J Sign Process Syst
DOI 10.1007/s11265-009-0434-4

Study of Algorithmic and Architectural Characteristics of Gaussian Particle Filters

Miodrag Bolić · Akshay Athalye · Sangjin Hong · Petar M. Djurić

Received: 9 October 2007 / Revised: 5 November 2009 / Accepted: 5 November 2009

Abstract In this paper, we analyze algorithmic and architectural characteristics of a class of particle filters known as Gaussian Particle Filters (GPFs). GPFs approximate the posterior density of the unknowns with a Gaussian distribution which limits the scope of their applications in comparison with the universally applied sample-importance resampling filters (SIRFs) but allows for their implementation without the classical resampling procedure. Since there is no need for resampling, we propose a modified GPF algorithm that is suitable for parallel hardware realization. Based on the new algorithm, we propose an efficient parallel and pipelined architecture for GPF that is superior to similar architectures for SIRF in the sense that it requires no memories for storing particles and it has very low amount of data exchange through the communication network. We analyze the GPF on the bearings-only tracking problem and the results are compared with results obtained by SIRF in terms of computational complexity, potential throughput, and hardware energy. We consider implementation on FPGAs and we perform detailed comparison of the GPF and SIRF algorithms implemented in different ways on this platform. GPFs that are implemented in parallel pipelined fashion on FPGAs can support higher sampling rates than SIRFs and as such they might be a more suitable candidate for real-time applications.

Keywords Gaussian particle filters · Sample-importance resampling filters · FPGA · Parallel architecture · Algorithmic modifications

1 Introduction

Sequential importance sampling based filters or particle filters (PFs) have recently been in the focus of extensive research [8]. These filters are applied to problems that can be formulated by dynamic state space (DSS) models. DSS models describe the evolution of a state of interest with time and the observations as a function of the state. PFs are used to estimate the unobserved states or their functions or to detect events described by the states. They outperform other filters in many important practical situations, and their flexibility in addressing a wide variety of problems makes them very attractive.

PFs are methods for sequential signal processing where the distributions of interest are approximated by discrete random measures which are represented by

This work was supported by the NSF under Awards CCR-9903120 and CCR-0220011.

M. Bolić (✉)
School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada
e-mail: mbolic@site.uottawa.ca

A. Athalye · S. Hong · P. M. Djurić
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794-2350, USA

A. Athalye
e-mail: athalye@ece.sunysb.edu

S. Hong
e-mail: snjhong@ece.sunysb.edu

P. M. Djurić
e-mail: djuric@ece.sunysb.edu

Published online: 21 November 2009

samples (particles) from the space of unknowns and weights associated to the particles. Particle filters implement three steps: 1) propagation of (proposal of new) particles, 2) computation of their weights (by using the principle of importance sampling), and 3) resampling. There are many variations of PFs, but in this paper we consider two types. One employs the sample-importance-resample (SIR) algorithm and we refer to these filters as sample-importance-resample filters (SIRFs). The other filters do not employ resampling but use Gaussian approximations of the desired densities, and we call these filters Gaussian particle filters (GPFs) [19].

Applications of the GPF are restricted to problems where the filtering densities can be approximated by Gaussian distributions. The GPF can be applied to any problem where the Extended Kalman Filter (EKF) or the unscented Kalman filter (UKF) can be used. In [28] it was stated that the GPF is asymptotically optimal, in contrast to the EKF and UKF, and that its performance is better for as long as enough random particles are employed by the GPF. The better performance of the GPF with respect to that of the EKF and UKF has been demonstrated in [26], where as metrics for comparison were employed the Kullback-Leibler and χ^2 distances between the filtering distributions obtained by these filters and the one estimated by the standard particle filtering method.

The number of applications where one can use GPFs is large and they include localization, tracking, and navigation. Examples of navigation include map based localization of objects [25] and those of tracking comprise target tracking based on bearing measurements [1], ballistic radar data [17, 23] or blind doppler data [23]. Examples of navigation include global positioning systems [1] and navigation of road vehicles [18]. Other applications are in data association and fusion [22].

Recently, the GPF was used for effective message representation/approximation in belief propagation algorithms for a wireless networks application [30]. In [5], the GPF was emplyed for the implementation of a probability hypothesis density filter (a multiple-target filter that recursively estimates the number of targets and their state vectors from sets of observations). In [11], the GPF was exploited for tracking of a maneuvering target and it showed excellent performance even for complex target maneuvering patterns. In [13], the GPF was used for in-motion alignment of the inertial navigation system when the observation variable is the velocity. The simulation results showed that the GPF is much more robust than the UKF in presence of initial misalignment.

Since all of the above applications are real-time applications and particle filters are expected to perform better than traditional algorithms, it is worthwhile analyzing and improving the implementation aspects of particle filters. PFs are very computationally extensive and that restricts their application to real-time problems. Previous work dealt with several different aspects of implementation of PFs and it was focused mainly on SIRFs. Several modifications of the particle filtering algorithm that improve execution time and simplify the particle filtering steps were proposed in [2]. Novel particle filters designed to overcome the barrier of sequential implementation and brought closer to a fully parallel implementation were developed and studied in [3]. Architectures for particle filters applied to the bearings-only tracking problem were proposed in [3] and [15]. The hardware implementation improved the speed of the particle filter in comparison with the implementation of the same filter on digital signal processors by 50 times.

In some of our previous work, it was shown that many applications require parallel implementation of SIRF in order to achieve real-time requirements. Parallel SIRFs were implemented with multiple processing elements (PEs), where each PE processes a fraction of the total number of particles, and a single central unit controls the operations of the PEs. The sample and importance steps of SIRFs are inherently parallel; therefore most of the effort is focused on modifying the resampling procedure so that it is more suitable for parallel implementation and on modifying the algorithm so that the data exchange between the parallel PEs is reduced and made deterministic [3].

The GPF has some properties that makes it very suitable for parallel implementation with multiple PEs. GPFs do not require a resampling procedure and therefore there is no need for exchanging particles between the PEs. So, the data exchange in GPFs, though not negligible, is significantly lower than in SIRFs and is deterministic.

In this paper, the GPF algorithm is first modified using loop fusion and some additional transformations so that it is suitable for pipelined hardware implementation with multiple PEs. These modifications result in an attractive feature of the GPFs in that they can be implemented without storing particles between successive time instants. This eliminates the need for memories in hardware and provides freedom of using a large number of particles for processing without having constraints by memory size. The latter might be important for embedded applications. In SIRFs, the number of used particles are restricted by the practical limits on

memory capacity determined by the available chip area and speed.

In the paper we also analyze the algorithms and high level architectures of SIRFs and GPFs from a hardware implementation viewpoint on the basis of execution throughput, memory requirements and complexity of data exchange between CUs and PEs in case of parallel implementation. The application that is used for the analysis is the bearings-only tracking problem [12]. We consider the FPGA design platform for single PE and multiple PE configurations. We have determined that SIRFs are more energy efficient for low capacity single PE implementations and that GPFs are superior in executing a large number of particles with multiple PEs and at high sampling frequencies.

The choice between traditional SIRFs and GPFs for hardware implementation will always involve a tradeoff [4]. Although GPFs have the aforementioned advantages for parallel hardware implementation, they suffer from increased complexity due to some operations that are particular to the GPF algorithm. These include a more complicated particle generation (sampling) step as compared to the one of SIRFs, and Cholesky decomposition of covariance matrices. Complexity analyses of GPFs and SIRFs are provided as well as elaboration on how to choose the right filter for an application where the critical features of the filters are the number of particles and the model dimension.

