_k becomes available and will be used to update the prior via Bayes' rule

$$p(x_{k+1}|Z_{k+1}) = \frac{p(z_{k+1}|x_{k+1})p(x_{k+1}|Z_k)}{p(z_{k+1}|Z_k)} \tag{2.6}$$

In the above the likelihood function $p(z_{k+1}|x_{k+1})$ is defined by the measurement model (2.4).

The above recursive propagation of the posterior density is only a conceptual solution in that in general, it cannot be determined analytically. Solutions do exist in a restrictive set of cases.

When both $f()$ and $h()$ are linear functions, the problem at hand is a linear problem and otherwise it is a nonlinear one. For a nonlinear problem that has a prior, process noise and measurement noise being Gaussian, the problem is classified as a linear Gaussian problem. This kind of problem presents a very important class that can be tractable in recursive Bayesian estimation framework, which is easily implemented by Kalman filter. For non-linear problems, several classes of estimators have been developed. Extended Kalman Filter (EKF) linearizes the system function and/or observation function at each time step and uses the Kalman filter to do the filtering. Another class deals with nonlinearity and non-Gaussianity by discrete approximations of the posterior density, which is grid-based filters. They associate with each grid point with a probability value. The last class consists of Monte Carlo methods. They do not approximate the posterior density directly, but sample it.

2.3.1 Kalman Filter

The Kalman filter assumes that the state and measurement models are linear and the initial state error and all the noises entering into the system are Gaussian and, hence, parameterized by a mean and covariance [4]. Under the above assumptions, if $p(x_k|Z_k)$ is Gaussian, it can be proved that $p(x_{k+1}|Z_{k+1})$ is also Gaussian.

Then, the state and measurement equations are given by

$$x_{k+1} = F_k x_k + \nu_k \tag{2.7}$$

$$z_k = H_k x_k + \omega_k \tag{2.8}$$

If F_k and H_k are known matrices, $\nu_k \sim \mathcal{N}(0, \Gamma_k)$ and $\omega_k \sim \mathcal{N}(0, \Sigma_k)$, the Kalman filter algorithm can then be viewed as the following recursive relationship [4]

$$p(x_k|Z_k) = \mathcal{N}(x_k; m_{k|k}, P_{k|k}) \tag{2.9}$$

$$p(x_{k+1}|Z_k) = \mathcal{N}(x_{k+1}; m_{k+1|k}, P_{k+1|k}) \tag{2.10}$$

$$p(x_{k+1}|Z_{k+1}) = \mathcal{N}(x_{k+1}; m_{k+1|k+1}, P_{k+1|k+1}) \tag{2.11}$$

where

$$m_{k+1|k} = F_{k+1} m_{k|k} \tag{2.12}$$

$$P_{k+1|k} = \Gamma_k + F_{k+1} P_{k|k} F_{k+1}^T \tag{2.13}$$

$$m_{k+1|k+1} = m_{k+1|k} + K_{k+1}(z_{k+1} - H_{k+1} m_{k+1|k}) \tag{2.14}$$

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1} H_{k+1} P_{k+1|k} \tag{2.15}$$

with

$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + \Sigma_{k+1} \tag{2.16}$$

$$K_{k+1} = P_{k+1|k} H_{k+1}^T S_{k+1}^{-1} \tag{2.17}$$

In the above, $\mathcal{N}(x; m, P)$ is a Gaussian density with argument x, mean m and covariance P

This is the optimal solution to the tracking problem if the above assumptions hold. The implication is that no algorithm can do better than a Kalman filter in this linear Gaussian environment.

In many situations of interest the assumptions made above do not hold. Hence the Kalman filter cannot be used as described above, and approximations are necessary.

2.3.2 Extended Kalman Filter

If the functions in (2.3) and (2.4) are nonlinear, then a local linearization of the equations may be a sufficient description of the nonlinearity. Local linearizations of the above functions are

$$\hat{F}_k = \left.\frac{df_k(x)}{dx}\right|_{x=m_{k-1|k-1}} \tag{2.18}$$

$$\hat{H}_k = \left.\frac{dh_k(x)}{dx}\right|_{x=m_{k|k-1}} \tag{2.19}$$

The EKF is based on the assumption that $p(x_k|Z_k)$ is approximated by a Gaussian. Then all the equations of the Kalman filter can be used with this approximation and the linearized functions [4].

If the true density is substantially non-Gaussian, then a Gaussian approximation can never describe it well. In such cases, particle filters will yield an improvement in performance in comparison to the EKF.

2.3.3 Grid Based Filters

The grid based approach is a numerical method that represents the distributions as numbers on a fixed grid. The grid is simply a large series of points and to each grid-point is associated a number, which is the density of the distribution in the grid-point. Grid-based methods provide the optimal recursion of the filtered density, $p(\mathbf{X}_k|\mathbf{Z}_k)$, if the state space is discrete and consists of a finite number of states. Suppose the state space at time $k-1$ consists if discrete states $\mathbf{X}_{k-1}^i, i = 1, ..., N$. For each state $\mathbf{X}_{k-1}^i$, let the conditional probability of that state, given measurements up to time

$k-1$, be denoted by $w^i_{k-1|k-1}$; that is, $P\{\mathbf{X}_{k-1} = \mathbf{X}^i_{k-1}|\mathbf{Z}_{k-1}\} \triangleq w^i_{k-1|k-1}$. Then, the posterior pdf at $k-1$ can be written as

$$p(\mathbf{X}_{k-1}|\mathbf{Z}_{k-1}) = \sum_{i=1}^{N} w^i_{k-1|k-1}\delta(\mathbf{X}_{k-1} - \mathbf{X}^i_{k-1}) \tag{2.20}$$

where $\delta(.)$ is the Dirac delta function. Then the prediction and update equations will be

$$p(X_k|Z_{k-1}) = \sum_{i=1}^{N} w^i_{k|k-1}\delta(X_k - X^i_k) \tag{2.21}$$

$$p(X_k|Z_k) = \sum_{i=1}^{N} w^i_{k|k-1}\delta(X_k - X^i_k) \tag{2.22}$$

respectively.

The grid must be sufficiently dense to get a good approximation to the continuous state space. As the dimensionality of the state space increases, the computational cost of the approach therefore increases dramatically.

2.3.4 Particle Filter

The particle filter represents the distribution $p(x_k|Z_k)$ as a set of m particles, where each particle consists of an hypothesized state, $x^{(i)}_k$, and an associated weight, $w^{(i)}_k$ [12]. The weights sum to unity over the m particles at each revisit time. From this representation, one can then obtain an estimate of the expected value of any function, $g(x_k)$:

$$\mathbb{E}[g(x_k)] \approx \sum_{i=1}^{m} g\left(x^{(i)}_k\right) w^{(i)}_k \tag{2.23}$$

If $g(x_k) = x_k$ then the above will give an estimator of the mean whereas if

$$g(x_k) = \left(x_k - \mathbb{E}[x_k]\right)\left(x_k - \mathbb{E}[x_k]\right)^T \tag{2.24}$$

it will give an estimate of the covariance.

The detailed description of the multitarget particle filter is given in Chapter 3.

Chapter 3

MULTITARGET PARTICLE FILTER (MPF)

As mentioned in Chapter 1, the augmented state vector of target dynamic is used as particle state in multitarget tracking. In the multitarget particle filter, each particle represents the joint state vector of target dynamics. Let $X_k^j = [x_k^j, \dot{x}_k^j, y_k^j, \dot{y}_k^j]'$ be the state vector of the j^{th} target at time k, then the joint state vector is given by

$$X_k = \begin{bmatrix} \mathbf{x}_k^1 \\ \mathbf{x}_k^2 \\ \vdots \\ \mathbf{x}_k^t \end{bmatrix} \tag{3.25}$$

Here, t is the number of targets in the surveillance region. Further,$\mathbf{x}_k^t$ can also be refereed to as the t^{th} partition of particle.

The Particle Filter [12] (see also [2] and [10]) provides a mechanism for representing the density $p(X_k|Z_k)$ of the state vector X_k at sampling time k as a set of random samples (particles) $\{X_k{}^{(i)} : i = 1,2,\ldots,m\}$, with associated weights $\{w_k{}^{(i)} : i = 1,2,\ldots,m\}$, where m is the number of

particles. It then uses the principle of Importance Sampling to propagate and update these particles and their associated weights as and when new measurements become available [9].

3.1 Sequential Importance Sampling

Importance sampling is a general Monte Carlo (MC) integration method that we apply to perform nonlinear filtering. The resulting sequential importance sampling (SIS) algorithm is a MC method that forms the basis for most sequential MC filters developed to date. This sequential Monte Carlo approach is also known as bootstrap filtering [12], condensation algorithm [21], particle filtering [2]. It is a technique for implementing a recursive Bayesian filter by Monte Carlo simulations. The key idea is to represent the required posterior density function by a set of random samples with associated weights and to compute estimates based on these samples and weights. As the number of samples becomes very large, this Monte Carlo characterization becomes an equivalent representation to the usual functional description of the posterior pdf, and the SIS filter approaches the optimal Bayesian estimator.

