J Sign Process Syst (2017) 88:29–42
DOI 10.1007/s11265-016-1110-0

# DART: Distributed Particle Filter Algorithm with Resampling Tree for Ultimate Real-Time Capability

**Qinglin Tian** [1] · **Yun Pan** [2] · **Zoran Salcic** [3] · **Ruohong Huan** [4]

Received: 6 December 2015 / Revised: 14 January 2016 / Accepted: 28 January 2016 / Published online: 12 February 2016


**Abstract** A novel Distributed Particle Filter Algorithm with Resampling Tree, called DART, is proposed in this paper, where particles are resampled by Branch Resampling and Root Resampling in a flexible tree-like structure. Though sampling and weight calculation can be executed in parallel on a group of Processing Elements, resampling is the bottleneck for distributed particle filters since it requires the knowledge of the whole particle set. Conventional approaches to accelerate resampling on distributed platforms often introduce extra procedure other than the standard processing flow and achieve acceleration limited by linear speedup. By introducing the proposed algorithm, where Branch Resampling can be executed in parallel with sampling and weight calculation, the number of particles in the final sequential implemented Root Resampling can be reduced in an exponential relationship with the depth of the tree. With the same linear speedup in sampling and weight calculation steps, the overall speedup achieved in DART surpasses linear boundary and outperforms state-of-art approaches. The corresponding implementation architecture, which possesses unique features of hardware efficiency and scalability, is also presented. The prototype of the algorithm with 8 PEs is implemented on a Xilinx Virtex-IV Pro FPGA (XC4VFX100-12FF1152) under BOT system. With 8192 particles, the input observation can achieve 63.3 kHz at a clock speed of 80 MHz.

**Keywords** Resampling tree · Distributed particle filters · Parallel

✉ Yun Pan
panyun@vlsi.zju.edu.cn

Qinglin Tian
tianql@vlsi.zju.edu.cn

Zoran Salcic
z.salcic@auckland.ac.nz

Ruohong Huan
huanrh@zjut.edu.cn

[1] College of Electrical Engineering, Zhejiang University, Hangzhou, China

[2] College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China

[3] Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand

[4] College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China

## 1 Introduction

Particle filters (PFs) [1], consisting of iterative sampling, weight calculation and resampling, are proved to be a powerful tool dealing with non-linear/non-Gaussian Bayesian state estimation problems and have been widely applied in video tracking, indoor positioning, robotics systems and SLAM [2–5]. In indoor positioning context, the localization error can be reduced by about 40 % with the help of sensor information fusion by PFs [6] and its effectiveness in non-linear tracking is proved in [7]. [7] also shows by simulation that the position errors decrease with increasing number of particles up to 50000. Moreover, particle filter based map-matching algorithm is frequently used in literature [8–11]. Most studies [6, 7, 12, 13] uses simulation/offline-processing to verify the accuracy of tracking systems since PF is a Monte Carlo approximation based method and the high computational complexity becomes the bottleneck in implementation for such systems that have stringent real-time requirements.

For conventional centralized PFs, where sampling, weight calculation and resampling are handled by a single processing element (PE), i.e. sequentially, the processing time of one filtering recursion increases significantly with the number of particles used, which is unrealistic for real-time systems as tens of thousands particles are required usually. Distributed PFs with a number of PEs that operate in parallel are proposed, taking advantage of the inherent parallelism in the algorithm, to accelerate the overall process of filtering by processing sampling and weight calculation for the large group of particles simultaneously on several PEs. But resampling is difficult to parallelize since it requires a joint process on the whole particle set [14]. proposes Resampling with Proportional Allocation (RPA) and Resampling with Non-proportional Allocation (RNA) for distributed PFs to allow concurrent resampling on several PEs while the additional particle redistribution/exchange presents a challenge for hardware realization [15]. divides particles into a sub-group of local PFs and process them with independent sampling, weight calculation and resampling processes to increase parallelism. However, extra particle mixing procedures among local PFs are inevitably introduced to maintain filtering accuracy. The mixing step is optimized in [16] by only exchanging representative particles with larger weights. Speedup is typically used as the metric for comparisons of execution time and the speedup achieved in distributed PF architecture refers to the relative execution time decrease compared to the centralized (sequential) implementations. For instance [14–16], accelerate sampling, weight calculation and resampling by parallel processing in multiple PEs. A linear speedup is achieved since the execution time is reduced linearly with the number of PEs used. Despite the achieved speedup, the extra particle redistribution steps involved in [14–16] decrease the overall algorithm acceleration. The approach presented in [17] proposes a Hierarchical Resampling (HR) algorithm that decomposes resampling into two granular hierarchies and achieves a linear speedup without introducing extra steps. Therefore, it stands out as the state-of-art reference algorithm amongst distributed PFs.

In this paper, we are aiming at further accelerating particle filtering process to increase its real-time capability, which is the key for real-life use of the PFs. A Distributed PF Algorithm with Resampling Tree (DART) is proposed. DART keeps the same processing flow of sampling and weight calculation steps as in classical distributed PFs such as RPA and RNA [14], while implements resampling with a novel Resampling Tree Scheme (RTS). Particles are resampled with one or more Branch Resampling (BR) and one Root Resampling (RR) in a tree structure. In DART, any BR can be executed in parallel with sampling and weight calculation. RR is the only sequential part of resampling after sampling and weight calculation and the number of particle RR has to handle can be exponentially reduced by BR process. Furthermore, RR indirectly resamples the whole particle group, introducing no extra procedure. Consequently, DART makes the breakthrough by achieving a further exponential acceleration and becomes suitable for ultimate real-time capacity of PFs.

Besides the algorithm, this paper also presents the corresponding implementation architecture for DART. In distributed platforms, scalability of hardware architecture is an important and favorable characteristic. Keeping this in mind, the proposed DART architecture supports configurations with varying number of PEs. Though PF systems range from simple one-dimensional to complex multiple-dimensional scenarios, they differ in the PEs' sampling and weight calculation function. At the same time, the resampling processes share common characteristic that processing is mainly based on weight of the particles. So, the architecture can be readily modified and extended to run in different applications that require customized configurations. Finally, hardware efficiency is achieved by minimizing the hardware design cost of the proposed algorithm.

By proposing DART and its hardware architecture, the main contributions of the paper are:

- The proposed algorithm for distributed PFs resamples particles with a novel Resampling Tree Scheme. An exponential speedup in processing time of resampling is achieved, which outperforms existing state-of-art approaches for distributed PFs.
- Resampling is achieved through a Resampling Tree Scheme, RTS, which has significant flexibility in the way that a different tree structure with distinctive tree depth or branches corresponds to an alternative implementation of the algorithm.
- The proposed architecture possesses high scalability to satisfy various configurations of the target system. Achieved hardware efficiency is reflected through efficient consumption of hardware resource of FPGA used for the implementation.
- The proposed approach to PFs implementation achieves the ultimate goal of real-time capability at reasonable cost lacking in other approaches.

The rest of the paper is organized as follows. First, a summary of related works are presented in Section 2. Section 3 gives a detailed description of the proposed DART algorithm. The hardware architecture designed is then illustrated in Section 4. Section 5 presents the evaluation and experimental results of the algorithm as well as comparison with previous studies. And finally, conclusion and future works are presented in Section 6.

## 2 Related Work

Proposed in [1], particle filter is a Monte Carlo based method and uses a group of weighted particles to estimate the probability density function (PDF) of the target Bayesian state

system recursively. To solve the problem of particle degradation and sample impoverishment [18], refines the filtering process with an efficient resampling step known as Systematic Resampling (SR) to discard particles with negligible weights.

Due to its Monte Carlo nature, the accuracy of PFs increases with larger amount of particles used. For instance, the localization error rate drops more than 20 % to around 5 % when the number of particles used increases from 256 to 131072 [19]. In [16], one million particles are required to reduce an average tracking error of a robotic arm to 3 mm. In real-time indoor tracking applications using particle filters, only tens or hundreds particles are used due to limitations in real-time capability. A total number of 64 particles are used in [8, 20] uses 100 particles in map-matching algorithm and up to 250 particles are used in [9]. It is proved in [7], by simulation, that increasing number of particles to 50000 helps in reduction of position errors in indoor tracking scenario. Therefore, PFs with better real-time capability are needed.

