Journal of Real-Time Image Processing

Scalable Implementation of Particle Filter based Visual Object Tracking on Network-on-chip (NoC)

--Manuscript Draft--

Manuscript Number:	RTIP-D-18-00057
Full Title:	Scalable Implementation of Particle Filter based Visual Object Tracking on Network-on-chip (NoC)
Article Type:	Original Research Paper
Section/Category:	2 Hardware architectural and implementation issues to achieve real-time throughput (e.g., FPGA, DSP)
Keywords:	Particle filter; Object tracking; FPGA; Network on Chip (NoC); Real-time image processing; Scalability; Low power; Smart Camera
Abstract:	Particle filter algorithms have been successfully used in various visual object tracking applications. They handle non-linear model and non-Gaussian noise but are computationally demanding. In this paper, we propose a scalable implementation of particle filter algorithm for visual object tracking, using scalable interconnect such as network on chip (NoC) on a FPGA platform. Here, several processing elements execute parallelly to handle large number of particles. We propose two designs and implementations, with one optimized for speed and other optimized for area. These implementations can easily support different image sizes, object sizes and number of particles, without modifying the complete architecture. Multi-target tracking is also demonstrated for four objects. We validated the particle filter based visual tracking with video feed from a Petalinux based system. With image size of 320x240, frame rates of 348 fps and 310 fps were achieved for single object tracking of size 17x17 and 33x33 pixels, respectively, with a reasonable low power consumption of 1.7 mW/fps on Zynq XC7Z020 (Zedboard) with an operating frequency of 69 MHz. This makes our implementation a good candidate for low-power, visual object tracking using FPGA, especially in low-power, smart camera applications.

Author Biographies

Pinalkumar Engineer is Research scholar in Electrical engineering department at Indian Institute of Technology (IIT) Bombay, India. He is also working as Assistant Professor in Electronics engineering department at S.V. National Institute of Technology (SVNIT), Surat since 2007. He holds a master degree in electrical engineering with specialization in Microprocessor systems and applications (M. S. University of Baroda, 2002). His research interests include embedded systems, high performance computing architectures, reconfigurable computing and real-time image processing.

Rajbabu Velmurugan is an associate professor in the Department of Electrical Engineering, Indian Institute of Technology Bombay. He received his Ph.D. in electrical and computer engineering from Georgia Inst. of Tech., USA in 2007. He was a graduate engineer trainee at L&T, India from 1995 to 1996 and a software engineer at The MathWorks, USA from 1998 to 2001. He joined IIT Bombay in 2007. His research interests are broadly in signal processing, including image, video, speech, and audio processing. His current research focus includes inverse problems with application in image and audio processing, such as blind deconvolution and source separation. He also works on low-level image processing and visual tracking. Another of his research interest is in front-end of processing of speech signals for enhancement using multi-microphone arrays. In addition he is interested in developing efficient hardware systems for signal processing applications.

Sachin Patkar is a professor in the Department of Electrical Engineering, Indian Institute of Technology Bombay. He received the B.Tech., M.Tech. and Ph.D. degree from Department of Computer Science, Indian Institute of Technology (IIT), Bombay, India (1986), IIT Madras (1987) and IIT Bombay (1992) respectively. He was a fellow of Alexander von Humboldt Foundation, at Research Institute of Discrete Mathematics, University of Bonn, Germany, between 1993 and 1994. He joined the Department of Mathematics at IIT Bombay, and subsequently Department of Electrical Engineering, where he serves as a Professor, and heads the High-Performance Computing lab. He has published more than 70 technical papers. His current research emphasis is on System Design, Combinatorial optimization, High Performance Computing, and Algorithms Design and Analysis.

Journal of Real-time Image Processing manuscript No.
(will be inserted by the editor)

Pinalkumar Engineer · Rajbabu Velmurugan · Sachin Patkar

Scalable Implementation of Particle Filter based Visual Object Tracking on Network-on-chip(NoC)

Received: date / Revised: date

Abstract Particle filter algorithms have been successfully used in various visual object tracking applications. They handle non-linear model and non-Gaussian noise but are computationally demanding. In this paper, we propose a scalable implementation of particle filter algorithm for visual object tracking, using scalable interconnect such as network on chip (NoC) on a FPGA platform. Here, several processing elements execute parallelly to handle large number of particles. We propose two designs and implementations, with one optimized for speed and other optimized for area. These implementations can easily support different image sizes, object sizes and number of particles, without modifying the complete architecture. Multi-target tracking is also demonstrated for four objects. We validated the particle filter based visual tracking with video feed from a Petalinux based system.