3.2 Degeneracy Problem

Ideally, the importance density function should be the posterior distribution $p(\mathbf{X}_k|\mathbf{Z}_k)$ itself. The variance of the importance weights can increase over time. This has a harmful effect on the accuracy of the estimates and may lead to a common problem associated with the Sequential Importance Sampling (SIS) particle filter: the degeneracy phenomenon.In practical terms this means that after a certain number of recursive steps, all but one particle will have negligible normalized weights. The degeneracy implies that a large portion of the computation is devoted to updating particles whose contribution to the approximation of $p(\mathbf{X}_k|\mathbf{Z}_k)$ is almost zero. One suitable measure of

degeneracy of an algorithm is the effective sample size N_{eff}, which can be estimated as follows:

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N} (w_k^i)^2} \tag{3.26}$$

where w_k^i is the normalized weight. It is straightforward to verify that $1 \leq N_{eff} \leq N$ with the following two extreme cases:

- If the weights are uniform (i.e., $w_k^i = \frac{1}{N}$ for $i = 1, ..., N$) then $N_{eff} = N$
- if $\exists j \; \varepsilon \{1, ..., N\}$ such that $w_k^j = 1$, and $w_k^i = 0$ for all $i \neq j$, then $N_{eff} = 1$

Hence, small N_{eff} indicates a severe degeneracy and vice versa. The next subsection presents a strategy to overcome degeneracy of samples in SIS.

3.3 Resampling

Whenever significant degeneracy is is observed (i.e., when N_{eff} falls below some threshold N_{thr}, resampling is required in the SIS algorithm. Resampling eliminates samples with low importance weights and multiplies samples with high importance weights. It involves a mapping of random measures $\{\mathbf{X}_k^i, w_k^i\}$ into random measure $\{\mathbf{X}_k^{i*}, \frac{1}{N}\}$ with uniform weights. The new set of random samples $\{\mathbf{X}_k^{i*}\}_{i=1}^N$ is generated by the resampling N times from an approximate discrete representation of $p(\mathbf{X}_k|\mathbf{Z}_k)$ is given by

$$p(\mathbf{X}_k|\mathbf{Z}_k) \approx \sum_{i=1}^{N} w_k^i \delta(\mathbf{X}_k - \mathbf{X}_k^i) \tag{3.27}$$

so that $P\{\mathbf{X}_k^{i*} = \mathbf{X}_k^j\} = w_k^j$. The resulting sample is an i.i.d. sample from the discrete density 3.27, and hence the new weights are uniform.

Although the resampling step reduces the effects of degeneracy, it introduces other practical problems. First, it limits the opportunity to parallelize the implementation since all the particles must be combined. Second, the particle that have higher weights w_k^i are statistically selected many times. This leads to a loss of diversity among the particles as the resultant sample will contain many repeated points. This problem is known as sample impoverishment, and it is severe in the case where the process noise in state dynamics is very small. It leads to the situation where all particles will collapse to a single point within a few iterations. However, this problem can be overcome by dithering. Third, since the diversity of the paths of the particles is reduced, any smoothed estimate based on the particles' paths will degenerate.

3.4 MPF Algorithm

In this thesis we take the Importance Density to be the prior $p(X_k|X_{k-1})$ and use the method of Sampling Importance Resampling (SIR) to produce a sample of equally weighted particles that approximate $p(X_k|Z_k)$, i.e.,

$$p(X_k|Z_k) \approx \frac{1}{m}\sum_{i=1}^{m}\delta(X_k - X_k{}^{(i)}) \tag{3.28}$$

where $\delta(.)$ is the Dirac delta function. The SIR method works as follows:

- **Prediction**: For each particle $X_{k-1}^{(i)}$, generate $\nu_{k-1}^{(i)}$ according to the known distribution of the transition noise and then a sample $X_{k|k-1}^{(i)}$ from the prior distribution $p(X_k|X_{k-1})$ can be obtained using the state propagation equation.

- **Weighting:** The information given by the observation can be utilized to find the importance weights. Each particle is given an importance weight $w_k{}^{(i)}$ using the formula

$$w_k{}^{(i)} = p(Z_k|X_{k|k-1}^{(i)}) \tag{3.29}$$

- **Resampling:** The weighted samples will be resampled to eliminate those with low weights, multiply those with high weights and regenerate those with equal weights. The new m particles are sampled with replacement from $\{X_{k|k-1}^{(1)}, X_{k|k-1}^{(2)}, \ldots, X_{k|k-1}^{(m)}\}$ so that the probability of sampling particle i is proportional to ${w_k}^i$. Then new samples $\{X_k^{(1)}, X_k^{(2)}, \ldots, X_k^{(m)}\}$ will have equal weights $(1/m)$.

At each stage, the mean of the posterior distribution is used to determine an estimate $\hat{X}_k$ of the target state X_k, i.e.,

$$\hat{X}_k = \mathbb{E}[X_k|Z_k] \tag{3.30}$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} {X_k}^{(i)} \tag{3.31}$$

A common problem with the SIR filter is the degeneracy phenomenon, where the particle set quickly collapses to just a single particle. To overcome this problem, regularization [24] can be used.

Chapter 4

PRIMARY-SECONDARY MODEL FOR THE PARTICLE FILTER

4.1 Background

Most tracking algorithms can be considered as iterative or recursive ones, in which the same set of steps is repeated at each update time with different parameters. For example, in particle filtering algorithms, importance sampling, prediction, update, and resampling steps occurs over and over again. Thus, multitarget particle filter can be treated as iterative computation and, in order to parallelize, the scheduling of processes across a number of processors is considered at each scan.

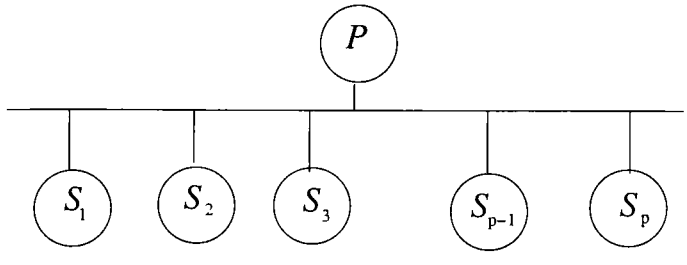

Figure 4.1: Primary-secondary architecture

Symbol	*Description*
T_i^c	Time required to communicate one particle from the primary processor to secondary processor i
T_i^p	Time required to perform the necessary computations on one particle by processor i
β_i	Latency (overhead) of processor i to initiate communication from the primary processor
n_i	Number of particles assigned to processor i
N	Total number of particles
R_i	Total communication time from the primary processor to secondary processor i
S_i	Total communication time from secondary processor i to the primary processor.
C_i	Total computation time for secondary processor i
p	Total number of available processors
q	Selected number of processors for computation

Table 4.1: Notations

In the primary-secondary node model that we consider, since the secondary processors are connected via a bus architecture, exclusive mode communication is possible. That is, at most one communication will take place between the primary node and a secondary node at any time step [6]. Furthermore, we characterize point-to-point communication by the classical linear model (given by $\beta_i + n_i T_i^c$) [11] and the computing power is assumed to be constant and can be written as $n_i T_i^p$, where n_i is the number of particles to be assigned to secondary processor i and other parameters are constant. The communication latency β_i is significant especially for networks of workstations. The actual latency and transmission times are dependent upon the specific computer system. Although the processors are inter-connected via the bus architecture, the processors themselves and the inter-processor communication among them may be heterogeneous. The communication speed depends not only on the interconnection architecture, but also on the power of the processors as well. Our model is developed to specifically handle the heterogeneity in the network due to the differences in the communication speeds of secondary processors.

Scheduling of primary-secondary computations has been studied in [11] and [19]. However, they consider the problem of mapping tasks onto a fixed number of processors, which is not realistic in all problems. In addition, they do not consider the effect of communication loads. In particle filtering, using a huge processor pool for computation is not always efficient. The excessive communication between processors may result in some processors being idle. In order to make the best use of the available communication bandwidth and processor power, optimal resource allocation has to be carried out in an efficient manner. In the multitarget particle filter, the data transferred between the primary and the secondary nodes vary according to the number of targets. Therefore, the challenge here is to select the number of processors and to find the number of particles to be mapped onto the selected processors. As the number of targets increases, the computational requirement for each particle increases exponentially. This is due to the data association taking place in each particle. Thus, the optimal number of processors will be different for different number of targets. Because of the communication overhead, using an unnecessary number of processors will only degrade performance.

