Parallel Implementation of Modified Rao-Blackwellised Particle Filter

Md. Suruz Miah, *Student Member, IEEE,* and Miodrag Bolic, *Member, IEEE*

***Abstract*—**

***Index Terms*—**

I. Introduction

PARTICLE filter algorithm has become as an open and challenging problem over the last decades with respect to their computational complexity. Despite the significant advances in this field, researchers are yet to reach a satisfactory level of computational overhead. Usually, particle filtering methods are by nature computationally expensive [1] when the dimension of the state vector is large, as it may be the case for speech processing application, tracking applications, and to name a few. In such cases, a large number of particles is typically required to achieve a performance that would justify the use of particle filtering. Rao-Blackwellised Particle Filters (RBPF) [2] are alternative to PF algorithm that requires much less particles to achieve the similar performance but instead, it requires more computations per particle. The further improvement of RBPF (modified RBPF) is conducted by [3], where the fundamental difference from the generic RBPF lies in the way of propagating the particles from one iteration to the next iteration.

The main goal of this paper is to design and implement the modified RBPF algorithm using multiprocessing system in order to reduce the computational complexity and to find the optimal number of Processing Elements (PE). The rest of the manuscript is organized as follows: A brief literature review in the field of RBPF is presented in section II. The overview proposed system architecture is described in section III. Section IV illustrates the generic particle filter and RBPF algorithms. Our proposed parallel algorithm to implement the modified RBPF is discussed in section V. A thorough evaluation of our algorithm is provided in section VI and finally the paper is concluded with some further improvements in section VII.

II. Related Work

Particle filter has become a popular tool in different tracking and localization applications. It outperforms over some conventional methods such as the Kalman filter in nonlinear/non-Gaussian environments. In recent years, a significant research body has been conducted on mobile robotics that incorporate several sensors, landmarks as observation media in the environment and on signal processing applications based on different particle filtering algorithm.

Md. Suruz Miah is a Masters student and a research member of the Machine Intelligence, Robotics, and Mechatronics (MIRaM) Laboratory at the School of Information Technology and Engineering, University of Ottawa, Ontario, Canada. E-mail address: smiah069@site.uottawa.ca.

Miodrag Bolic is with the Signal Processing Oriented Technology (SPOT) research group at the School of Information Technology and Engineering, University of Ottawa, Ontario, Canada. E-mail address: mbolic@site.uottawa.ca.

III. System Architecture

The general high level architecture of our system consists of several processing elements and one central unit. The central unit is responsible to control the operations of all processing elements connected to it. The processing elements are independent of each other while generating particles and weights. Each processing element executes modified RBPF algorithm independently. We will mainly consider two main topology for the implementation of modified RBPF algorithm in parallel. In both topology, the processing elements are responsible to generate particles and compute weight for each particle and the central unit is responsible to perform resampling [4]. The first topology is straight forward architecture which is shown in Figure 1, where one central unit is connected to p processing elements. The central unit controls the operation of all processing elements connected to it. The interconnections between processing elements are necessary simply because of load balancing. The term load balancing comes from the fact that each processing element has to generate equal number of particles and compute weights of the respective particles.

For example, consider the case where the total number of particles is $N = 32$ and the number of processing elements is p =4. These particles have to be generated and then resampled for the next iteration. The central unit initially distributes equal number of particles (8 in this case) to the processing elements PE_1, PE_2, PE_3 and PE_4. Let, N_{PE_p} indicates the set of particles in PE_p, then the indexing of the particles in each processing elements are performed as $N_{PE_1} = \{1, 2, ..., 8\}$, N_{PE_2} =$\{8, 9, ..., 16\}$, $N_{PE_3} = \{17, 18, ..., 24\}$ and $N_{PE_4} = \{25, 26, ..., 32\}$. Each PE executes the modified RBPF algorithm locally and generates new and equal set of particles. Once the computations of all PEs are finished, an ACK signal is sent to central unit in order to perform resampling. After resampling, the central unit sends only valid particles to the respective PEs with corresponding replication factor [4]. Let assume that in this phase, the central unit produces only particles at indexed 15 and 26 with the corresponding replication factors 20 and 12 respectively which belong to the processing elements PE_2 and PE_4 respectively. Now PE_2 sends 8 particles to PE_1 and 4 particles to PE_3 and finally PE_4 sends 4 particles to PE_3

