Resampling Algorithms and Architectures for Distributed Particle Filters

Miodrag Bolić*, Petar M. Djurić, and Sangjin Hong

Department of Electrical and Computer Engineering

Stony Brook University

Stony Brook, NY 11794-2350

mbolic, djuric, snjhong@ece.sunysb.edu

EDICS: 2-IMPL Algorithm Implementation in Hardware and Software, 5-PARA Parallel Processing Architectures, 5-PARI Algorithms for Parallel Implementation

This work has been supported under Awards CCR-9903120 and CCR-0220011.

Abstract

In this paper, we propose novel resampling algorithms with architectures for efficient distributed implementation of particle filters. The proposed algorithms improve the scalability of the filter architectures affected by the resampling process. Problems in the particle filter implementation due to resampling are described and appropriate modifications of the resampling algorithms are proposed so that distributed implementations are developed and studied. Distributed resampling algorithms with proportional allocation (RPA) and non-proportional allocation (RNA) of particles are considered. The components of the filter architectures are the processing elements (PEs), a central unit (CU) and an interconnection network. One of the main advantages of the new resampling algorithms is that communication through the interconnection network is reduced and made deterministic, which results in simpler network structure and increased sampling frequency. Particle filter performances are estimated for the bearings-only tracking applications. In the architectural part of the analysis, the area and speed of the particle filter implementation are estimated for different number of particles and different level of parallelism with FPGA implementation. In this paper only sampling importance resampling (SIR) particle filters are considered, but the analysis can be extended to any particle filters with resampling.

I. Introduction

Particle filters (PF) are very suitable for non-linear and/or non-Gaussian applications. They show great promise in addressing a wide variety of complex problems [6], [18]. However, their application in real-time systems is limited due to their inherent computational complexity. The main goal of this paper is to develop distributed PF algorithms and to propose corresponding parallel architectures which allow for shorter particle-filter execution time. We show that the parallel architectures can be implemented on state-of-the-art FPGA chips. By showing that fast implementation of PFs is feasible, we hope that the gap that exists between PF theory and their hardware implementation will be reduced.

The SIR algorithm [7] is composed of three steps:

1. sampling step – generation of new particles, in which M particles $x^{(m)}$ for $m = 1, ..., M$ are drawn from an importance function $\pi(x)$,
2. importance step – computation of particle weights $w^{(m)}$ for $m = 1, ..., M$, and
3. resampling step – drawing of M particles $\widetilde{x}^{(m)}$ from the set $x^{(m)}$ for $m = 1, ..., M$ according to the resampling function $a^{(m)}$ whose support is defined by the particles $x^{(m)}$ [17]. Commonly $a^{(m)} = w^{(m)}$ for $m = 1, ..., M$.

The resampling step is critical in every implementation of particle filtering because without it, the variance of the particle weights quickly increases, i.e., very few normalized weights are substantial. Then, the inference is degraded because it is made by using only a very small number of particles. The idea of resampling is to remove the particle trajectories with small weights and replicate the trajectories with large weights. Resampling was proposed for use in particle filtering in various works including [2], [3], [13], [14], [15], [16]. The following problems are recognized and addressed for distributed implementation of resampling: (a) there is no natural concurrency among iterations because the new iterations depend on the previous ones, (b) communication among the PEs after resampling is extensive, and (c) connections among the PEs are not known before the run-time and are changed after each sampling period. Modifications of the resampling algorithms that intend to overcome these barriers and move towards a fully distributed implementation are developed and studied.

The main design goal here is to minimize the execution time of the PF. This is done through exploiting data parallelism and pipelining of operations. In Section II, a parallel architecture for PFs is introduced and the minimum execution time is defined. In order to decrease PF execution time, an algorithm that allows for distributed resampling and reduced communication in the network is proposed. This algorithm is presented in Section III and is named distributed Resampling with Proportional Allocation (RPA). It yields the same resampling result as the sequential resampling method (for example systematic resampling). Further improvement of the execution time is achieved through making the communication through the network deterministic and local. These algorithms use non-proportional sampling (Resampling with Non-proportional Allocation - RNA), and they are presented in Section IV. Different architectures suitable for distributed RPA and RNA algorithms are discussed in Section V. The objective in these architectures is to pipeline the communication through interconnection network (particle routing) with the subsequent sampling step. There, we also evaluate architecture parameters on an FPGA platform.

II. Distributed PFs

A. Distributed architecture

The distributed architecture for the PFs is shown in Figure 1. It consists of processing elements (PEs) and a central unit (CU). Since there are no data dependencies during particle generation and calculation of the weights, these steps can be easily parallelized and pipelined. This segment of particle filtering is a data parallel single instruction multiple data (SIMD) algorithm [4]. As such, particle generation and weight calculation for the M particles can be partitioned in K PEs, where $1 \leq K \leq M$. Each PE performs the same operations in time on different particles and each PE is responsible for processing $N = M/K$ particles where both K and N are integers. The CU carries out partial or full resampling and particle routing as well as overall control. Full resampling means that the overall resampling procedure is performed by one logic unit. In the following sections, we will show that resampling can be distributed to PEs and that the CU is then responsible only for a small portion of resampling.

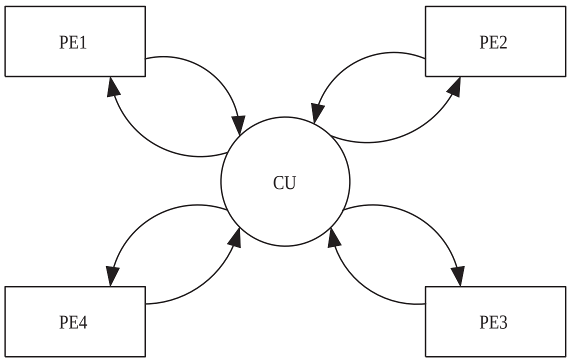

Fig. 1. Architecture of the distributed PF with a CU and four PEs.

We distinguish three operations that carry out the resampling task:

1. Computation – involves the bare resampling procedure whose result is an array of indexes which show the replicated particles and their addresses.
2. Communication – represents exchanging of particles among the PEs based on the resampling results. We refer to it as particle routing. Particle routing defines the protocol and the network architecture for exchanging particles and it is the main focus of the paper.
3. Scheduling – includes (a) determination of which particles in the PEs are routed and which are stored locally, (b) placing of particles in the destination PEs, and (c) addressing used for indexes.