Distributed PF is a natural and promising solution based on the fact that sampling and weight calculation for the group of particles can be executed in parallel. Studies on using both FPGAs [14, 17] and multi-core processor/GPU [15, 16, 19] can be found in literature, showing significant speedup compared to centralized implementation. These works focus on the bottleneck of real-time use of PFs, i.e. resampling, as it requires the knowledge of the whole particle set [14]. proposed Resampling with Proportional Allocation (RPA) and Resampling with Non-proportional Allocation (RNA) to accelerate filtering in the way that sampling, weight calculation and resampling are all done in parallel in PEs. But they also introduce additional process of particle redistribution in RPA and particle exchange in RNA. For RPA, the main drawback is that the execution time of the particle redistribution is not deterministic, which is unsuitable for hardware implementation. Besides, the memory overhead is also tremendous since it has to be designed to handle the worst case. In case of RNA with a simplified process of particle exchange, i.e., local exchange, performance deterioration can be observed since the covariance of particle weights is usually larger. Another way to improve parallel execution is proposed in [15] by running several local PFs concurrently on PEs. Still, particle mixture among PEs is added to maintain filtering accuracy. The mixing procedure is simplified in [16] in the way that weight sorting is performed for particles before mixture and only particles with larger weights are transferred among PEs. It reduces the communication traffic of the network at the same time. To overcome the disadvantage of additional introduced procedures [17], proposes a Hierarchical Resampling (HR) based resampling scheme for distributed PFs that achieves a processing flow free from extra steps. Also, Residue Cumulative Resampling (RCR) is proposed in [17] according to the fact that the replication factor of a particle is related to its own weight as well as the cumulative weight residue of the previous particle.

For distributed PFs with $K$ PEs, Fig. 1 gives a simplified illustration of timing for a PF system with $N$ particles. Clearly seen from centralized and distributed PFs, a linear speedup on sampling, weight calculation and resampling is achieved by scaling their processing time to $1/K$. Figure 1(b) presents algorithms, summarized as Type 1, that require extra steps other than the standard flow [14–16], as presented by the dashed part of $E$ as these steps may appear before and/or after resampling and consume a time of $t_0$ and $t_1$, respectively. Figure 1(c) is referred to the HR based algorithm, Type 2, where no extra steps are introduced and achieving a linear speedup when omitting the startup latencies [17].

To the best of our knowledge, linear speedup is the boundary for existing state-of-art algorithms. While in the practical domain of PFs, simulation/offline processing is commonly seen. A recent work [5] implements a complete real-time vision-based SLAM system with PF, running at 30Hz by accelerating PF using GPU. Clearly seen, a distributed PF algorithm with better capability in real-time performance is needed for future systems with higher frame rates and processing volume. By proposing DART in this paper, the current limit of linear speedup is pushed further and an exponential speedup for resampling is attained with the Resampling Tree Scheme (RTS).

## 3 Proposed Algorithm

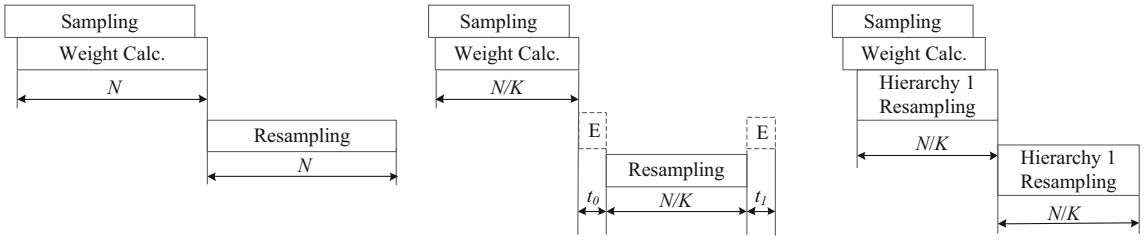
This section gives a comprehensive description of the proposed DART algorithm and the RTS employed.

### 3.1 DART Overview

For a distributed PF with $N$ particles and $K$ PEs, sampling and weight calculation is processed simultaneously on $K$ PEs, while resampling of the weighted particles is performed with the technique that employs RTS. RTS is responsible for generating the replication factors of every particle. Thus, by replicating corresponding particles of the previous iteration, the sampling of next iteration can be initiated.

### 3.2 Resampling Tree Scheme

In RTS, particles are resampled in a tree structure. Two kinds of resampling steps, i.e., Branch Resampling (BR) and Root Resampling (RR), constitutes the whole resampling. BR and RR resample particles and generate the corresponding replication factor of every particle according to its weight. The resampling tree is formed starting from the leaf nodes, which represent the weighted particles directly generated by PEs. As illustrated in Fig. 2, a sub-group of leaf nodes are resampled by BR process, generating an intermediate node at a higher level as the parent of the nodes involved in the BR. The weight




(a) Centralized PF (b) Distributed PF, Type 1 (c) Distributed PF, Type 2

**Figure 1** Timing comparison of different PFs.

of the intermediate node equals to the sum of weights of the nodes within the BR step, i.e., all its children. Sub-group of intermediate nodes can be again resampled by BR process and generate a parent node at a higher level in similar way. BR processes can be repeated until reaching the level next to root, where RR takes place by resampling all particles on this level and producing its replication factors.

By performing BR and RR, the resampling tree is formed with node representing a weighted particle and any nodes except the leaf ones have a weight equivalent to the sum of weights of its children. In a resampling tree with depth $D$ as in Fig. 2, certain constraints are necessary when constructing the tree. For intermediate nodes on depth 2 to $D$, they are generated by BR processes from the immediate previous level. The branching factor of these nodes defines the number of children, which is also the number of particles taken in a BR. In RTS, it is required that all nodes on the same level have the same branching factor. Therefore, in the example of Fig. 2, the branching factor of nodes on depth 2 is 2 as every node is generated by resampling 2 nodes on depth 1 and the branching factor is 3 for nodes on depth 3.

### 3.2.1 Determine the Tree Topology

With the above constraint that branching factor remains the same for all nodes on one level, the topology of a resampling tree with $N$ leaf nodes can be determined by the tree depth $D$ together with a set of branching factor on each level, $f_b^d$ $(2 \le d \le D)$. The formed resampling tree has a height of $D+1$ with root included and the tree is both a complete and full tree. Every node on level $d$ $(2 \le d \le D)$ has $f_b^d$ child nodes.

It can be seen, the number of nodes on level $d$ $(2 \le d \le D)$ scales to $1/f_b^d$ the number on level *d-1*. In order to keep an integer number of nodes on each level, the total number of leaf nodes/particles $N$ is preferred to have multiple divisors. The resampling tree is impossible to grow up from a leaf node by BR when $N$ is a prime number. Thus, the branching factor for each level should satisfy the requirement as in

$$f_b^d \Big| n_{d-1} \ (2 \le d \le D), \tag{1}$$

where $n_d$ is the number of nodes on level $d$, i.e., the branching factor on level $d$ is a divisor of the number of nodes on previous level *d-1*. The analytical form of $n_d$ is further expressed as

$$n_d = \begin{cases} N, & d = 1 \\ \dfrac{N}{\prod_{j=2}^{d} f_b^j}, & 2 \le d \le D \end{cases}. \tag{2}$$

Apart from branching factor, the depth $D$ defines the height of the tree. A level of nodes resampled by BR will increment