Another high level star topology for implementing the same problem is presented in Figure 2, where the central unit and all

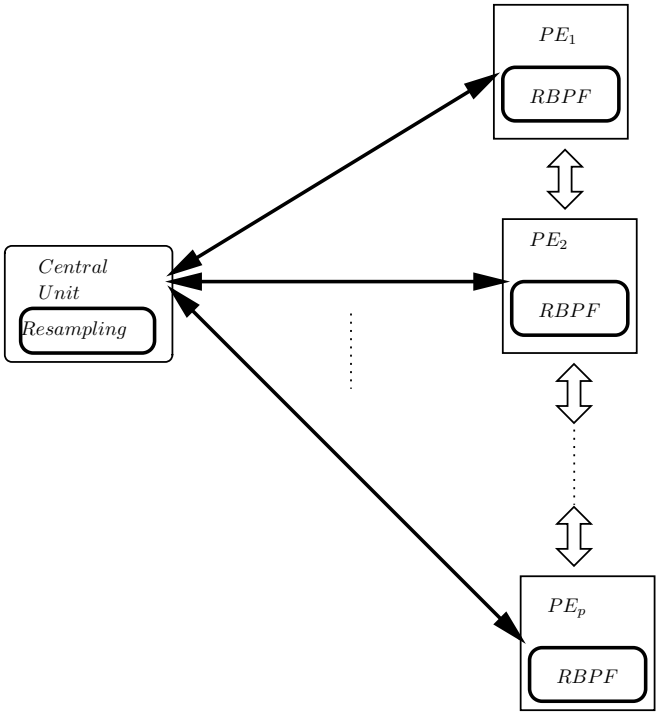

Fig. 1. The topology 1 of parallel RBPF algorithm.

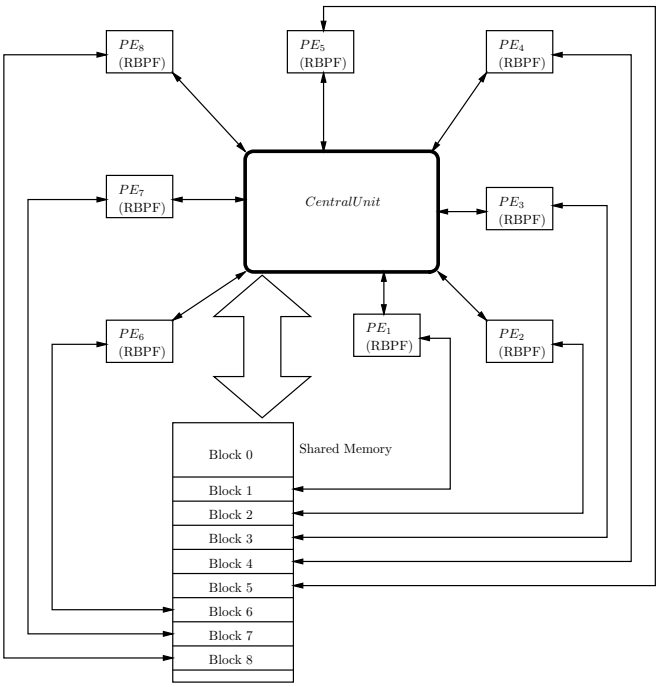

Fig. 2. The topology 2 of parallel RBPF algorithm.

the processing elements are sharing the global memory. The global memory is responsible to store all the particles and the corresponding weights. It is partitioned into several blocks (*Block 0, 1, 2,...*) where each block is assigned for a particular processing element. The processing elements have access to their pre-assigned blocks (e.g. PE_1 can access to *block 1*, PE_2 can access to *block 2*, and so on). The central unit use the *block 0* of the global memory as an additional storage block for its own processing purpose. All processing elements are connected to the central unit. Similar to the previous architecture, the central unit is responsible to control the operations of all the processing elements. In this architecture, two communication channels are used where one is dedicated for the central unit and the other one is shared by all the processing elements. In order to better understanding, the Figure 2 shows that each processing elements has dedicated access to the particular block of the memory. The central unit has the access in the whole memory simply because it will resample based on the total number of particles' weights.