With image size of 320×240, frame rates of 348 fps and 310 fps were achieved for single object tracking of size 17×17 and 33×33 pixels, respectively, with a reasonable low power consumption of 1.7 mW/fps on Zynq XC7Z020 (Zedboard) with an operating frequency of 69 MHz. This makes our implementation a good candidate for low-power, visual object tracking using FPGA, especially in low-power, smart camera applications.

Keywords Particle filter; Object tracking; FPGA; Network on Chip (NoC); Real-time image processing; Scalability; Low power; Smart Camera

Pinalkumar Engineer, Rajbabu Velmurugan, Sachin Patkar
Electrical Engineering Department,
Indian Institute of Technology Bombay,
Mumbai, INDIA
E-mail: pje@ee.iitb.ac.in

Rajbabu Velmurugan
E-mail: rajbabu@ee.iitb.ac.in

Sachin Patkar
E-mail: patkar@ee.iitb.ac.in

1 Introduction

Object tracking is an important aspect of several computer vision applications such as robotic control or visual surveillance. It estimates object positions across frames in a video. A survey of popular object tracking algorithms can be found in [37, 39]. One successful approach is to pose this problem as a state estimation problem and use a Bayesian framework such as the particle filter (PF) algorithm [4, 20, 23, 36, 41].

Particle filter is an efficient algorithm to handle non-Gaussian noise and non-linearity in state transition or measurement model. Particle filter algorithm represents the distributions in Bayesian estimation with a set of samples called particles and their corresponding weights. PF based visual tracking was first proposed in [15]. A color based particle filter tracker was proposed in [26] and was shown to outperform the mean-shift tracker in terms of reliability, at the price of increased computational time.

One main concern for real-time particle filter (PF) based visual tracking is to obtain results in a reasonable computational time with desired accuracy. This is only possible with a hardware realization [6, 18]. Furthermore, computation time can be drastically reduced by exploiting parallelization, which is essential for many PF based applications requiring large number of particles and high dimensional state space. Most stages in particle filter algorithm, are highly amenable to parallelization and hence appropriate for parallel implementation, where simultaneous operations can be performed on many samples (particles). In hardware realizations, another aspect of importance is scalability to accommodate more particles or object size changes or varying number of targets or processing elements, depending on the objective without modifying the complete architecture. Our objective in this work is to handle both parallelization and scalability, to achieve a computationally efficient particle filter framework for visual tracking of multiple targets. Among the various parallel computing platforms (like multicore processors, computer clusters, GPUs and field programmable gate arrays (FPGAs)) available to realize real-time systems, we consider FPGAs in this work.

The proposed design considers particle filters as a network of simple processing elements and realized on a Network on Chip (NoC). The ensuing algorithm depends on local computations in the processing elements and related communication. Our framework has homogeneous processing elements (PEs) connected to the interfaces of NoC with varying scale of the system.

Our contributions are as follows:

– We propose a scalable implementation using network on chip (NoC) for particle filter based visual tracking of single object and varying object sizes
– This framework is extended to handle multi-object tracking demonstrating the scalability of the architecture
– The architecture is optimized for low-power, enabling embedded smart camera operation

We provide a brief review of color based particle filter for visual tracking in Section 2 and related work in Section 3.Proposed implementation of particle filter based object tracking on NoC is detailed in Section 4. Two design options for implementation on NoC are presented in Section 5, followed by scalability analysis in Section 6. Simulation results and hardware validation for the proposed design are shown in Section 7. Hardware validation of single/multiple object tracking is also demonstrated in Section 7.2.

