Journal of Signal Processing Systems manuscript No.
(will be inserted by the editor)

Algorithm and Parallel Implementation of Particle Filtering and its Use in Waveform-Agile Sensing

Lifeng Miao · Jun Jason Zhang · Chaitali Chakrabarti · Antonia Papandreou-Suppappola

the date of receipt and acceptance should be inserted later

Abstract Sequential Monte Carlo particle filters (PFs) are useful for estimating nonlinear non-Gaussian dynamic system parameters. As these algorithms are recursive, their real-time implementation can be computationally complex. In this paper, we analyze the bottlenecks in existing parallel PF algorithms, and propose a new approach that integrates parallel PFs with independent Metropolis-Hastings (PPF-IMH) resampling algorithms to improve root mean-squared estimation error (RMSE) performance. We implement the new PPF-IMH algorithm on a Xilinx Virtex-5 field programmable gate array (FPGA) platform. For a one-dimensional problem with 1,000 particles, the PPF-IMH architecture with four processing elements uses less than 5% of a Virtex-5 FPGA's resource and takes 5.85 μs for one iteration. We also incorporate waveform-agile tracking techniques into the PPF-IMH algorithm. We demonstrate a significant performance improvement when the waveform is adaptively designed at each time step with 6.84 μs FPGA processing time per iteration.

Keywords Particle filter · Waveform-agile sensing · Parallel architecture · Field programmable gate array · Target Tracking

This work was partly supported by NSF under Grant No. 0830799 and MURI Grant AFOSR FA9550-05-1-0443.

The parallel particle filter implementation was discussed in our 2010 IEEE Workshop on Signal Processing Systems paper [1]. This work also presents: the new algorithm and hardware implementation described in more detail (Section 3); the effect of the number of processing elements and number of groups in each processing element on the parallel particle filter algorithm performance (Section 5.2, Section 6.1); the waveform-agile sensing algorithm (Section 4.1), the waveform-agile tracking application (Section 4.2), and its hardware (FPGA) implementation (Section 4.3); new simulation and hardware implementation results on waveform-agile tracking ((Section 5.4) and (Section 6.2)).

School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ
E-mail: lifeng.miao@asu.edu · jun.zhang.ee@asu.edu · chaitali@asu.edu · papandreou@asu.edu

1 Introduction

Particle filtering is a sequential Bayesian estimation technique that has been used for estimating parameters of nonlinear and non-Gaussian dynamic systems in applications such as target tracking and biomedical signal processing [2–4]. Although Kalman filters can provide optimal parameter estimates for linear dynamic systems in additive Gaussian noise [5], they are not applicable when the systems are highly nonlinear. Extended versions of Kalman filters provide local linearization techniques, but particle filtering has been found to provide more accurate estimation performance [2, 6–8]. One disadvantage of particle filters (PF), however, is that they are computationally intensive as they are sequential Monte Carlo techniques. Our objective in this paper is to present a method that can be used to parallelize the PF for real-time implementation with minimum loss in algorithm performance.

An important application of particle filtering is waveform-agile sensing, where the waveform is adaptively configured at each time. The sensing performance has been shown to increase when the parameters of transmitted waveforms (in active sensing) are adaptively designed or the parameters of observed waveforms (in passive sensing) are optimally selected at each time step [9–11]. However, as the waveform parameters need to be adaptively updated at each time step, the computational complexity of waveform design is very high. When waveform-agility is integrated into particle filtering, the computational complexity can become unmanageable. However, if the PF can be implemented in parallel and efficiently, real-time implementation of adaptive waveform design schemes will become more feasible.

There are three major operations in PF processing: particle generation, weight calculation, and resampling. As shown in [12,13], the bottleneck in real-time PF implementation is the resampling operation. Several modifications of the resampling algorithm, such as residual-systematic resampling and threshold based resampling, were proposed to reduce computational complexity [14–16]. The threshold based resampling algorithm in [14] was modified to obtain the compact resampling algorithm that helped improved tracking performance in [17–19]. A systematic resampling algorithm with non-normalized weights was proposed in [16] to improve the PF pipelined implementation. In [20], a particle-tagging quantization scheme was used to make the number of particles a power of two and thus reduce the hardware complexity of the PF residual resampling algorithm.

The aforementioned resampling algorithms are modified versions of the systematic resampling algorithm [8] or residual resampling algorithm [21]. For both algorithms, resampling cannot be computed unless knowledge of all particle weights is available, and that poses a considerable challenge for pipelined implementation. In order to eliminate this bottleneck, independent Metropolis-Hastings (IMH) resampling can be employed as it can start as soon as the first particle weight is available [22, 23]. Another important issue in PF hardware implementation is the ability to parallelize the PF computation. While parallel architectures have been proposed in [14, 16, 23], the communication between parallel processing units and the central processing unit is a significant overhead. In our previous work, we proposed an algorithm which significantly reduced the communication overhead though at the cost of degradation in estimation performance [24].

In this paper, we develop a parallel PF algorithm that can improve estimation performance with minimal overhead. The algorithm can be mapped onto a parallel and pipelined architecture and is capable of meeting the requirements of real-time processing. We apply the proposed algorithm to waveform-agile sensing to improve dynamic state estimation performance. Our contributions are as follows.

- *Algorithmic Enhancements.* In order to efficiently parallelize the PF computation, we propose an algorithm which uses the independent Metropolis-Hastings (IMH) sampler with the parallel PF (PPF) algorithm. We analyze the performance of the new PPF-IMH algorithm and show that it has superior performance when compared to the PPF in [24], and minimal performance degradation when compared to a non-parallel PF. We also demonstrate that the new PPF-IMH algorithm significantly reduces the communication overhead when mapped onto a parallel architecture.
- *Hardware Implementation.* We present a pipelined and parallel architecture to implement the proposed PPF-IMH algorithm and map it onto a Xilinx FPGA hardware platform. Experimental results show that it can meet the requirements of real-time processing with fairly low resource usage, for instance, 5% of the slice resource of a Xilinx Virtex-5 FPGA.
- *Application in Waveform-agile Sensing.* We incorporate waveform agile sensing techniques into PPF-IMH algorithm in order to increase the state parameter estimation performance. We demonstrate the estimation performance improvement using a waveform-agile tracking application. We also implement the proposed integrated waveform-agile PPF-IMH system on an FPGA platform and show that it can be used for real-time processing applications.

The rest of the paper is organized as follows. We first review the particle filtering algorithm for estimating dynamic system state parameters in Section 2. In Section 3, we propose the hardware and FPGA implementation of the parallel particle filter with independent Metropolis-Hastings resampling. In Section 4, we apply the PPF-IMH in waveform-agile sensing and present its FPGA implementation for a target tracking application example. In Section 5 and 6, we demonstrate our numerical and experimental results for both algorithmic and hardware performance improvements.

2 Particle Filtering

Particle filtering is a sequential Monte Carlo method that is used to estimate the dynamic state parameters of nonlinear and/or non-Gaussian systems [6, 7]. The estimation is performed by approximating the posterior probability density function of the unknown state parameters at each time step given measurements up to that time step. Specifically, we consider a dynamic system described by the following state space model

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{n}_k \tag{1}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k\,, \tag{2}$$

where $\mathbf{x}_k$ is the vector of N_x unknown parameters at time step k, $\mathbf{z}_k$ is the vector of N_z measurements at time step k, $f(\cdot)\colon \mathbb{R}_{N_x} \to \mathbb{R}_{N_x}$ is a (possibly) nonlinear state-transition function, $h(\cdot)\colon \mathbb{R}_{N_x} \to \mathbb{R}_{N_z}$ is a (possibly) nonlinear function that relates the state vector with the measurement vector, $\mathbf{n}_k$ is the state modeling error vector, and $\mathbf{v}_k$ is the measurement noise vector. The state estimate could be obtained directly in closed form using a Kalman filter if both functions in (1) and (2) are linear and both the modeling error and measurement noise are Gaussian processes [5]. When the Kalman filter cannot be used, the particle filter (PF) has been shown to approximate the joint posterior probability density function of $\mathbf{x}_k$ at time k using a set of N random samples or particles, $\mathbf{x}_k^{(i)}$, and their corresponding weights, $w_k^{(i)}$, $i = 1, \ldots, N$, as:

$$p(\mathbf{x}_k|\mathbf{z}_k) \approx \sum_{i=1}^{N} w_k^{(i)}\, \delta(\mathbf{x}_k - \mathbf{x}_k^{(i)}) \,.$$

where $\delta(\cdot)$ is a Dirac delta function. Using this approximation, the estimated state parameter vector can be obtained as $\hat{\mathbf{x}}_k \approx \sum_{i=1}^{N} w_k^{(i)}\, \mathbf{x}_k^{(i)}$.

There are different PF algorithms, depending on the choice of importance density used to compute the weights [7,8]. One of the most commonly used algorithms is the sequential importance resampling (SIR) PF that consists of the following basic three steps:

1. *Particle generation.* The particles $\mathbf{x}_k^{(i)}$ are drawn from an importance density function $q(\mathbf{x}_k|\mathbf{x}_{k-1}^{(i)}, \mathbf{z}_{1:k})$, where $\mathbf{z}_{1:k} = \{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$.
2. *Weight computation.* The corresponding weights are calculated as

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p\left(\mathbf{z}_k|\mathbf{x}_k^{(i)}\right)\; p\left(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)}\right)}{q\left(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)}, \mathbf{z}_{1:k}\right)}$$

   and then normalized so that $\sum_{i=1}^{N} w_k^{(i)}=1$. Note that the importance density is often chosen to be the prior density function $q(\mathbf{x}_k|\mathbf{x}_{k-1}^{(i)}, \mathbf{z}_{1:k}) = p(\mathbf{x}_k|\mathbf{x}_{k-1}^{(i)})$. This simplifies the weight computation to $w_k^{(i)} \propto w_{k-1}^{(i)} p(\mathbf{z}_k|\mathbf{x}_k^{(i)})$.
3. *Resampling.* The particles are resampled to avoid particle degeneracy, which occurs when most particle weights are close to zero, resulting in a poor representation of the posterior probability density function [7]. Resampling avoids degeneracy by eliminating particles with low importance weights and replicating particles with high importance weights.