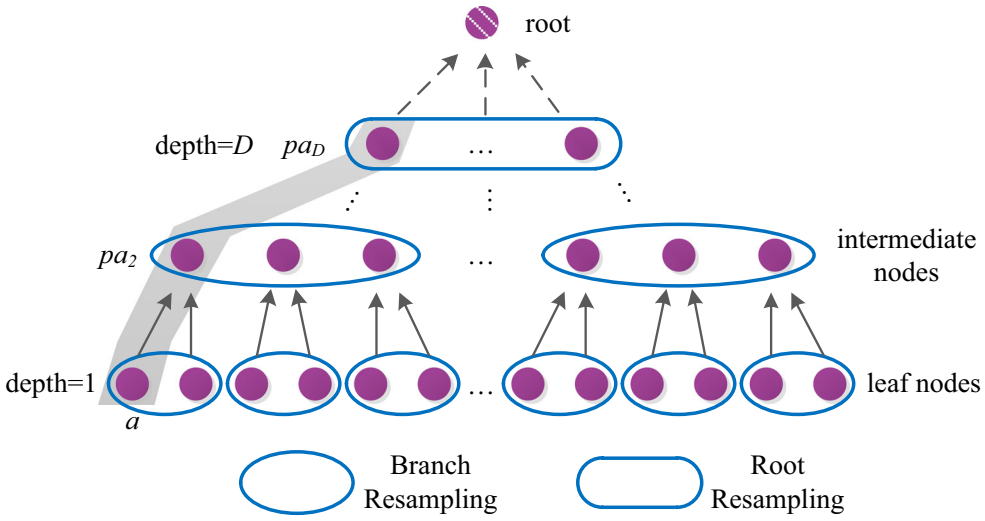



**Figure 2** Resampling tree scheme.

the value of $D$ by 1. The height of the tree can grow as long as an appropriate branching factor satisfying Eq. (1) can be selected. In practice the topology of the tree corresponds to an implementation of a resampling process and $D$ should be selected with care in terms of hardware consumption and filtering accuracy. The influences of $D$ on these issues are further discussed in Section 5.

The flexibility of RTS is reflected in the tree topology. Starting from the same set of leaf nodes, a different tree depth $D$ or distinctive branching factor for different levels would form an alternative tree structure. In this way, the scheme is configurable in terms of depth and branching factor, resulting in diverse topologies of the resampling tree.

#### 3.2.2 Calculating the Final Replication Factor

After the tree topology is determined, the resampling can be conducted by BR and RR as described before. Every node except the root will have a replication factor generated in its relevant BR or RR process. The final replication factor of a leaf node, which corresponds to a particle, is obtained as the product of the replication factor of itself and all its ancestors on level 2 to $D$. As depicted in the shaded area of Fig. 2, the leaf node $a$ on depth $d = 1$ has an ancestor $pa_d$ on each level with depth $d$ ($2 \le d \le D$). So the final replication factor of $a$ is the product of the replication factor of $a$, $pa_2$ throughout $pa_D$.

At this stage, the RTS can be represented by $RTS(N, D, f_b^d)$ which resamples $N$ weighted particles with a resampling tree topology defined by $D$ and $f_b^d$, generating the replication factor $r^i$ ($1 \le i \le N$) for each particle.

#### 3.2.3 Statistical Analysis

This part gives a statistical proof that RTS is equivalent to standard SR [18].

Consider employing the basic resampling technique of SR for all resampling processes, the replication factor expectation of a particle in standard SR is represented in

$$E(r^i) = \frac{N\omega^i}{\sum_{j=1}^{N}\omega^j} \tag{3}$$

from a statistical viewpoint where $N$ is the total number of particles, $E(\cdot)$ is the expectation function, $\omega^i$ and $r^i$ is the weight and replication factor of $i^{th}$ particle, respectively.

While in the proposed RTS with tree depth of $D$ and branching factor $f_b^d$ for the BR process on depth $d$ ($2 \le d \le D$), one particle is resampled, both directly and indirectly, by BR on depth 1 through $D$-1 and RR on depth $D$. For the BR on depth $d$, it resamples on $f_b^{d+1}$ particles so the expectation of intermediate replication factor for one particle with weight $\omega_d^i$ is expressed in

$$E_d = \frac{f_b^{d+1}\omega_d^i}{\sum_{j=1}^{f_b^{d+1}}\omega_d^j} (1 \le d \le D-1). \tag{4}$$

Besides, it also generates a particle on the next depth $d+1$ with weight equals to the sum of all particles within the resampling as shown in

$$\omega_{d+1}^i = \sum_{j=1}^{f_b^{d+1}} \omega_d^j (1 \le d \le D-1). \tag{5}$$

RR resamples all the particles on depth $D$ and the number of particles is $n_D$ in Eq. (2). Similarly, the replication expectation of a particle in RR is given in

$$E_D = \frac{n_D\omega_D^i}{\sum_{j=1}^{n_D}\omega_D^j}. \tag{6}$$

Combining the knowledge of how final replication factor of a particle is calculated and Eqs. (2), (4) ~ (6), the expectation can be reduced to the same form as in Eq. (3).

$$\begin{aligned} \mathrm{E}(r^i) &= \textstyle\prod_{d=1}^{D-1} E_d \cdot E_D \\ &= E_1 \cdot E_2 \cdots E_{D-1} \cdot E_D \\ &= \frac{f_b^2\omega_1^i}{\sum_{j=1}^{f_b^2}\omega_1^j} \cdot \frac{f_b^3\omega_2^i}{\sum_{j=1}^{f_b^3}\omega_2^j} \cdots \frac{f_b^D\omega_{D-1}^i}{\sum_{j=1}^{f_b^D}\omega_{D-1}^j} \cdot \frac{\frac{N}{\prod_{j=2}^{D} f_b^j}\omega_D^i}{\sum_{j=1}^{\frac{N}{\prod_{j=2}^{D} f_b^j}}\omega_D^j} \\ &= \frac{N\omega_1^i}{\sum_{j=1}^{n_D}\omega_D^j} = \frac{N\omega^i}{\sum_{j=1}^{N}\omega^j}. \end{aligned} \tag{7}$$

As can be seen from the above equations, the replication factor in both RTS and traditional resampling has the same expectation statistically. This guarantees that the proposed algorithm accelerates resampling process without sacrificing filtering accuracy. The performance is further discussed in Section 5.

### 3.3 Accelerating Particle Filtering

In a DART particle filtering system, the processing time of one recursion can be significantly reduced using the proposed RTS. Resampling is done with a series of BR steps plus a final RR procedure. As described above, BR only resamples among a sub-group of nodes, i.e., particles. Therefore, any BR can be initiated as long as the nodes it deals with are prepared. When PEs have done sampling and weight calculation for a sub-group of particles, BR can start resampling them while PEs are sampling remaining particles of the whole particle set.

This feature is also observable in Fig. 2; BR is only dependent on the nodes it takes in, so BR is processed in parallel with sampling and weight calculation steps. RR is the only resampling procedure that requires the knowledge of the whole particle set, thus it is sequentially implemented after all weights of particles are calculated. It can be seen that the number of particles RR has to handle is drastically reduced. Referring to Eq. (2) where depth equals $D$, the number of nodes on the level next to root is $n_D$. By setting a constant branching factor $f_b^{const}$, $n_D$ is exponentially reduced with tree depth $D$ as in

$$n_D = N/f_b^{const^{D-1}}. \tag{8}$$

Since the processing time of resampling has a linear relationship with the number of particles, the time consumed by the final sequentially implemented resampling also achieves an exponential decrease. Furthermore, as RR indirectly resamples the whole particle set, no extra processing is needed and this also proved in Section 3.2.3 that RTS is equivalent to standard SR.

As a summary, in a DART implementation of $K$ PEs and $N$ particles $DART(N, K)$, the filtering process is presented in the pseudo-code shown in Table 1 when resampling is done with $RTS(N, D, f_b^d)$ $(2 \le d \le D)$.

## 4 Proposed Hardware Architecture

This section presents the hardware architecture designed for the proposed $DART(N, K)$ algorithm under an resampling implementation of $RTS(N, D, f_b^d)$ $(2 \le d \le D)$.

The distributed implementation of particle filter has $K$ PEs, so $K$ particles with their corresponding weights are generated simultaneously. Despite the flexibility in RTS that branching factor of BR process on each level can choose distinctive values, the branching factor on depth 2, $f_b^2$, is set to $K$ to ensure a minimal design cost. In this way, every time $K$ weighted particles are sampled by PEs, the BR on depth 1 is initiated without extra delay and the control logic is also simplified.