2 Visual object tracking using particle filter

2.1 Particle filter (PF) algorithm

Particle filter algorithm is used to solve a Bayesian estimation problem by modelling the state x_k at time instant k and the observation z_k as two stochastic processes. Under Markovian assumptions (i.e., given the observations $z_{1:k-1}$, the current state x_k depends only on x_{k-1}), the observation equation is

$$z_k = g_k(x_k,\ \nu_k), \tag{1}$$

and the state equation describing the dynamics of x_k is

$$x_k = f_k(x_{k-1},\ \omega_{k-1}), \tag{2}$$

where $\{\omega_k\}_{k=1,\ldots.}$ and $\{\nu_k\}_{k=1,\ldots.}$ are independent and identically distributed process and measurement noise, respectively. When f_k and g_k are non-linear functions, we can consider a solution based on a Monte Carlo approximation. The densities $p_{k|k}(x_k \mid z_{1:k})$ are approximated using N particles, as a sum of N delta functions (the particles) centered at $\left\{x_k^{(i)}\right\}_{i=1}^{N}$

$$p_{k|k}(x_k \mid z_{1:k}) \approx \sum_{i=1}^{N} \omega_k^{(i)} \delta(x_k - x_k^{(i)}), \tag{3}$$

where the function $\delta(x)$ is the Kronecker delta and $\left\{\omega_k^{(i)}\right\}_{i=1}^{N}$ are the weights associated with the particles $i = 1, \ldots, N$ and defined as

$$\omega_k^{(i)} = \frac{p_{k|k}(x_k^{(i)} \mid z_{1:k})}{q_k(x_k^{(i)} \mid z_{1:k})} \ i = 1, \ldots, N. \tag{4}$$

$q_k(.)$ is the importance density function defined as the density that generated the current set of particles. From (4), the recursive formulation to propagate the particle weights is given as

$$\omega_k^{(i)} \propto \frac{p_k(z_k \mid x_k^{(i)}) p_{k|k-1}(x_k \mid z_{1:k-1})}{q_k(x_k^{(i)} \mid z_{1:k})} \tag{5}$$

In the algorithm for visual target tracking [15], the particles are drawn from the predicted prior, i.e.,

$$q_k(x_k \mid z_{1:k}) = p_{k|k-1}(x_k \mid z_{1:k-1}). \tag{6}$$

This simplifies the weight computation (Eq. 5) to

$$\omega_k^{(i)} \propto p_k(z_k \mid x_k^{(i)}), \tag{7}$$

where the weights are now proportional to the likelihood. This estimation based on Monte Carlo approximation and recursive Bayes equations is known as the particle filter (PF).

2.2 Color based particle filter for visual tracking

The steps in particle filter based visual tracking can be summarised as follows:

– **Prediction** for each particle according to dynamic model (according to Equation 6)
– **Update** the weight of each particle according to likelihood model (according to Equation 7)
– **Estimate** the posterior state of the target given the new frame (according to Equation 3)

2.2.1 Models

For visual tracking, various models have been used to represent target's motion and its relation to the measurements or observations. Target state, corresponding to x_k in Equation 1, is represented as the center of region of interest (ROI) in the spatial domain, $S = (x, y)$. We can define the dynamic model, which predicts the target state for next observation as

$$S_{t+1} = A\ S_t + \eta_t,\ \ t = 0, 1, 2, \ldots \tag{8}$$

where, A is the state update model, S_t is target state of current observation and η_t is Gaussian noise. Initial target state is represented as S_0. We assume a random walk model. Other models that include features such as velocity, acceleration, scale and rotation can be considered for more accurate tracking.

Observation model

In case of color based particle filter, the color features (histograms) of the target object can be considered as measurements. After normalisation, this histogram provides the color distribution in region of interest (ROI). In computing color distribution, pixels near the center are given more weight compared to the pixels farther from the center, which is helpful to handle occlusion at the border of the target object in the image. This is achieved by using a kernel $k(.)$ of the form,

$$k(r_j) = \begin{cases} 1 - r_j^2, & r_j \leq 1 \\ 0, & otherwise \end{cases} \tag{9}$$

where r_j is the normalized distance from the target center $c = \{C_x, C_y\}$ to actual pixel location v_j. Assuming color distribution of the ROI, n-bin histogram with center at pixel location c be computed as:

$$p(c)^u = C \sum_{j=1}^{R_c} k(r_j)\delta[b(v_j) - u], \; u = 1, \ldots, n \tag{10}$$

where R_c is the number of pixels in the ROI, C is a normalization constant so that $\sum p(c)^u = 1$, $k(.)$ is the monotonically decreasing kernel in (9), and $b(v_j)$ is a function that assigns color histogram bin to pixel located at v_j. Reference histogram (p_0) (for the first frame) and candidate histograms ($p(c)$) (for consecutive frames) are calculated using (10).