Even with the simplified weight computation, the SIR PF can be very computationally intensive as the number of particles is large. For example, in a radar tracking problem, a PF using $N \approx 1,000$ particles requires about $30N$ additions, $20N$ multiplications, and N exponential calculations per iteration. Thus, the overall computational complexity is very high.

Real-time implementation of SIR PF requires the use of pipelining and parallel processing. Particle generation and weight calculation can be easily parallelized and pipelined since they do not have any data dependencies [14]. The bottleneck is systematic resampling for the following reasons. First, systematic resampling requires the knowledge of all normalized weights which makes it hard to be pipelined with other steps. Secondly, systematic resampling requires a large volume of particle information exchange in the resampling process, resulting in a huge communication overhead [16].

Algorithm 1 Metropolis-Hastings algorithm [23]

```
Choose a starting point $\left(\mathbf{x}_k^{(0)}, w_k^{(0)}\right)$
for $i = 1$ to $N$ do
    From $\mathbf{x}_k^{(i)}$, draw samples (and compute corresponding weights) $(\mathbf{x}_k^*, w_k^*)$ from $q\left(\mathbf{x}_k^*|\mathbf{x}_k^{(i)}\right)$
    Compute probability $\alpha\left(\mathbf{x}_k^{(i)}, \mathbf{x}_k^*\right) = \min\left\{\frac{p(\mathbf{x}_k^*)\ q\left(\mathbf{x}_k^{(i)}|\mathbf{x}_k^*\right)}{p\left(\mathbf{x}_k^{(i)}\right)\ q\left(\mathbf{x}_k^*|\mathbf{x}_k^{(j)}\right)}, 1\right\}$
    $\left(\mathbf{x}_k^{(i+1)}, w_k^{(i+1)}\right) = \begin{cases} (\mathbf{x}_k^*, w_k^*), & \text{with probability } \alpha\left(\mathbf{x}_k^{(i)}, \mathbf{x}_k^*\right) \\ \left(\mathbf{x}_k^{(i)}, w_k^{(i)}\right), & \text{with probability } 1 - \alpha\left(\mathbf{x}_k^{(i)}, \mathbf{x}_k^*\right) \end{cases}$
end for
```

3 Parallel PF with Independent Metropolis-Hastings Sampling

In this section, we propose a new particle filtering algorithm and its hardware architecture with low communication overhead, good tracking performance comparable to PF with systematic resampling, and support for parallel and pipeline processing.

3.1 Metropolis-Hastings Algorithm

We use the Metropolis-Hastings (MH) algorithm to perform PF resampling in order to overcome the hardware implementation limitation [1]. As the MH resampling computation can start as soon as the first particle weight becomes available [23]. Specifically, the Metropolis-Hastings (MH) algorithm does not require all the particles as it can generate a Markov chain in which the current state $\mathbf{x}_k^{(i+1)}$ depends on the previous state $\mathbf{x}_k^{(i)}$ [25]. In particular, the MH algorithm can draw samples from a desired probability density function $p(\mathbf{x}_k)$ given a proposal probability density function $q(\mathbf{x}_k)$. The steps of the MH algorithm are described in Algorithm 1.

In Algorithm 1, the step of accepting the sample $\mathbf{x}_k^*$ can be implemented by first generating the uniform sample $\mathbf{u} \sim U(0,1)$, and then performing [22]

$$\left(\mathbf{x}_k^{(i+1)}, w_k^{(i+1)}\right) = \begin{cases} (\mathbf{x}_k^*, w_k^*) & \mathbf{u} \leq \min\left\{w_k^*/w_k^{(i)}, 1\right\} \\ \left(\mathbf{x}_k^{(i)}, w_k^{(i)}\right) & \mathbf{u} > \min\left\{w_k^*/w_k^{(i)}, 1\right\} \end{cases}.$$

The independent Metropolis-Hastings (IMH) algorithm can be obtained when $q(x_k^*|\mathbf{x}_k^{(i)})$ is independent of $\mathbf{x}_k^{(i)}$ in Algorithm 1. Note that, as there is no need to wait for all the particles and their weights to become available [22], the IMH algorithm is suitable for pipeline and parallel hardware implementation.

3.2 PPF-IMH Hardware Implementation

We propose a parallel PF algorithm which can be mapped into a multiple processing element architecture. The processing elements (PE) perform the major PF computational workload (particle generation, weight evaluation and resampling), and a central unit (CU) performs global computations and coordinates the PEs activities. If the PF is implemented by computing the systematic resampling in the CU, all the importance weights would have to be transferred, resulting in a huge communication overhead. In [24], we developed a method which can significantly reduce the amount of data communication overhead. The main idea was to divide the particles into several groups in each PE and use the average of each group as the new particle. However, this method results in an estimation performance degradation. To improve estimation performance while keeping the communication overhead low, we propose to use the IMH resampling in each PE before communication with the CU. Using the IMH resampling, the particles $\tilde{\mathbf{x}}_k^{(i)}$, $i = 1, \ldots, N$, are resampled to obtain $\mathbf{x}_k^{(i)}$ in order to more accurately represent the posterior probability density function. The information of the resampled particles is then sent to the CU. Also, since the IMH resampler can be easily pipelined with the other steps, the processing period is not increased.

The new PPF-IMH algorithm is described next in detail. We distribute M particles to P PEs, so $N{=}M/P$ particles are assigned to each PE. The mth PE, $m = 1, \ldots, P$, executes the processing steps in Algorithm 2 (sampling, weights computation, and IMH resampling) to generate the resampled particle set $\mathbf{x}_{k,m}^{(i)}$, $i = 1, \ldots, N$. Note that in Algorithm 2, we use $(N + N_b)$ particles since at the end of the processing, we will discard N_b samples from the start of the sequence as they may not converge to a good estimate [23].

Next, we present the one-dimensional grouping method that is used to reduce the communication overhead in Algorithm 3. First, we find the local minima and local maxima of the mth PE as $x_{\min,m} = \min_i\ x_{k,m}^{(i)}$ and $x_{\max,m} = \max_i\ x_{k,m}^{(i)}$, respectively, and then transmit them to the CU. The CU then finds the global maxima x_{Max} and global minima x_{Min}, and sends them back to all the PEs. Based on x_{Max} and x_{Min}, the particles in each PE are divided into $G = \lceil (x_{\text{Max}} - x_{\text{Min}})/\delta \rceil$ groups, where $\lceil a \rceil$ represents the smallest integer greater than a [24]. Note that δ provides the range of each group; if δ is large, then the number of groups in each PE is small and thus the algorithm precision is low. The mth PE calculates the average particle value $x_{\text{mean},j,m}$ and particle weight $w_{\text{mean},j,m}$ of group j, $j = 1, \ldots, G$, and transmits them to the CU. The CU uses these values to compute the particle replication factor ρ_j. It also ensures that the replication factor is an integer number by simple rounding-off operations and that $\sum_{j=1}^{G} \rho_j = N$.

This grouping method can be extended to multi-dimensional problem by operating Algorithm 3 on each of the dimensions of the particles. Assuming $\mathbf{x}_k^{(i)} = [x_{1,k}^{(i)}\ x_{2,k}^{(i)}\ \ldots\ x_{D,k}^{(i)}]^T$ is a D dimensional particle, then $\min_i\ \mathbf{x}_k^{(i)} = [\min_i x_{1,k}^{(i)}\ \min_i x_{2,k}^{(i)}\ \ldots\ \min_i x_{D,k}^{(i)}]^T$ and $\max_i\ \mathbf{x}_k^{(i)} = [\max_i x_{1,k}^{(i)}\ \max_i x_{2,k}^{(i)}\ \ldots\ \max_i x_{D,k}^{(i)}]^T$. Local extrema $\mathbf{x}_{\min,m} = \min_i\ \mathbf{x}_{k,m}^{(i)}$ and $\mathbf{x}_{\max,m} = \max_i\ \mathbf{x}_{k,m}^{(i)}$ of the mth PE are transmitted to CU and global extrema $\mathbf{x}_{\text{Max}}$ and $\mathbf{x}_{\text{Min}}$ are sent back to PE. Here $\mathbf{x}_{\min,m}$, $\mathbf{x}_{\max,m}$, $\mathbf{x}_{\text{Max}}$ and $\mathbf{x}_{\text{Min}}$ are all D dimensional vectors. For each dimension, particles are divided into G groups based on $\mathbf{x}_{\text{Max}}$ and $\mathbf{x}_{\text{Min}}$ and thus, there are $G \times D$ groups. Then the average particle value $\mathbf{x}_{\text{mean},j,m}$ and particle weight $w_{\text{mean},j,m}$ of group j, $j = 1, \ldots, G \times D$, are calculated and transmitted to the CU for calculating the particle replication factor ρ_j.