In this paper we define the execution time of PFs as the time necessary to process one observation by the PF, and it corresponds to the sampling period.

In order to achieve minimum execution time, one-to-one mapping between the PF operations and hardware resources is done which allows for utilizing operational concurrency. Hence, several operations can be executed at the same time and their blocks are pipelined in hardware implementation. The execution time of the generation and weight computation of every particle is LT_{clk}, where T_{clk} is the clock period and L in the latency due to pipelining. Thus, the first particle is available at the output of the importance step block after LT_{clk}, and every next particle, due to pipelining, after T_{clk}. Resampling cannot start until the sum of all the weights is calculated so that resampling cannot be overlapped in time with the sampling and importance steps. The internal operations of resampling are also pipelined so that they take M clock cycles as well. Hence, the minimum execution time of non-distributed PF is $(2M + L)T_{clk}$ [1].

Further reduction of execution time is achieved by replicating hardware resources (parallelism). When K PEs are used, the minimum execution time is $(2M/K + L)T_{clk}$. The main goal of this paper is to develop algorithms and architectures that can reach the minimum execution time. Our strategy towards achieving the minimum execution time is to allow for deterministic communication during particle routing. Then, we can overlap the particle routing and the next sampling step to allow for pipelining in hardware of their operations, so that the particle routing will not increase the execution time of the PF.

Next, we show why the communication pattern is non-deterministic and the connections among the PEs are changed after each sampling period. Let the number of particles that PE_k produces after resampling be $N^{(k)}$ for $k = 1, ..., K$, $0 \leq N^{(k)} \leq M$ and $\Sigma_{k=1}^{K} N^{(k)} = M$. It is important to note that $N^{(k)}$ is a random number which depends on the overall distribution of the weights. The PEs with $N^{(k)} > N$ have surplus of particles and they need to exchange particles with the PEs with shortage of particles for which $N^{(k)} < N$. The number $N^{(k)}$ changes after each sampling period so that it is necessary to connect different PEs in order to perform particle routing. The number of particles that have to be exchanged among the PEs is $N_M = \Sigma_{k:N^{(k)}>N}^{K}(N^{(k)} - N) = \Sigma_{k:N^{(k)}<N}^{K}(N - N^{(k)})$.

B. Centralized resampling

In centralized resampling, particle generation and weight calculation are performed in parallel in PEs and resampling is sequential and it is carried out by the CU. The sequence of operations and directions of communication are shown in Figure 2(a). The CU collects the N weights from each PE (M weights overall) in order to perform resampling and returns N replication factors to each PE (M replication factors overall).

The number of particles transferred between PE_k and the CU is $|N^{(k)} - N|$ for $k = 1, ..., K$. The direction of communication is from the PE to the CU for the PE with particle surplus after resampling ($N^{(k)} - N > 0$) and from the CU to the PE for the PE with particle shortage ($N^{(k)} - N < 0$). While the communication of weights and indexes is deterministic, the particles are routed in a non-deterministic fashion. The overall amount of particles that has to be transferred through the network is $M/2$ for the worst case. Even in the fully connected network, the scalability of the implementation is significantly affected by the sequential resampling and particle routing. One version of centralized resampling which is implemented on a network of personal computers is described in [21].

III. Distributed RPA

In this section, a method based on stratified sampling with proportional allocation is described. The sample space is divided into K disjoint areas or strata, where each stratum corresponds to a PE. The density of particle weights can then be written as a mixture of K densities restricted to the corresponding strata. Proportional allocation among strata is used, which means that more samples are drawn from the strata with larger weights. After the weights of the strata are known, the number of particles that each stratum replicates is calculated using residual systematic

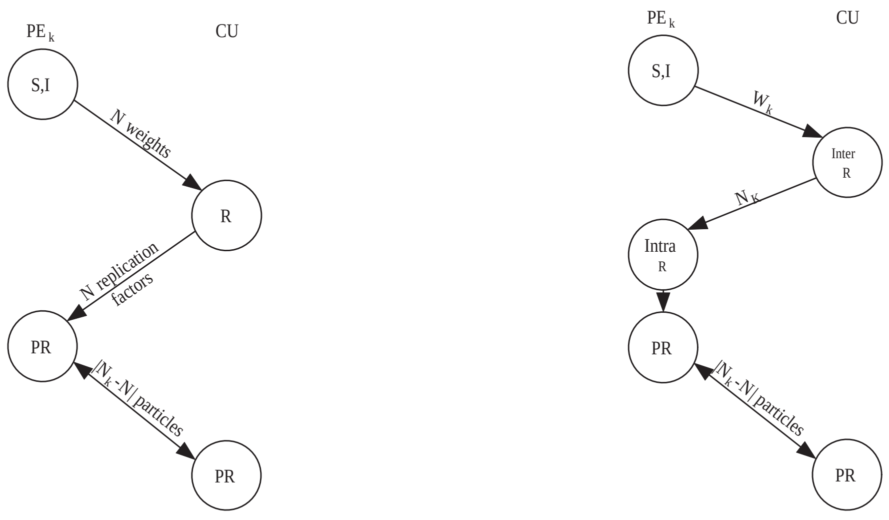

Fig. 2. Sequence of operations performed by the k-th PE and the CU for (a) centralized resampling and (b) RPA. The direction of communication as well as data that are sent are presented. The abbreviations are: S-sampling, I-importance computation, R-resampling, PR-particle routing.

resampling (RSR) described in [1], and this process is denoted as inter-resampling since it treats the PEs as single particles. Finally, resampling is performed inside the strata which is referred to as intra-resampling. So, the resampling algorithm is accelerated by using loop transformation or specifically loop distribution [22], which allows for having an inner loop that can run in parallel on the PEs (intra-resampling) with small sequential centralized pre-processing (inter-resampling). The weight of the PE is calculated as a sum of the weights of the particles inside the PE, i.e. $W^{(k)} = \sum_{i=1}^{N} w^{(i,k)}$ for $k = 1, ..., K$. A diagram and the sequence of operations performed by the PE and the CU are shown in Figure 2(b).