A hardware implementation of GPFs is presented in [16]. In this paper, the original GPF algorithm is implemented in hardware in pipelined but not parallel fashion. The paper also provides a novel architecture and a mapping method for the control path of the design. This paper is different from [16] in that it shows algorithmic modifications of GPFs and analyzes different algorithmic/architectural parameters of both SIRFs and GPFs especially for parallel implementation.

The rest of the paper is organized as follows. Section 2 describes the way the GPF is implemented in the paper and the main design parameters that are used for quantifying the performance of the implementation. Section 3 provides a brief review of the theory of GPFs. Section 4 presents the proposed modifications of GPFs and an analysis of the algorithmic complexity of GPFs in terms of temporal and spatial concurrency. Section 5 discusses implementation issues for sequential and concurrent implementations of both SIRFs and GPFs. A data flow analysis and high level architecture characterization are given in this section. A lower level comparison between SIRFs and GPFs in terms of resource utilization and speed is presented in Section 6. This section is intended to give an estimate of the resource usage and energy consumption of SIRFs and GPFs based on the algorithm and high level architectures. Section 7 concludes the paper.

2 Design Metrics and Their Relation to PFs

In order to fulfill the real-time requirements, we consider parallel pipelined implementation of GPFs and SIRFs on an FPGA platform. Next, we describe several concepts that are used in the paper.

Concurrency of operation means that different instructions or different parts of the instructions can be executed at the same time on different hardware resources (pipelining) [21]. In order to apply pipelining efficiently, it is necessary to work with streams of data. The longer the streams are, the more efficient the pipelining is because the time overhead associated with initiating and ending the pipelining is less important. GPFs and SIRFs are stream based algorithms where the streams are particles and not the input observations as it is common with other DSP algorithms.

Data parallelism means that the same instructions can be executed on different data sets using different hardware resources. In this paper we analyze algorithms for which we use time-multiplexed hardware for parts of the algorithms with data dependencies. For the parts of the algorithm without data dependencies, the one-to-one mapping method is used, which means that every operation in the algorithm has its own block in hardware. One-to-one mapping results in significant speed improvements if both concurrency of operations and data parallelism can be utilized.

The reasons (from hardware standpoint) for using the design of SIRF as a reference are: 1. it has relatively simple independent operations in comparison with more advanced particle filters, which entails smaller area requirements and 2. it has a relatively large number of particles per PE in comparison with the pipelining depth, and that justifies pipelined and parallel implementation. In the sequel we show that the GPF is even better suited for parallel hardware implementation.

In this paper, we analyze the algorithms and high level architectures of SIRFs and GPFs from a hardware implementation viewpoint on the basis of the following metrics: (a) execution throughput, (b) memory requirements, and (c) complexity of data exchange between CUs and PEs in case of parallel implementation. These metrics are evaluated against model dimension and number of used particles. We selected these metrics

because increasing any of them also increases the complexity of the hardware implementation.

Execution throughput is defined as the input sampling frequency that can be achieved for a specified number of particles. The main design criterion in this paper is high speed, so we try to maximize the input sampling frequency for a given number of particles. Regarding data exchange, we consider the number of data exchanges between the CU and the PEs as well as the type of exchange which can be either deterministic (known before run time) or random (unknown before run time). The data exchange in GPFs, though not negligible, is significantly lower than in SIRFs and is deterministic. This makes the GPF a *better* candidate for parallel implementation. Memory requirements include the amount of memory that is needed to store the particles and their weights.

On the implementation side, energy and area requirements for FPGA are analyzed.

3 Theory of Gaussian Particle Filtering

PFs are used in problems which are represented using DSS models. These models involve a state equation which shows how the state evolves with time and an observation equation that relates the noisy observations to the state. These equations have the form:

$$\mathbf{x}_n = \mathbf{f}_n(\mathbf{x}_{n-1}, \mathbf{u}_n)$$
$$\mathbf{y}_n = \mathbf{g}_n(\mathbf{x}_n, \mathbf{v}_n) \tag{1}$$

where $n \in \mathbb{N}$ is a discrete-time index, $\mathbf{x}_n$ is a signal vector of interest, and $\mathbf{y}_n$ is a vector of observations. The symbols $\mathbf{u}_n$ and $\mathbf{v}_n$ are noise vectors, and $\mathbf{f}_n$ and $\mathbf{g}_n$ are a signal transition function and a measurement function, which are assumed known. In a particle filtering framework, the objective is to estimate *recursively* in time the signal $\mathbf{x}_n$, $\forall n$, from the observations $\mathbf{y}_{1:n}$, where $\mathbf{y}_{1:n} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$.

The particle filters base their operation on representing relevant densities by particles drawn independently from an importance function $\pi(\mathbf{x}_n|\mathbf{x}_{n-1}^{(m)}, \mathbf{y}_{1:n})$ that has the same support as the posterior density. Each particle has a weight associated with it. Accordingly, if the particles $\{\mathbf{x}_{0:n}^{(m)}; m = 1, 2..., M\}$[1] are drawn from the importance function $\pi(\mathbf{x}_{0:n}|\mathbf{y}_{1:n})$, then an estimate of $E(\mathbf{h}(\mathbf{x}_{0:n}))$,

$$I = E(\mathbf{h}(\mathbf{x}_{0:n})) = \int \mathbf{h}(\mathbf{x}_{0:n}) p(\mathbf{x}_{0:n} \mid \mathbf{y}_{1:n}) d\mathbf{x}_{0:n} \tag{2}$$

where $\mathbf{h}(\mathbf{x}_{0:n})$ is some function of $\mathbf{x}_{0:n}$, can be obtained by the following expression [8, 9]:

$$\widehat{I}_M(\mathbf{h}) = \sum_{m=1}^{M} \mathbf{h}\left(\mathbf{x}_{0:n}^{(m)}\right) w_n^{(m)} \tag{3}$$

$$w_n^{(m)} = \frac{\tilde{w}_n^{(m)}}{\sum_{j=1}^{M} \tilde{w}_n^{(j)}} \tag{4}$$

$$\tilde{w}_n^{(m)} = \frac{p\left(\mathbf{y}_{1:n} \mid \mathbf{x}_{0:n}^{(m)}\right) p\left(\mathbf{x}_{0:n}^{(m)}\right)}{\pi\left(\mathbf{x}_{0:n}^{(m)} \mid \mathbf{y}_{1:n}\right)} \tag{5}$$

where $\mathbf{x}_{0:n}^{(m)}$ denotes the $m-$th stream of particles of the unobserved state of the system, $\mathbf{y}_{1:n}$ is the sequence of observed data, and $\tilde{w}_n^{(m)}$ is the non-normalized weight of the $m-$th particle.

Densities that play a critical role in sequential signal processing are the *filtering density*, $p(\mathbf{x}_n|\mathbf{y}_{1:n})$, and the *predictive density*, $p(\mathbf{x}_{n+l}|\mathbf{y}_{1:n})$, $l \geq 1$. It can be seen that SIRFs operate by propagating the approximations of the desired densities recursively in time. GPFs operate in the same way but by making the additional assumption that the filtering and predictive densities are Gaussians. In the latter case, therefore, only the parameters of the densities are propagated recursively in time (the mean vector and the covariance matrix). In the implementation of GPFs, Monte Carlo simulations are employed to obtain the estimates of the necessary density parameters and these estimates are recursively updated in time [19, 20]. Propagation of the first two moments only instead of the whole particle set simplifies the parallel implementation of the GPF significantly.