IV. Particle Filter and RBPF algorithms

A. The Particle Filter Algorithm

B. Regular RBPF Algorithm

C. modified RBPF Algorithm

V. Parallel RBPF algorithm

An algorithm is needed to act as a supervisory control layer to process and coordinate among central unit and multiple processing elements. A flowchart of parallel RBPF algorithm is provided in Figure 3. The following is a description of the different steps of the algorithm.

Step 1: The particles are initialized with their respective

Algorithm 1: The Particle Filter Algorithm

Input: N particles, in initial state.
Output: Set of estimated states.
N = Total number of Particles.
k = State index.
X_k = State variable.
TK = Total number of states.
$p(X_k)$ = probability density function (pdf) of state X_k.
begin
  // Initialize particles
  $k = 0$
  for i = 1 *to* N **do**
    Generate $X_{k,i}$ and $p(X_k)$
  //Main iteration starts here
  for $k = 1$ *to* TK **do**
    // Importance sampling step
    for $i = 1$ *to* N **do**
      $X_{k,i} \sim p(X_k|X_{k-1,i})$
      $\hat{X}_{0:k,i}$ = $\{X_{0:k-1,i}; X_{k,i}\}$
      $w_{k,i}$ = $p(z_k|\hat{X}_{k,i})$
      // Normalize the importance weights
      $\hat{w}_{k,i} = \frac{w_{k,i}}{\sum_{j=1}^{N} w_{k,j}}$
    // Resample with Replacement N particles
    for $i = 1$ *to* N **do**
      draw $X_{k,i}$ from $\hat{X}_{k,i}$ according to $\hat{w}_{k,i}$
end