The algorithm for RPA is shown by Pseudocode 1. The inputs of the algorithm are the PE weights and the output is the number of particles $N^{(k)}$ that each PE will produce after resampling, where $E(N^{(k)}) = MW^{(k)}$ for $k = 1, ..., K$. The RSR algorithm is applied to get $N^{(k)}$, for $k = 1, 2, ..., K$

by propagating the uniform random number in a similar fashion as in the systematic resampling algorithm. In the algorithm, $N^{(k)}$ is obtained by truncating $(W^{(k)} - U_k) \cdot M$. The minimum value of the truncated product is -1 so that the minimum value of $N^{(k)}$ is zero. Resampling is performed in each PE in parallel during the intra-resampling step. The input of the intra-resampling algorithm is the number of particles that should be generated in the resampling procedure. We have to stress that there is no difference in results between RPA and sequential resampling.

```
DISTRIBUTED RPA ALGORITHM
Purpose:  Calculation of the number of particles $N^{(k)}$ for the intra-resampling algorithm.
Input:    Array of PE weights $W^{(k)}$ for $k = 1, ..., K$.
Method:
   Generate random number $U_1 \sim U[0, 1/M]$
   for $k = 1$ to $K$
      $N^{(k)} = \lfloor (W^{(k)} - U_k) \cdot M \rfloor + 1$
      Send $N^{(k)}$ to $PE_k$
      $U_{k+1} = U_k + \frac{N^{(k)}}{M} - W^{(k)}$
   end
   do in parallel
      Intra-resampling for all PEs
   end
```

Pseudocode 1. A distributed RPA algorithm that utilizes the RSR approach.

The RSR algorithm is very attractive for hardware implementation since it has only one loop (there are two loops in systematic resampling), it can be easily pipelined so that it can calculate a replication factor per clock cycle, and it easily deals with different number of particles at the input and at the output. In systematic resampling *while* loop has unknown number of iterations which makes it difficult to apply pipelining. Of course, the same resampling result would be obtained if residual or systematic resampling are applied as the inter-resampling algorithms.

An example of particle exchange for the RPA algorithm is shown in Figure 3. The PF architecture with four PEs is considered, where each PE processes $N = 100$ particles. The distribution of the normalized PE weights before resampling is presented in the table. After inter-resampling, the number of particles that each PE will produce is determined and it is $200, 50, 105$ and 45 respectively. So, PEs 1 and 3 have surpluses of particles. In this example, PE_1 sends 50 particles to both PE_2 and PE_4, and PE_3 sends 5 particles to PE_4.

Weights of the PEs before resampling

	PE weights before resampling
$W^{(1)}$	0.5
$W^{(2)}$	0.125
$W^{(3)}$	0.2625
$W^{(4)}$	0.1125
Sum	1

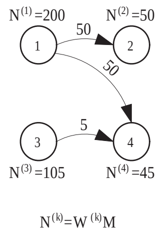

Fig. 3. An example of particle exchange for the RPA algorithm.

The main advantage of distributed RPA over centralized resampling lies in reducing the amount of deterministic communication and in the distributed resampling where the resampling is executed concurently in the PEs instead in the CU (Figure 2). The time for the resampling procedure in distributed RPA is reduced $M/(M/K + K)$ times, where M/K corresponds to the intra-resampling time and K is a time for inter-resampling. It can readily be shown that maximum reduction is achieved when $K = \sqrt{M}$. It is important to note that inter-resampling requires global communication among the PEs, while intra-resampling is completely local within the PEs. The $2M$ words representing weights and indexes that are exchanged in the centralized resampling are reduced to

$2K$ words ($W^{(k)}$ and $N^{(k)}$) in RPA. However, scalability of the implementation is still affected by the particle routing step, which is unchanged. If we assume equal clock period for resampling and the other PFs steps, then $T_{ex} = (2M/K + L + K + M_r)T_{clk}$, where K represents the delay due to inter-resampling and M_r is the delay due to particle routing. When the PEs and the CU are connected with a single bus, then the delay M_r becomes dominant. Scalability of the design is affected so much by the bus structure, that there is almost no gain in pursuing parallel implementation. An efficient architecture that uses K buses and supports pipelining of the particle routing with the sampling step is proposed in Section V-A.

IV. Distributed RNA

Even though distributed RPA allows for distributed and parallel implementation of resampling, it requires a complicated scheme for particle routing which implies a complex CU design and area increase. Besides, there is a need for an additional global pre-processing step (inter-resampling) which introduces an extra delay. These problems can be solved by using an RNA algorithm. The main advantage of RNA is that routing of particles can be deterministic and planned in advance by a designer.

A. RNA algorithm

Here, we introduce the term group where a group is formed from one or more PEs. In RPA, the number of particles drawn is proportional to the weight of the stratum. On the other hand, in RNA the number of particles within a group after resampling is fixed and equal to the number of particles per group, $N_k = N$. So, full independent resampling is performed by each group.

The general PF algorithm with RNA is outlined by Pseudocode 2.

```
1. Exchange particles among groups deterministically.
2. Generate particles in each group in parallel by sampling $\mathbf{x}_t^{(k,i)} \sim \pi(\mathbf{x}_t)$ for $k = 1, \ldots, K$ and $i = 1, \ldots, N$.
3. Perform the importance step in each group in parallel. The weights are calculated by
   $w_t^{*(k,i)} = w_{t-1}^{(k,i)} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(k,i)})p(\mathbf{x}_t^{(k,i)}|\mathbf{x}_{t-1}^{(k,i)})}{\pi(\mathbf{x}_t^{(k,i)})}$ for $k = 1, \ldots, K$ and $i = 1, \ldots, N$.
4. Normalize the weights of the particles with the sum of the weights in the group:
   $w_t^{(k,i)} = \frac{w_t^{*(k,i)}}{W^{(k)}}$ where $W^{*(k)} = \sum_{j=1}^{N} w_t^{*(k,j)}$ and $W^{(k)} = W^{*(k)}/(\sum_{j=1}^{K} W^{*(k)})$ for $k = 1, \ldots, K$.
5. Perform resampling inside the groups and obtain new random measures $\{\widetilde{\mathbf{x}}_{1:t}^{(k,i)}, \widetilde{w_t}^{(k,i)} = W^{(k)}\}$ for
   $k = 1, \ldots, K$ and $i = 1, \ldots, N$.
6. Go to step 1.
```

Pseudocode 2. PF steps for distributed RNA.