4.2 Optimality Condition

In the optimal scheduling, if we assume that the total work can be divided into finely decomposable tasks (divisible load), the idling time is zero and the corresponding timing diagram for optimal mapping is shown in Figure 4.2. The communication mode is exclusive and thus, the primary processor can send or receive data to or from only one secondary node at a time. With optimal scheduling, after each node performs its computation, there may not be any idling [25]. However, this assumption is valid only if the load is finely divisible.

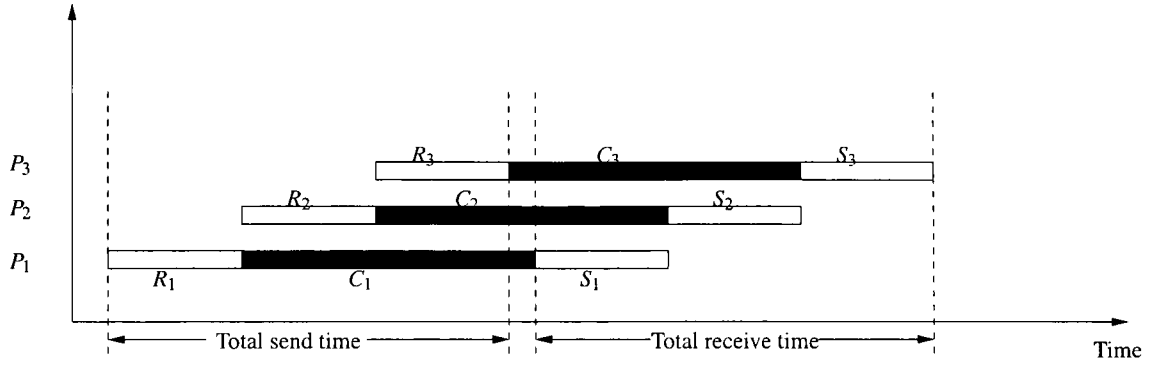

Figure 4.2: Timing diagram for finely decomposable tasks

4.3 Approximation

Particle scheduling of a particle filter is a special case of parallel task mapping. Since we have a large number of small tasks of equal computational complexity, it is reasonable to assume that our tasks can be decomposed finely. If the tasks are not identical, the above assumption is no longer valid. In typical multitarget tracking problems, we use thousands of particles, and the assumption of fine decomposability and the subsequent approximations are valid. If the load is not divisible finely as in the particle filtering case, the computational load cannot be adjusted so as to avoid processor idling. We show in the following section that this assumption makes the problem real-time feasible.

Chapter 5

SCHEDULING PROBLEM FORMULATION

With a large pool of processors in the system and problems like particle filtering, where computation-to-communication ratio (CCR) is low, the timing diagram becomes as shown in Figure 5.1. In this case, the primary processor cannot start receiving data from a secondary processor even after the previous secondary processor finishes computation because of high communication load. Under these circumstances, using all available processors for computation is not efficient. Then one has

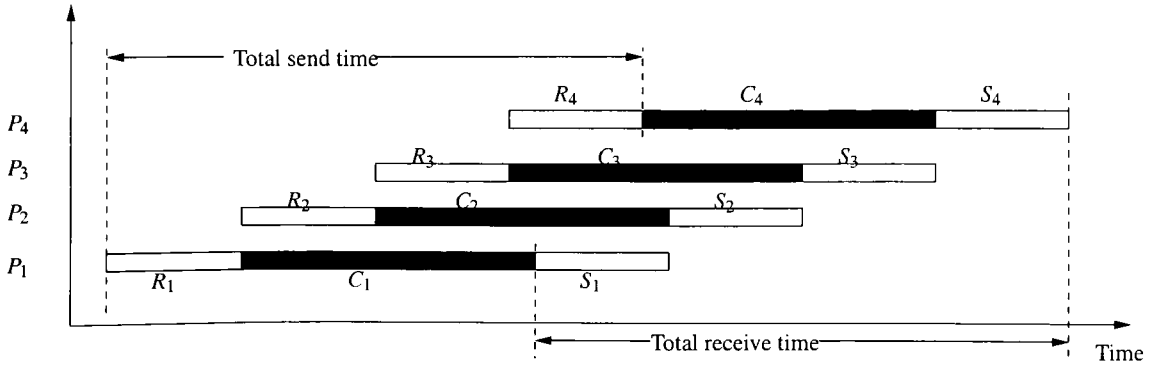

Figure 5.1: Timing diagram for a large number of processors

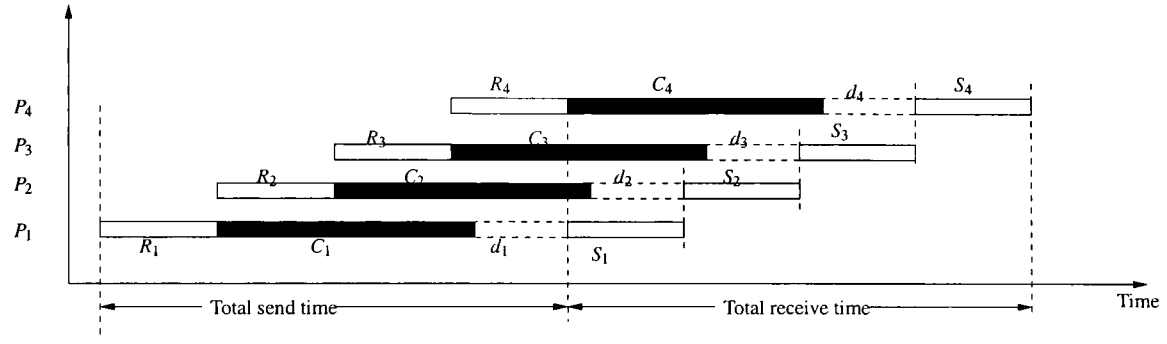

Figure 5.2: Timing diagram with optimal mapping

to go an for optimization formulation to find the optimal number of processors and number of particles to be mapped among the selected processors. From Figure 5.2 we can write the problem as

$$\min R_1 + C_1 + d_1 + \sum_{i=1}^{p} S_i \tag{5.32}$$

subject to:

$$\sum_{j=1}^{p} n_j = N \tag{5.33}$$

$$d_i \geq 0 \tag{5.34}$$

$$C_1 + d_1 - \sum_{i=2}^{p} R_i \geq 0 \tag{5.35}$$

$$C_i + S_i + d_i - R_{i+1} - C_{i+1} - d_{i+1} = 0, \quad \text{i=1 ... p-1} \tag{5.36}$$

Here,

$$R_i = \beta_i + n_i T_i^c \tag{5.37}$$

$$S_i = \beta_i + n_i T_i^c \tag{5.38}$$

$$C_i = n_i T_i^p \tag{5.39}$$

$$\tag{5.40}$$

The above formulation will select the required number of secondary processors as well as the number of particles to be mapped on each selected processor. However, the latencies of the unused secondary processors are included in the above formulation. We introduce a binary variable ρ_i, which indicates the selected processors, and eliminate the unused latency in the formulation. Note that

$$\rho_i = \begin{cases} 1 & \text{if processor } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{5.41}$$

Therefore, the optimization problem will be as follows:

$$\min_{n_i,\rho_i} \quad \beta_1\rho_1 + n_1 T_1^c + n_1 T_1^p + d_1 + \sum_{i=1}^{p} (\beta_i\rho_i + n_i T_i^c + \rho_i T_{\text{penalty}}) \tag{5.42}$$

subject to:

$$\sum_{j=1}^{p} n_j = N \tag{5.43}$$

$$d_i \geq 0 \tag{5.44}$$

$$n_i T_i^p + d_1 - \sum_{i=2}^{p} (\beta_i\rho_i + n_i T_i^c) \geq 0 \tag{5.45}$$

$$n_i T_i^p + \beta_i\rho_i + n_i T_i^c + d_i - \beta_{i+1}\rho_{i+1} \tag{5.46}$$

$$-n_{i+1} T_{i+1}^c - n_{i+1} T_{i+1}^p - d_{i+1} = 0 \tag{5.47}$$

$$n_i \leq \rho_i N \tag{5.48}$$

In (5.42), we introduce a penalty term T_{penalty}, which is necessary to account for the communication overhead and to determine the resulting optimum number of processors. As we increase the number of processors, there should be a significant improvement in overall computation time.

This T_{penalty} will decide how beneficial it is to add one more processor for computation while taking into account the communication overhead. In order to add a processor, the reduction in the overall computation time should be greater than T_{penalty}. In the simulation results, it is clearly seen that we have the problem of diminishing returns in terms of the overall computational time as we increase the number of processors.