The PPF-IMH algorithm has advantages both in terms of algorithm and hardware performance. In each PE, the particles $\mathbf{x}_k^{(i)}$, $i = 1, \ldots, N$ are resampled using the IMH; thus particles with high weights are replicated and particles with low

Algorithm 2 Parallel Particle Filter with IMH

```
Input $\mathbf{z}_k$ and initial set $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$, $i = 1, \ldots, N$

for $k = 1$ to $K$ time step do

  Sampling {Generate particles and weights}
  for $i = 1$ to $(N + N_b)$ do
    $J(i) \sim \mathcal{U}[1, N]$ {Discrete uniform distribution}
    $\tilde{\mathbf{x}}_k^{(i)} \sim p\left(\mathbf{x}_k^{(i)} | \mathbf{x}_{k-1}^{(J(i))}\right)$
    Calculate $\tilde{w}_k^{(i)} = p\left(\mathbf{z}_k | \tilde{\mathbf{x}}_k^{(i)}\right)$
  end for

  IMH resampling
  Initialize the chain $\left(\bar{\mathbf{x}}_k^{(1)}, \bar{w}_k^{(1)}\right) = \left(\tilde{\mathbf{x}}_k^{(1)}, \tilde{w}_k^{(1)}\right)$
  for $i = 2$ to $(N + N_b)$ do
    $u \sim \mathcal{U}(0, 1)$
    $\alpha\left(\bar{\mathbf{x}}_k^{(i-1)}, \tilde{\mathbf{x}}_k^{(i)}\right) = \min\left\{\tilde{w}_k^{(i)} / \bar{w}_k^{(i-1)}, 1\right\}$
    $\left(\bar{\mathbf{x}}_k^{(i)}, \bar{w}_k^{(i)}\right) = \begin{cases} \left(\tilde{\mathbf{x}}_k^{(i)}, \tilde{w}_k^{(i)}\right), & u \le \alpha\left(\bar{\mathbf{x}}_k^{(i-1)}, \tilde{\mathbf{x}}_k^{(i)}\right) \\ \left(\bar{\mathbf{x}}_k^{(i-1)}, \bar{w}_k^{(i-1)}\right), & u > \alpha\left(\bar{\mathbf{x}}_k^{(i-1)}, \tilde{\mathbf{x}}_k^{(i)}\right) \end{cases}$
  end for
  Assign $\left\{\left(\mathbf{x}_k^{(i)}, w_k^{(i)}\right), i = 1, \ldots, N\right\}$ to $\left\{\left(\bar{\mathbf{x}}_k^{(i)}, \bar{w}_k^{(i)}\right), i = (N_b + 1), \ldots, (N + N_b)\right\}$
end for
```

weights are discarded. The remaining particles represent the posterior probability density function more accurately, resulting in improved performance. The PPF-IMH also results in reduced communication overhead. Specifically, in a traditional parallel architecture, M weights and M index factors have to be shared between the PEs and the CU, and, in the worst case scenario, there could be $M/2$ inter-PE communications [14]. For comparison, in the PPF-IMH, only the mth PE range factors $\mathbf{x}_{\min,m}$, $\mathbf{x}_{\max,m}$, $\mathbf{x}_{\text{Min}}$, and $\mathbf{x}_{\text{Max}}$, the average weights $w_{\text{mean},j,m}$, $j = 1, \ldots, G \times D$, and the replication factors ρ_j, $j = 1, \ldots, G \times D$ need to be transferred between the mth PE and the CU. Also, there is no inter-PE communication. As a result, the communication is reduced to $(2G \times D \times P) + (4 \times P)$, where G is the number of groups in each PE, D is the vector dimension and P is the number of PEs. Also, since the IMH resampler does not need all the normalized weights, resampling can start once the first weight is computed. Thus the computation time of the PPF-IMH method increases very mildly when compared to the parallel PF algorithm in [24].

3.3 PPF-IMH FPGA Implementation

The overall block diagram of the proposed PPF-IMH hardware implementation architecture is shown in Figure 1 which consists of four PEs and one CU. Local PF processing steps, such as particle generation, weight evaluation and IMH resampling, are executed in each PE. Global processing steps, such as computing global range and replication factors, are executed in the CU. Each PE commu-

Algorithm 3 Grouping method

```
Given particles and weights of the $m$th PE $\left(x_{k,m}^{(i)}, w_{k,m}^{(i)}\right)$, $m = 1, \dots, P$
*Find the local extrema at the mth PE*
**for** $m = 1$ to $P$ **do**
    $x_{\min,m} = \min_i \; x_{k,m}^{(i)}$
    $x_{\max,m} = \max_i \; x_{k,m}^{(i)}$
    Transmit $x_{\min,m}$ and $x_{\max,m}$ to the CU
**end for**

*Find global extrema in the CU*
$x_{\text{Min}} = \min_m \; x_{\min,m}$
$x_{\text{Max}} = \max_m \; x_{\max,m}$
Send $x_{\text{Min}}$ and $x_{\text{Max}}$ back to the PEs

*Divide particles into groups based on global extrema*
*Calculate the averages for each group in the PEs*
**for** $j = 1$ to $G$ **do**
    $x_{\text{mean},j,m} = \frac{1}{N_j} \sum_{i \in \text{Group}_j} x_{k,m}^{(i)}$
    $w_{\text{mean},j,m} = \frac{1}{N_j} \sum_{i \in \text{Group}_j} w_{k,m}^{(i)}$
    Send $w_{\text{mean},j,m}$ to the CU
    Calculate replication factor $\rho_j$ based on $w_{\text{mean},j,m}$
    Send $\rho_j$ to each PE (operate in the CU)
**end for**
```

nicates with the CU, but there is no communication among PEs. Figure 1 also shows the data that is transferred between the PE and the CU.

3.4 Processing Element Architecture

The PE block diagram is shown in Figure 2. The PE processes the input particles and executes the sampling, weighting and IMH sampling steps. After sampling, the particles are stored in the particle memory (PMEM), and the replicated particle index factors are stored in the replicated particle index memory (RMEM). Using the index from RMEM, each PE reads the resampled particles from PMEM, computes the local range factors $\mathbf{x}_{\max,m}$, $\mathbf{x}_{\min,m}$ and transmits them to the CU. After receiving the global range factors $\mathbf{x}_{\text{Min}}$, $\mathbf{x}_{\text{Max}}$, the resampled particles are divided into G groups, and the average particles $\mathbf{x}_{\text{mean},j,m}$ and average weights $w_{\text{mean},j,m}$ for the jth group are calculated. Next, the average weights of each group $w_{\text{mean},j,m}$ are sent to the CU to compute the replication factor ρ_j. The mean particles $\mathbf{x}_{\text{mean},j,m}$ are read from the mean particle memory (MPMEM) and sent to the sampling unit for generating particles at the next time step.

Figure 3 shows the IMH sampler architecture. When computing the acceptance probability, we use the modified method in [26] to avoid division computation. In particular, in our case, we accept particles following the procedure

$$\left(\mathbf{x}_k^{(i)}, w_k^{(i)}\right) = \begin{cases} \left(\tilde{\mathbf{x}}_k^{(i)}, \tilde{w}_k^{(i)}\right), & u\, w_k^{(i-1)} \le \tilde{w}_k^{(i)} \\ \left(\mathbf{x}_k^{(i-1)}, w_k^{(i-1)}\right), & u\, w_k^{(i-1)} > \tilde{w}_k^{(i)}, \end{cases}$$

where $u \sim \mathcal{U}(0,1)$. Specifically, the weight of a newly generated particle is first compared with the product of the uniformly distributed random variable u and

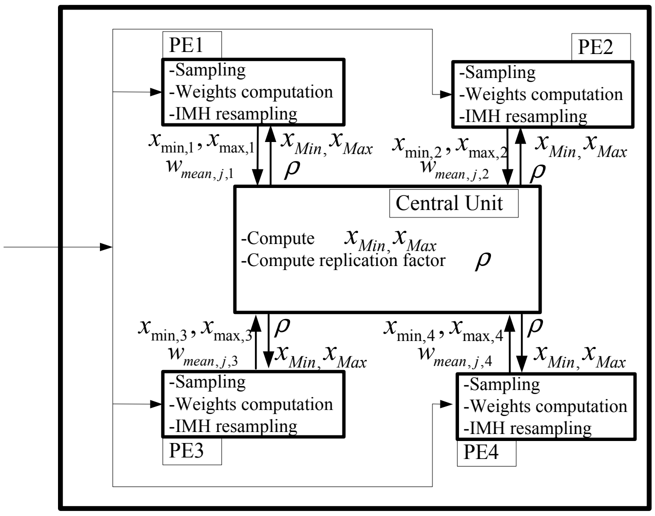
PE1
-Sampling
-Weights computation
-IMH resampling
PE2
PE3
PE4
Central Unit
-Compute x_{Min}, x_{Max}
-Compute replication factor ρ
$x_{\min,1}, x_{\max,1}$ $w_{mean,j,1}$
x_{Min}, x_{Max} ρ
$x_{\min,2}, x_{\max,2}$ $w_{mean,j,2}$
$x_{\min,3}, x_{\max,3}$ $w_{mean,j,3}$
$x_{\min,4}, x_{\max,4}$ $w_{mean,j,4}$

Fig. 1 PPF-IMH architecture with four PEs: PE1, PE2, PE3, and PE4. The mth PE, $m = 1, \dots, 4$ sends the average weights $w_{\text{mean},j,m}$, local minima $\mathbf{x}_{\text{min},m}$, local maxima $\mathbf{x}_{\text{max},m}$ to the CU and CU sends global minima $\mathbf{x}_{\text{Min}}$ and global maxima $\mathbf{x}_{\text{Max}}$ to the PEs.

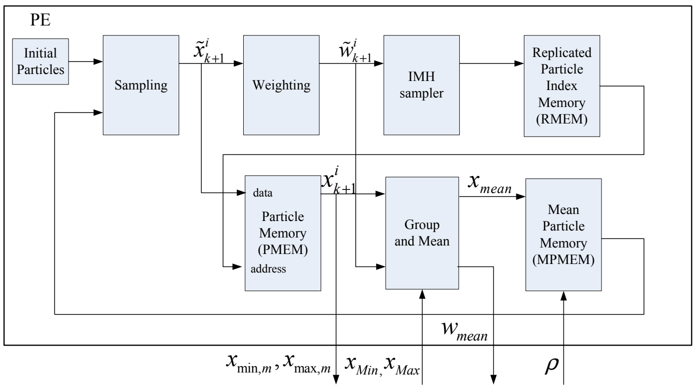

Fig. 2 Block diagram of a PE.

the weight of the last accepted particle in the chain. If the new particle weight is larger, it remains in the chain and its index is assigned to a new replicate index labeled r_i; otherwise, it is replicated once more, and the replicate index r_i remains unchanged.