There are several differences in comparison with the original SIR filter and the RPA algorithm. Here, normalization is performed with the local sum $W^{(k)}$. Resampling is performed locally per each group and the weights are equal inside the group. A characteristics of RNA is that the weights after resampling are not equal to $1/M$, but they are equal inside the groups $\widetilde{w_t}^{(k,i)} = W^{(k)}$, for $k = 1, 2, \ldots, K$ and $i = 1, 2, \ldots, N$. In addition, routing of particles among the groups after resampling is necessary due to the possibility of having very unequally distributed weights among groups.

We distinguish between three methods of particle exchange after resampling: regrouping, adaptive regrouping and local exchange. These methods are presented in Figure 4 which is based on the same example described in Figure 3. The description of these methods is provided in the sequel.

A.1 Distributed RNA with regrouping

In RNA with regrouping, resampling and particle routing are performed inside the groups using the RPA method. For example, in Figure 4(a) PE_1 and PE_2 form one and PE_3 and PE_4 another

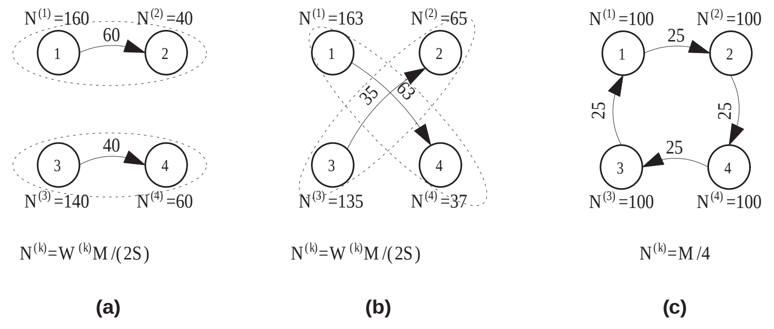

Fig. 4. An example of particle exchange for RNA algorithms with (a) regrouping, (b) adaptive regrouping and (c) with local exchange. Here, S is the sum of weights in the group.

group. The RPA algorithm is applied for both groups. As a result, PE_1 and PE_2 produce 160 and 40 particles after resampling, so that 60 particles from PE_1 are transferred to PE_2. At the next sampling instant, the PEs are rearranged so that they form different groups. For example, the new groups can be composed of PE_1 and PE_3, and PE_2 and PE_4. After each time instant, regrouping is performed so that particles are exchanged among PEs and the variance is reduced.

An example with $K = 9$ PEs and $R = 3$ PEs per group is shown in Figure 5. Only the group that consists of PE_1, PE_4 and PE_7 has particles with non-negligible weights after the importance computation and resampling and these PEs are drawn darker in the figure. At the next time instant, new groups are formed so that the particles with significant weights are propagated to all PEs. One period of regrouping is denoted as distribution factor D. When all the particles with non-zero weights are in one PE and the mesh architecture is used, D determines the number of cycles needed that these particles propagate to all the other PEs. In Figure 5, $D = 2$.

Since the simplicity of the controllers is one of the design goals, we restrict the number of PEs per groups to be 2. If the number of PEs in the group is larger, a very complicated controller is necessary in order to perform fast particle routing as described in Section V-A. When $R = 2$,

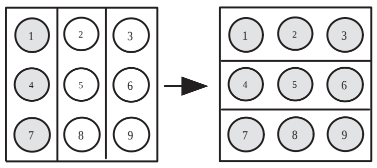

Fig. 5. Routing in RNA with regrouping for the mesh architecture with $K = 9$, $R = 3$ and $D = 2$.

the local controllers are simple because there is only one PE with surplus and one with shortage of particles. Choosing so small value for R could cause high distribution factor and large number of periods until full propagation of particles is achieved. If $R = 2$ and $K = 16$, the minimal distribution factor is $D = 6$.

A.2 RNA with adaptive regrouping

RNA with regrouping uses the predefined fixed rules to form the groups and does not take advantage of knowing the distribution of the group weights. By utilizing this knowledge, it is possible to reduce the variance after resampling. RNA with adaptive regrouping forms groups from the PEs with the largest and the smallest PE weights. For example, in Figure 4(b) PE_1 and PE_3 have the largest and the smallest PE weights so that they form one group. The other group is formed from the remaining PEs. Inside the groups, the RPA algorithm is applied. Weights after resampling are calculated based on step 5 of Pseudocode 2. This method utilizes the Randez-Vouz load balancing algorithm [8], which is a simple greedy algorithm that associates the heavily and the lightly loaded groups. The main disadvantages of RNA with adaptive regrouping are that groups contain only two PEs ($R = 2$) and the connections among the PEs are not local in general.

A.3 Distributed RNA with local exchange

In RNA with regrouping, the RPA algorithm is still performed inside groups so that the particle routing process is still random, even though it is done on a smaller set of particles. Randomness during particle routing makes it difficult for pipelining between the particle routing and sampling steps.

The example of the RNA algorithm with local exchange is shown in Figure 4(c). Resampling is done inside the PE and then particles are exchanged in a deterministic way only among the neighboring PEs. Routing is done through local communication. The amount of particles sent between PEs is fixed and defined in advance. In the example, it is $N/4 = 25$. This is a very important difference in comparison with the RNA with regrouping where particles are routed among the PEs in the group non-deterministically (except when $R = 2$). Since groups are formed from one PE, the weights after resampling are set to $W^{(k)}/N$. Local communication can give rise to a large number of periods until full resampling is achieved, which restricts the level of parallelism.