The GPF is much easier to implement when the prior density is used as importance function. This means that $\pi(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{y}_{1:n})$ is given by $p(\mathbf{x}_n|\mathbf{x}_{n-1})$. In this case, the GPF performs the steps shown in Pseudocode 1.

4 Algorithmic Modifications and Complexity Characterization

4.1 Temporal Concurrency

It is observed that the GPF contains four loops of M iterations, where each loop is used for calculation of one step in Pseudocode 1. Since the results from step one are used in the following steps, all M values of the states and weights must be saved in the memory for further processing. However, all four steps have the

[1] The notation $\mathbf{x}_{0:n}^{(m)}$ represents the set $\{\mathbf{x}_0^{(m)}, \mathbf{x}_1^{(m)}, \cdots, \mathbf{x}_n^{(m)}\}$.

Pseudocode 1 GPF algorithm with the prior density as importance function.

Input: The observation $\mathbf{y}_n$ and previous estimates $\boldsymbol{\mu}_{n-1}$ and Σ_{n-1}.
Setup: Mean $\boldsymbol{\mu}_0$ and covariance Σ_0 based on prior information.
Method:
GPF - Time update algorithm.
1. Draw conditioning particles from $\mathcal{N}(\mathbf{x}_{n-1}; \boldsymbol{\mu}_{n-1}, \Sigma_{n-1})$ to obtain $\{\mathbf{x}_{n-1}^{(m)}\}_{m=1}^{M}$.
2. Generate particles by drawing particles from $p(\mathbf{x}_n \mid \mathbf{x}_{n-1}^{(m)})$ to obtain $\{\mathbf{x}_n^{(m)}\}_{m=1}^{M}$.
GPF - Measurement update algorithm
3. (a) Calculate weights by $\tilde{w}_n^{(m)} = p(\mathbf{y}_n \mid \mathbf{x}_n^{(m)})$.
(b) Normalize the weights by $w_n^{(m)} = \tilde{w}_n^{(m)} / \sum_{m=1}^{M} \tilde{w}_n^{(m)}$.
4. Estimate the mean and covariance of the filtering distribution by
(a) $\boldsymbol{\mu}_n = \sum_{m=1}^{M} w_n^{(m)} \mathbf{x}_n^{(m)}$
(b) $\Sigma_n = \sum_{m=1}^{M} w_n^{(m)} (\mathbf{x}_n^{(m)} - \boldsymbol{\mu}_n)(\mathbf{x}_n^{(m)} - \boldsymbol{\mu}_n)^{\top}$.

same number of iterations and as such they are suitable for loop fusion [24].

The steps 1, 2 and 3(a) in Pseudocode 1 can be easily fused. Weight normalization (step 3(b)) requires that all the weights are known in order to form the sum of the weights and as such is not appropriate for loop fusion. However, we can modify the algorithm so that in calculating the mean and covariance coefficients we first use the non-normalized weights and perform normalization at the end with the sum of weights W_n. Step 4(b) cannot be fused in its original form since the mean is not known during the computation of the covariance coefficients until all of the M particles are processed. However, this step can be rewritten as follows:

$$\Sigma_n = \frac{1}{W_n} \sum_{m=1}^{M} \tilde{w}_n^{(m)} \mathbf{x}_n^{(m)} \mathbf{x}_n^{\top(m)} - \boldsymbol{\mu}_n \boldsymbol{\mu}_n^{\top}. \tag{6}$$

The second term on the right is constant, and it can be calculated outside the loop. The first term of the modified step 4(b) can be fused with the previous three steps. The fused steps are presented in Pseudocode 2. Note that the particles $\mathbf{x}_n^{(m)}$ do not need to be saved, which is advantageous for hardware implementation. There is additional processing outside the main loop that is still necessary (Pseudocode 3) and it involves: final computation of the mean and covariance coefficients (step 2), calculation of the final covariance coefficients using Eq. 6 (step 3) and performing the square root of a matrix (step 4). The square root of a matrix is needed because the Gaussian random number generation in step 1 of Pseudocode 2 utilizes $\mathbf{C}_n$ which is the square root of the covariance matrix. For that step, we selected the Cholesky decomposition as one of the possible solutions for square rooting of a matrix. In non-parallel implementations (i.e., with a single PE), step 1 in Pseudocode 3 is not necessary.

4.2 Spatial Concurrency

Spatial concurrency can be exploited for developing parallel architectures by mapping independent operations to multiple hardware units that operate in parallel. While temporal concurrency is exploited for reducing storage requirement and increasing throughput, spatial concurrency is exploited to increase throughput only. The throughput is increased by making all hardware units operate independently. Spatial concurrency is also exploited for multiple PEs execution.

It can be seen from the GPF algorithm that the particle and weight calculation steps for each of the M particles are independent of the other particles. The operations of each iteration of the loop in Pseudocode 2 are independent. Moreover the loop does not have any loop-carried dependence [14]. Thus, the loop level parallelism can be exploited for increasing throughput. This is done by having multiple independent PEs each processing a fraction of the total number of particles. The number of PEs, K, defines the degree of parallelism. The maximum degree of parallelism is obtained when each PE consists of only one particle ($K = M$).

There is dependence among the iterations of the step related to estimation of the mean and the covariance. To parallelize this step, partial sums of the mean and covariance $(\boldsymbol{\mu}_n^k$ and $\Sigma_n^k)$ are calculated by the PEs and then added in the end in the CU (step 1 of Pseudocode 3). In the pseudocode, quantities with subscript k represent the result of operations in the $k-$th PE. During the final addition of partial sums, the PEs communicate with the CU and the amount of data transferred corresponds to the dimension of the state space model.

In summary, with the modified algorithm, in the PEs we perform concurrent operations on the particles whereas in the CU, we carry out sequential post processing.

Pseudocode 2 Part of the GPF algorithm that runs in parallel on PEs after loop fusion is applied. The symbol k denotes the $k-$th PE and K denotes the total number of PEs.

Input: The observation $\mathbf{y}_n$ and previous estimates $\boldsymbol{\mu}_{n-1}$ and matrix $\mathbf{C}_{n-1}$ s.t. $\boldsymbol{\Sigma}_{n-1} = \mathbf{C}_{n-1}\mathbf{C}_{n-1}^{\top}$. For $n = 1$, mean $\boldsymbol{\mu}_0$ and covariance $\boldsymbol{\Sigma}_0$ are based on prior information.
Setup: Sum of weight, mean and covariance elements in the $k-$th PE: $W_n^k = 0$, $\boldsymbol{\mu}_n^k = 0$, $\boldsymbol{\Sigma}_n^k = 0$.
Method:

for $m = 1$ **to** M/K

1. Drawing conditioning particles from $\mathcal{N}(\mathbf{x}_{n-1}; \boldsymbol{\mu}_{n-1}, \Sigma_{n-1})$ to obtain $\mathbf{x}_{n-1}^{(m)}$.
2. Generation of particles: Draw a particle from $p\big(\mathbf{x}_n \mid \mathbf{x}_{n-1}^{(m)}\big)$ to obtain $\mathbf{x}_n^{(m)}$.
3. (a) Weight calculation $\tilde{w}_n^{(m)} = p\big(\mathbf{y}_n|\mathbf{x}_n^{(m)}\big)$.
   (b) Update the current sum of weights by $W_n^k = W_n^k + \tilde{w}_n^{(m)}$.
4. Updating mean $\boldsymbol{\mu}_n^k$ and covariance Σ_n^k by
   (a) $\boldsymbol{\mu}_n^k = \boldsymbol{\mu}_n^k + \tilde{w}_n\mathbf{x}_n$
   (b) $\Sigma_n^k = \Sigma_n^k + \tilde{w}_n\mathbf{x}_n\mathbf{x}_n^{\top}$.

end

4.3 SIR Algorithm

For the purpose of comparison, we provide the modified SIR algorithm in Pseudocode 4. Algorithmic modifications to extract temporal and spatial concurrency have also been applied to the traditional SIRF. These modifications have been detailed in [3]. The presented SIR algorithm is also considered to run in parallel on K PEs, where each PE processes M/K particles.