The group-and-mean unit is used to divide the particles into different groups, based on the global ranges, and to calculate particle and weight averages in each group. For the one-dimensional problem, the architecture of this unit is shown in Figure 4. First, using the global range factors x_{Min}, x_{Max}, and the number of groups G, the range for each group, $\delta = (x_{\text{Max}} - x_{\text{Min}})/G$ is computed. Then, the thresholds γ of each group are generated based on δ as $\gamma_j = x_{\text{Min}} + (j-1)\delta$, $j = 1, \dots, G$. Each particle is then compared to the thresholds and placed in the

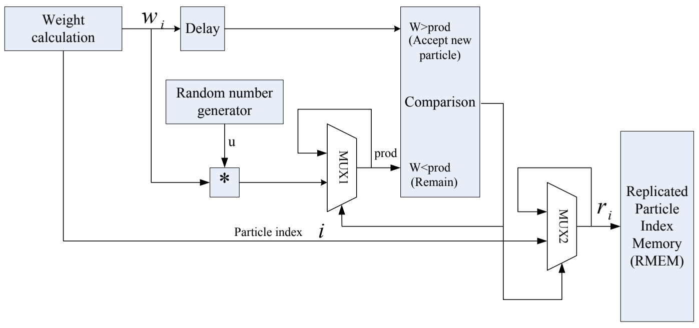

Fig. 3 Block diagram of the IMH sampler.

corresponding group. The particle values are accumulated, and the number of particles is counted in each group. Finally, the mean value $x_{\mathrm{mean},j}$ and the mean weight $w_{\mathrm{mean},j}$ are computed for each group. For multi-dimensional problem, since the computations for each dimension are independent, we apply this procedure to each dimension in parallel.

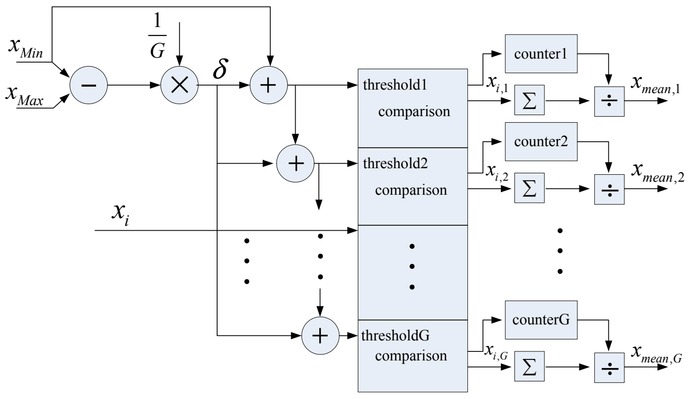

Fig. 4 Block diagram of the group-and-mean unit.

3.5 Central Unit Architecture

The CU executes global computations such as global range and replication factor computations. Its architecture is shown in Figure 5. Two comparators and multiplexers (MUXs) are used to generate $\mathbf{x}_{\mathrm{Min}}$ and $\mathbf{x}_{\mathrm{Max}}$. If the new local range $\mathbf{x}_{\mathrm{Min}}$ is smaller than the last accepted global range $\mathbf{x}_{\mathrm{Min}}$, we assign $\mathbf{x}_{\mathrm{min},m}$ to $\mathbf{x}_{\mathrm{min}}$, or keep the last value of $\mathbf{x}_{\mathrm{Min}}$ and a similar procedure is used to find $\mathbf{x}_{\mathrm{Max}}$. We use an accumulator and a multiplier to compute the replication factor. The accumulator inputs, $w_{\mathrm{mean},j,m}$, are normalized to guarantee that $\sum_{j=1}^{G\times D} \rho_j = N$. Thus, after each iteration, the number of PE particles is unchanged.

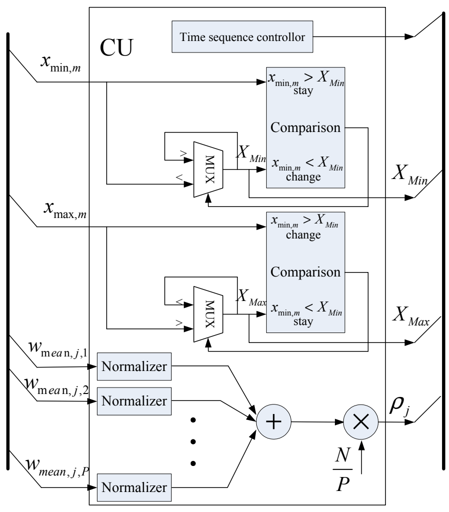

Fig. 5 Block diagram of the CU.

4 Waveform-agile Sensing and Implementation

4.1 Waveform-agile Sensing Algorithm

The dynamic system described by the state-space equations (1) and (2) assumes that measurements $\mathbf{z}_k$ are observed at time step k. In certain applications, these measurements could be determined by transmitted waveform $s_k(t; \boldsymbol{\theta}_k)$, with known fixed parameters $\boldsymbol{\theta}_k$. One possible way to improve the estimation performance of the state parameters is to adaptively control the transmit waveform parameters $\boldsymbol{\theta}_k$ at each time step k. Specifically, waveform-agile sensing is a closed-loop feedback optimization procedure that allows adaptive selection of the waveform parameters to be transmitted at the next time-step in order to optimize a cost function [9–11]. As our objective here is to accurately estimate the dynamic state $\mathbf{x}_k$, we can choose the cost function to be the MSE for the next time step.

We assume that the waveform $s_k(t; \boldsymbol{\theta}_k)$ to be transmitted at time step k has a parameter vector $\boldsymbol{\theta}_k$ that can be adaptively selected. The received waveform is analyzed to obtain the measurement vector $\mathbf{z}_k$ in Equation (2), and, consequently, the measurement noise vector $\mathbf{v}_k$ in Equation (2) is assumed to have a covariance matrix $\mathbf{R}(\boldsymbol{\theta}_k)$ that depends on $\boldsymbol{\theta}_k$. Using $\mathbf{z}_k$, we can obtain an estimate of the target state, $\hat{\mathbf{x}}_k$; thus the estimation error depends on the choice of $\boldsymbol{\theta}_k$. The proposed PPF-IMF approach can be applied to derive an efficient implementation of waveform-agile sensing. In particular, we will use the proposed PPF-IMF formulation to draw particles $\mathbf{x}_k^{(i)}$ from an importance density $q(\mathbf{x}_k | \mathbf{x}_{k-1}^{(i)}, \mathbf{z}_k, \boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_k)$, estimate the posterior probability density function and adaptively choose the waveform parameter $\hat{\boldsymbol{\theta}}_k$ that optimizes the predicted MSE in estimating $\mathbf{x}_k$ [10].

The covariance matrix for the target state estimate at time step k is given by

$$\mathbf{P}(\boldsymbol{\theta}_k) = E_{\mathbf{x}_k, \mathbf{z}_k | \mathbf{z}_{1:k-1}} [(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T (\mathbf{x}_k - \hat{\mathbf{x}}_k)]$$

where $E[\cdot]$ is the expectation operator and $\hat{\mathbf{x}}_k$ is the estimate of $\mathbf{x}_k$ given the measurement sequence $\mathbf{z}_{1:k-1}$. Under a high signal-to-noise ratio (SNR) assumption, the covariance matrix of error estimation can be approximated by the posterior Cramér-Rao lower bound (PCRLB) [2, 27–29]

$$\mathbf{P}(\boldsymbol{\theta}_k) \approx \mathbf{PCRLB}(\boldsymbol{\theta}_k) \tag{3}$$

that depends on the waveform parameter vector $\boldsymbol{\theta}_k$. The PCRLB can be computed from the predicted Fisher information matrix $\mathbf{I}_k$ using [2]

$$\mathbf{PCRLB}(\boldsymbol{\theta}_k) = \mathbf{I}_k^{-1}(\boldsymbol{\theta}_k)$$

where

$$\begin{aligned}\mathbf{I}_k(\boldsymbol{\theta}_k) &= \mathbf{Q}^{-1} + E\left[\tilde{\mathbf{H}}_k^T \mathbf{R}^{-1}(\boldsymbol{\theta}_k)\tilde{\mathbf{H}}_k\right] - \mathbf{Q}^{-1}\mathbf{F}\left(\mathbf{I}_{k-1}(\boldsymbol{\theta}_{k-1}) + \mathbf{F}^T\mathbf{Q}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{Q}^{-1} \\ &= \left(\mathbf{Q}^{-1} + F^T\mathbf{I}_{k-1}^{-1}(\boldsymbol{\theta}_{k-1})\mathbf{F}\right)^{-1} + E\left[\tilde{\mathbf{H}}_k^T\mathbf{R}^{-1}(\boldsymbol{\theta}_k)\tilde{\mathbf{H}}_k\right],\end{aligned}$$

$\tilde{\mathbf{H}}_{k+1} = [\nabla_{\mathbf{x}_{k+1}} h_{k+1}^T(\mathbf{x}_{k+1})]^T$ and ∇ denotes gradient operation. The specific representation of the measurement noise covariance $\mathbf{R}$ is related to the waveform type and parameters, and will be describe in Equation (7). Thus, the covariance matrix of error estimation can be calculated iteratively as

$$\mathbf{P}(\boldsymbol{\theta}_k) \approx \left(\left(\mathbf{Q}^{-1} + \mathbf{F}^T\mathbf{P}(\boldsymbol{\theta}_{k-1})\mathbf{F}\right)^{-1} + E\left[\mathbf{H}_k\mathbf{R}^{-1}(\boldsymbol{\theta}_k)\mathbf{H}_k\right]\right)^{-1} \tag{4}$$

where $\mathbf{H}_k = \nabla_{\mathbf{x}_k} h_k^T(\mathbf{x}_k)$.