B. Effects of resampling on obtained estimates

In PFs, the output estimate before resampling can be calculated as: $\overline{g} = \sum_{m=1}^{M} w^{(m)} g(x^{(m)})$, where $x^{(m)}$ are the states of the particles, $g(\cdot)$ is an arbitrary function, and $w^{(m)}$ represents a normalized importance weight [9], [10]. For parallel implementation, the estimate can be written in the form: $\overline{g} = \sum_{k=1}^{K} W^{(k)} \sum_{i=1}^{N} w^{(k,i)} g(x^{(k,i)})/W^{(k)} = \sum_{k=1}^{K} W^{(k)} \overline{g}^{(k)}$, where $\overline{g}^{(k)}$ represents the expected value of $g(x)$ from a distribution $w^{(k,i)}$ in the $k-$th PE. The estimate after applying distributed RPA is of the form: $\widetilde{g} = 1/M \sum_{k=1}^{K} \sum_{i=1}^{N} N^{(k,i)} g(x^{(k,i)})$ where $N^{(k,i)}$ represents the number of times the particle k, i is replicated after resampling and $E(N^{(k,i)}) = w^{(k,i)} M$. The estimate after applying distributed RNA is of the form: $\widehat{g} = 1/N \sum_{k=1}^{K} W^{(k)} \sum_{i=1}^{N} N^{(k,i)} g(x^{(k,i)})$, where the number of replications of each particle is calculated as $E(N^{(k,i)}) = w^{(k,i)}N/W^{(k)}$. It is easy to show that $E(\widetilde{g}) = E(\widehat{g}) = \sum_{k=1}^{K} W^{(k)}\overline{g}^{(k)}$, which is equal to $\overline{g}$. This means that both estimates of $\overline{g}$ are unbiased. The result is expected for both types of sampling due to Theorem 5.1 from Cochran [5], which claims that if in every stratum the sample estimate is unbiased, then the overall estimate too, is an unbiased estimate of the population mean.

PFs with full and without resampling can be considered as special cases of the RNA algorithm. In the first case, $K = 1$ and the whole resampling is performed inside one PE. In the second case, $K = M$ so that resampling is performed on a single particle. Since the input and output of the resampling is only one particle, there is actually no resampling.

It is not easy to compare $Var(\widetilde{g})$ and $Var(\widehat{g})$ in general. It was observed by simulations that there was almost no difference in the variances if the weights are equally distributed among the PEs. However, the variance of the RNA algorithm was much greater in the case when there was only one PE with non-zero weights. This problem can be resolved by exchanging the particles between PEs after resampling deterministically (step 1 of the RNA algorithm).

C. Performance analysis

In this section, the performances of the sequential PF and the PF with distributed RNA with local exchange with different number of PEs are compared. The architectural model that was chosen for the PF with distributed RNA was the 2-cube torus type network [19]. We considered 2-ary, 4-ary and 8-ary torus networks. In the model it was assumed that each PE had a single input and output port. The deterministic particle routing was implemented in a way that each PE exchanged particles with the PE above and on the PE left. In this way, particles were routed with a statically scheduled communication pattern. The number of particles that was exchanged is the

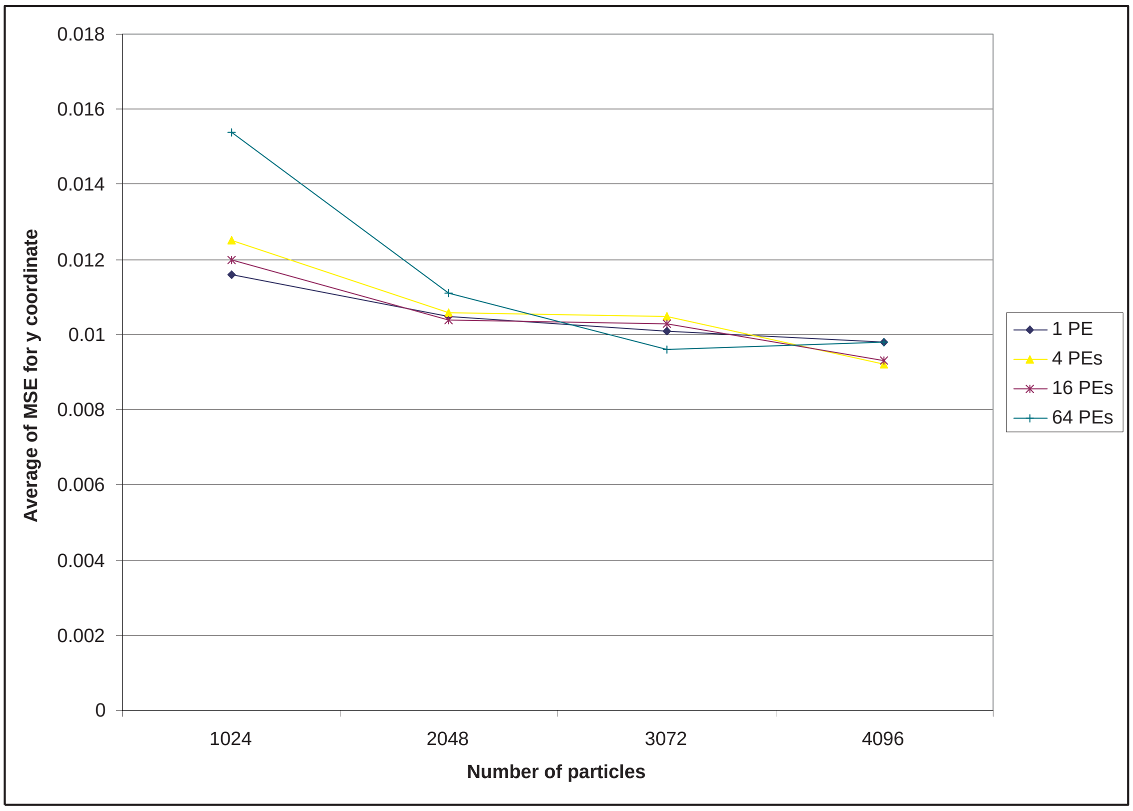

Fig. 6. MSE versus the number of particles for different levels of parallelism. In the case of 4, 16 and 64 PEs, RNA with local exchange is applied.

half of the number of particles in PEs $N/2$. Particles were exchanged in full duplex mode which means that $N/2$ resampled particles from one PE were sent to another and at the same time $N/2$ of resampled particles from the second PE were sent to the first one.