Figure 2 gives an overall view of the architecture for DART with $K = 4$ Processing Elements (PEs) as an example. Each PE is responsible for sampling and weight calculation steps while the central part, Resampling Tree Module (RTM), implements the resampling using RTS, completing a whole iteration of the PF. BRM #$d$ $(1 \le d \le D\text{-}1)$ corresponds to a module responsible for the BR on depth $d$ and RRM is the module implementing the final resampling of RR. Finally, the Replication Factor Generation (RFG) module calculates the final replication factor for each particle combining the results of RR and all previous BRs. As PEs responsible for sampling and weight calculation are similar in generic distributed implementations, the following subsections mainly focus on the RTM that implement the RTS for DART.

### 4.1 First BR Module

Although BR on different depth levels does similar processing, the First BR Module (FBRM) is designed in a different way from Remaining BR Modules (RBRMs) in terms of hardware implementation and efficiency. In order to avoid the situation where replication factor for every particle needs to be calculated and stored, the architecture of Intermediate Resampling proposed in [17] is adopted as the FBRM to cut down the overhead of required memory. As a result, particle states together with weights are transferred between PEs and RTM as shown in Fig. 3. Figure 4 shows the architecture for the FBRM, which pipelines the BR on depth 1 when $K = 4$ particles and weights are produced. It employs the basic resampling strategy of SR to produce resampled particle states. Upon receiving particle weights $w_0 \sim w_3$, the temporary sum of weights (TSW) is calculated where $\mathrm{TSW}_i$ equals to the sum of the first $i + 1$ $(0 \le i \le 3)$ weights and the sum of weights $S = \mathrm{TSW}_3$. Also, weight mean value $S/K$ is used to update the variable $U$ used in SR. Every IR stage compares all TSWs with the resampling variable $U_i$ $(0 \le i \le 3)$ to determine which particle is resampled. As depicted in the detailed structure of one FBRM stage in the right side of Fig. 4, when $U_3$ and TSWs has the value provided, the resampled particle is $p_3$ since $\mathrm{TSW}_3$ is the first one no less than $U_3$. Then, according to the select signal, resampled particle states are sent back to each PE to be stored in the particle state memory. On the other hand, the sum of weights is also passed down to the next BR unit as a single weight prepared for the BR process on next depth level. In this way, $K$ particle states with the same index in each PE share the same final replication factor generated by RTM since they are resampled within the same BR process. So, the memory consumption for recording replication factors is significantly reduced a factor of $1/K$. Interested readers can also refer to [17] for details.

### 4.2 Remaining BR Module

For the Remaining BR Modules (RBRM) resample particles on depth 2 to $D$-1 of the resampling tree also employ the same resampling technique with the FBRM. But, differences can be observed that the first BR is pipelined and resamples $K$ particles every cycle throughout the sampling and weight calculation process while BR on depth other than 1 takes place less often. Generally in $RTS(N, D, f_b^d)$ with the above presented hardware implementation ($f_b^2$ set to $K$), the BR on depth $d$ $(2 \le d \le D\text{-}1)$ need to resample $f_b^{d+1}$ particles every $\prod_{j=2}^{D-1} f_b^{d+1}$ cycles. As a result, the timing constraint is relaxed and RBRM can be implemented without pipeline and simplified to

**Table 1** Pseudo-code for DART.

*DART*(*N*, *K*)

**-do** in parallel in *K* PEs for *N* particles

sample a new particle $x_t^{i,j}$ from the proposal distribution

$$x_t^{i,j} \sim q(x_t | x_{t-1}^{i,j}, z_t), j = 1,2,\ldots,K$$

calculate its corresponding weight $w_t^{i,j}$, prepare it as one particle for the BR on depth 1

$$w_t^{i,j} = \frac{p(z_t | x_t^{i,j}) p(x_t^{i,j} | x_{t-1}^{i,j})}{q(x_t^{i,j} | x_{t-1}^{i,j}, z_t)}$$

**-end**

Resampling *N* particles by *RTS*(*N*, *D*, $f_b^d$), generate replication factors $\{w_t^i\}_{i=1}^N$

****for*** *d*=1:*D*-1

**-do** BR on this level:

when $f_b^{d+1}$ particles with weights $\{w_t^{j,d}\}_{j=1}^{f_b^{d+1}}$ are prepared, resample according to $\{w_t^{j,d}\}_{j=1}^{f_b^{d+1}}$ and generate corresponding replication factor $\{r_t^{j,d}\}_{j=1}^{f_b^{d+1}}$

generate a particle with weight $w_t^{i,d+1} = \sum_{j=1}^{f_b^{d+1}} w_t^{j,d}$ prepared for the next level *d*+1

**-end**

***end *for***

**-do** RR on depth *D*

resample $N/\prod_{j=2}^{D} f_b^j$ particles and generate corresponding replication factor $\{r_t^{j,D}\}_{j=1}^{N/\prod_{j=2}^{D} f_b^j}$

**-end**

Calculate final replication factor for each particle $\{w_t^i\}_{i=1}^N$

reduce hardware consumption. Every RBRM only needs one stage of the FBRM architecture for the main resampling task.

As depicted in Fig. 5(a) with branching factor equals to 4, four prepared weights $w_0 \sim w_3$ initialize the BR. Same as described