The optimal waveform to be transmitted at the next time step is then obtained by optimizing the predicted MSE using $\mathbf{P}(\boldsymbol{\theta}_k)$ in Equation (4). The waveform-agile sensing problem can thus be stated as the selection of the waveform parameter

$$\hat{\boldsymbol{\theta}}_k = \arg\min_{\boldsymbol{\theta}_k} \operatorname{Tr}\bigl(\mathbf{P}(\boldsymbol{\theta}_k)\bigr),$$

where $\hat{\boldsymbol{\theta}}_k$ is the optimally chosen waveform parameter vector and $\operatorname{Tr}(\cdot)$ is the matrix trace.

4.2 Waveform-agile Tracking Application

We consider a waveform-agile tracking application problem, where a target's position and velocity in a 2-D Cartesian coordinate system need to be estimated. The target is tracked using a phased-array radar system, transmitting waveforms from a class of generalized frequency-modulated (GFM) waveforms with complex Gaussian envelopes [30]. A GFM waveform, at time step k, is given by

$$s_k(t;\boldsymbol{\theta}_k) = (\pi\alpha_k^2)^{-1/4}\ e^{-0.5(t/t_r)^2/\alpha_k^2}\ e^{j2\pi\beta_k\xi(t/t_r)}, \tag{5}$$

where α_k is the shape parameter of the Gaussian envelope, β_k is the frequency modulation (FM) rate, $\xi(t/t_r)$ is the time-varying phase function, and $t_r = 1$ s is a reference time. The waveform parameter vector that can be configured is given by $\boldsymbol{\theta}_k = [\alpha_k\ \ \beta_k]^T$. An example of waveforms we use are linear FM (LFM) waveforms; these are waveforms with quadratic phase function $\xi(t/t_r) = (t/t_r)^2$.

The target state at time k can be represented as $\mathbf{x}_k = [x_k \; y_k \; \dot{x}_k \; \dot{y}_k]^T$, where (x_k, y_k) and $(\dot{x}_k, \dot{y}_k)$ are the position and velocity of the target, respectively, in 2-D Cartesian coordinates. For this system, the state-transition is linear, so Equation (1) can be rewritten as $\mathbf{x}_k = \mathbf{F}\,\mathbf{x}_{k-1} + \mathbf{n}_k$. Here, the process noise $\mathbf{n}_k$ has covariance matrix given by $\mathbf{Q}$. The state transfer function $\mathbf{F}$ and $\mathbf{Q}$ are given by

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \delta_t & 0 \\ 0 & 1 & 0 & \delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = q \begin{bmatrix} \delta_t^2/3 & 0 & \delta_t^2/2 & 0 \\ 0 & \delta_t^2/3 & 0 & \delta_t^2/2 \\ \delta_t^2/2 & 0 & \delta_t & 0 \\ 0 & \delta_t^2/2 & 0 & \delta_t \end{bmatrix}, \tag{6}$$

where δ_t is the step interval and q is the intensity factor. If the radar is located at position (0, 0), the nonlinear relation between $\mathbf{x}_k$ and $\mathbf{z}_k$ is given by

$$\begin{aligned} \mathbf{z}_k &= [r_k \;\; \dot{r}_k \;\; \varphi_k]^T + \mathbf{v}_k \\ &= \left[\left(x_k^2 + y_k^2\right)^{1/2} \quad (\dot{x}_k\, x_k + \dot{y}_k\, y_k)/r_k \quad \arctan\left(y_k/x_k\right) \right]^T + \mathbf{v}_k \end{aligned}$$

and $\mathbf{v}_k$ is measurement noise with zero mean and covariance matrix $\mathbf{R}_k(\boldsymbol{\theta}_k)$.

In order to compute the covariance matrix of error estimation in (4), we first compute $\tilde{\mathbf{H}}_k = [\nabla_{\mathbf{x}_k} h_k^T(\mathbf{x}_k)]^T$ as

$$\nabla_{\mathbf{x}_k} h_k^T(\mathbf{x}_k) = \begin{bmatrix} \frac{\partial r_k}{\partial x_k} & \frac{\partial \dot{r}_k}{\partial x_k} & \frac{\partial \varphi_k}{\partial x_k} \\ \frac{\partial r_k}{\partial y_k} & \frac{\partial \dot{r}_k}{\partial y_k} & \frac{\partial \varphi_k}{\partial y_k} \\ \frac{\partial r_k}{\partial \dot{x}_k} & \frac{\partial \dot{r}_k}{\partial \dot{x}_k} & \frac{\partial \varphi_k}{\partial \dot{x}_k} \\ \frac{\partial r_k}{\partial \dot{y}_k} & \frac{\partial \dot{r}_k}{\partial \dot{y}_k} & \frac{\partial \varphi_k}{\partial \dot{y}_k} \end{bmatrix} = \begin{bmatrix} \dfrac{2\,x_k}{c\,r_k} & \dfrac{2\,f_c}{c}\left(\dfrac{\dot{x}_k}{r_k} - \dfrac{\dot{r}_k x_k}{r_k^2}\right) & -\dfrac{y_k}{r_k^2} \\ \dfrac{2\,y_k}{c\,r_k} & \dfrac{2\,f_c}{c}\left(\dfrac{\dot{y}_k}{r_k} - \dfrac{\dot{r}_k y_k}{r_k^2}\right) & \dfrac{x_k}{r_k^2} \\ 0 & \dfrac{2\,f_c}{c}(x_k/r_k) & 0 \\ 0 & \dfrac{2\,f_c}{c}(y_k/r_k) & 0 \end{bmatrix},$$

where f_c is the carrier frequency of the waveform and c is the waveform speed of propagation in the medium. The noise covariance matrix $\mathbf{R}(\boldsymbol{\theta}_k)$ is a 3×3 matrix that, for the GFM waveform in (5), is given by [30]

$$\mathbf{R}(\boldsymbol{\theta}_k) = \eta_k \begin{bmatrix} \frac{1}{2\alpha_k^2} + g(\beta_k) & 2\pi d(\beta_k) & 0 \\ 2\pi d(\beta_k) & (2\pi)^2 \alpha_k^2/2 & 0 \\ 0 & 0 & \psi \end{bmatrix}. \tag{7}$$

Here, η_k is the SNR, ψ is determined by the radar array properties and is independent of waveform parameter θ_k and

$$\begin{aligned} g(\beta_k) &= (2\pi\beta_k)^2 \int_{-\infty}^{\infty} \frac{1}{\alpha_k\sqrt{\pi}} \; \exp\left(-t^2\; \xi^2(t)/\alpha_k^2\right) dt \\ d(\beta_k) &= (2\pi\beta_k)^2 \int_{-\infty}^{\infty} \frac{t}{\alpha_k\sqrt{\pi}} \; \exp\left(t^2\; \xi'(t)/\alpha_k^2\right) dt\,. \end{aligned}$$

We can optimally choose the waveform parameter $\boldsymbol{\theta}_k$ to minimize the PCRLB using the following three steps.

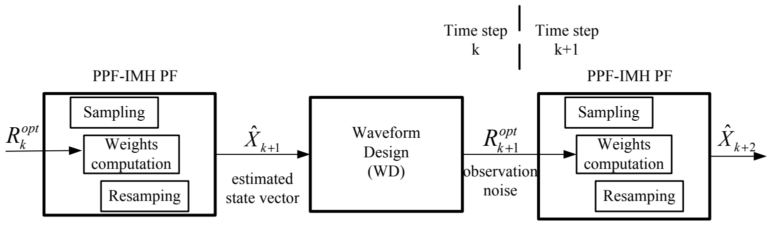

Fig. 6 Block diagram of tracking with waveform-agile design.

- *Prediction.* Predict the target state at time k as $\tilde{\mathbf{x}}_k = \mathbf{F}\hat{\mathbf{x}}_{k-1}$, where $\hat{\mathbf{x}}_{k-1}$ is the estimated state at time $(k-1)$ using the PPF-IMH algorithm.
- *Optimization.* Use $\tilde{\mathbf{x}}_k$ to calculate $E[\tilde{\mathbf{H}}_k^T \mathbf{R}^{-1}(\boldsymbol{\theta}_k)\tilde{\mathbf{H}}_k] \approx \mathbf{H}_k^T(\tilde{\mathbf{x}}_k)\mathbf{R}^{-1}(\boldsymbol{\theta}_k)\mathbf{H}_k(\tilde{\mathbf{x}}_k)$. Calculate $\mathbf{PCRLB}(\boldsymbol{\theta}_k)$ for every possible waveform parameter and choose $\boldsymbol{\theta}_k^{\text{opt}}$, which minimizes $\mathbf{PCRLB}(\boldsymbol{\theta}_k)$.
- *Updating.* Update observation noise covariance $\mathbf{R}(\boldsymbol{\theta}_k^{\text{opt}})$.

We can see that the computational complexity of the waveform-agile design method is fairly high as 2 matrix additions, 5 matrix multiplications and 5 matrix inversions (including a 4×4 matrix inversion) are included for each waveform parameter set. In the next section, we will modify the algorithm and make it amenable for FPGA hardware implementation.

4.3 FPGA implementation of waveform-agile design

The overall block diagram of the hardware architecture for waveform-agile design is shown in Figure 6. It consists of a PPF-IMH PF unit (described in Section 3.2) and a waveform-agile design unit. At each time step k, we use the PPF-IMH to obtain $\hat{\mathbf{x}}_k$, which is the estimation of $\mathbf{x}_k$ given measurements $\mathbf{z}_0$ to $\mathbf{z}_k$. Waveform-agile design steps such as prediction, optimization and updating, are operated in the waveform-agile design unit.