4.4 Computational Complexity Characteristic

A comparison of number of operations and memory requirements for one recursion of SIRFs and GPFs is shown in Table 1. The number of operations for particle generation and weight calculation is not presented because it is the same for both filters. Additional complexity for GPFs is added due to the generation of conditioning particles and computation of the mean vector and covariance matrix of the states. We would like to stress here that the complexity of these steps is related to the dimensionality of the model N_s as $O(N_s^2)$. For the GPF, the number of multiplication operations for the computations of the mean vector and covariance matrix is equal to $N_s(N_s + 1)$, whereas for the SIRF, the number of multiplications for calculation of the mean is N_s. On the other hand, the memory requirements of SIRFs are dominant. They increase with the increase of the dimension of the model as shown in [6]. In the table, $(N_s + 2)$ data per particle are used for storing N_s states, a weight and an index that is a result of the resampling process.

In presenting the amount of data exchange required for SIRFs and GPFs, we consider SIRFs with parallel resampling described in [3]. In both filters, operations without data dependencies are mapped to the PEs in order to allow for parallel processing. In GPFs, these are the four operations described by Pseudocode 2. In SIRFs with parallel resampling, the particle generation, weight calculation and partial resampling are mapped to the PEs. The parallel architecture for GPFs that consists of four PEs and the CU is shown in Fig. 1.

For an N_s dimensional model, the amount of data acquired by the CU from each of the K PEs is: $N_s(N_s + 1)/2 + N_s + 1 = (N_s^2 + 3N_s)/2 + 1$, where $N_s(N_s + 1)/2$ is the number of data in the covariance matrix $\Sigma_n^{(k)}$ (which is symmetric), N_s is the number of

Pseudocode 3 Part of the GPF algorithm that runs sequentially on the CU and exchanges data with K PEs.

Input: W_n^k, $\boldsymbol{\mu}_n^k$ and Σ_n^k for $k = 1, ..., K$.
Setup: sum of the weights $W_n = 0$, initial mean $\boldsymbol{\mu}_n = 0$ and covariance $\Sigma_n = 0$.
Method:

1. Collect data including update central sum of weights, mean and covariance

for $k = 1$ **to** K

   (a) $W_n = W_n + W_n^k$.
   (b) $\boldsymbol{\mu}_n = \boldsymbol{\mu}_n + \boldsymbol{\mu}_n^k$
   (c) $\Sigma_n = \Sigma_n + \Sigma_n^k$

end

2. (a) Scale mean and covariance
   (a) $\boldsymbol{\mu}_n = \boldsymbol{\mu}_n / W_n$
   (b) $\Sigma_n = \Sigma_n / W_n$
3. Compute the covariance estimate $\Sigma_n = \Sigma_n - \boldsymbol{\mu}_n\boldsymbol{\mu}_n^{\top}$
4. Compute the Cholesky decomposition of the matrix Σ_n in order to obtain $\mathbf{C}_n$.

Pseudocode 4 The SIR algorithm with concurrent operations on the PEs and sequential operations on the CU.

Input: The observation $\mathbf{y}_n$ and previously resampled particles $\{\tilde{\mathbf{x}}_{n-1}^{(m)}\}_{m=1}^{M/K}$.
Setup: Initial sum of weights $W_n^k = 0$.
Method:

PE Operation	CU Operation
for $m = 1$ **to** M/K	
1. Draw a particle $\mathbf{x}_n^{(m)}$ from $p(\mathbf{x}_n \vert \tilde{\mathbf{x}}_{n-1}^{(m)})$.	
2. Calculate weights by $\tilde{w}_n^{(m)} = w_{n-1}^{(m)} \cdot p(y_n \vert \mathbf{x}_n^{(m)})$.	
3. Update the partial sum of weights by $W_n^k = W_n^k + \tilde{w}_n^{(m)}$	
end	
4. Transfer W_n^k for $k = 1, ..., K$ to the CU	
	5. Compute the total sum W_n
	6. Compute the number of particles each PU will produce after resampling as $M_n^k = \lfloor W_n^k \cdot M / W_n \rfloor$
	7. The CU sends M_n^k for $k = 1, ..., K$ to the PEs
8. Resample M/K particles to obtain a new set of particles $\{\tilde{\mathbf{x}}_n^{(m)}\}_{m=1}^{M_n^k}$ and $w_n^{(m)} = W_n^k \cdot K/(W_n \cdot M)$	.
	9. Redistribute the particles $\tilde{\mathbf{x}}_n^{(m)}$ from the PEs with surplus to PEs with deficit of particles.

data in the mean vector $\boldsymbol{\mu}_n^{(k)}$, and 1 corresponds to the sum of weights $W_n^{(k)}$. The CU sends back to the PEs the calculated mean $\boldsymbol{\mu}_n$ and the Cholesky factorization $\mathbf{C}_n$ of the final covariance matrix. This is an additional $N_s(N_s+1)/2 + N_s$ data. Since, the same data are sent to each PE, it is beneficial to use mechanisms that allow for simultaneous transfer of data from the CU to all the PEs. For the bearings-only tracking problem, when there is one target, $N_s = 4$, so the number of data sent from each PE to the CU is 15 and the number of data sent from the CU back to the PEs is 14. This data exchange is depicted in Fig. 1. Overall, however, the number of data that are transferred during data exchange is much smaller than in the case of SIRFs, and it is fixed over time. This feature greatly increases the scalability [14] of the filter and also ensures that data exchange is never the dominant operation of GPFs. In contrast, in the worst case of SIRFs, $M/2$ particles are sent during data exchange [3].

The complexity characteristics of SIRFs and GPFs are summarized in Table 2. In the table, the effects of the increase in model dimension on different particle filter parameters are shown together with their resources requirements. The summary of the effects is as follows:

1. For GPFs, the number of operations per particle increases quadratically with model dimension. This significantly affects the complexity of the units for drawing conditioning particles and covariance estimation as well as the complexity of the CU. For SIRFs, the number of operations increases linearly with the model dimension.
2. The number of particles needed to achieve a required accuracy increases with the model dimension and that affects the sampling periods of both SIRFs and GPFs. However, SIRFs are more affected since there is an additional time for accessing the memories. There is a significant area increase in the spatial implementation of SIRFs, because physical memories are necessary to store particles and weights.

Table 1 Comparison of the number of operations and memory requirements of SIRFs and GPFs in a PE.

Algorithms	Gaussian random number generator	Multiplication operations		Memory requirements
		Drawing conditioning particles	Computation of estimate	
GPF	$2N_sM$	$N_s(N_s+1)M/2$	$N_s(N_s+1)M$	0
SIRF	N_sM	0	N_sM	$(N_s+2)M$

One sampling period of PF is analyzed. The operations that are the same in the particle generation and weight calculation steps are not considered.