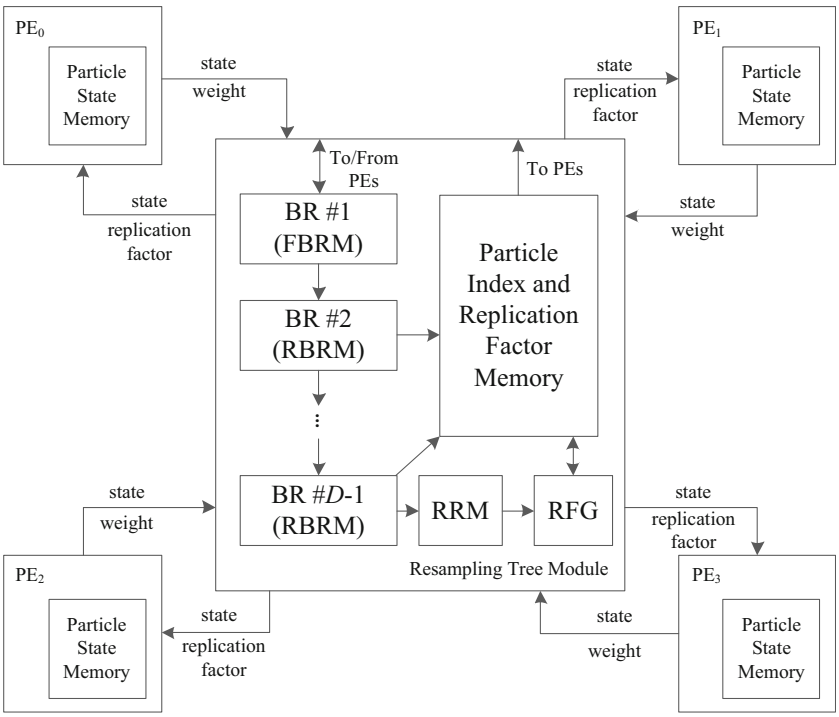



**Figure 3** Architecture of DART.

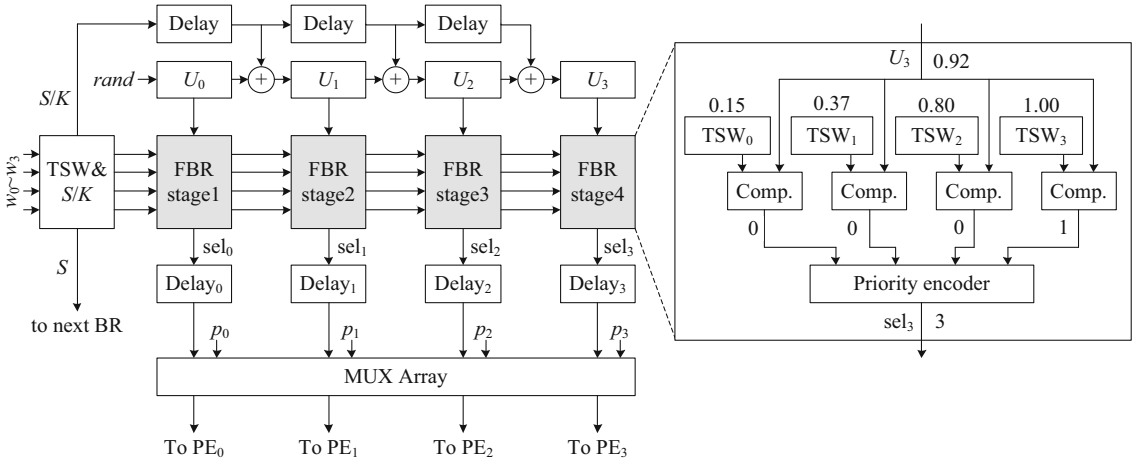



**Figure 4** Architecture of the first BR module.

in FBRM, all TSW and *S*/*K* are calculated and used to resample particles. Every cycle, one particle is resampled and the variable *U* is updated. And the select signal is used to generate the replication factor for each weight in the BR process. When all factors are generated, they are stored in the intermediate replication factor memory for this level and used in calculating the final replication factor later on. At the same time, the sum of all weights is sent to the next BR as a single weight prepared except for the last level of BR on depth $D-1$, the weight sum is stored in collective weight memory for RR to process.

### 4.3 RR Module

When all process of BR is completed and necessary collective weights are saved, the final RR Module is invoked to resample the last $n_D$ (Eq. (2)) particles. There are many ways of implementing the architecture of RRM with the main goal to generate the replication factor of all particles based on weights. Figure 5(b) is a RRM architecture employing a RCR based scheme to generate replication factors, refer to [17] for detailed processing flow. With prepared parameter $Q = n_D / S$ where $S$ is the sum of all $n_D$ particles, every time a weight $w_i$ is read out from the collective weight memory, the corresponding replication factor $r_i$ is obtained as the integer part of *temp*, which is calculated by $Q^* w_i + l$. $l$ is the current residue whose value is a random number ~ $U(0,1)$ for the first weight. For weights followed-up, the residue equals to the decimal part of *temp* generated by the previous weight and is recursively updated every time a replication factor is calculated.

For implementing RCR based resampling, the finite precision effect studied in [21] which would lead to the situation that the sum of all calculated replication factors is smaller than the total number of particles. The issue is handled the same way as in [17] so that the last replication factor is the difference between total number of particles and the sum of all already calculated replication factors.

### 4.4 Replication Factor Generation Module

Once a replication factor is generated by RR, it is used in calculating final replication factors related to it. RR will only

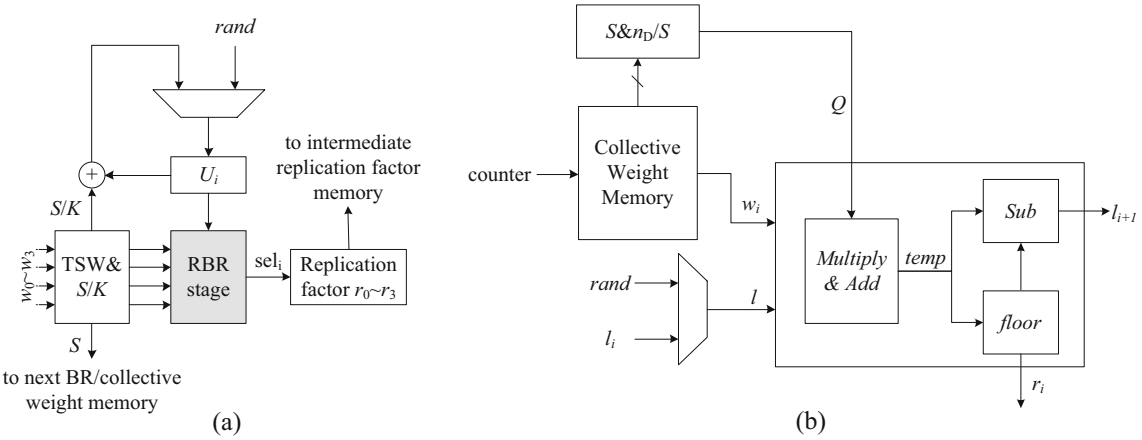



**Figure 5** **a** Architecture of RBRM **b** Architecture of RRM.

generate $n_D$ replication factors so each one is related to $\prod_{j=3}^{D} f_b^j$ final factors under the replication factor sharing mechanism by FBRM. The final replication factor of a particle is obtained in the way shown in Fig. 6. The counter, whose value increments from 1 to $n_D$, is used to index the corresponding weight for RR as well as the group of generated intermediate replication factors by BR. RR generate replication factor $r_{\mathrm{iD}}$ and a group of factors $r_{\mathrm{id}}$ $(2 \le d \le D\text{-}1)$ is indexed out from the memory for intermediate factors with number of $r_{\mathrm{id}}$ equals to $n_r^d$ $(2 \le d \le D\text{-}1)$ for each level $d$. All these replication factors are processed in the multiplier matrix, generating $\prod_{j=3}^{D} f_b^j$ results to be stored in the final replication factor memory.

$$n_r^d = n_d / n_D = \prod_{j=d+1}^{D} f_b^j \ (2 \le d \le D-1) \ . \tag{9}$$

### 4.5 Hardware Scalability

With the above presented architecture of DART which pipelines the BR on depth 1, hardware scalability can be observed. FBRM is the module interacting with PEs and it can be easily extended by adding more stages to support more PEs connecting to the RTM. At the same time, the number of RBR units in RTM reflects the tree depth $D$ in RTS and every RBRM responsible for BR on depth $d$ is able to resample particles according to the branching factor defined. So, the proposed architecture is able to support implementing RT algorithm with various tree topologies. Also, it has features of deterministic execution with no extra procedure similar to particle redistribution or mixture and the replication factor sharing scheme in FBRM guarantees efficient consumption and use memory.

## 5 Evaluation & Experimental Result

In this section, experimental results regarding the evaluation of the proposed algorithm are presented in terms of filtering performance, processing time and resource consumption.

### 5.1 Filtering Performance

Firstly, the simulated filtering performance of the DART algorithm is illustrated in Figs. 7 and 8 in comparison to RPA [14] since RPA is equivalent to standard SR theoretically. With Root Mean Square Error (RMSE) as criterion, the experiment is conducted under BOT [18] model using particle numbers varying from 1024 to 8192 with a 1024 step size. For simplicity in setting the variable in a $DART(N, K)$ algorithm with $RTS(N, D, f_b^d)$, branching factors $f_b^d$ $(2 \le d \le D)$ are all set to $K$, referred to as $RTS(N, D, K^{const})$. Figure 7(a) presents DART with $K = 4$ PEs and two tree depth $D = 3$ and 4 are studied. While for Fig. 7(b), $K = 2, 4, 8$ with a fixed tree depth $D = 3$ are depicted. When filtering with a minimum of 1024 particles with $D = 4$ and $K = 8$, there are only 2 particles for the last resampling of RR, therefore, no further increment to $D$ and $K$ is made in the experiment.