Chapter 6

DISTRIBUTED RESAMPLING

The obvious impediment in the direct parallelization of the particle filter is that a huge amount of data has to be sent back and forth at every time step among the processors. This is done in order to perform resampling, which avoids the degeneracy of particles. The resampling step statistically multiplies and/or discards particles at each time step to adaptively concentrate particles in regions of high posterior probability. In tracking problems, the data have to be combined at every time step and in this section we introduce a less communication intensive method, which does not require the particles to be sent between the primary and secondary nodes. Furthermore, in this method, the resampling can be performed at each secondary node instead of doing it at the primary one. The target pdf is distributed among the secondary nodes and the data from all secondary nodes are combined at the primary one to find the estimates of targets. The idea is to have the pdf of target motion independently at each secondary node and combine the local estimates to get the global estimate, which is discussed in [5]. This is very similar to running several particle filters independently with fewer particles and finding the the overall estimate by combining the local estimates. However, in this case, there will be a tendency for filter divergence as the number of particles in each local node is not enough to prevent particle depletion. In [5], another method

called the Compressed Distributed Particle Filter (CDPF), which enables significantly less data exchange between the primary and secondary nodes than direct parallelization. The idea is to avoid sending duplicate particles that are generated when resampling. However, there is no guarantee that particle duplication will occur at every time step. In such situations, CDPF will be almost as complex as direct parallelization.

In [5] another algorithm known as the Local Distributed Particle Filter (LDPF) was also presented. In LDPF samples are drawn and importance weights are calculated and normalized at secondary nodes. Further, resampling is also performed locally, thus the need for sending particles to the primary node is avoided. Each node sends only the local estimate to the primary node, where global estimate is calculated. However, there is no exchange of information between the particles at various nodes before resampling (i.e., resampling is done independently at each node), which degrades the tracking performance. Simulation results for a selected problem in [5] show that the LDPF performs better than the CDPF.

In this paper, we introduce the Distributed Resampling Particle Filter (DRPF), which reduces the amount of data exchange among the processors. In this method, a modified version of resampling (compared with the standard one) is performed at secondary nodes. In the multitarget particle filter, each particle represents the joint state vector of target dynamics. Let $\mathbf{x}_k^j = [x_k^j, \dot{x}_k^j, y_k^j, \dot{y}_k^j]'$ be the state vector of j^{th} target at time k, then the joint state vector is given by

$$X_k = \begin{bmatrix} \mathbf{x}_k^1 \\ \mathbf{x}_k^2 \\ \vdots \\ \mathbf{x}_k^t \end{bmatrix} \tag{6.49}$$

Here, t is the number of targets in the surveillance region. Further, $\mathbf{x}_k^t$ can also be refereed to as the

t^{th} partition of particles. The prediction of particles at time k can be written as

$$X_k = \begin{pmatrix} A_k^1(\mathbf{x}_{k-1}^1, \mathbf{w}_k^1) \\ A_k^2(\mathbf{x}_{k-1}^2, \mathbf{w}_k^2) \\ \vdots \\ A_k^t(\mathbf{x}_{k-1}^t, \mathbf{w}_k^t) \end{pmatrix} \tag{6.50}$$

Let $\{X_k^{i_s}, w_k^{i_s}\}_{i=1}^{N_s}$ denote the characterization of the joint posterior pdf $p(X_k^s|z_k)$ of targets in node s, where $w_k^{i_s}$ are the unnormalized weights of each particle.

Let $\{\tilde{X}_k^{s,i_s}, \tilde{w}_k^{s,i_s}\}_{i_s=1}^{N_s}$ denote the set of importance sampling and their unnormalized weights of node s

$$\hat{X}_k^s = \sum_{i_s=1}^{N_s} w_k^{s,i_s} \tilde{X}_k^{s,i_s} \tag{6.51}$$

$$P_k^s = \sum_{i_s=1}^{N_s} w_k^{s,i_s} (\tilde{X}_k^{s,i_s} - \hat{X}_k^s)(\tilde{X}_k^{s,i_s} - \hat{X}_k^s)' \tag{6.52}$$

where

$$w_k^s = \sum_{i_s=1}^{N_s} \tilde{w}_k^{s,i_s} \tag{6.53}$$

$$w_k = \sum_{s=1}^{S} w_k^s \tag{6.54}$$

$$w_k^{s,i_s} = \tilde{w}_k^{s,i_s} / w_k \tag{6.55}$$

$$\tilde{w}_k^s = w_k^s / w_k \tag{6.56}$$

From the above, the primary node will find the global estimates as follows:

$$\hat{X}_k = \sum_{s=1}^{N_s} \tilde{w}_k^s \hat{X}_k^s \tag{6.57}$$

$$P_k = \sum_{s=1}^{N_s} \tilde{w}_k^s [P_k^s + (\hat{X}_k^s - \hat{X}_k)(\hat{X}_k^s - \hat{X}_k)'] \tag{6.58}$$

In the distributed resampling method, the resampling is performed at each node s according to N_{eff}^s. At each time step, once the summation of local particle weights is available at the primary processor, the load balancer is invoked. Depending on the predicted number of particles that will be available after resampling, the load balancer decides how the particles should be migrated among the nodes.

6.1 Algorithm

- At the primary node
  - Send measurement set (z_k) to each node $s = 1, 2 \cdots S$
- At each secondary node $s = 1, \cdots, S$
  - Perform importance sampling on X_{k-1}^s to obtain $\bar{X}_k^s$
  - Evaluate importance weights $\tilde{w}_k^s$
  - Evaluate sum of local weights w_k^s
  - Compute the local estimate $\hat{X}_k^s$
  - Send $(\hat{X}_k^s, w_k^s)$
- At the primary node
  - Send $w_k = \sum_{s=1}^{S} w_k^s$ to each node $s = 1, \cdots, S$

- Compute the global estimate $\hat{X}_k$
- If number of targets changes, call the scheduler for rescheduling
- Else call load balancer when the system is not balanced

- At each secondary node $s = 1, \cdots, S$
  - Compute the number of samples $N_k^s = [N(\tilde{w}_k^s)]$
  - Resample the particles and get N_k^s particles
  - Perform particle migration if the primary node requests it

In the distributed resampling particle filter, the number of particles for a specific node varies after resampling. Thus, it creates a load imbalances in the previously balanced system. Therefore, an efficient load balancing technique is required to balance system load and to enable the efficient use of multiple processors. The number of particles on each node after resampling can be calculated once the local wights from each node arrives at the primary processor. While the resampling occurs at secondary nodes, the load balancer can be invoked to estimate the number of particles to be migrated. This will facilitate load balance across the network. The efficient load balancing algorithm is explained in the following section.

Chapter 7

LOAD BALANCING

In the primary-secondary mapping strategy, we developed algorithms for selecting the number of processors and the number of particles for the corresponding secondary nodes. However, in our distributed resampling method at every time step after resampling is done, the number of particles will not be as optimal at each node. Therefore, we need to consider load balancing by migrating the particles among the set of selected secondary nodes. Since the resampling is a random process, it is impossible to estimate a priori the number of particles residing at each node after resampling. Therefore, the load balancing in the parallel particle filter has to be done dynamically at real time. A general four-phase dynamic load balancing (DLB) model is presented in [22] and another approach is explained in [8].

In this paper, we present a new DLB algorithm for parallel particle filtering, which enables us to have the stable load-balanced system obtained by primary-secondary task mapping as explained in Section 5.

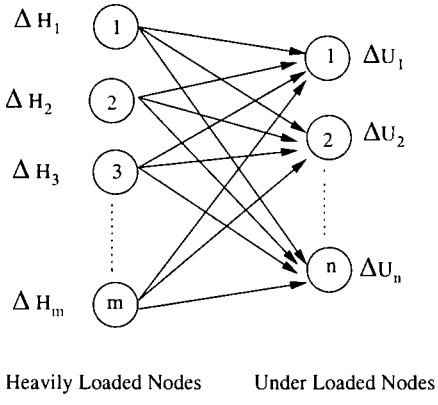

Figure 7.1: The load balancer

7.1 Processor Load Evaluation

The load at each processor has to be determined at each time step of particle filtering in order to maintain the optimal number of particles at each node. These load values are used as inputs to the load balancer to detect load imbalances and to perform task migration. In the distributed resampling method, the load (number of particles) at each node after resampling can be estimated at the primary node using the corresponding weights from each node. This enables us to run the load balancer while the secondary nodes perform resampling. The predictable characteristic of the number of particle at each node is of great benefit for the load balancer in deciding how the particles have to be moved among the processors. The load of node i at a particular step L_i can be written as

$$L_i = \frac{w_i \times N}{\sum_{j=1}^{q} w_j} \times \text{computation per particle} \tag{7.59}$$

7.2 Load Balancing Profitability Determination

In this phase, we estimate the potential speedup which is obtainable through load balancing. The *load imbalance factor* $\Phi(t)$ is used in the profitability determination process, which is weighted

against load balancing overhead to determine whether the load balancing should be performed or not. Therefore, we do load balancing if $\Phi(t) > T_{\text{overhead}}$

$$\Phi_i(t) = T_{\text{unbalance}} - T_{\text{bal}} \tag{7.60}$$

We use our primary-secondary model to determine the *load imbalance factor* and the overhead and then decide whether to invoke the load balancer or not. The load balancing decision has to be taken within a very short time since the load balancer is invoked in real time. T_{bal} is already known and $T_{\text{unbalance}}$ can be calculated using the primary-secondary model. The load balancing has to be done so as to minimize T_{overhead}, which requires an efficient task migration strategy.