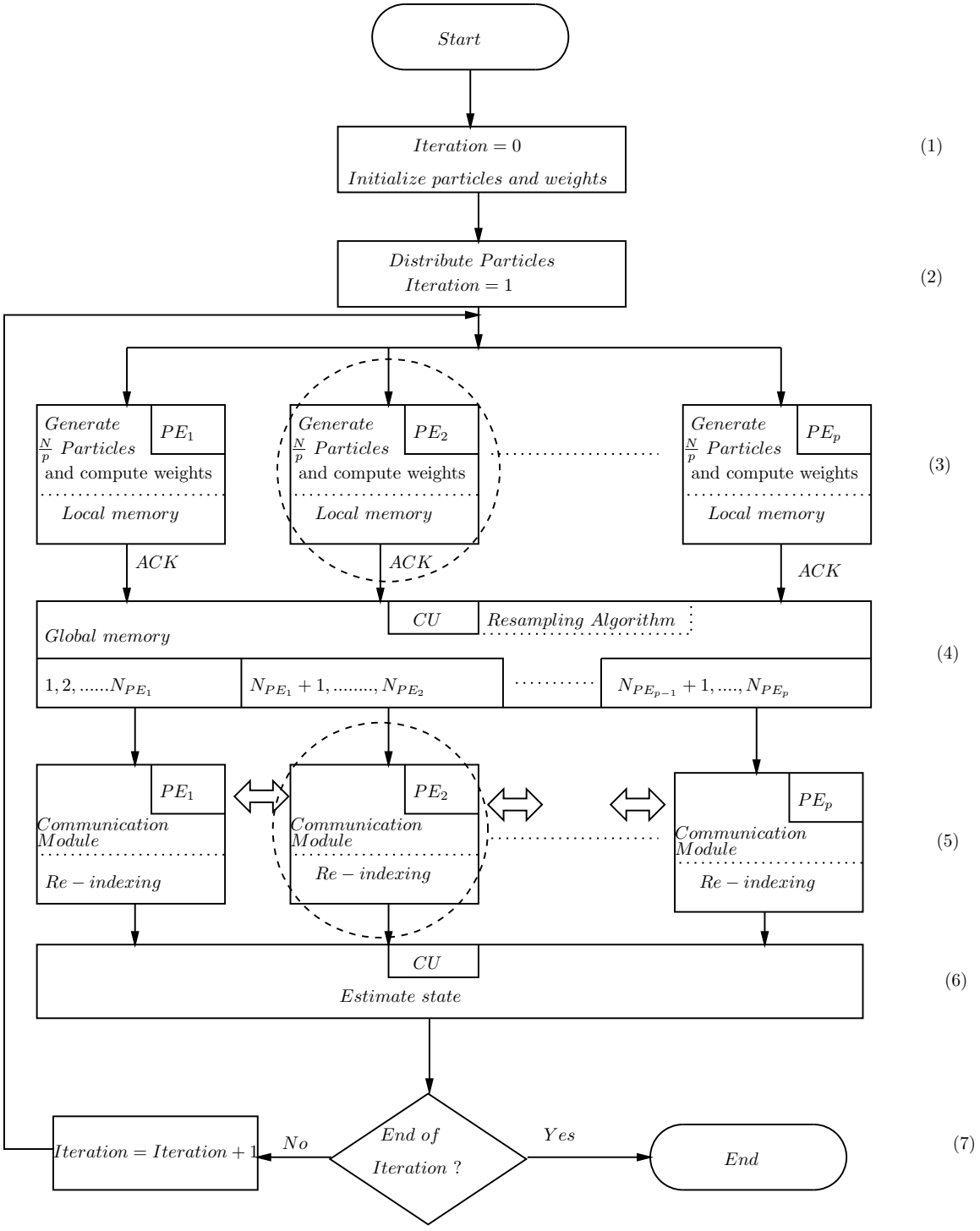

Fig. 3. Flowchart of the proposed parallel RBPF algorithm.

weights and all weights are equal in this stage.

Step 2: This step starts with $iteration = 1$ and distributes equal number of initially generated particles among processing elements.

Step 3: A number of processing elements are assigned to regenerate the particles and to compute the weights. In this step, each processing elements executes the modified RBPF algorithm which is shown in algorithm **??**. Each processing element stores their particles and corresponding weights into their local memory. Once the particle generation and weight computations are finished, all processing elements send the acknowledgment signal (ACK) with their corresponding weight vectors to the central unit.

Step 4: The central unit has a global memory which is partitioned into a number of segments which is equal to the number of processing elements. All the weight vectors are stored at the global memory of the central unit in predefined addresses. The resampling algorithm (see algorithm **??**) takes the whole weight vector as argument and produces only valid particles with the corresponding replication factors. Finally, it sends the valid particle indices with the replication factors to the respective processing elements.

Step 5: According to the particle indices and replication factors sent by the central unit, each processing element delete or replicates particles in their local memory. Now the communication module in each processing element is responsible to balance the number of particles. In our architecture, communication always starts from PE_1 and ends with PE_p. *Re-indexing* is done for sake of simplicity in *Step 3* and *Step 4*. The term *re-indexing* means that indexing of the total number of particles (N) is set to $1, 2, ..., N$. Algirithm 2 performs the redistribution of particles among processing elements and indices of the particles are rearranged as $1, 2, ..., N$.

Step 6: In this step, central unit estimates the current state based on the particles available in the global memory.

Step 7: The last step is to check whether the algorithm finished the desired number of iteration. If the more iterations are yet to be done, then it rollbacks to the *Step 3* and continues to execute the above steps as well.

Algorithm 2: The particle re-distribution algorithm

Input: Pr: array of replication factors of particles.
Output: Equal number of particles to all processing elements.
N = Total number of Particles.
Pr_{ir} = i^{th} particle with replication factor r.
N_{PE_p} =number of particles assigned for processing element PE_p.
P = total number of processing elements.
N_c = current number of particles.

```
begin
    i = 1
    N_c = Pr_ir
    for p = 1 to P do
        if N_c > N_PE_p then
            send N_c − N_PE_p copies of the i^th particle to PE_p
            N_c = N_c-N_PE_p
            set indices from (p − 1) ∗ N_c to p ∗ N_c
        else
            while N_c ≤ N_PE_p do
                send N_c copies of the i^th particle to PE_p
                i = i + 1
                cp = Pr_ir
                N_c = N_c + cp
            set indices from (p − 1) ∗ N_c to p ∗ N_c
            N_c = Pr_ir
end
```

VI. Experimental Results

A set of numerical experiments are conducted to test effectiveness of the parallel RBPF algorithm and to demonstrate its performance. The simulations are carried out using MATLAB. We are focusing to find the optimal number of processing elements running on $N = 150$ particles. For the sake of simplicity, we are considering the problem of tracking a maneuvering target. The outcome of our experiments are divided into two phases. The first phase is conducted to represent timing characteristics and in the second phase, the model estimation parameters are presented.