PFs were applied to the bearings only tracking problem with the model from [11]. As performance metrics we chose the mean square error (MSE). The simulation results are shown in Figure 6. We can see that all the MSEs are comparable.

V. PF architectures with distributed resampling

A. Distributed RPA architectures

One possible architecture for distributed RPA with four PEs that allows for pipelining the particle routing step with the next sampling step is shown in Figure 7. The main idea is to store the particles that will be routed among the PEs into dedicated memories in the CU and to have very fast interface capable of reading particles from the CU and routing them to the PEs in one clock cycle.

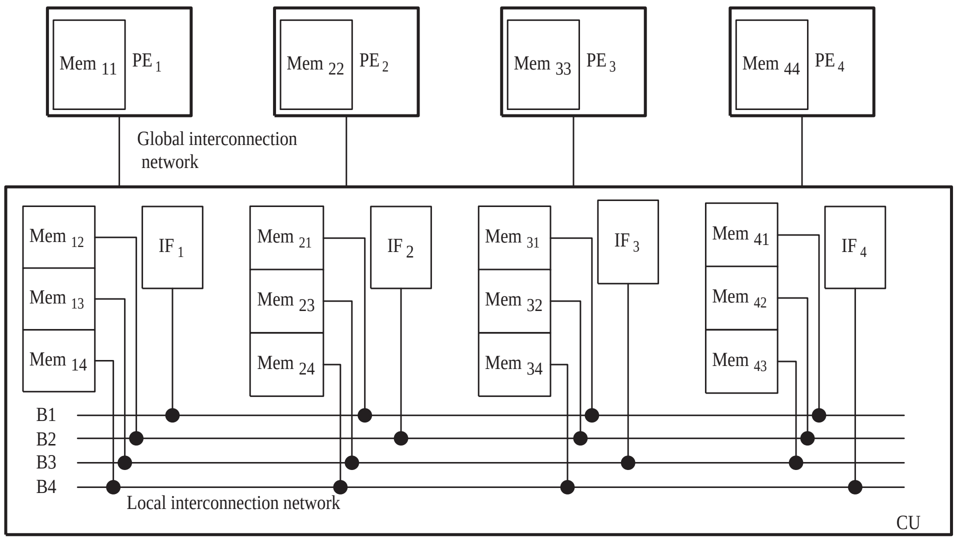

Fig. 7. Architecture of the PF with distributed RPA with four PEs. The CU is implemented to support pipelining between the particle routing and sampling steps.

The particles that are replicated as a result of the resampling for PE_k are stored into local memories Mem_{kk} for $N^{(k)} < N$. When there is a surplus of particles, these particles are stored in CU memories Mem_{ki} for $i = 1, ...K$ and $k \neq i$. For example, the memory Mem_{12} is used to store the surplus of particles from PE_1 that should be routed to PE_2. If there is a shortage of particles in PE_k, then PE_k reads particles from the interface IF_k which is connected to the memories Mem_{ik}.

Routing is performed through three steps. First, particles from the PEs with the particle surplus are sent to the CU through the global interconnection network. Then, routing is performed through the IF block inside the CU using the buses B_i for $i = 1, ..., 4$. Each IF is connected to the corresponding memories with a bus and it acts as a master on the bus. Finally, particles are transferred to the destination PEs through the global interconnection network. The size of the memories is determined for the worst case (when one PE acquires all the N particles from another PE) and it is N words, where each word consists of the particles and their replication factors. So, the overall memory requirements are $16N = 4M$ words which is 4 times more than in the sequential case.

The timing diagram for the PE with particle shortage together with its communication with CU is presented in Figure 8. Resampling is performed using the following steps:

1. CU performs inter-resampling and sends the output number of particles $N^{(k)}$ to PE_k for $k = 1, ..., K$. The CU also calculates the amount of data that should be transferred among the PEs.
2. The PEs perform intra-resampling so that the first $N^{(k)} \leq N$ particles are stored into the local memory Mem_{kk} and when $N^{(k)} > N$, the surplus is sent to the CU.
3. The particles are allocated to the corresponding memories Mem_{ki}. The PEs have no information how the particles are further routed in the CU.
4. During the sampling step, the PE reads the particles first from the local memories. The PEs with the shortage of particles, acquire the rest of particles from the IF as shown in Figure 8.

This architecture has an execution time very close to the minimum execution time at the expense of increased resources. There are four parallel buses from the PEs to the CU and four parallel buses inside the CU. The area is also increased because particles are additionally stored inside the CU. The clock speed is limited by the memory access and by the complexity of the CU. The design methodology and implementation results for the distributed RPA in ASIC are given in [12].

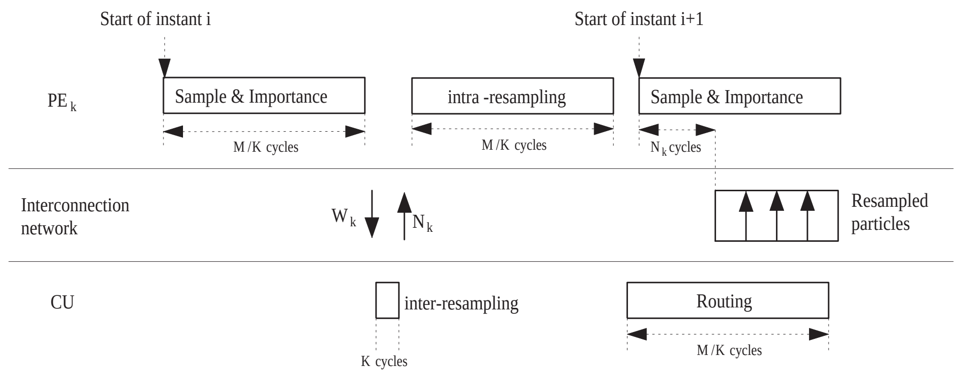

Fig. 8. Timing diagrams for the PF with distributed RPA. Communication through the interconnection network is shown for the PE_k with shortage of particles.