In the primary-secondary mapping model, we considered the optimality condition for the balanced system to schedule the particles. However, those equations are is not useful in finding the time required to compute when the system is unbalanced. The total time T_p required for balanced or unbalanced systems can be written as [11]

$$T_p = R_1 + C_1 + \sum_{i=1}^{p} S_i + \sum_{i=1}^{p} \max(0, d_i) \tag{7.61}$$

where $d_1 = 1$ and, for $i > 1$,

$$d_i = \begin{cases} g_i, & \text{if } d_{i-1} \geq 0 \\ g_i - |d_{i-1}|, & \text{otherwise} \end{cases} \tag{7.62}$$

with $g_i = (R_i + C_i) - (C_{i-1} + S_{i-1})$. d_i holds the accumulated delay between the first $i-1$ processors and processor i and g_i stands for the gap between processor $i-1$ and processor i.

However, the profitability determination mentioned above needs a lot of computation. The unbalanced time and the particle migration pattern are to be calculated at every scan to determine whether the load balancing has to be done or not. The whole exercise of distributed resampling

method may be rendered meaningless if we do not reduce the total computation time. Therefore, we use following method to decide the load balancing requirement.

In this method, instead of considering the load at the current time step, we consider the past few steps of load (i.e., particles) on each node from the time particle migration occurred previously. Let Δ be the number of overloaded particles or under-loaded particles at processor p_i. Furthermore, if $\Delta_i > 0$ the overloaded particles $\Delta H_i = \Delta$ and if $\Delta_i < 0$, the under-loaded particles $\Delta U_i = |\Delta|$. Thus, we can write the load imbalance factor as

$$\phi(k) = \sum_{i=1}^{q} |\Delta| \tag{7.63}$$

In this method, the load balancer is invoked when $\phi(k) > \phi_{\text{Threshold}}$.

7.3 Particle Migration

Particle migration in the load balancing phase can be formulated as a search for appropriate pairing between processors that are heavily loaded and those that are under-loaded. The first task is to classify the processor pool as overloaded or under-loaded. This can be determined very easily using the number of optimally scheduled particles on each node and the estimated number of particles that will be generated after resampling on each node. The second task is handing particle migration or mapping between overloaded and under-loaded processors. Particle migration determines the processors that are involved in load transfer and the number of particles that are moved during each transfer. The communication overhead associated with particle migration depends on the communication mechanisms supported by the parallel machine. The goal is to minimize this overhead. Then,

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (\beta_{ij}\rho_{ij} + c_{ij}n_{ij}) \tag{7.64}$$

Subject To:

$$\sum_{j=1}^{n} n_{ij} = \Delta H_i \, \forall i \tag{7.65}$$

$$\sum_{i=1}^{m} n_{ij} = \Delta U_j \, \forall j \tag{7.66}$$

Here,

c_{ij} is the time taken to communicate one particle from processor i to ptocessor j

β_{ij} is the latency between processor i and j

n_{ij} is the number of particles to be migrated between processor i to processor j.

Chapter 8

PERFORMANCE MEASURES

The performance measures that show the effectiveness of a parallel algorithm are the speedup factor and parallel efficiency. These measures depend on how a given set of tasks is mapped onto the multiprocessor architecture. The speedup of a parallel algorithm is given by

$$\lambda = \frac{\text{Execution time using one processor}}{\text{Execution time on a multiprocessor system}} \tag{8.67}$$

The algorithm efficiency η is given as

$$\eta = \frac{\text{Actual speedup}}{\text{Number of processors used (q)}} \tag{8.68}$$

$$= \frac{\lambda}{q} \tag{8.69}$$

For a bus-connected homogeneous processor system, our scheduling algorithm selects q number of processors out of p existing processors according to the number of targets present in the scenario. The scheduler schedules n_i number of particles on each selected processor, and thus, the total number of particles N can be written as $\sum_{i=1}^{q} n_i = N$. Also, Tc_i is the average computational time taken to perform computation on one particle at node P_i. Thus, from the above definition of

speedup, the maximum achievable speedup $\bar{\lambda}$ of our algorithm can be given as

$$\bar{\lambda} = \frac{N \min_i Tp_i}{\beta_1 + n_1(Tc_1 + Tp_1) + \sum_{i=1}^{q}(\beta_i + n_i Tc_i)} \tag{8.70}$$

In the following, we assume that the idling time of each processor is negligible and that the optimality condition is satisfied. Further, in the proposed DRPF method, the whole set of particles are not transmitted and the maximum achievable speedup will be

$$\bar{\lambda} = \frac{N \min_i Tp_i}{\beta_1 + mTm_1 + n_1 Tp_1 + \sum_{i=1}^{q}(\beta_i + eTe_i)} \tag{8.71}$$

Here, m is the average number of measurements received at each scan and e is the number of targets within the surveillance region. Note that the additional overhead in performing the load balancing is not considered in the equation for the speedup $\bar{\lambda}$ of the DRPF. Therefore, the maximum speedup will not be achieved, i.e., unity efficiency is not possible. However, simulation results will show that the speedup achieved using the DRPF is much higher than that of the exact implementation of the particle filter.

Chapter 9

SIMULATION RESULTS

This section presents a two dimensional tracking example to illustrate the new mapping technique and to compare the distributed resampling with the direct implementation of the parallel particle filter. The single target Markov transition model that characterizes the j^{th} target dynamic at time $k$is given by

$$\mathbf{x}_k^j = A_k^j \mathbf{x}_{k-1}^j + \mathbf{w}_k^j \tag{9.72}$$

where $\mathbf{x}_k^j = [x_k^j, \dot{x}_k^j, y_k^j, \dot{y}_k^j]$ is the state of the the j^{th} target, which consists of target position (x_k^j, y_k^j) and target velocity $(\dot{x}_k^j, \dot{y}_k^j)$ at time step k, and $\mathbf{w}_k^j$ is an i.i.d. sequence of zero-mean Gaussian noise vectors with covariance Σ_k^j. The matrices A_k^j and Σ_k^j are given by

$$A_k^j = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.73}$$

and

$$\Sigma_k^j = \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} & 0 & 0 \\ \frac{T^2}{2} & T & 0 & 0 \\ 0 & 0 & \frac{T^3}{3} & \frac{T^2}{2} \\ 0 & 0 & \frac{T^2}{2} & T \end{bmatrix} l \tag{9.74}$$

where $l = 1\mathbf{x}10^{-4}\mathrm{m}^2\mathrm{s}^{-3}$.

The observations are taken from a single sensor located at the origin of the reference coordinate system. The measurements are available at discrete time sampling interval $T = 5$ seconds. The target-generated measurements corresponding to target j on sensor i are given by

$$z_k^{i,j} = \begin{bmatrix} \sqrt{(x_k^j - x_S^i)^2 + (y_k^j - x_S^i)^2} \\ \tan^{-1}((y_k^j - y_S^i)/(x_k^j - x_S^i)) \end{bmatrix} + v_k^i \tag{9.75}$$

where v_k^i is zero-mean Gaussian noise vector of dimension 2 with covariance diag[1×10^4 m^2, 3×10^{-4} rad^2]. (x_k^j, y_k^j) and (x_S^i, y_S^i) denote the locations of target j and sensor i at time step k, respectively.

We consider a varying number of targets in the scenario to illustrate parallelization efficiency. Simulation results shows how the number of processors are selected depending on the number of targets in the surveillance region. Furthermore, the selection of processors is different for the distributed resampling method. This is due to the communication reduction in the proposed algorithm.

The parallel platform that we use to analyze the performance of our algorithm consists of a cluster of 128 nodes with dual 2.4 GHz Pentium Xeon processors and 1 GB of memory on each node. The cluster is connected via a high speed bus of 1 GB/s. All machines run the Debian Linux operating system. The communication characteristics of the processors is determined by a ping-pong experiment between them. Figure 9.1 shows the communication characteristics of the

parallel architecture on which we perform our simulations. The latency β and the communication speed τ are calculated from the communication characteristics graph. The time required for one particle on each node is calculated by taking the average time on each node.

The simulation results given below show that the optimal number of processors varies depending on the number of targets. In order to justify the optimal number of processors we get using our algorithm, we fix the number of processors and change $n_i \leq \rho_1 N$ to $n_i > 0$ in (5.48) and find the number of particles on each processor. Now we run the filter according to the above mapping and find the required time for computation. By doing this, it is possible to explain how meaningful the solution we get from the mapping algorithm is.