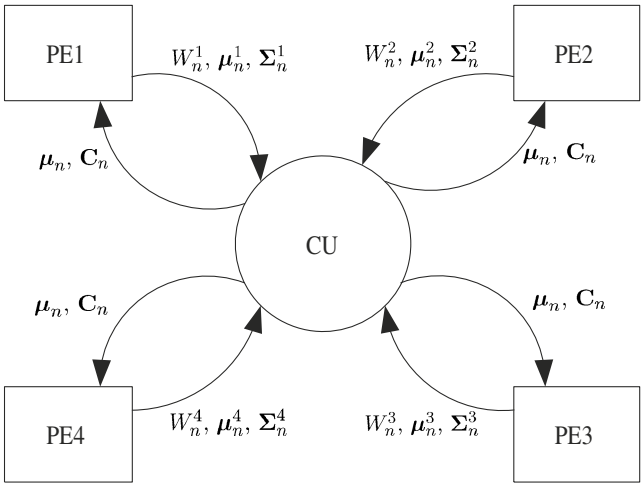


Figure 1 A parallel GPF model with $K = 4$ PEs.

3. For GPFs, the data exchange requirements increase quadratically with the model dimension, but the amount of data that is transferred between the PEs and the CU is several orders of magnitude lower than that for SIRFs. Besides, the data exchange pattern of GPFs is deterministic.
4. For GPFs, the complexity of mathematical operations increases resulting in a very large word length for finite precision processing. In such cases, floating point implementation is the more feasible option which implies requirements for increased area and/or increased sampling period. The finite precision processing of SIRFs is less affected by the increase in model dimension.

5 Implementation Issues

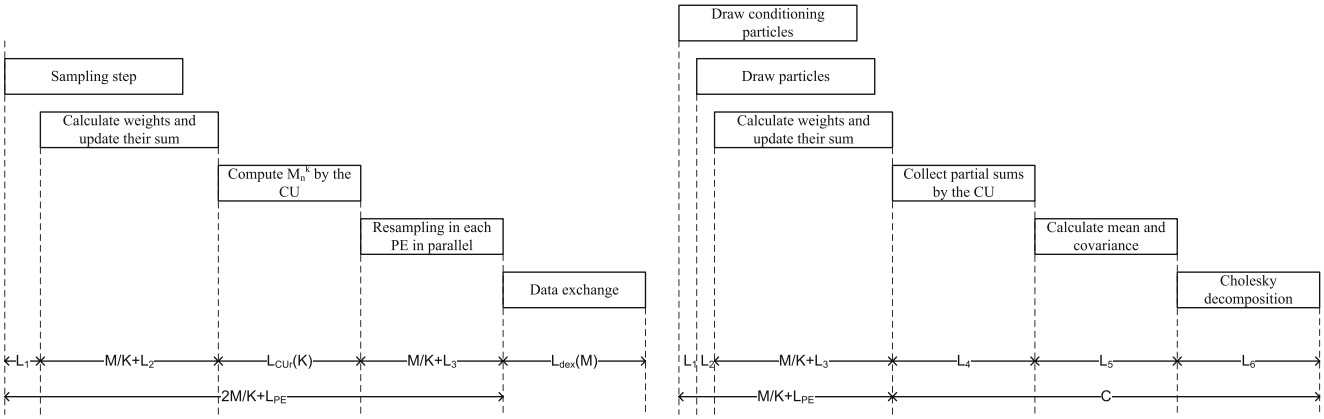
5.1 Latency and Sampling Period

Figure 2 shows the timing diagrams with the latencies of various operations of SIRFs and GPFs including the data transfer. For the GPF, the outputs of the first three logic blocks of the data flow are generated at clock speed, while the output of the updating mean and covariance block (step 4 in Pseudocode 2) and the output of the CU are generated with the PF sampling speed. In case of the SIR, the results of steps 1–4 in Pseudocode 4 are calculated at clock speed while the output of the resampling unit in step 5 is calculated at the PF sampling speed. The minimum sampling period that can be achieved with parallel SIRFs and GPFs for different number of particles and PEs is presented in Fig. 3. For SIRFs whose resampling is distributed to the PEs, the minimum sampling period that can be achieved is $\left(\frac{2M}{K} + L_{SIR} + L_{dex}(M)\right) \cdot T_{clk}$, where $\frac{2M}{K} + L_{SIR}$ is the latency of processing in the PEs. This is because each PE processes M/K particles, and it does sampling and weight calculation on each particle. After the summation of the weights of each PE is completed, the CU takes all the sums and computes the number of particles that each PE must produce after resampling [4]. The latency of the CU resampling is a function of the number of PEs represented by $L_{CUr}(K)$. Once this processing is completed, another M/K cycles are needed to perform resampling in each PE. The term $L_{SIR} = \sum_{i=1}^{3} L_i + L_{CUr}(K)$ accounts for the start up latencies of each block and is inherent to any pipelined feed forward data path. The $L_{dex}(M)$ represents the latency of the data exchange after particle allocation, which takes place in order to redistribute particles after resampling.

In the case of GPFs, the minimum sampling period is $\left(\frac{M}{K} + L_{GPF} + C\right) \cdot T_{clk}$, where, L_{GPF} is the net latency of the critical path. The latency $L_{GPF} = \sum_{i=1}^{3} L_i$ accounts for the start up latencies of the various blocks inside the PEs. The constant term $C = \sum_{i=4}^{6} L_i$ accounts for the latency of both the CU and the data exchange between the PEs and the CU. Thus, we see that though GPFs have a larger constant latency of the CU, for large number of particles the latency of the GPFs will be smaller than that of SIRFs. This is primarily because SIRFs require resampling, which is not only sequential and dependent on the result of processing *all* the particles in the PE, but also necessitates particle redistribution that has an execution time proportional to M. In our simulations, we assume that the clock period is equal for both filters $T_{clk} = 10$ns and that $L_{GPF} = 3 \cdot L_{SIR} = 300$ and $Ldex(M) = 0$ for SIRF. In Fig. 3, the minimum execution time for SIRFs is presented by black line and for GPFs by gray line. The large total latency affects the scalability of GPFs

Table 2 The effect of the number of particles and the number of states in the model on number of operations, number of particles, number of data exchanged and finite precision wordlength.

Parameter	Effects on algorithmic parameters	
	SIRF	GPF
Number of operations	Linear increase with the N_s	Quadratic increase with N_s
Number of particles	Increase with model dimension as in [6]	
Data exchange	Proportional to M	Proportional to N_s^2
Finite precision word length	Increase with the number and complexity of operations	



(a) SIRF Timing (b) GPF Timing

Figure 2 Timing diagrams for **a** SIRF and **b** GPF.

when M is small. On the other hand, when M is large, the sampling period of GPFs is almost twice smaller than the one of SIRFs. Hence GPFs are appropriate for high speed applications that require large number of particles on platforms that have enough resources for spatial parallel implementation.