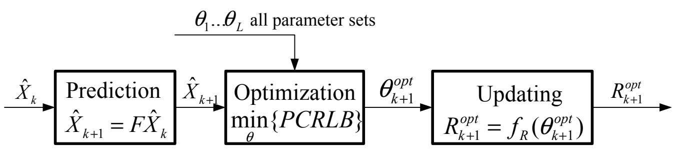

Fig. 7 Architecture of waveform-agile design unit.

The waveform-agile design block diagram is shown in Figure 7. The most computationally intensive step is the optimization. In the original optimization method, we find $\mathbf{R}(\boldsymbol{\theta}_k^{\text{opt}})$ by

$$\boldsymbol{\theta}_k^{\text{opt}} = \min_{\boldsymbol{\theta}_k} \operatorname{Tr}\left\{ \left((\mathbf{Q}^{-1} + \mathbf{F}^T \mathbf{I}_{k-1}^{-1}(\boldsymbol{\theta}_{k-1})\mathbf{F})^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{R}^{-1}(\boldsymbol{\theta}_k)\tilde{\mathbf{H}}_k \right)^{-1} \right\}.$$

This involves a 4×4 matrix inversion, which is difficult to implement in hardware. Using Woodbury's matrix identity [31], we modify the algorithm in order to reduce the computational complexity as

$$\begin{aligned}\boldsymbol{\theta}_k^{\text{opt}} &= \min_{\boldsymbol{\theta}_k} \operatorname{Tr}\left\{\left(\left(\mathbf{Q}^{-1} + \mathbf{F}^T \mathbf{I}_{k-1}^{-1}(\boldsymbol{\theta}_{k-1})\mathbf{F}\right)^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{R}^{-1}(\boldsymbol{\theta}_k)\tilde{\mathbf{H}}_k\right)^{-1}\right\} \\ &= \min_{\boldsymbol{\theta}_k} \operatorname{Tr}\left\{\mathbf{C} + \mathbf{C}\tilde{\mathbf{H}}^T\left(-\mathbf{R}(\boldsymbol{\theta}_k) - \tilde{\mathbf{H}}_k\mathbf{C}\tilde{\mathbf{H}}_k^T\right)^{-1}\tilde{\mathbf{H}}_k\mathbf{K}\right\} = \min_{\boldsymbol{\theta}_k} \operatorname{Tr}\left\{\mathbf{C}\tilde{\mathbf{H}}_k^T\mathbf{D}^{-1}\tilde{\mathbf{H}}_k\mathbf{C}\right\}.\end{aligned}$$

Here, $\mathbf{C}$ is a symmetric matrix that is given by

$$\mathbf{C} = \mathbf{Q}^{-1} + \mathbf{F}^T \mathbf{I}_{k-1}^{-1}(\boldsymbol{\theta}_{k-1})\mathbf{F} = \begin{bmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ b & 0 & d & 0 \\ 0 & b & 0 & d \end{bmatrix},$$

where a, b and d do not depend on $\boldsymbol{\theta}_k$ and $\mathbf{D} = -\mathbf{R}(\boldsymbol{\theta}_k) - \tilde{\mathbf{H}}_k\mathbf{C}\tilde{\mathbf{H}}_k^T$ is a 3×3 matrix. As a result, we simplify the 4×4 matrix inverse problem into a 3×3 matrix inverse problem. Furthermore, using

$$\tilde{\mathbf{H}}_{k+1}\mathbf{C}\tilde{\mathbf{H}}_{k+1}^T = \begin{bmatrix} 4a/c^2 & 4bf_c/c^2 & 0 \\ 4bf_c/c^2 & 4d{f_c}^2/c^2 & 0 \\ 0 & 0 & a/r_{k+1}^2 \end{bmatrix},$$

and by substituting $\mathbf{R}(\boldsymbol{\theta}_k)$, and simplifying the matrix computation, we obtain

$$\mathbf{D} = -\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & 0 \\ 0 & 0 & B \end{bmatrix} = -\begin{bmatrix} 2/\alpha_k + 4a/c^2 & 4bf_c/c^2 & 0 \\ 4bf_c/c^2 & 2\alpha_k + 4df_c^2/c^2 & 0 \\ 0 & 0 & \psi + a/r_k^2 \end{bmatrix}.$$

The inverse matrix can be represented as

$$\mathbf{D}^{-1} = -\begin{bmatrix} A_{22}/\zeta & -A_{12}/\zeta & 0 \\ -A_{12}/\zeta & A_{11}/\zeta & 0 \\ 0 & 0 & 1/B \end{bmatrix}$$

where $\zeta = |A_{11}A_{22} - A_{12}^2|$. Thus, the 3×3 matrix inversion requires only 13 multipliers, 5 adders and 2 dividers.

5 Algorithm Performance Results

5.1 PPF-IMH Simulation

We demonstrate the performance of our proposed PPF-IMH system using two dynamic state-space examples that have been previously used in the literature for comparison. The first system, state-space Model 1, depends on a 1-D dynamic state parameter x_k and is described by the following equations [32]

$$\begin{aligned} x_{k+1} &= 1 + \sin(0.04\pi k) + 0.5\,x_k + v_k \\ z_k &= \begin{cases} 0.2\,x_k^2 + n_k, & \text{if } k \leq 30 \\ 0.5\,x_k - 2 + n_k, & \text{if } k > 30 \end{cases} . \end{aligned} \tag{8}$$

Here, v_k is a random process modeled by a Gamma random variable with shape parameter 3 and scale parameter 2, and n_k is zero-mean, additive white Gaussian noise with variance 10^{-5}. The second example, state-space Model 2, is also a 1-D state space system in [7]

$$\begin{aligned} x_{k+1} &= 0.5\,x_k + 25\,\frac{x_k}{1+x_k^2} + 8\cos(1.2\,k) + v_k \\ z_k &= \frac{1}{20}x_k^2 + n_k \end{aligned} \tag{9}$$

where v_k and n_k are zero-mean, Gaussian random variables with variances $\sigma_v^2 = 10$ and $\sigma_n^2 = 1$, respectively.

The performance is given by RMSE, computed as

$$\text{RMSE} = \left(\frac{1}{K} \sum_{k=1}^{K} \frac{1}{\text{MC}} \sum_{l=1}^{\text{MC}} \left(\hat{\mathbf{x}}_{k,l} - \mathbf{x}_k \right)^2 \right)^{1/2} .$$

Here, $K = 30$ is the simulation path length, MC $= 100$ is the number of Monte Carlo simulations, $\mathbf{x}_k$ is the true state k and $\hat{\mathbf{x}}_{k,l}$ is the estimated state parameter in the lth Monte Carlo iteration at time k.

5.2 Effect of Number of Groups

In the proposed PPF-IMH algorithm, we divide the particles in each PE into G groups and use the average of each group as the new particle. The choice of G is crucial as it impacts the estimation accuracy. Figure 8 shows the RMSE tracking performance with respect to G for Model 1. Here the number of particles is chosen to be 1,000 and 2,000, and the number of PEs is chosen to be 1, 2 and 4. In all cases, we can see that as G increases, the RMSE decreases. But when G is greater than an optimal value G_{opt}, then there is no significant improvement in the RMSE. The G_{opt} value depends on the number of particles in each PE. From Figure 8, we can see that for N=1,000 particles, when P=4 PEs then $G_{\text{opt}} \approx 10$, when P=2 then $G_{\text{opt}} \approx 15$ and when P=1 then $G_{\text{opt}} \approx 20$. For N=2,000 particles, when P=4 then $G_{\text{opt}} \approx 15$; this is similar to the case of N=1,000 and P=2. Furthermore, since for large G the hardware resource utilization is also higher, here we choose G=10.

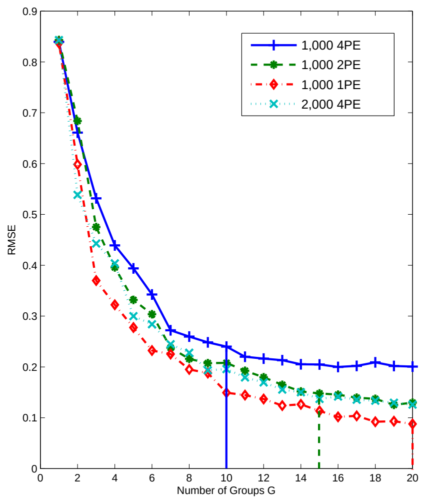

Fig. 8 RMSE performance of different group numbers and PE numbers

5.3 Estimation Performance

We use a P=4 PE parallel architecture for numerical simulations, where each PE processes 250 particles. We apply the parallel algorithm in [24] and also the new PPF-IMH algorithm to Model 1 and Model 2 systems. The corresponding estimation results are shown in Figure 9 and Figure 10. Table 1 shows the RMSE performance for the two models. As we can see, the RMSE performance of the PPF-IMH algorithm is significantly better when compared to the parallel algorithm in [24] for both models. In addition, the RMSE performance of the PPF-IMH is close to the PF with systematic resampling, which means that the performance degradation due to parallelization in [24] is compensated by IMH resampling.

Table 1 Comparison of RMSE performances.