As the dimension of the state vector of the particle filter increases, the data transmission between processors increases linearly. However, the computational requirement for each particle increases exponentially due to the data association in each particle. Therefore, the optimal number of processors does not vary linearly with the number of targets, which we can clearly observe from the simulation results in Figures 9.2, 9.3 and 9.4. In the distributed resampling implementation, the computational time is always below the direct parallel implementation. This time reduction is achieved by reducing the huge amount of data transmission at each time step.

Figure 9.5 shows the efficiency of the parallel algorithm. From the curves we can note that the distributed resampling method is highly efficient — it reduces data transfer significantly and selects the number of processors more efficiently than the direct parallel implementation of the particle filter.

Figure 9.6 compares the root mean squared errors (RMSE) of the two methods over 50 Monte Carlo runs. Both methods show almost the same error. That is, the distributed resampling implementation of the particle filter does not result in any performance degradation.

Because of the approximation resulting from the independent resampling at local nodes, LDPF's performance is worse than that of the the serial particle filter (although the former outperforms the

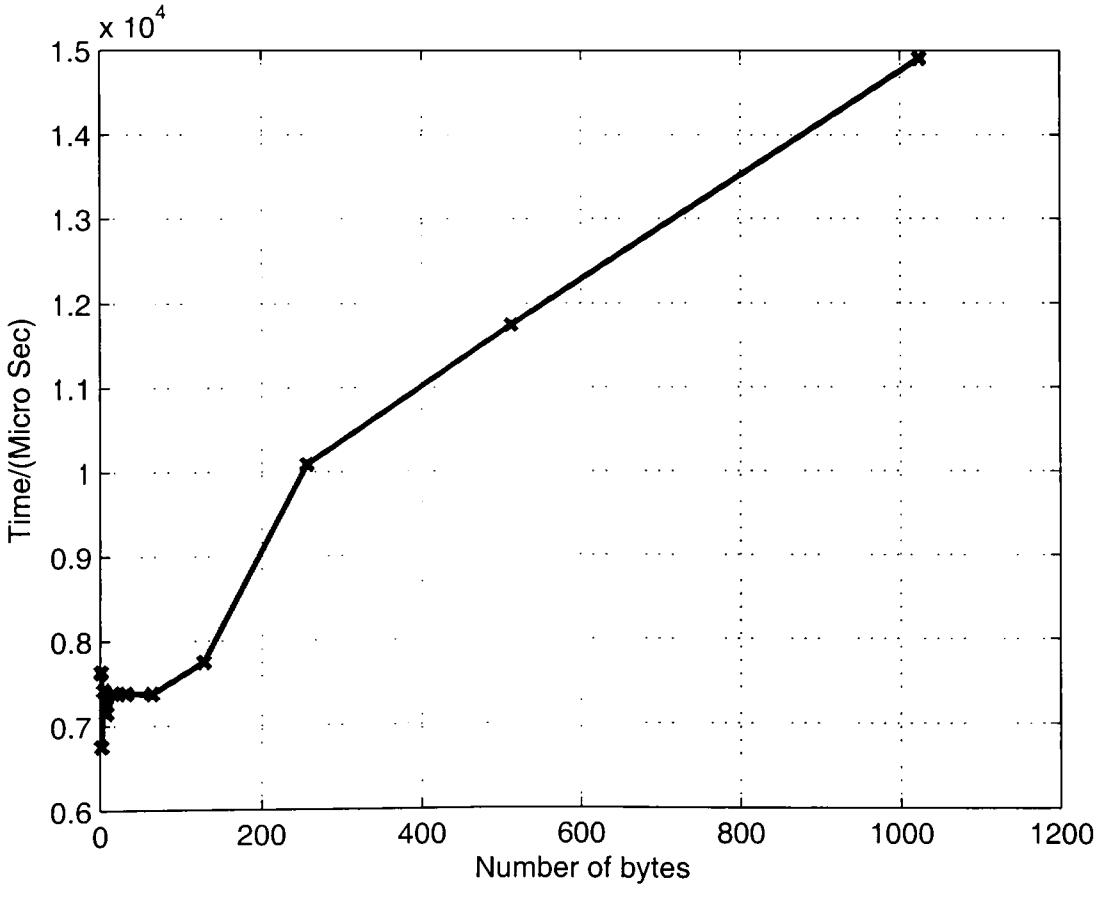

Figure 9.1: Communication characteristics between the primary and secondary nodes