The Figure 4 provides the performance of our parallel RBPF algorithm's performance which shows the total execution time versus number of processing elements running on 150 particles. Let, t_{gen_i} = *generation time of particle* i, t_{w_i} = *time required to compute weight of particle* i, t_c = *total communication time among processing elements*, t_{e_k} = *execution time for iteration* k, and t_e = *total execution time*. So, the execution time for one iteration (k) is given by Equation 1 and total execution time is given by Equation 2.

$$t_{e_k} = \sum_{i=1}^{\frac{N}{p}} (t_{gen_i} + t_{w_i}) + t_c \tag{1}$$

$$t_e = \sum_{k=1}^{TK} t_{e_k} \tag{2}$$

We assume that one time unit is required to generate and to compute weight of each particle ($i.e.\ t_{gen_i} + t_{w_i} = 1$) and the communication time (for one particle) between two processing elements is 0.25 time unit. Figure 4 represents the total execution time (in seconds), where we consider that $1\ time\ unit = 1ms$. We notice that at the beginning, the total execution time decreasing drastically as the number of precessing elements increases and then remain constant. After a while, the execution time goes high with the increasing number of processing elements because the communication time is higher than the time required to generate and to compute weights of particles in parallel. Our algorithm finds the optimal number of processing elements (10 in this experiment) where the total execution time is minimum (4.2 seconds in this case).

In the maneuvering target problem, the position and velocity at instant k are given by a continuous random vector

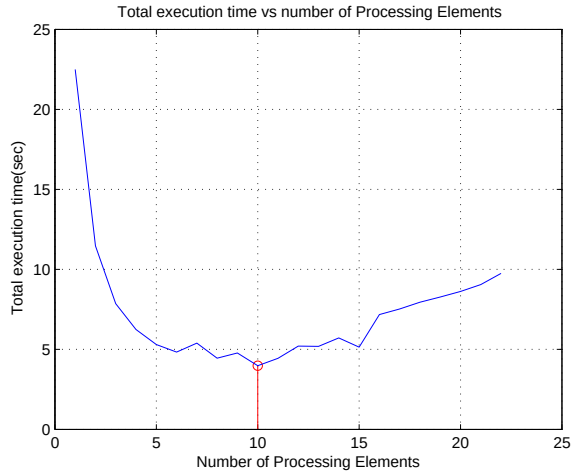

Fig. 4. The parallel RBPF algorithm's performance.

$x2k \in \mathbb{R}^{n-1}$, and where the maneuver/regime of the target is represented by the discrete random variable $x1k \in \mathbb{R}$. The state to be estimated is $x_k = \{x_{1k}; x_{2k}\}$. The model is as follows:

The true and estimated trajectory of the tracking maneuvering target problem is presented in Figure 5.

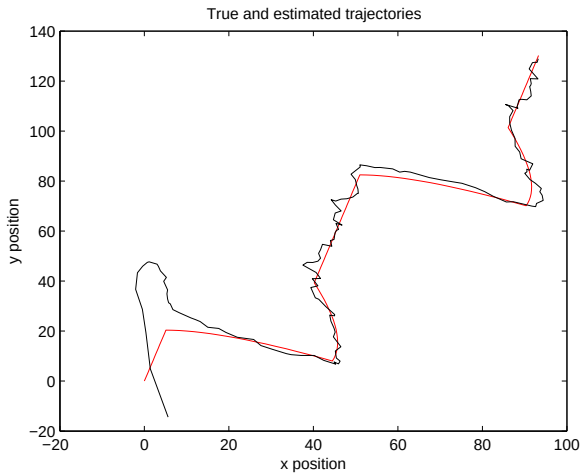

Fig. 5. The true and estimated trajectory. The estimates are in dotted lines.

VII. Conclusion

In this paper, we present a novel parallel RBPF algorithm using multiple processing units. The experimental results from the first architecture demonstrates the effectiveness of our algorithm. This study will open the doors for the optimal parallel implementation of RBPF algorithm. A potential future research avenue to extend this work is to append the algorithm for the several architectures with different number processing elements and the comparison of different architectures with some real parameters.

References

[1] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: particle filters for tracking applications*, B. Ristic, Ed. Artech house, 2004.

[2] X. Xu and B. Li, "Rao-blackwellised particle filter for tracking with application in visual surveillance," in *International Conference on Computer Vision*, Beijing, October 2005.

[3] F. Mustiere, M. Bolic, and M. Bouchard, "A modified rao-blackwellised particle filter," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Toulouse, France, May 2006.

[4] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2442– 2450, July 2005.