As can be seen from the two graphs, the performance of DART is lower compared to RPA with fewer particles. The reason for the phenomenon is that the RTS employed in DART will make the resulting resampled particles less diverse. Since the final replication factor of a particle after resampling in RTS is the product of intermediate replication factors itself and all its ancestors, any factor equals to zero would lead to a zero final replication factor. The less diverse particle would cause sample impoverishment and reduce the filtering accuracy. However, for larger number of particles, DART consistently shows slightly better accuracy. For particle number greater than 3072, the filtering performance is

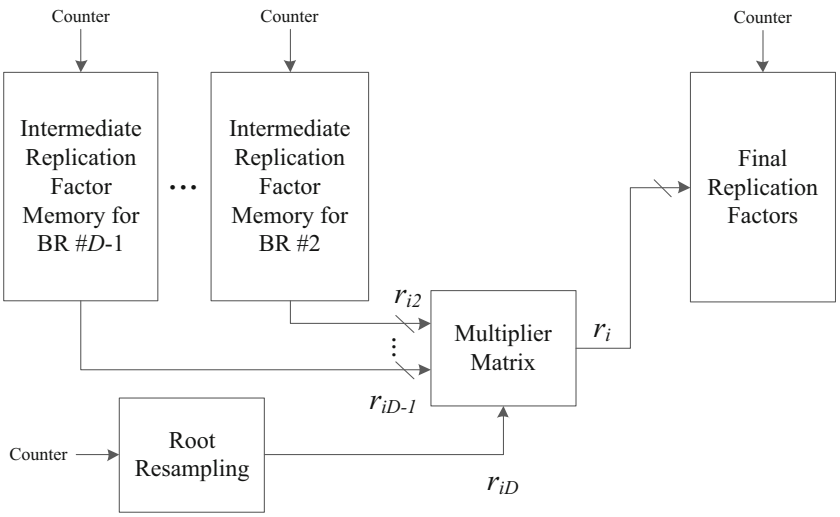



**Figure 6** Architecture of replication factor generation.

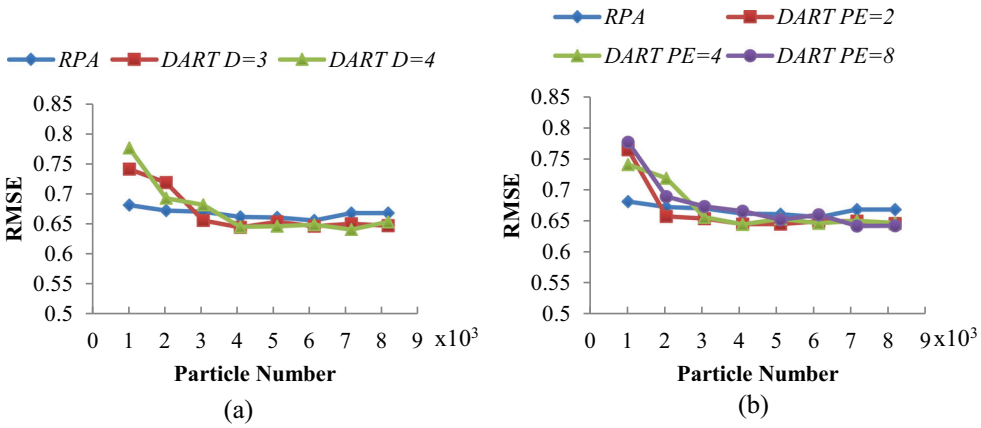



**Figure 7** Filtering performance of RPA and DART with different topology.

maintained when compared to RPA with particle diversity above 40 %. Figure 8 presents the particle diversity, which is the percentage of particles having non-zero replication factors after resampling, for RPA and DART during the simulation. For all presented tree topologies in DART, a particle diversity drop of around 10 % is observed. The diversity will increase as the particle number rises from 1024 to 8192. For the same number of particles with different tree topology, the diversity will decrease for deeper tree topology. The reason for this is that the possibility of a zero final replication factor will increase as the number of intermediate replication factors involved in calculating a final one grows linearly with tree depth. This can be mitigated by increasing branching factors as seen in Fig. 8 that diversity increases together with branching factor (since branching factors are set to the same value of PEs). For a larger branching factor, the number of BR processes will decrease and the influence of BR on final replication factors will also decrease. Consequently, for a given number of particles, the tree topology can be tuned for the desired filtering performance as well as to required hardware implementation resource, which can be found in Section 5.3.

### 5.2 Timing Analysis

Processing timing reduction is the most significant contribution of the algorithm. For $DART(N, K)$ with $RTS(N, D, K^{const})$ implemented with the above described architecture in Section 4, the timing of processing is analyzed to better illustrate how BR is processed in parallel with sampling and weight calculation steps. As presented in Fig. 9, one filtering recursion starts with sampling particles. After an initial latency of $L_S$, weight calculation can be initiated. Also, a latency of $L_W$ is observed before PEs generate the weights of the corresponding particles. Weight calculation for all $N$ particles will continue for $N/K$ cycles.

Meanwhile, as soon as the first few particles with weights are generated, BR on level 1 will start resampling of particles with $f_b^2 = K$. Since BR on level 1 is pipelined by FBRM, it will also consume a total of $N/K$ cycles after the startup latency of $L_{FBR}$. As for BR on level 2, it needs to resample $K$ every $K$ cycles. Without a pipelined structure, the RBRM for level 2 will resample a total number of $N/K$ particles in $N/K$ cycle. An initial weight sum delay of 1 cycle is added to calculate the sum of weights for the BR after the $L_{FBR}$ latency.

In order to make the timing for a whole recursion better visible from Fig. 9, every BR on level 3 through $D$-1 spans a period of $N/K$ cycles. Every time the level goes up by 1, an extra cycle delay is added for calculating the sum of weight. As BR on deeper levels takes place less often, the RBRM on these levels only needs to resample $K$ particles when their weights are fully prepared. The period before BR on these levels actually starts resampling is denoted by the BR IDLE

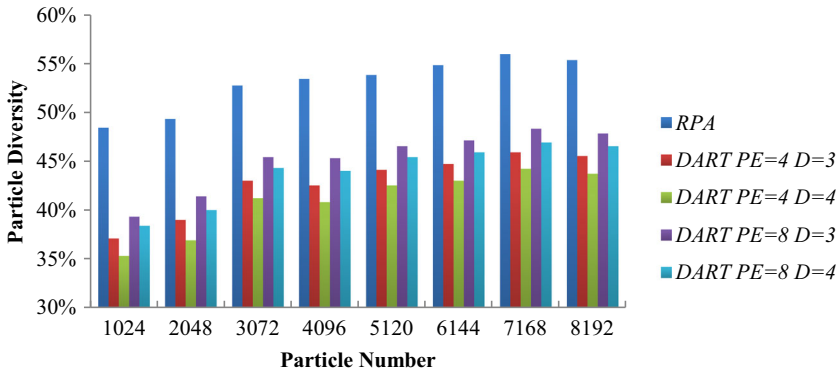



**Figure 8** Particle diversity after resampling in RPA and DART with different topology.

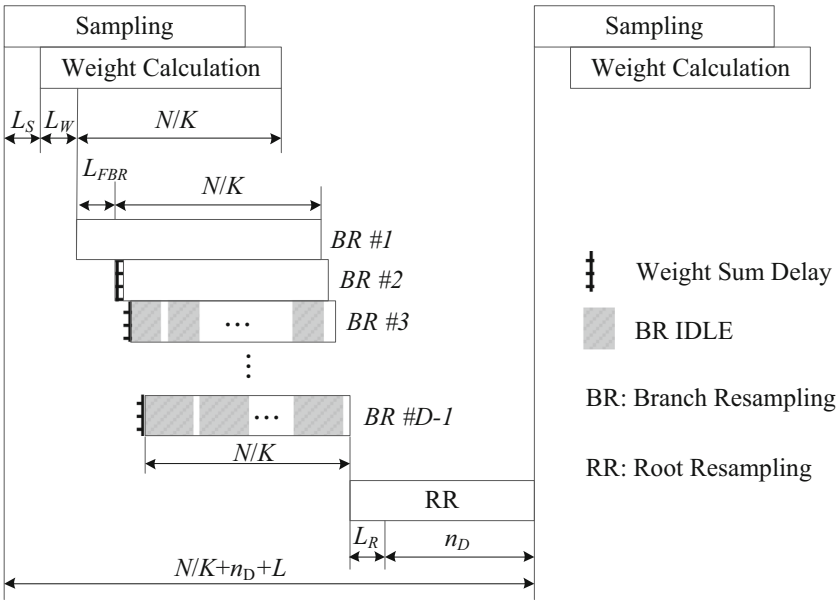



**Figure 9** Timing of DART.

period as in Fig. 9 after the weight sum delay. The IDLE period also represents the time consumed waiting weights to be calculated by previous level of BR.