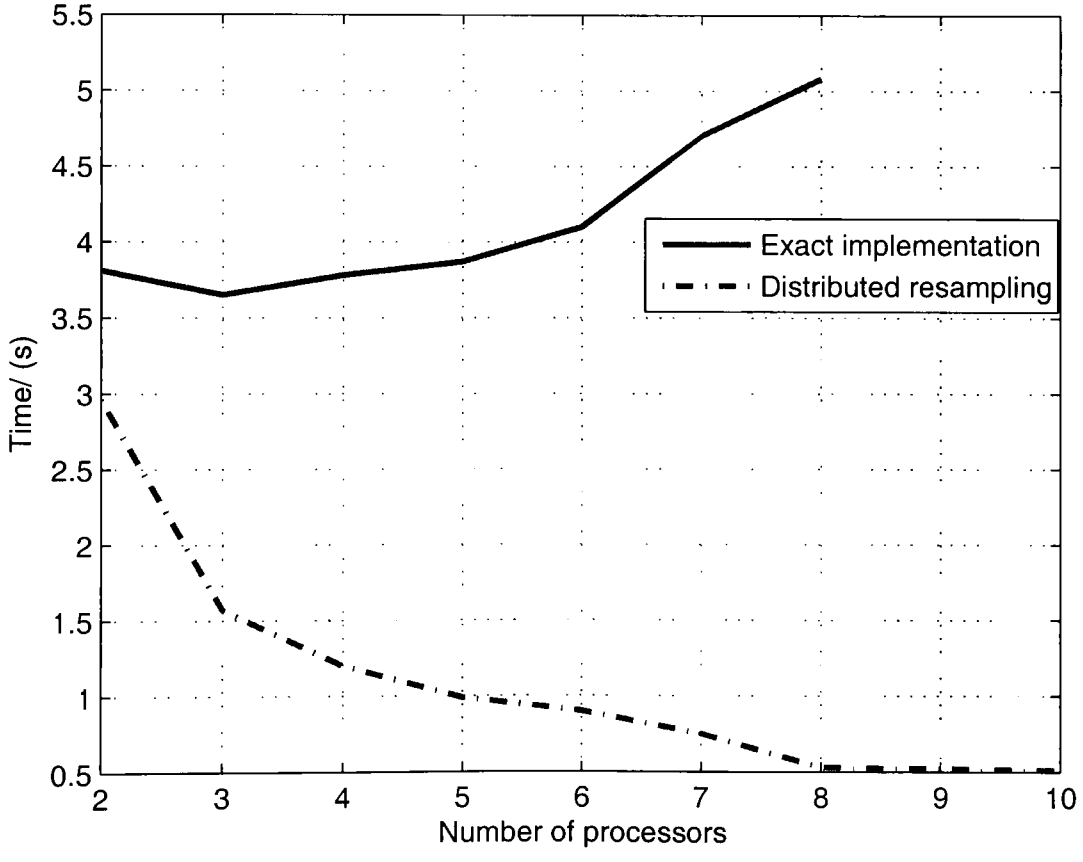

Figure 9.2: Time comparison for a single target

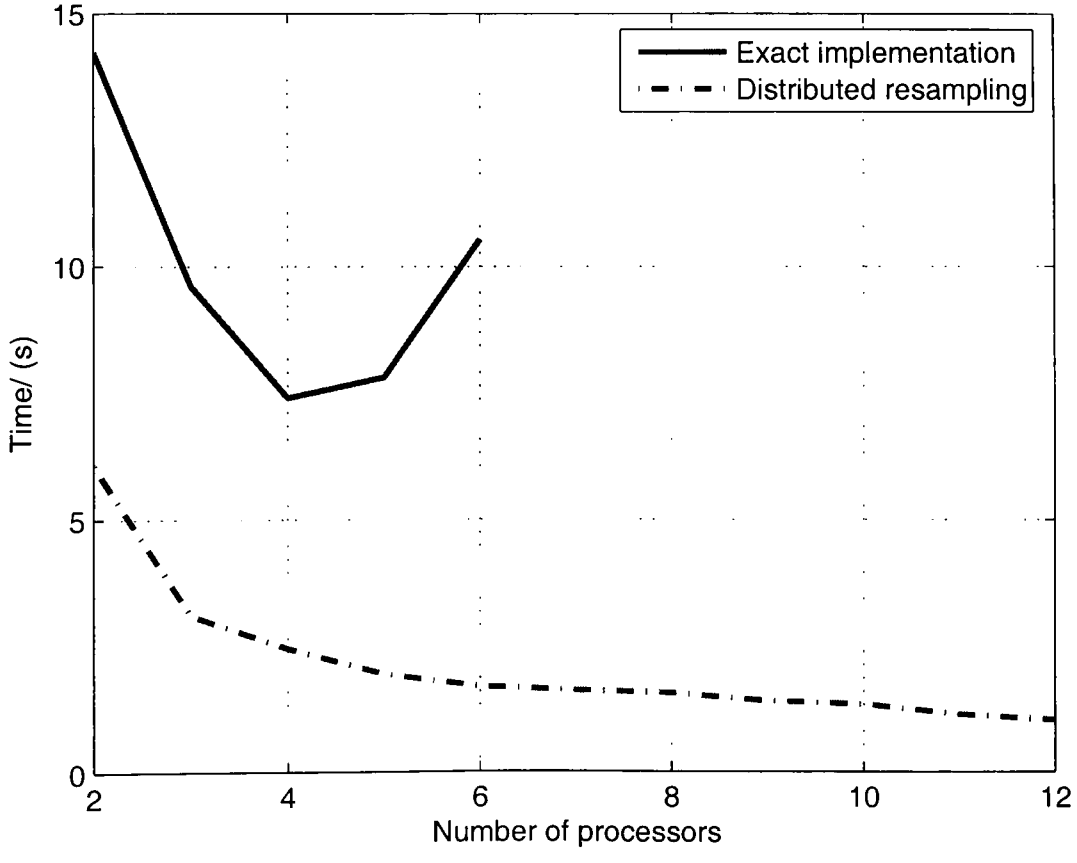

Figure 9.3: Time comparison for two targets

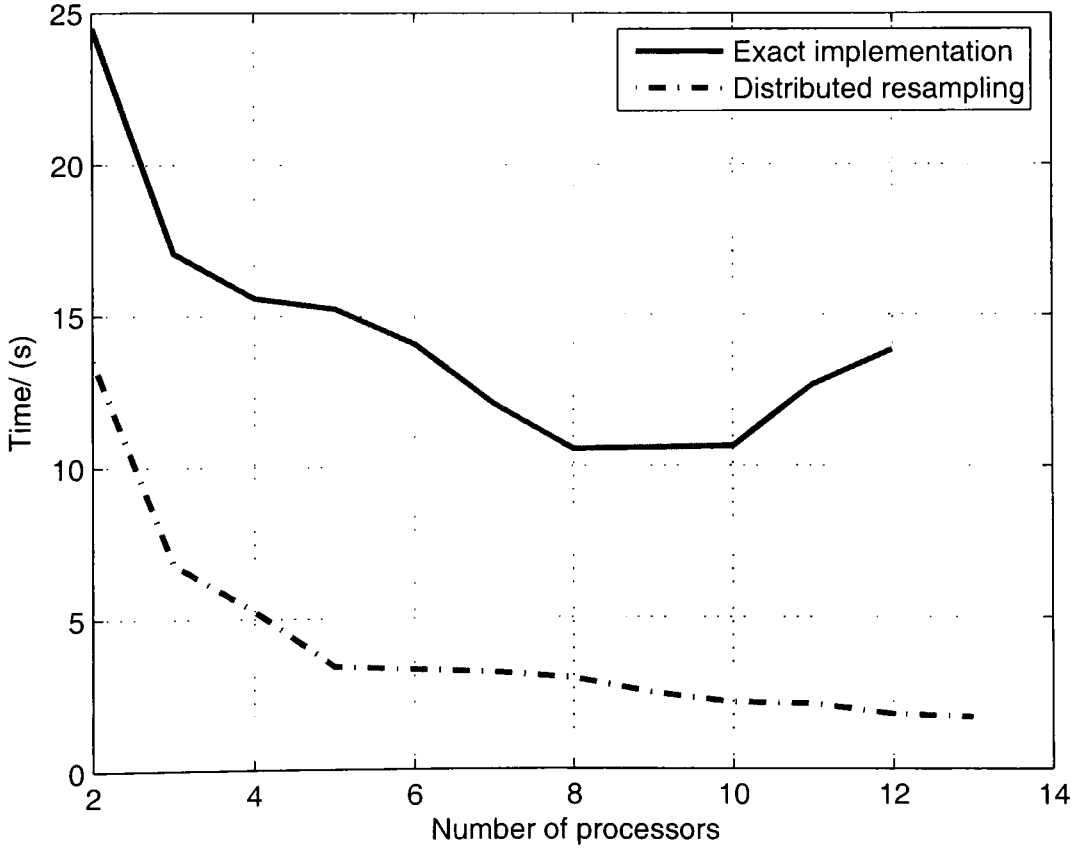

Figure 9.4: Time comparison for three targets

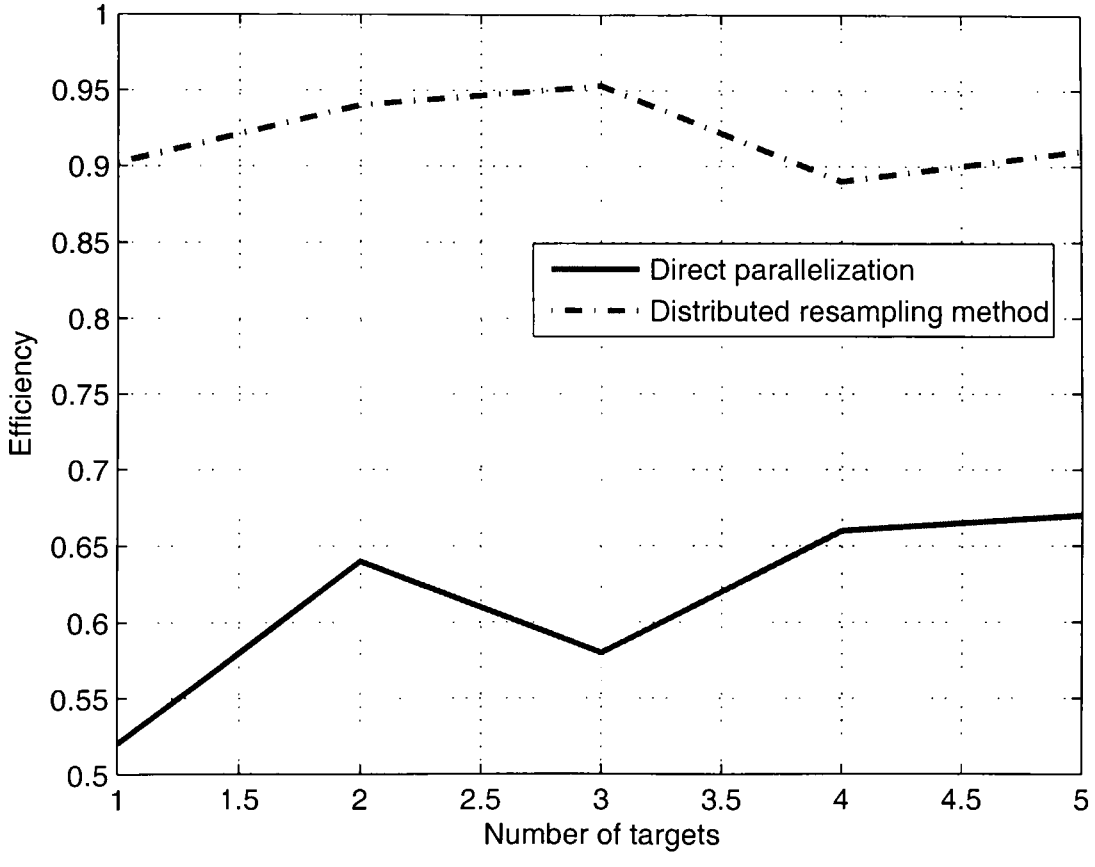

Figure 9.5: Efficiency comparison of two parallel algorithms

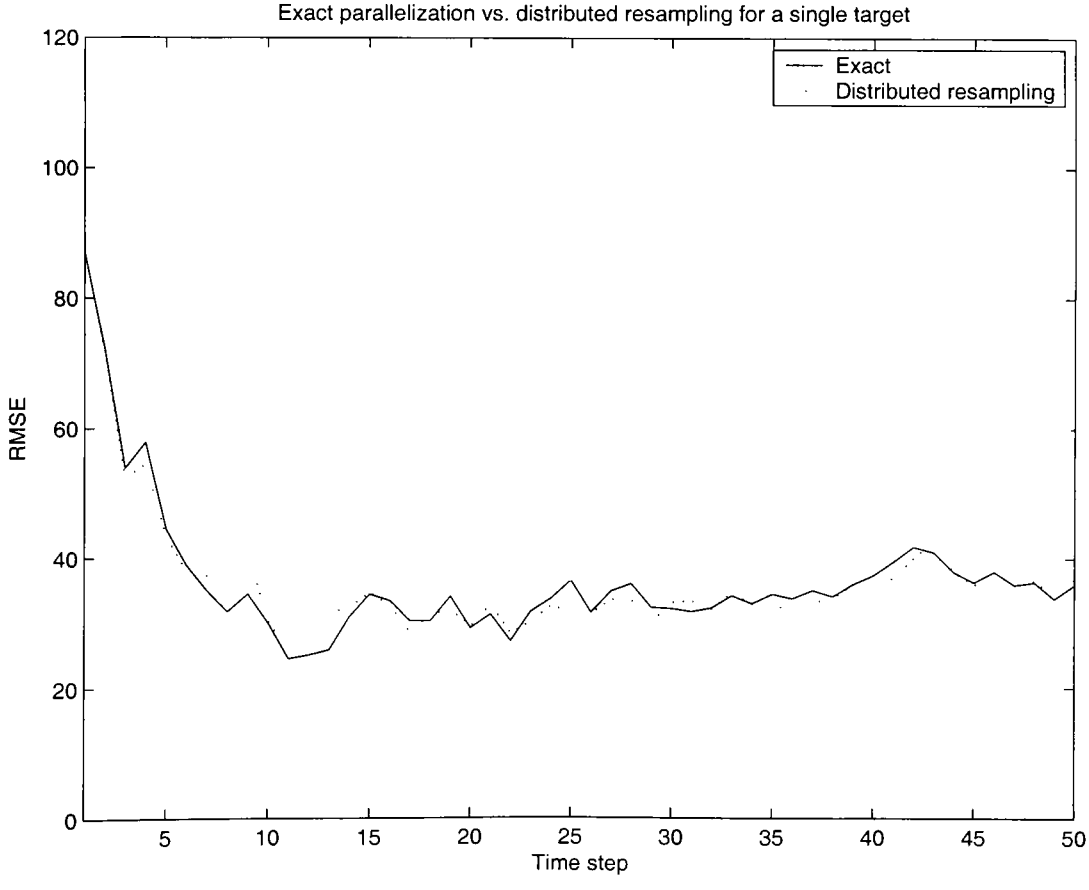

Figure 9.6: RMSE comparison for a single target

CDPF according to [5]). Theoretically, the proposed DRPF's performance will be the same as that of the serial particle filter, which is verified through simulations. Thus, direct comparisons against CDPF and LDPF are not performed.

Chapter 10

CONCLUSIONS

This thesis considered the parallelization of a particle filter for multitarget tracking problems with non-linear non-Gaussian dynamics. The high computational load of standard multitarget particle filters, which typically consists of stacked state vectors, is made tractable for real-time applications through parallelization in a primary-secondary architecture using optimization techniques. Furthermore, the proposed Distributed Resampling Particle Filter (DRPF) is shown to be more efficient in terms of resource utilization. In the DRPF, the data transfer between the primary node and secondary ones is reduced significantly without any apparent degradation in tracking performance. However, the DRPF needs load balancing as the number of particles on each node after resamling may not be optimal. A load balancing algorithm is proposed to make the overall algorithm efficient and real-time feasible.

Bibliography

[1] R. Andrei and J.C. Arjan, "Low-cost task scheduling for distributed-memory machines", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 6, pp. 648–658, June 2002.

[2] S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking",*IEEE Transactions on Signal Processing*, Vol. 50, No. 2, pp. 174–188, Feb. 2002.

[3] Y. Bar-Shalom (ed.), *Multitarget-Multisensor Tracking: Applications and Advances*, Vol. 2, Artech House, Boston:MA, 1990.

[4] Y. Bar-Shalom, X.R. Li and T. Kirubarajan, *Estimation with Application to Tracking and Navigation*, Wiley, New York:NY, 2001.

[5] A.S. Bashi, V.P. Jilkov, X.R. Li and H. Chen, "Distributed implementations of particle filters", *Proc. ISIF International Conference on Information Fusion*, Cairns, Australia, pp. 1164–1171, July 2003.

[6] O. Beaumont, A. Legrand and Y. Robert, "The master-slave paradigm with heterogeneous processors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 9, pp. 897–908, Sept. 2003.

[7] M. Bolic, P.M. Djuric and S. Hong, "New resampling algorithms for particle filters", *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 589–592, Hong Kong, April 2003.

[8] M. Cermele, G. Colajanni and G. Necci, "Dynamic load balancing of distributed SPMD computations with explicit message-passing", *Proceedings of the Heterogeneous Computing Workshop*, Vol. pp. 2–16, Geneva, April 1997.

[9] A. Doucet, S. J. Godsill and C. Andrieu. "On sequential Monte Carlo sampling methods for Bayesian filtering", *Statistics and Computing*, Vol. 10, No. 3, pp. 197–208, 2000.

[10] A. Doucet, B. Vo, C. Andrieu and M. Davy, "Particle filtering for multitarget tracking and sensor management", *Proceedings of the 5th International Conference on Information Fusion*, Vol. 1, pp. 474–481, Annapolis, Maryland, July 2002.

[11] A. Francisco, G. Daniel and M. Luz Marina, "The master-slave paradigm on heterogeneous systems: A dynamic programming approach for the optimal mapping", *Proc. 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 266–272, Spain, Feb. 2004.

[12] N. J. Gordon, D. J. Salmond and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation", *IEE Proceedings-F, Radar and Signal Processing*, Vol. 140, No. 2, pp. 107–113, April 1993.