5.2 Architectures and Resource Requirement

We consider spatial implementations of PFs applied to bearings-only tracking [12], with a one-to-one mapping between each operation and the hardware resource. The range of tracking is restricted to the region $[-2, -2] \times [2, 2]$ resulting in a 16-bit representation of the hidden states. We use the mean square error (MSE) of tracking as a performance evaluation criterion for fixed precision analysis, where the error due to finite precision arithmetic is kept within the limits of $\pm 10\%$ of the floating point value. Our simulations indicate

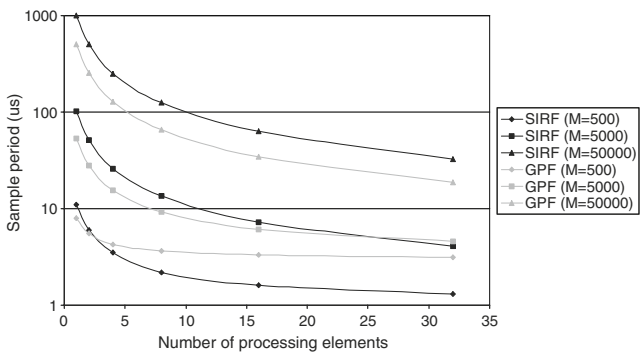


Figure 3 Minimum sampling period versus number of PEs of parallel GPFs and SIRFs for $M = \{500, 5000, 50000\}$. Spatial implementation of PFs is assumed.

that the steps 1–3 in Pseudocode 2 are less sensitive to finite precision effects. The MSE within defined limits is achieved using 16 bit representations for the operations of these steps. However, step 4 in Pseudocode 2 and all the steps of Pseudocode 3 are very sensitive to finite-precision effects. One of the operations is Cholesky decomposition which requires that the input matrix is positive definite, a condition that may not be satisfied for representations which use small number of bits. In order to alleviate this problem, 40 bits of precision are necessary for operations in step four of Pseudocode 2. Similarly, all the operations that are executed in the CU (Pseudocode 3) require more than 50 bits.

Figure 4 shows the basic architecture of the GPF's PE, where each operation in the algorithm has a dedicated hardware unit. The steps of drawing conditioning particles and particle generation require four and two random number generators (RNGs), respectively. These RNGs produce Gaussian random numbers with properties as per the requirement of the model. The RNGs are based on a look up table approach detailed in [10]. Inputs of the drawing conditioning particles steps are the 16-bit mean vector and covariance matrix from the CU and 16-bit inputs from the RNGs. The particle generation step comprises of taking the conditioning particles which contain four 16-bit buses and producing final particles (four 16-bit buses) which are the inputs for weight generation and updating of the mean and the covariance matrix. The weight calculation takes also input observations (16-bit bus) at its input and computes the 16-bit weights. In this step, the calculation of the exponential and arctangent functions is attained by using Coordinate Rotation Digital Computer (CORDIC) algorithms [27]. The output and the internal operations of the updating of

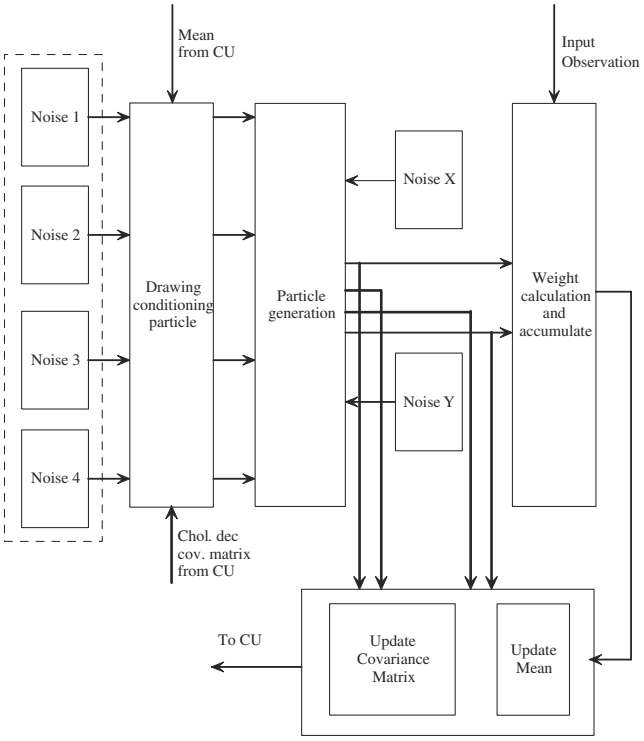


Figure 4 Architecture of a GPF's PE.

the mean and covariance matrix are represented using 40 bits. Since there are 14 40-bit outputs which are generated once in a recursion, we assume a single 40-bit bus that connects the PE and the CU.

The PEs calculate the covariance and mean coefficients in one clock cycle, and so a fully spatial design is used for them. The percentage of required resources for the hardware blocks of PEs is shown in Fig. 5. We used the Xilinx Virtex II pro FPGA platform [29] to estimate the resource and area requirements. The considered resources are the number of occupied slices and the number of used block multipliers. The only reason for domination of the step in which the covariance matrix and mean are estimated, is the large number of bits used for fixed point representation. To be able to calculate four mean and 10 covariance coefficients per clock cycle, 14 multipliers are required. With a 40-bit representation, each multiplier occupies 9 multiplier blocks each with 18×18 bits on the Virtex II Pro chip. However, for lower dimensional models, the ratio of resources in hardware blocks will look different. For example, for two dimensional tracking models only three covariance coefficients and two mean coefficients are necessary for estimation, while the importance step is almost the same. In this case, the area of the importance step would dominate.

The operations that take place in the CU are sequential due to data dependency and have computationally intensive tasks such as square rooting, division, multiplication and addition. Thus, the CU is a good candidate for time-multiplexed implementation where hardware resources are shared in time by various operations. This implementation minimizes hardware without degrading execution throughput. We estimate that the overall sampling frequency of the GPF is reduced about 4% for time-multiplexed implementation in comparison to spatial implementation. However, area saving in the CU is about 90%.

With a very tight performance criterion, the necessary precision of the GPF's CU is more than 50 bits. This is because the coefficients of the covariance matrix are very small and their truncation may entail violation of the positive definiteness of the covariance matrix. If there is a violation, the matrix cannot be decomposed and hence the recursion cannot proceed. So, when the CU is realized with the floating-point library [7] of Xilinx II Pro for estimating the area and latency, the clock frequency of the slowest floating point block (divider) is twice less than the speed of the slowest

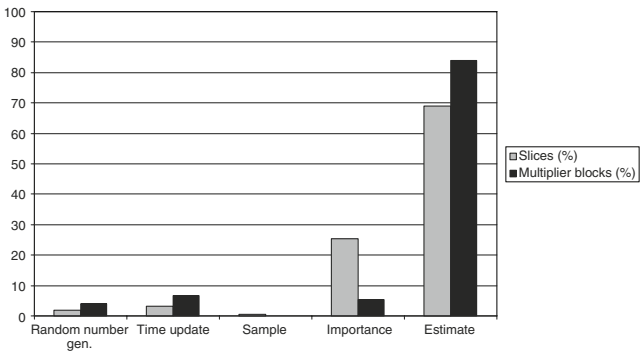


Figure 5 Percentage of number of slices and multipliers of each block in PEs estimated for Xilinx Virtex II Pro chips.

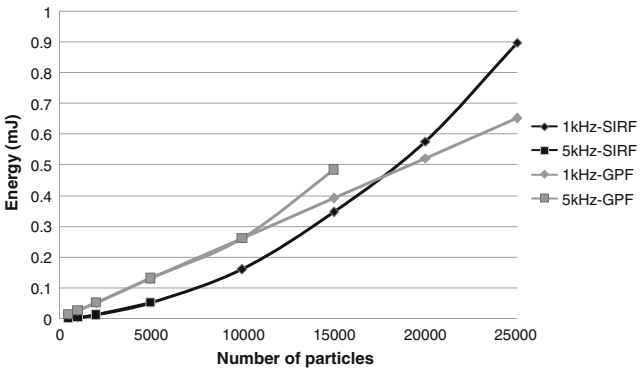


Figure 6 Energy versus number of particles for SIRFs and GPFs with a single PE for different maximum sampling rates.