After BR on level $D$-1 finishes, all collective weight for RR is obtained. As RR only resamples $n_D$ weights, the time needed is $n_D$ together with a startup latency of $L_R$. Finally, the total time for an iteration in DART can be represented by $T$ as expressed in

$$\begin{aligned} T &= L_S + L_W + L_{FBR} + D-1 + N/K + L_R + n_D \\ &= N/K + N/K^{D-1} + L, \end{aligned} \tag{10}$$

where $L$ denotes the total startup latency for all processes together with $D$-1 included since these parts are independent of $N$.

For better comparison with other PF architectures, the time required for one filtering iteration in different distributed PFs [14–17] with $N$ particles and $K$ PEs is shown in Table 2 in analytical form. $L$ in all expressions includes the startup latency and other minor parts which are independent from $N$. From the table we see that distributed PFs proposed in [14–16] share the same feature that final resampling of particles is conducted utilizing all PEs and achieve linear speedup. Nevertheless, they all introduce extra steps after the distributed resampling, which are mainly designed for maintaining filter accuracy by particle redistribution [10-RPA], exchange [10-RNA, 14], and mixing [15]. The extra time needed is denoted with $T_{RPA}$, $T_{RNA}$, $T_{MIX}$ and $T_{EXCHANGE}$, respectively. HR [17] outperforms [14–16] by eliminating the extra process in distributed PFs, and achieves a linear speedup. For DART with $RTS(N, D, K^{const})$, the part $N/K$ is further reduced to $N/K^{D-1}$, which is the number of particles resampled by RR and will decrease exponentially when the tree depth grows linearly.

The speedup achieved is further illustrated in Fig. 10 compared to centralized PF when omitting the constant latency of $L$. HR has a linear speedup proportional to the number of PEs against centralized implementation, which is also the limit of current algorithms. The speedup of RPA and RNA is below the linear boundary due to extra processing time of $T_{RPA}$ and $T_{RNA}$,

**Table 2** Time for one iteration in different PFs architectures.

| Architecture | Time for one iteration |
|---|---|
| RPA [14] | $2N/K + L + T_{RPA}$ |
| RNA with local exchange [14] | $2N/K + L + T_{RNA}$ |
| Distributed resampling with particle mixing [15] | $2N/K + L + T_{MIX}$ |
| Distributed resampling with representative particles exchange [16] | $2N/K + L + T_{EXCHANGE}$ |
| HR [17] | $2N/K + L$ |
| DART with $RTS(N, D, K^{const})$ | $N/K + N/K^{D-1} + L$ |

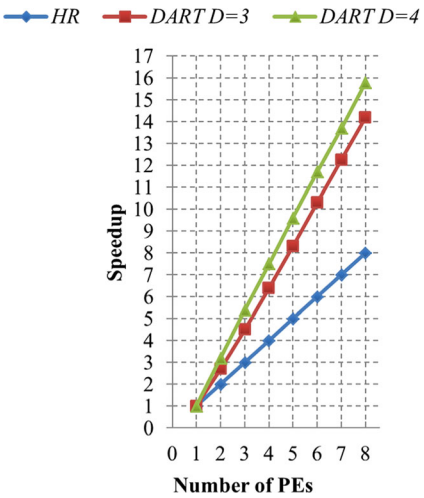



**Figure 10** Speedup of HR and DART compared to centralized PF.

respectively. For DART with $RTS(N, D, K^{const})$, tree depth $D = 3$ and $D = 4$ are presented in the figure. As part of the processing time can be reduced exponentially with increasing resampling tree depth, the overall speedup achieved surpasses the linear boundary and is more noticeable with deeper tree topology.

A prototype of DART is developed on a Xilinx Virtex-IV Pro FPGA (XC4VFX100-12FF1152), which implements a distributed PF under the classical Bearings-Only Tracking (BOT) [18] scenario with $K = 8$ PEs and a total number of $N = 8192$ particles. Each dimension of particle states and particle weights use 18 and 16 bits respectively [14]. The tree depth $D$ is chosen as 3 and all branching factors for BR are set to $K$, so the central RTM module in Fig. 3 implements FBRM, one RBRM and RRM. In [17], the implemented PF needs 2150 clock cycles to complete one iteration with $L = 102$ in HR based architecture. For DART based architecture, the iteration time is 1257 clock cycles with $L = 105$ where $L_S = 5$ cycles, $L_W = 49$ cycles, $L_{FBR} = 12$ cycles and $L_R = 37$ cycles. The part of the final sequential resampling in [17] $N/K = 1024$ cycles is reduced to $N/K^{D-1} = 128$ cycles in DART and a significant overall processing time reduction of 41.5 % is achieved. The prototype is able to support a maximum clock frequency of 94 MHz and given the clock frequency at 80 MHz, the input observation frequency can be up to 63.6 kHz.

### 5.3 Resource Consumption

While achieving significant speedup, two extra things regarding the hardware resource requirement to implement DART should be noted. The first one is the extra memory required for storing intermediate replication factors generated by BR. With the help of replication factor sharing mechanism, the memory needed to store replication factors for $N$ particles is reduced to $N/K$ size in [17], which is denoted by $M$. Since the same mechanism is also employed in DART, the memory used for final replication factors remains the same. Extra memory is used for intermediate replication factors generated by BR on depth 2 through $D$-1. BR on depth 2 needs exactly the same memory $M$ for storing $N/K$ replication factors ($f_b^2 = K$). While for BR on depth 3 to $D$-1, $n_d$ ($3 \le d \le D$-1) replication factors are generated on depth $d$ as analyzed before. The size scales by $1/f_b^{d+1}$ when depth increments from $d$ to $d + 1$. Therefore, as shown in

$$n_{d+1} = n_d / f_b^{d+1} \quad (2 \le d \le D-1) \tag{11}$$

and

$$lim_{D\to\infty} \sum_{i=2}^{D-1} n_i + M \; < \; M * lim_{D\to\infty} \sum_{i=2}^{D-1} \left(\frac{1}{2}\right)^{i-2} + M \; = \; 3M, \tag{12}$$

**Table 3** Resource consumption for centralized, HR and RT based PF.

| Resource | Centralized | HR [17] | RT |
|---|---|---|---|
| Slices | 20008 | 20731 | 21444(50.84 %) |
| Slice flip flops | 30360 | 30008 | 30438(36.08 %) |
| 4-input LUTs | 35344 | 36667 | 37855(44.88 %) |
| Block RAMs | 80 | 52 | 53(14.10 %) |
| DSP48s | 72 | 58 | 66(41.25 %) |

the total memory required to store replication factors in DART will not exceed 3 $M$ as the minimum value of any branching factor is 2.

Another resource requirement is the multiplier needed in the multiplication matrix in Replication Factor Generation. It is noted that the number of multipliers needed in the matrix is $N_m$, equals to $\prod_{j=3}^{D} f_b^j$, in order to calculate $N_m$ final replication factors simultaneously. So, the branching factor and tree depth should be chosen with care to avoid excessive hardware resource consumption as $N_m$ rises significantly when $D$ or $f_b^d$ ($3 \le d \le D$) increases linearly.

The resource requirements of the prototype implementation with 8192 particles is also presented and compared to other works shown in Table 3, where the centralized column are data 8 times the resource of a centralized particle filter with 1024 particles, which corresponds to the system running 8 centralized PFs simultaneously, keeping the same 8192 particles in total. The percentage in brackets under RT column is the utilization of the FPGA chip. Compared to HR [17], the prototype requires slightly more hardware resources and control logic for execution of extra BR and generating final replication factors during RR. As a result, RT consumes slightly more slices, slice flip flops and 4-input LUTs than the other two. While regarding the requirement for block RAMs, one more block RAM is needed to store the intermediate replication factors generated by BR process. Also increased with respect to HR is the DSP48 component used for generating the final replication factors in the multiplier matrix where $N_m = \prod_{j=3}^{D} f_b^j = 8$ with $D = 3$ and $f_b^3 = 8$. Still, the block RAMs and DSP48s consumption are less than centralized implementation since replication factor sharing mechanism among PEs is also utilized the same with HR [17].