B. Distributed RNA architectures

In Figure 9(a), a PF architecture that can be used for all RNA algorithms with four PEs is presented. Since the connections are not local, it is especially suitable for RNA with adaptive regrouping. Two lines in the figure represent buses used for particle routing. The algorithm running on the CU configures switches so that only two PEs access one bus at any given time. In the case of RNA with fixed regrouping, the switches are configured in fixed order. For example, if $D = 2$, the switches can be configured so that the following sequence is repeated: 12 and 34, 13 and 24. In RNA with adaptive regrouping, the switches are configured so that they connect the PEs with largest and smallest weights. The RNA with local exchange can also be run on the same architecture. We must stress here that the buses consist of a significant number of lines. For example, for the aforementioned bearings-only tracking problem, there are at least four 24-bit lines for transferring particles.

A simpler architecture is shown in Figure 9 (b). The network topology that is chosen is a 2×2 mesh. The network is static and based only on local interconnections. The CU is simple and its functions are collecting partial sums of weights and outputs, returning the final sum of weights to the PEs and the overall control. The CU is connected to the PEs through a single bus. However, the RNA with adaptive regrouping cannot be applied because not all the PEs are physically connected.

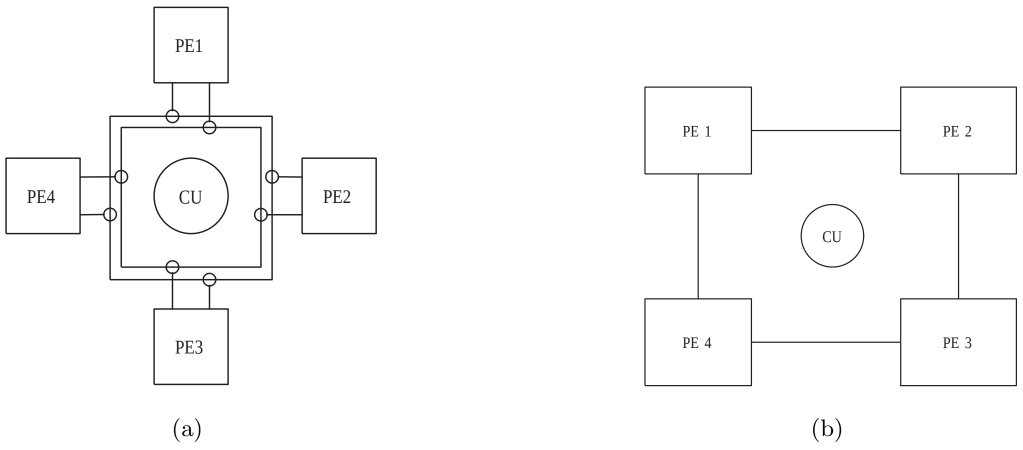

Fig. 9. Architectures for PFs with $K = 4$ PEs that support (a) all RNA algorithms and (b) does not support RNA with adaptive regrouping. The number of lines for each bus is four 24-bit lines for the four-dimensional bearings-only tracking problem.

The architectures become more complex for a higher level of parallelism. A scalable architecture that can support both methods of RNA with regrouping (adaptive and fixed) for $K \leq 4$ and their ASIC implementation is presented in [12].

C. Area and speed of distributed PF with RNA with local exchange

The area and speed of the distributed PF with RNA with local exchange are estimated for the bearings-only tracking problem. The same parameters and model are used as in [11]. The range of interest is restricted to the region $[-32, 32] \times [-32, 32]$. As a benchmark, the chosen hardware platform is Xilinx Virtex-II Pro [23]. The resources are analyzed as a combination of the number of logic slices, multiplier blocks and memory bits.

The finite precision approximation of variables is performed in the SystemC language [20]. The particles are represented using 24 bits with one sign bit, five bits to the left and 18 bits to the right of the decimal point, while the weights are represented with 16 bits with one bit in decimal and 15 bits in the fractional part. The final memory requirements are: four $24 \times M$ memories for storing hidden states, one $16 \times M$ memory for storing weights, and two $16 \times M$ memory for storing replication factors and indexes for resampling. So, the overall storage space is $144 \times M$ bits. The complex mathematical functions are implemented using CORDIC, and the Gaussian random number generator is implemented using the Box-Muller method. The implementation is parallel in order to achieve maximum speed.

In Figure 10 we present the execution times as functions of K. The latency and the clock period that are used are $L = 100$ and $T_{clk} = 10ns$. The area of the graph bounded by the bold line represents the design space area for the Virtex II Pro family. For smaller M, the design space is determined by the logic blocks which increases with the level of parallelism, and for large M by the memory size.

It is interesting to compare the number of memory slices, number of multiplers and the number of bits with the corresponding values from the Virtex II Pro family, which are shown in Table I. In the table, the number of particles is $M = 10000$. The number of slices for components in the dataflow is calculated and is multiplied by the factor of 1.5 in order to take into account the controllers and unused slices. The approximate number of block RAM modules is calculated as $\lceil BM/(KS) \rceil KS$, where B is the number of bits and S is the size of block RAM memory which is 18Kbit. The symbol '*' represents the parameter of the memory, number of slices or multipliers blocks that determines the choice of the Xilinx chip. In the same table, the corresponding Xilinx chip is shown as well. For a lower level of parallelism ($K \leq 4$), the design is memory dominated,

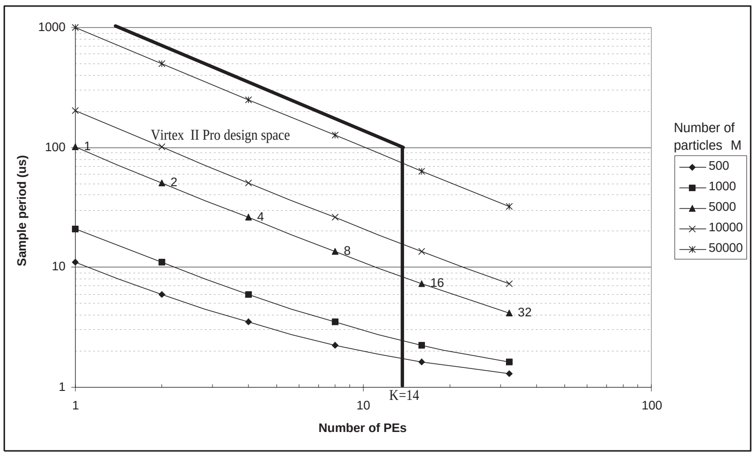

Fig. 10. (a) Execution time as a function of the number of PEs for RNA with local exchange for M =500, 1000, 5000, 10000 and 50000 particles.

while for a higher level of parallelism ($K > 4$), it is logic dominated. The design with $K > 14$ PEs cannot fit into commercial Virtex II Pro FPGAs.