[13] S. Hong, S. Chin, and S. Magesh, "A flexible resampling mechanism for parallel particle filters", *Proc. International Symposium on VLSI Technology, Systems, and Applications*, Vol. 1, pp. 288–291, Romania, July 2003.

[14] S. Hong, M. Bolic and P. Djuric, "An efficient fixed-point implementation of residual resampling scheme for high-speed particle filter", *IEEE Signal Processing Letters*, Vol. 11, No. 5, pp. 288–291, May 2004.

[15] C. Hue, J. Le Cardre and P. Perez, "Sequential Monte Carlo methods for multiple target tracking and data fusion", *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, pp. 309–325, Feb. 2002.

[16] T. Kirubarajan, Y. Bar-Shalom, K. Pattipati, I. Kadar, "Ground target tracking with variable structure IMM estimator",*IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36, pp. 26-46, Jan. 2000

[17] C. Kreucher, K. Kastella and A.O. Hero III, "A Bayesian method for integrated multitarget tracking and sensor management", *Proc. ISIF International Conference on Information Fusion*, Vol. 1, pp. 704–711, Cairns, Australia, July 2003.

[18] C. Kreucher, K. Kastella and A.O. Hero III, "Multitarget tracking using a particle filter representation of joint multitarget density", *Submitted to IEEE Transactions on Aerospace and Electronic Systems*, 2005.

[19] A. Legrand, H. Renard, Y. Robert and F. Vivien, "Mapping and load-balancing iterative computations", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 6, pp. 546–558, June 2004.

[20] K. Li and Y. Pan, "Probabilistic analysis of scheduling precedence constrained parallel tasks on multicomputers with contiguous processor allocation", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 49, No. 10, pp. 1021–1030, Oct. 2000.

[21] J. MacCormick and A. Blake, "A probabilistic exclusion principle for tracking multiple objects", *International Journal of Computer Vision*, Vol.39, No.1, pp. 57–71, August 2000.

[22] H. Marc and P.R. Anthony, "Strategies for dynamic load balancing on highly parallel computers", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 9, pp. 979-993, Sept. 1993

[23] A.D. Marrs, S.R. Maskell and Y. Bar-Shalom, "Expected likelihood for tracking in clutter with particle filters", *Signal and Data Processing of Small Targets, Proceedings of SPIE*, Vol. 4728, pp. 230–239, Orlando, Florida, April 2002.

[24] C. Musso, N. Oudjane and F. Le Gland, "Improving regularised particle filters", in *Sequential Monte Carlo Methods in Practice*, A. Doucet, F. G. de Freitas and N. J. Gordon, Eds, Springer-Verlag, 2001.

[25] A.R. Nolan, B. Everding and W. Wee, "Scheduling of low level computer vision algorithms on networks of heterogeneous machines", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 9, pp. 352–358, Sept. 1995.

[29] R. Popp, K. Pattipati, Y. Bar-Shalom and M. Yeddanapudi, "Parallelization of a multiple model multitarget tracking algorithm with superlinear speedups", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 33, No. 1, pp. 281–290, Jan. 1997.

[30] M. Sadasivam, J. Wagholikar and S. Hong, "A look-up based low-complexity parallel noise generator for particle filter processing", *Proc. International Symposium on Signals, Circuits and Systems*, Vol. 2, pp. 621–624, Romania, July 2003.

[31] A. Sinha, N. Nandakumaran, S. Sutharsan and T. Kirubarajan, "Tracking spawning targets with a tagged particle filter", *Proc. SPIE Conference on Signal Processing, Sensor Fusion, and Target Recognition*,Vol. 5429, pp. 13-22, Orlando, FL, April 2004.

[32] H. Wang, T. Kirubarajan, Y. Bar-Shalom, "Precision large scale air traffic surveillance using IMM/assignment estimators", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 35, No. 1, pp. 255-266, Jan. 1997.