Algorithms	RMSE for Model 1	RMSE for Model 2
Systematic resampling	0.24	4.06
Parallel algorithm in [24]	0.36	6.19
Proposed PPF-IMH algorithm	0.26	4.34

5.4 Waveform-Agile Target Tracking Algorithm Simulation

The simulation setup consists of a single target moving in a 2-D plane. The initial position and velocity of the target are $\mathbf{x}_0 = [5000 \quad 5000 \quad 100 \quad 100]^T$. We set the waveform parameters to $10^6 < \alpha_k < 10^{14}$ and $\beta_k = 0$. For the case without waveform design, we choose mid range value $\alpha_k = 10^9$. We use N=1,000 particles

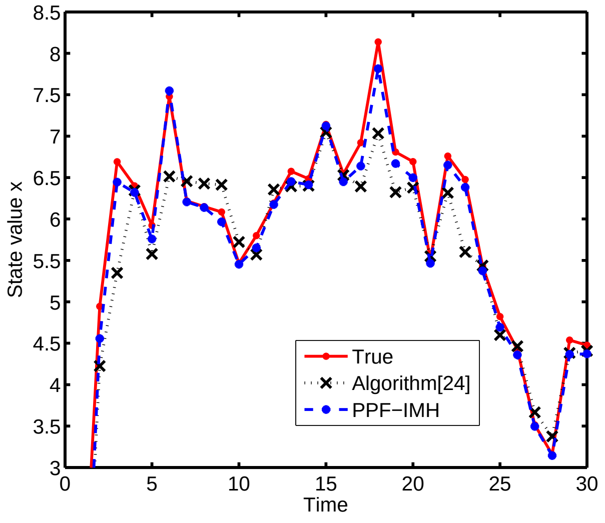

Fig. 9 Comparison of estimation performance for Model 1.

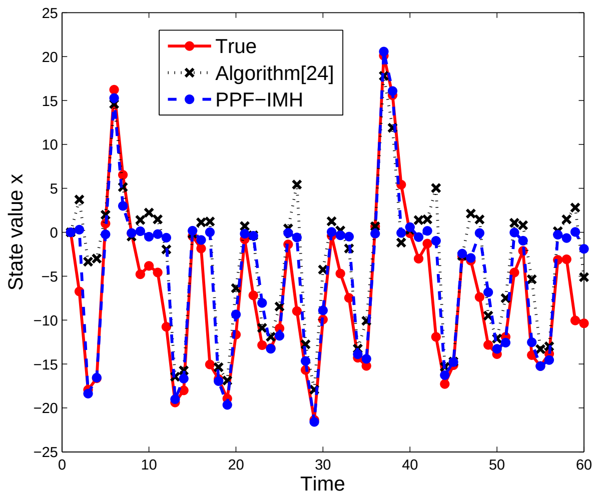

Fig. 10 Comparison of estimation performance for Model 2.

to track the target. The tracking results with and without waveform design for the x and y positions are shown in Figures 11(a) and 11(b), and Table 2 compares the tracking RMSE. We can see that the tracking performance with waveform design is much better and the RMSE is improved by about 10 times for x and y position estimation.

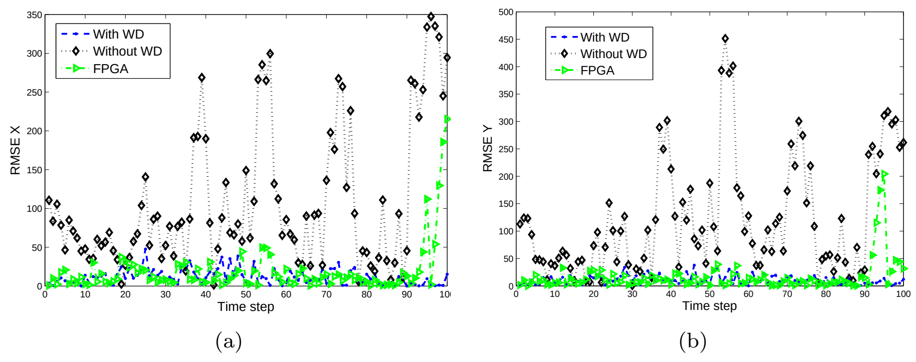

Fig. 11 RMSE of the (a) x-position and (b) y-position at each time step, demonstrating the improvement in performance when the waveform is adaptively selected at each time step.

Table 2 Comparison of RMSE performances.

State Parameter	Numerical simulation without Waveform-agility	Numerical simulation with Waveform-agility	FPGA implementation with Waveform-agility
x-position	141.82	15.12	37.57
y-position	161.52	13.91	33.27
$\dot{x}$-velocity	30.53	17.46	22.56
$\dot{y}$-velocity	37.00	16.18	20.23

6 Hardware Performance Results

6.1 PPF-IMH Implementation

The PPF-IMH hardware architecture for the system state estimation in Model 1 is implemented using Verilog HDL and synthesized on Xilinx Virtex-5 device (XC5VSX240T). The design was verified using Modelsim. Both the P=1 PE serial architecture and the P=4 PE parallel architectures were implemented. The RMSE values for the P=1 and P=4 PE architectures are 0.2686 and 0.3172, respectively. The RMSE is higher than the MATLAB generated numerical results because of the 14 bits fixed-point FPGA implementation.

Resource utilization: Table 3 summarizes the P=1 and P=4 PE architecture resource utilization. The sinusoidal and exponential functions are implemented using CORDIC units, and the rest of the units are implemented using DSP cores. For P=4 PE implementation, the PE and CU occupied slices utilizations are 408 (1%) and 420 (1%), respectively. Our resource usage is fairly low, for instance, only about 5% of the slice resource in a Xilinx Virtex-5 FPGA. Thus, such an implementation can support a much larger number of particles or multiple PFs which are required in biomedical signal processing applications [4].

Execution Time: Figure 12 shows the timing for one iteration of the proposed method for a system using N=1,000 particles and P=4 PEs. For our implementation, L_s = 21 is the sampling step delay determined by the sinusoid calculation time, L_w = 24 is the weighting latency determined by the time for calculating the exponential functions, L_r = 2 is the latency of the global range calcula-

Table 3 Resource utilization comparison.

Unit	Occupied Slices	Slice Registers	Slice LUTs	Block RAM	DSP48Es
P=1 processing element	398 (1%)	1,292 (1%)	1,363 (1%)	5 (1%)	10 (1%)
P=4 processing element	2,052 (5%)	5,749 (3%)	6,352 (4%)	18 (3%)	46 (4%)

tion, $L_m = 18$ is the time for computing the average value, and $L_\rho = 20$ is the time for calculating the replication factor. Thus, one PPF-IMH iteration takes $L_s + L_w + N + L_r + N + L_m + L_\rho = 585$ cycles. For a system clock rate of 100 MHz, the total processing period for one iteration is T_{total}=5.85 μs.

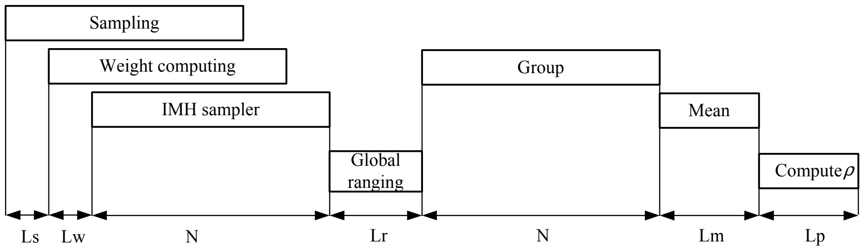

Fig. 12 Execution time of proposed method.

Communication overhead: The communication overhead of the proposed algorithm for a system using N=1,000 particles, P=4 PEs and G=10 groups is 96 bytes. This is a significant reduction compared with the traditional algorithm whose communication overhead is 2,500 bytes.

Scalability: Figures 13(a) and 13(b) show the execution time and communication overhead, respectively, for one processing iteration with respect to the number P of PEs for the proposed parallel architecture. The processing period curve saturates when P is large because there is no significant speedup when M/P approaches the constant latency L. In this case, the latency is given by $L = L_s + L_w + L_r + L_m + L_\rho =$ 85 cycles. From Figure 8, the RMSE performance slightly decreases as P increases. Thus, for a N=1,000 particle system, the P=4 PEs is a good choice.

In many applications such as in biomedical signal processing, the dimension of the state space is very large [4]. Consequently, a very large number of particles is required for satisfactory performance. In such cases, the processing time can be further reduced by using more PEs. Figure 13(a) shows the processing period for N=2,000 and N=4,000 particles. For these cases, a P=8 PEs architecture is a good choice.

From Figure 13(b), we can see that the communication overhead curve increases linearly with respect to P, and the slope is equal to $2G + 4$, where G is the number of groups in each PE. Thus, a lower value of G is more desirable for lower communication overhead. Unfortunately, a lower value of G results in degraded RMSE performance and thus the choice of G is a compromise between RMSE performance and communication overhead.

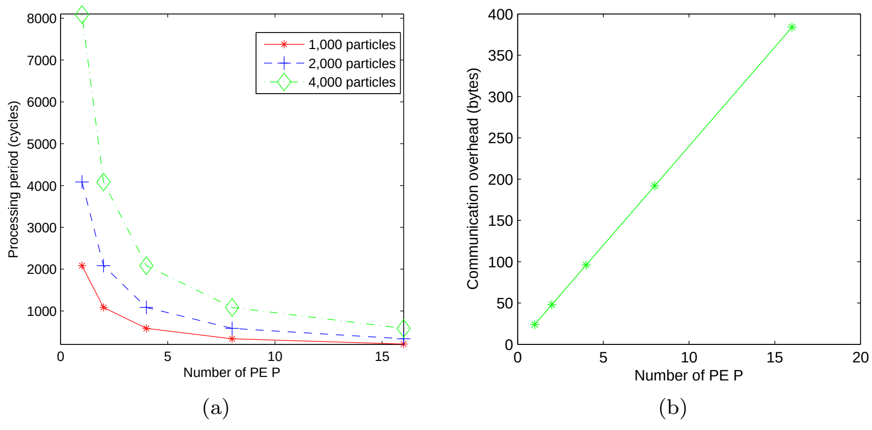

Fig. 13 Scalability of the proposed parallel architecture.