## 6 Conclusion & Future Work

This paper proposes a novel distributed Particle Filter algorithm with Resampling Tree Scheme, which

parallelize Branch Resampling of sub-groups of particles with sampling and weight calculation steps. The final sequential implemented Root Resampling has much fewer particles to handle. Specifically, the number of particles in Root Resampling, as well as the processing time, could decrease exponentially with the depth of the resampling tree when compared to sequential centralized PFs. Therefore, with the same linear speedup in sampling and weight calculation procedures, the overall speedup of DART surpasses linear boundary and outperforms state-of-art distributed PFs.

The algorithm also has high flexibility as featured in the resampling tree topology. Moreover, the proposed hardware architecture possesses favorable features of deterministic execution, hardware efficiency and scalability. The case study of filtering performance in Bearings-Only Tracking scenario proves its lossless performance compared to standard resampling with larger particle numbers.

Future work will be implementing the algorithm under a more application specific context of indoor localization based on fusion of information coming from different sensors (inertial sensors and radio frequency signals) to realize a real-time system for accurate positioning and tracking of a moving object using particle filters.

**Acknowledgments** This research is supported by the National Natural Science Foundation of China (61204030, 61302129), Zhejiang Provincial Natural Science Foundation of China (LY15F020008), Zhejiang Provincial Nonprofit Technology Research Projects (2014C31045) and China Scholarship Council.

## References

1. Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F Radar and Signal Processing, 140*(2), 107–113.
2. Sankaranarayanan, A. C., Srivastava, A., & Chellappa, R. (2008). Algorithmic and architectural optimizations for computationally efficient particle filtering. *IEEE Transactions on Image Processing, 17*(5), 737–748.
3. Gentner C., Munoz E., Khider M., Staudinger E., Sand S., & Dammann A. (2012). Particle filter based positioning with 3GPP-LTE in indoor environments. *Position Location and Navigation Symposium (PLANS)*, IEEE/ION, 301–308.
4. Hongjun, Z., & Sakane, S. (2008). Sensor planning for mobile robot localization-a hierarchical approach using a Bayesian network and a particle filter. *IEEE Transactions on Robotics, 24*(2), 481–487.
5. Kai, W., Yun-Hui, L., & Luyang, L. (2014). A simple and parallel algorithm for real-time robot localization by fusing monocular vision and odometry/AHRS sensors. *IEEE/ASME Transactions on Mechatronics, 19*(4), 1447–1457.
6. Sz-Pin H., Jun-Wei Q., Chi-Chung L., & Yu-Chee T. (2014). Wearable localization by particle filter with the assistance of inertial and visual sensors. *11th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 52–57.
7. Arshad, I., Syed, W. S., & Shamin, K. (2014). Non-linear moving target tracking: a particle filter approach. *International Journal of Computer and Communication System Engineering, 1*(1), 20–26.
8. Tian, Q., Salcic, Z., Wang, K. I., & Pan, Y. (2015). A hybrid indoor localization and navigation system with map matching for pedestrians using smartphones. *Sensors, 2015*, 30759–30783.
9. Putta R., Misra M., & Kapoor D. (2015). Smartphone based Indoor tracking using magnetic and indoor maps. *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 I.E. Tenth International Conference on*, pp. 1–6.
10. Qian J., Ma J., Ying R., Liu P., & Pei L. (2013). An improved indoor localization method using smartphone inertial sensors. *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, pp. 1–7.
11. Bao, H., & Wong, W. (2014). A novel map-based dead-reckoning algorithm for indoor localization. *Sensors, 3*, 44–63.
12. Valentin R., & Mahesh K. M. (2013). HiMLoc: indoor smartphone localization via activity aware pedestrian dead reckoning with selective crowdsourced WiFi fingerprinting. *International Conference on Indoor Positioning and Indoor Navigation*.
13. Yuan Y., Yubin Z., & Kyas M. (2014). GeoF: a geometric Bayesian filter for indoor position tracking in mixed LOS/NLOS conditions. *11th Workshop on Positioning, Navigation and Communication (WPNC)*, 1–6, 12–13.
14. Bolic, M., Djuric, P. M., & Hong, S. (2005). Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing, 53*(7), 2442–2450.
15. Zhang Y., Sathyan T., Hedley M., Leong P. H. W., & Pasha A. (2012). Hardware efficient parallel particle filter for tracking in wireless networks. *2012 I.E. 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC),* pp. 1734–1739.
16. Chitchian, M., Simonetto, A., Amesfoort, A. S., & Keviczky, T. (2013). Distributed computation particle filters on GPU architectures for real-time control applications. *IEEE Transactions on Control Systems Technology, 21*(6), 2224–2238.
17. Pan, Y., Zheng, N., Tian, Q., Yan, X., & Huan, R. (2013). Hierarchical resampling algorithm and architecture for distributed particle filters. *Journal of Signal Processing Systems, 71*(3), 237–246.
18. Carpenter, J., Clifford, P., & Fearnhead, P. (1999). Improved particle filter for nonlinear problems. *IEE Proceedings of Radar, Sonar and Navigation, 146*(1), 2–7.
19. Kerem P., & Oguz T. (2011). Parallelization of particle filter based localization and map matching algorithm on multicore/manycore architectures. *IEEE Intelligent Vehicles Symposium*, pp. 820–826.
20. Xu Y., Liu J., Ma L., & Peng L. (2010). WLAN indoor tracking method via improved particle filter algorithm. *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on*, pp. 1078–1082.
21. Bolic M., Hong S., & Djuric P. M. (2002). Finite precision effect on performance and complexity of particle filters for bearing-only tracking. *Conference Record of the 36th Asilomar Conference on Signals, Systems and Computers,* vol. 1. pp. 838–842.

**Qinglin Tian** received his B.S. degree in electrical engineering from Zhejiang University, China, in June 2011. Since then he has been pursuing the Ph.D. degree in Department of VLSI Design at College of Electrical Engineering, Zhejiang University, China. He was a visiting scholar at University of Auckland, New Zealand from 2014 to 2015. His research interests include design and implementation of indoor tracking and navigation systems, FPGA-based optimization and acceleration of signal processing algorithms in related area, computer architecture and hardware-software co-design.

**Yun Pan** received his B.S. degree in Department of Information Science & Electronic Engineering, Zhejiang University, China, in 2002, and Ph.D. degree in Department of Electronic Engineering, Tsinghua University, China, in 2008. He joined Institute of VLSI Design, College of Electrical Engineering, Zhejiang University in 2008 as a post-doctor. He is currently an Associate Professor in College of Information Science & Electronic Engineering, Zhejiang University. His current research interests include on-chip communication, mobile computing, application-specific heterogeneous architecture design, mobile embedded systems and healthcare micro systems. He has published more than 40 academic papers, coauthored 2 books and 1 chapter, and held more than 10 Chinese patents in these areas.

**Zoran Salcic** (SM IEEE) holds a chair in computer systems engineering at the University of Auckland. He has the BE (‘72), ME (‘74) and PhD (‘76) degree in electrical and computer engineering (Sarajevo University). His main research interests include complex digital systems, custom-computing machines, embedded systems and their implementation, design automation tools, hardware-software co-design, models of computation and languages for concurrent and distributed systems, and cyber-physical systems. He has published more than 300 peer-reviewed journal and conference papers and several books. He is a Fellow of the Royal Society New Zealand and recipient of Alexander von Humboldt Research Award in 2010.

**Ruohong Huan** received her B.S. degree in Department of Information Science and Electronic Engineering, Zhejiang University, China, in 2002, her M.S. degree in Department of Information Science and Electronic Engineering, Zhejiang University, China, in 2005 and her Ph.D. degree in Institute of Electronics, Chinese Academy of Sciences, China, in 2008. She is currently an Associate Professor in College of Computer Science and Technology, Zhejiang University of Technology, China. Her current research interests include image processing and target recognition, video processing and human behavior recognition. She has published over 30 academic papers in journals or proceedings.