VI. Conclusions

In this paper, two methods for distributing the resampling step suitable for distributed real-time FPGA implementation are proposed. The practical guidelines for choosing the resampling method depend primarily on the desired performance, communication pattern and complexity of the CU.

PF performance of centralized resampling and the RPA algorithm are the same as the sequentially implemented PF. However, there are no advantages in using centralized resampling since the RPA algorithm is faster and has a simpler CU. On the other hand, the RNA algorithm trades PF performance for speed improvement. So, RPA algorithm is a good choice when it is necessary to preserve performance, but with significant increase in complexity.

Communication pattern in the RPA algorithm is non-deterministic. As such, it requires point-to-point network to achieve the minimum execution time. The RNA algorithm can also achieve

Area/level of parallelism	1	2	4	8	16	32
Memory bits for M=10000 (Mbits)	1.53*	1.62*	1.94	2.3	2.59	5.1
Multiplier blocks	10	20	40	80	160	320
Number of slices (Kslices)	4	8	16*	32*	64*	128*
Xilinx chip that fits the design	XC2VP20	XC2VP30	XC2VP40	XC2VP70	-	-

TABLE I

The number of memory bits, slices and block multipliers for the distributed PF implementation with RNA with local exchange. The Virtex II Pro chips that can be fitted by the PF parameters are listed. The star shows which parameter determined in choosing the chip.

minimum execution time, but its architecture consists only of local connections. The communication pattern of the RNA algorithm with regrouping is somewhere in between the RNA algorithm with local exchange and the RPA algorithm. If the size of the group is larger than two, the RNA algorithm with regrouping also suffers from a non-deterministic communication pattern. However, the amount of particles that have to be exchanged inside groups is smaller than for the RPA algorithm.

The complexity of the CU of the RPA algorithm is very high since it has to implement a complex routing protocol through point-to-point network. The CU of the RNA algorithm with local exchange is simple and is not responsible for particle routing after resampling. The RNA algorithm with regrouping has to have control units in every group when groups contain more than two PEs. So, when speed is important and when it is required that design time is low (low complexity of the CU and of the scheduling and protocol in interconnection networks) the RNA algorithm with local exchange is the preferred solution.

References

[1] M. Bolić, P. M. Djurić, and S. Hong,"Resampling Algorithms for Particle Filters: A Computational Complexity Perspective," submitted to the EURASIP Journal of Applied Signal Processing, 2003.

[2] 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.

[3] J. Carpenter, P. Clifford, and P. Fearnhead, "An improved particle filter for non-linear problems," *IEE Proceedings on Radar and Sonar Navigation*, vol. 146, no.1, pp. 2-7, 1999.

[4] D. E. Culler and J. P. Singh, *Parallel Computer Architectures: A Hardware/Software Approach,* Morgan Kaufmann Publisher, 1999.

[5] W. G. Cohran, *Sampling Techniques,* London: Wiley, 3nd edition, 1963.

[6] P. M. Djurić, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, and J. Míguez, "Particle filtering," *IEEE Signal Processing Magazine*, vol. 20, no. 5, pp. 19-38, 2003.

[7] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*, New York: Springer Verlag, 2001.

[8] J. L. Dekeyser, C. Fonlupt, and P. Marquet, "Analysis of synchronous dynamic load balancing algorithms," *Advances in Parallel Computing*, vol 11, pp. 455-462, 1995.

[9] P. Fearnhead, *Sequential Monte Carlo Methods in Filter Theory*, Ph.D. Thesis, Merton College, University of Oxford, 1998.

[10] G. S. Fishman, *Monte Carlo: Concepts, Algorithms and Applications*, Springer series in operationsl research, Springer-Verlag, 1st edition, 1995.

[11] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "A novel approach to nonlinear and non-Gaussian Bayesian state estimation," *IEE Proceedings F*, vol. 140, pp. 107-113, 1993.

[12] S. Hong, S. S. Chin, M. Bolić, P. M. Djurić, "Design and Implementation of Flexible Resampling Mechanism for High-Speed Parallel Particle Filters," submitted to *Journal of VLSI Signal Processings*, 2003.

[13] G. Kitagawa, "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models," *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, pp. 1-25, 1996.

[14] A. Kong, J. S Liu, and W. H. Wong,"Sequential imputations and Bayesian missing data problems," *Journal of American Statistical Association*, vol. 89, no. 425, pp. 278-288, 1994.

[15] J. S. Liu and R. Chen, “Blind deconvolution via sequential imputations,” *Journal of the American Statistical Association,* vol. 90, no. 430, pp. 567-576, 1995.

[16] J. S. Liu and R. Chen, “Sequential Monte Carlo methods for dynamic systems,” *Journal of the American Statistical Association,* vol. 93, pp. 1032-1044, 1998.

[17] J. S. Liu, R. Chen, and T. Logvinenko, “A Theoretical Framework for Sequential Importance Sampling and Resampling,” in [7].

[18] B. Ristic, S. Arulampalam, N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House, 2004.

[19] S. G. Shiva, *Pipelined and parallel computer architectures*, Harper Collins College Publisher, 1996.

[20] *SystemC 2.0.1 Language Reference Manual*, Available from Open SystemC Initiative at http://systemc.org, 2003.

[21] V. Teuilere, O. Brun, “Parallelisation of the particle filtering technique and application to Dopler-bearing tracking of maneuvering sources,” *Parallel Computing*, Elsevier Science, vol. 29, pp. 1069-1090, 2003.

[22] M. Wolfe, *High Performance Compilers for Parallel Computing,* Addison-Wesley Publishing Company, 1996.

[23] Xilinx Inc., “Virtex-II Pro Patforms FPGA: Functional Description,” Available from www.xilinx.com, June 2003.