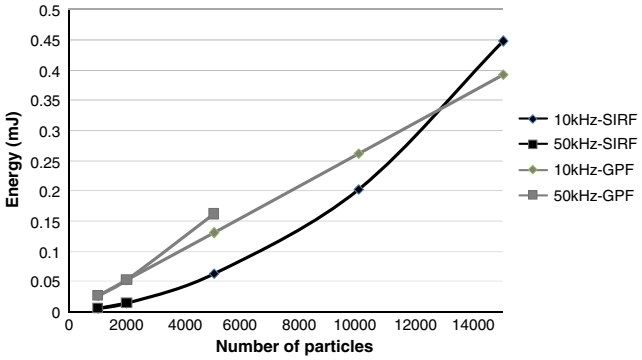


Figure 7 Energy versus the number of particles for SIRFs and GPFs implemented with four PEs for different maximum sampling rates.

synthesized fixed-point block using 50 bits precision. For a GPF implemented by a single PE with $M = 5000$ particles, the sampling frequency is decreased 1.0167 times, the latency is increased 1.3 times, and the logic area is increased 1.02 times in comparison with the area of the PF with a time multiplexed CU that uses 50-bits precision fixed-point arithmetics. Thus, the floating point implementation is an alternate solution when the maintenance of precision is the key issue. Then the throughput and area are not affected significantly.

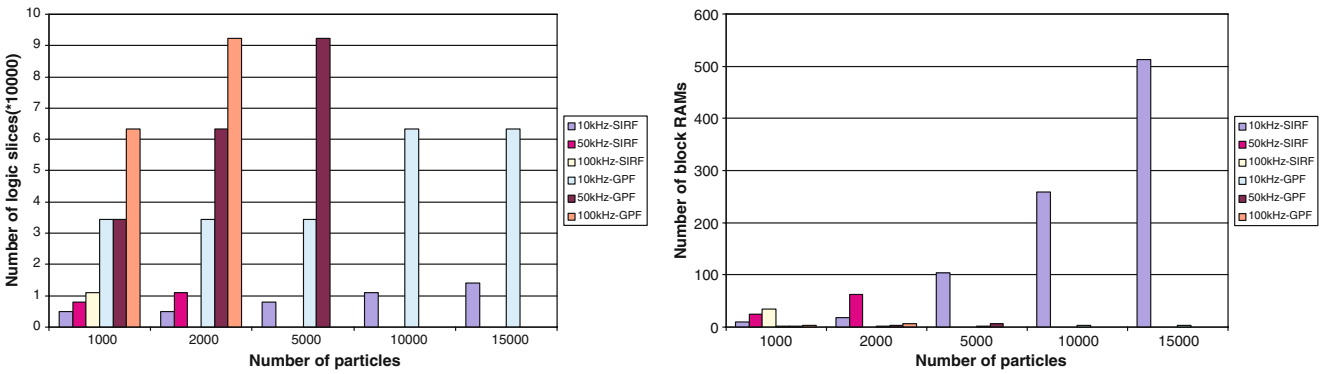
6 Comparisons and Tradeoffs Between SIRFs and GPFs

6.1 Energy with Speed Constraints

The modeling of high-level energy is performed on a module level by estimating the power functions of the elements associated with each module such as adders, multipliers, registers or memories. The power for each module is initially estimated using the Xilinx Spreadsheet Power Tools and verified using Xilinx XPower.

The energy of a single PE implementation of SIRFs and GPFs calculated for a PF sampling period is shown in Fig. 6. The filters are applied to the bearings-only tracking problem. The energy is estimated for various number of particles ($M = \{500, 1000, 2000, 5000, 10000, 15000, 20000, 25000\}$) and for various maximum sampling frequencies ($f_s = \{1, 5\}$kHz). The minimum depth of pipelining of the functional blocks that satisfies a given sampling frequencies is calculated and applied in order to reduce energy. There were 16 bits for all SIRF variables and steps 1–3 of Pesudocode 2 for the GPF. Step 4 of Pseudocode 2 and all the steps in Pseudocode 3 are represented in fixed point with 40 bits. The operating clock frequency is defined by the speed of the CORDIC which is the slowest individual unit in the data path. For our analysis, the clock frequency is set to 100 MHz. From Fig. 6, it is clear that for smaller number of particles and lower frequencies, SIRFs dissipate less energy than GPFs. It is important to note that with the increase of number of particles, the energy of SIRFs becomes comparable and even higher than the energy of GPFs (for more than 17,000 particles). There are two reasons for faster increase of the energy in SIRFs. Since SIRFs are memory dominant, with the increase of number of particles, the size of memory increase results in higher energy. Secondly, the number of operations of the CU of SIRFs is a function of M, while for GPFs the number of operations of the CU is a constant, so that the energy of the CU of SIRFs increases linearly and the energy of GPFs stays constant.

SIRF implementations with a single PE cannot achieve higher requirements such as processing of



(a) Area of SIRFs and GPFs

(b) Memory of SIRFs and GPFs

Figure 8 **a** Area in the number of slices and **b** number of block RAMs versus number of particles. The area is evaluated for different sampling frequencies and for necessary number of PEs so that the PFs satisfy sampling frequency requirements.

10,000 particles at 5 kHz. The energy for the parallel implementation of SIRFs and GPFs which utilize multiple PEs is presented in Fig. 7. The energy of SIRFs is calculated for the worst case data exchange among PEs and the CU which corresponds to transferring $[M - M/K]$ particles. The data exchange is modeled using a shared memory. Again, we can see that the energy of SIRFs is lower than the energy of GPFs when the number of particles is low. However, for 15,000 particles and 1kHz sampling speed, the energy of SIRFs is higher than the energy of GPFs.

6.2 Area Requirement in FPGA

We also evaluated and compared the area requirements of SIRFs and GPFs. Resource requirements are represented using the number of Virtex II Pro logic slices and block RAMs (Fig. 8). Here, the area is evaluated for various number of particles and various sampling frequencies. The number of PEs is adaptively changed so that the PFs meet the sampling frequency requirements. For example, for SIRFs with $M = 5000$ and $f_s = 1$ kHz we choose an implementation with 2 PEs because they are necessary to satisfy the requirements. The GPF algorithm does not contain memory for storing particles. However, it uses one block RAM per random number generator for implementing the Box-Muller method. From Fig. 8a and b, it is clear that SIRFs are memory dominated and GPFs are logic dominated.

7 Conclusion

In this paper, we proposed a physically realizable GPF algorithm and analyzed its design complexity. The proposed GPF allows for an implementation without storing particles in memories. We considered designs with FPGAs and provided results that facilitate the selection of PFs for various operating conditions and filter parameters. We indicated that SIRFs are memory limited whereas GPFs are logic limited. SIRFs are more energy efficient for low capacity single PE particle filtering, and GPFs are superior in both energy and in executing a large number of particles with multiple PEs at high sampling frequencies.