6.2 Target Tracking Hardware Synthesis Results

The waveform-agile radar tracking hardware architecture described in Section 4.2 is implemented using Verilog HDL and synthesized on a Xilinx Virtex-5 device (XC5VSX240T). The design was also verified using Modelsim. Here, we use a P=4 PEs PPF-IMH parallel architecture for a N=1,000 particle system. The particle weights are represented using 18-bit fixed-point. The target tracking result of the FPGA implementation is shown in Figure 14 to match well with the simulation results. The RMSE results from hardware experiments are shown in Table 2. Use of fixed-point data format degrades the performance since extremely small values are determined to be zero.

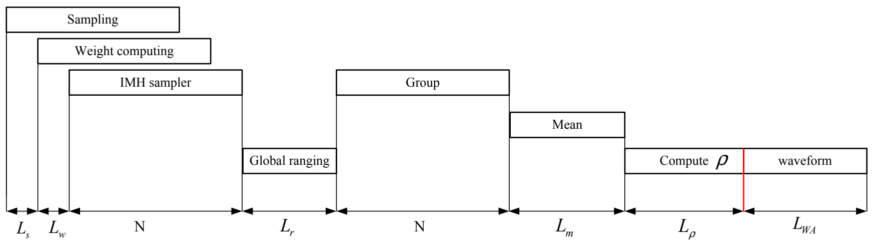

Fig. 14 Execution time of waveform-agile radar tracking problem.

Resource utilization: Table 4 summarizes the resource utilization for the waveform-agile design unit and the $P = 4$ PEs parallel architecture. The sinusoidal and exponential functions are implemented using CORDIC units, other calculations are implemented using DSP cores. We can see that the hardware resource utilization rate is fairly low; only about 10% of the total hardware resource is used. Thus, 10 such architectures should be able to fit onto a single Xilinx Virtex-5 platform.

Table 4 Resource utilization on Xilinx XC5VSX240T.

Unit	Occupied Slices	Slice Registers	Slice LUTs	Block Ram	DSP48Es
Waveform-agile design part	673 (1%)	735 (1%)	2229 (1%)	3 (1%)	45 (4%)
P=4 processing element	3,261 (8%)	7,590 (5%)	10,710 (7%)	48 (9%)	96 (9%)

Execution Time: Figure 14 shows the timing chart for one iteration of the proposed radar target tracking system. We can see that additional L_{WA} cycles are needed to obtain the optimal waveform parameter. In our design, $L_{\text{WA}} = 59$. In addition, $L_s = 4$ is the delay of the sampling step, $L_w = 56$ is the weighting latency determined by the calculation period of the exponential functions, $L_r = 2$ is the latency of the global range calculation, $L_m = 29$ is the time to compute the average value and $L_\rho = 34$ is the latency for calculating the replication factor. Thus, one iteration takes $L_s + L_w + N + L_r + N + L_m + L_\rho + L_{\text{WA}}$=684 cycles. For a system clock rate of 100 MHz, the total processing period for one iteration is $T_{\text{total}} = 6.84\ \mu$s.

7 Conclusions

In this paper, we proposed an efficient parallel architecture for implementing particle filters. This architecture can achieve both high speed and accurate estimation performance by using the independent Metropolis-Hastings sampler with the parallel PF implementation. The proposed method was also implemented on a Xilinx Virtex-5 FPGA platform. While it is difficult to give a fair comparison with other FPGA based implementations due to differences in models and number of particles, we can still claim that the proposed algorithm modification provided a reduced computational time with a slightly higher resource utilization. We integrated the waveform agile sensing technique into the new PPF-IMH algorithm to adaptively and efficiently increase dynamic state estimation performance. Simulations based on waveform-agile target tracking application demonstrated that the estimation performance is significantly improved and the processing speed is faster due to the PF parallelization.

References

1. L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "A new parallel implementation for particle filters and its application to adaptive waveform design," in *IEEE Workshop on Signal Processing Systems*, San Francisco, CA, October 2010, pp. 19–24.
2. B. Ristic, S. Arulampalam, and N. J. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House Publishers, Norwood, MA, 2004.
3. Z.-G. Shi, S.-H. Hong, J.-M. Chen, K.-S. Chen, and Y.-X. Sun, "Particle filter-based synchronization of chaotic Colpitts circuits combating AWGN channel distortion," *Circuits, Systems and Signal Processing*, vol. 27, pp. 833–845, 2008.
4. L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "Multiple sensor sequential tracking of neural activity: Algorithm and FPGA implementation," in *Asilomar Conference on Signals, Systems and Computers*, November 2010, pp. 369–373.
5. R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME*, vol. 82, pp. 35–45, March 1960.
6. N. J. Gordon, D. J. Salmon, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proceedings in Radar and Signal Processing*, 1992, vol. 140, pp. 107–113.
7. M. 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, pp. 173–449, February 2002.
8. A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, pp. 197–208, 2000.
9. D. J. Kershaw and R. J. Evans, "Optimal waveform selection for tracking systems," *IEEE Transactions on Information Theory*, vol. 40, pp. 1536–1550, September 1994.
10. S. P. Sira, A. Papandreou-Suppappola, and D. Morrell, "Dynamic configuration of time-varying waveforms for agile sensing and tracking in clutter," *IEEE Transactions on Signal Processing*, vol. 55, pp. 3207–3217, July 2007.
11. S. P. Sira, A. Papandreou-Suppappola, and D. Morrell, *Advances in Waveform-Agile Sensing for Tracking*, San Rafael, CA, Morgan & Claypool Publishers, 2009.
12. A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Architectures and memory schemes for sampling and resampling in particle filters," in *Digital Signal Processing Workshop*, August 2004, vol. 1, pp. 92–96.
13. A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP Journal of Applied Signal Processing*, vol. 17, pp. 2888–2902, 2005.
14. M. Bolić, *Architectures for Efficient Implementation of Particle Filters*, Ph.D. thesis, State University of New York at Stony Brook, 2004.
15. M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP Journal of Applied Signal Processing*, vol. 15, pp. 2267–2277, 2004.
16. M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Process.*, vol. 7, pp. 2442–2450, July 2005.
17. S. Hong, Z. Shi, J. Chen, and K. Chen, "Compact resampling algorithm and hardware architecture for particle filters," in *IEEE International Conference on Communications, Circuits and Systems*, 2008, vol. 2, pp. 886–890.
18. S. Hong, Z. Shi, J. Chen, and K. Chen, "Novel roughening algorithm and hardware architecture for bearings-only tracking using particle filter," *Journal of Electromagnetic Waves and Applications*, vol. 22, pp. 411–422, 2008.
19. S. Hong, Z. Shi, J. Chen, and K. Chen, "A low-power memory-efficient resampling architecture for particle filters," *Circuits, Systems and Signal Processing*, vol. 29, pp. 155–167, 2010.
20. S. Hong, M. Bolić, and P. M. Djurić, "An efficient fixed-point implementation of residual resampling scheme for high-speed particle filters," *IEEE Signal Processing Letters*, vol. 11, pp. 482–485, May 2004.
21. C. Berzuini, N. G. Best, W. R. Gilks, and C. Larizza, "Dynamic conditional independence models and Markov chain Monte Carlo methods," *Journal of the American Statistical Association*, vol. 92, pp. 1403–1412, 1997.

22. A. C. Sankaranarayanan, R. Chellappa, and A. Srivastava, "Algorithmic and architectural design methodology for particle filters in hardware," in *IEEE International Conference on Computer Design*, October 2005, pp. 275–280.
23. A. C. Sankaranarayanan, A. Srivastava, and R. Chellappa, "Algorithmic and architectural optimizations for computationally efficient particle filtering," *IEEE Transactions on Image Processing*, vol. 17, pp. 737–748, May 2008.
24. B. B. Manjunath, A. S. Williams, C. Chakrabarti, and A. Papandreou-Suppappola, "Efficient mapping of advanced signal processing algorithms on multi-processor architectures," in *IEEE Workshop on Signal Processing Systems*, October 2008, pp. 269–274.
25. C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, Springer-Verlag, New York, 2004.
26. S. Hong, Z. Shi, and K. Chen, "Easy-hardware-implementation MMPF for maneuvering target tracking: Algorithm and architecture," *Journal of Signal Processing Systems*, vol. 61, pp. 1–5, November 2009.
27. P. Tichavsky, C. H. Muravchik, and A. Nehorai, "Posterior Cramer-Rao bounds for discrete-time nonlinear filtering," *IEEE Transactions on Signal Processing*, vol. 46, pp. 1386–1396, 1998.
28. J. Zhang, B. Manjunath, G. Maalouli, A. Papandreou-Suppappola, and D. Morrell, "Dynamic waveform design for target tracking using MIMO radar," in *Asilomar Conference on Signals, Systems and Computers*, November 2008, pp. 31–35.
29. J. Zhang, Q. Ding, S. Kay, A. Papandreou-Suppappola, and M. Rangaswamy, "Agile multi-modal tracking with dependent measurements," in *Asilomar Conference on Signals, Systems and Computers*, November 2010.
30. S. P. Sira, A. Papandreou-Suppappola, and D. Morrell, "Time-varying waveform selection and configuration for agile sensors in tracking applications," in *IEEE International Conference of Acoustics, Speech and Signal Processing*, March 2005, vol. 5, pp. 881–884.
31. M. A. Woodbury, "Inverting modified matrices," *Statistical Research Group, Princeton University, Princeton, NJ*, vol. 42, pp. 4, 1950.
32. R. van der Merwe, A. Doucet, J. F. G. de Freitas, and E. Wan, "The unscented particle filter," in *Advances in Neural Information Processing Systems*, Dec. 2000, vol. 13, pp. 584–590.