References

1. Bar-Shalom, Y., Rong Li, X., & Kirubarajan, T. (2001). *Estimation with applications to tracking and navigation: Theory, algorithms and software*. New York: Wiley.
2. Bolić, M., Djurić, P. M., & Hong, S. (2004). Resampling algorithms for particle filters: A computational complexity perspective. *EURASIP Journal of Applied Signal Processing, 15*, 2267–2278.
3. Bolić, M., Djurić, P. M., & Hong, S. (2005). Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing, 53*(7), 2442–2450.
4. Bolić, M., Athalye, A., Djurić, P. M., & Hong, S. (2004). Algorithmic modification of particle filters for hardware implementation. In *Proceedings of the European signal processing conference* (pp. 1641–1646), Vienna, Austria.
5. Clark, D., Vo, B.-T., & Vo, B.-N. (2007). Gaussian particle implementations of probability hypothesis density filters. In *The proceedings of the IEEE aerospace conference*.
6. Daum, F., & Huang, J. (2002). *Curse of dimensionality and particle filters*. In *Fifth ONR/GTRI workshop on target tracking and sensor fusion*. Newport, RI, June 2002.
7. Digital Core Design Inc. (2009). *Pipelined floating point libraries*. www.dcd.pl.
8. Doucet, A., de Freitas, N., & Gordon, N. (Eds.) (2001). *Sequential Monte Carlo methods in practice*. New York: Springer.
9. Doucet, A., Godsill, S. J., & Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing, 10*, 197–208.
10. Danger, J. L., Ghazel, A., Boutillon, E., & Laamari, H. (2000). Efficient FPGA implementation of Gaussian noise generator for communication channel emulation. In *Proceedings of IEEE ICECS conference* (pp. 366–369). Laslik, Lebanon.
11. Ghirmai, T. (2007). Gaussian particle filtering for tracking maneuvering targets. In *The proceedings of the IEEE SoutheastCon* (pp. 439–443).
12. Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). A novel approach to nonlinear and non-Gaussian Bayesian state estimation. *IEE Proceedings F, 140*, 107–113.
13. Hao, Y., Xiong, Z., & Hu, Z. (2006). Particle filter for INS in-motion alignment. In *The proceedings of the 1ST IEEE conference on industrial electronics and applications*.
14. Hennessy, J. L., & Patterson, D. A. (2006). *Computer architecture: A quantitative approach* (3rd ed.). San Mateo: Morgan Kauffmann Publishers.
15. Hong, S., Chin, S.-S., Djurić, P. M., & Bolić, M. (2006). Design and implementation of flexible resampling mechanism for high-speed parallel particle filters. *The Journal of VLSI Signal Processing, 44*, 47–62.
16. Hong, S., Djurić, P. M., & Bolić, M. (2005). Simplifying physical realization of Gaussian particle filters with block level pipeline control. *EURASIP Journal of Applied Signal Processing, 4*, 575–587.
17. Howland, P. E. (1999). Target tracking using television-based bistatic radar. *IEE Procedings, Radar, Sonar and Navigation, 146*(3), 166–174.
18. Julier, S. J., & Durrant-Whyte, H. F. (1995) Navigation and parameter estimation of high speed road vehicles. In *Robotics and automation conference, Japan* (pp. 101–105).
19. Kotecha, J. H., & Djurić, P. M. (2003). Gaussian particle filtering. *IEEE Transactions on Signal Processing, 51*(10), 2592–2601.
20. Kotecha, J. H., & Djurić, P. M. (2003). Gaussian sum particle filtering. *IEEE Transactions on Signal Processing, 51*(10), 2602–2612.
21. Kumar, M.. (1988) Measuring parallelism in computation-intensive scientific/engineering applications. *IEEE Transactions on Computers, 37*(9), 1088–1098.

22. van Lawick van Pabst, J., & Krekel, P. F. (1993). Multisensor data fusion of points, line segments and surface segments in 3d space. In P. S. Schenker (Ed.), *Sensor Fusion VI, SPIE Proceedings* (Vol. 2059). Pasadena: Jet Propulsion.
23. Ristić, B., Arulampalam, S., & Gordon, N. (2004). *Beyond the Kalman filter: Particle filters for tracking applications*. Cormano: Artech House.
24. Shiva, S. G. (1996). *Pipelined and parallel computer architectures*. London: Harper Collins College.
25. Uhlmann, J. K. (1994). *Simultaneous map building and localization for real-time applications*. Technical report, University of Oxford.
26. Vemula, M., Bugallo, M. F., & Djurić, P. M. (2007). Performance comparison of Gaussian-based filters using information measures. *IEEE Signal Processing Letters, 14*(12), 1020–1023.
27. Volder, J. (1959). The CORDIC trigonometric computing technique. *IRE Trans. Electronic Computing, EC-8*, 330–334.
28. Wu, Y., Hu, X., Hu, D., & Wu, M. (2005). Comments on Gaussian particle filtering. *Transacations on Signal Processing, 53*(8), 3350–3351.
29. Xilinx Inc. (2003). Virtex-II Pro Patforms FPGA: Functional description. www.xilinx.com.
30. Zhang, Y., & Dai, H. (2007). Dynamic self-calibration in collaborative wireless networks using belief propagation with Gaussian particle Filtering. In *Proceedings of the 41st annual conference on information sciences and systems, CISS* (pp. 771–776).

Miodrag Bolić is an associate professor at School of Information Technology and Engineering at the University of Ottawa. He received his B.Sc. and M.Sc. degrees in electrical engineering from the University of Belgrade, Serbia in 1996 and 2001 and his Ph.D. degree in electrical engineering from Stony Brook University, U.S. in 2004. He has over 7 years of industrial experience related to embedded system design, instrumentation and signal processing. His research interests include computer architectures, hardware accelerators, signal processing for biomedical applications and RFID. He is a director of Radio Frequency Identification Systems Laboratory and Computer Architecture Research Group at the University of Ottawa.

Akshay Athalye received his Ph.D. in Electrical Engineering from the State University of New York, Stony Brook in 2007. His research interests lie in the development of dedicated hardware for signal processing applications. His work encompasses algorithm design, architecture development, and use of reconfigurable SoC design for various real-time signal processing applications. His secondary research interests lie in design and implementation of algorithms for efficient signal processing in RFID systems. He is currently a Research Associate at the Center of Excellence in Wireless in Information Technology (CEWIT) at Stony Brook, NY. He is also a co-founder of Astraion LLC, a Stony Brook, NY based start-up company working on the development of innovative RFID systems for a wide range of applications. He has served as an External Reviewer for various journals and conferences affiliated to the IEEE and EURASIP.

Sangjin Hong received the B.S and M.S degrees in EECS from the University of California, Berkeley. He received his Ph.D in EECS from the University of Michigan, Ann Arbor. He is currently with the department of Electrical and Computer Engineering at Stony Brook University - SUNY. Before joining SUNY, he has worked at Ford Aerospace Corp. Computer Systems Division as a systems engineer. He also worked at Samsung Electronics in Korea as a technical consultant. His current research interests are in the areas of low power VLSI design of multimedia wireless communications and digital signal processing systems, reconfigurable Systems on Chip design and optimization, VLSI signal processing, and low-complexity digital circuits. Prof. Hong served on numerous Technical Program Committees for IEEE conferences. Prof. Hong is a Senior Member of IEEE. He is also a member of Eta Kappa Nu and Tau Beta Pi Honor societies.

Petar M. Djurić received his B.S. and M.S. degrees in electrical engineering from the University of Belgrade, in 1981 and 1986, respectively, and his Ph.D. degree in electrical engineering from the University of Rhode Island, in 1990. From 1981 to 1986 he was a Research Associate with the Institute of Nuclear Sciences, Vinča, Belgrade. Since 1990 he has been with Stony Brook University, where he is Professor in the Department of Electrical and Computer Engineering. He works in the area of statistical signal processing and his primary interests are in the theory of modeling, detection, estimation, and time series analysis and its application to a wide variety of disciplines including wireless communications and biomedicine. Prof. Djurić has been elected Distinguished Lecturer of the IEEE Signal Processing Society. He has also been on the Editorial Boards of several journals. In 2007, he received a paper award for a paper published in the IEEE Signal Processing Magazine. Prof. Djurić is a Fellow of IEEE.