Likelihood model

A similarity measure (likelihood) is obtained by calculating the distance between candidate histograms and the reference histogram of the target object. The distance can be calculated as

$$d\left[p(c), p_0\right] = \sqrt{1 - \rho(p(c), p_0)} \tag{11}$$

where

$$\rho(p(c), p_0) = \sum_{u=1}^{n} \sqrt{p(c)^u p_0^u} \tag{12}$$

is the Bhattacharyya coefficient between the candidate color histogram $p(c)$ and the reference color histogram p_0. One common approach is to use the Bhattacharyya coefficient as a measure of the distance between the color distributions [7].

Estimation of target center

The target center (state) is estimated either by taking the mode or mean of the aposteriori estimate. Here we compute the mean over the weighted particles as

$$E\left[x_k \mid z_{1:k}\right] \approx \frac{1}{N} \sum_{i=1}^{N} \omega_k^{(i)} x_k^{(i)}, \tag{13}$$

where $\omega_k^{(i)}$ is obtained using Equation 12 (proportional to likelihood).

3 Related Work

Implementation of particle filters has been considered at various levels, starting from algorithmic to architecture level. Various scalable implementation of particle filter based object tracking on GPU platforms are proposed in [4, 21, 22, 35]. It has been observed in [11] that energy consumption and processing performance on a FPGA platform is comparatively better than multi-threaded CPU and GPU implementation, which makes FPGA a favorable platform for low power implementation. Since, this work is on hardware for particle filter, we briefly review hardware related work that use particle filter for visual tracking.

3.1 Particle filter framework using HW/SW on FPGA

Particle filter frameworks considering HW/SW design aspects on FPGA was proposed in [1, 9, 29, 32, 33] and object tracking has been implemented on FPGA platform [1, 29, 32, 33]. In [33], a hardware/software co-design based system is presented for grid based Fast-SLAM algorithm. Different approaches (varying from HW/SW co-design to fully HW design) are considered in [33] for particle filter acceleration on a FPGA platform. In [29], a parameterizable framework for 1-D and 2-D particle filter was implemented using processing elements (PEs) with memory management being their main focus. Optimized hardware architecture for particle filter based object tracking in gray scale images has been proposed in [1], where separate modules were implemented for particle generation, weight calculation, output calculation and resampling. In [9], parameterizable framework for color based object tracking algorithm using particle filters was implemented on a FPGA platform. By adjusting configuration parameters (such as number of particles, object size), the implementation can be ported for several smart camera applications.

3.2 Particle filter framework using CPU/FPGA

Various implementations of particle filter using CPU/FPGA are discussed in [13, 14, 19, 34]. In [13], a multithreaded framework for particle filter implementation on a hybrid CPU/FPGA platform has been proposed. Computational tasks in visual object tracking, are partitioned into hardware and software threads using reconfigurable operating system (ReconOS) in [19]. Runtime partitioning is one of the important features of this framework and this has been utilised in [14] to suggest various adaptive repartitioning approaches. Hardware implementation of particle filter based object tracking for PTZ camera is proposed in [34]. Histogram difference and sum-of-absolute-difference (SAD) is used as a feature for object tracking, achieving 116 fps.

3.3 Particle filter on network on chip (NoC)

Several image processing algorithms have been implemented on NoC based architecture [12, 16, 30, 38, 40]. On-line reconfigurable, distributed particle filters, using tree-based network on chip (NoC) is presented in [38]. Two distinct reconfiguration techniques were used for intra- and inter-resampling steps of particle filter algorithm to reduce the overall data traffic in four-ary tree based NoC implementation. Additionally, online reconfiguration in form of regrouping PEs into different clusters was also proposed in [38]. However, it's functionality is limited to resampling operation and scalability is still restricted to single FPGA.

4 Proposed implementation of particle filter based object tracking on NoC platform

As described in Section 3.1, different design options are explored for particle filtering applications on FPGA based parameterizable framework [1, 9, 29, 33]. But, they are restricted upto eight processing elements and architecture is limited to a single FPGA. Multithreaded hardware/software approach [14, 19] described in Section 3.2 allows flexible re-partitioning of hardware and software, which gives average performance limited to 7.18 fps with 100 particles. These approaches can be further improved by a more flexible framework based on hardware for assured performance. Goal of most existing implementation approaches is to design a particle filter framework for object tracking on various platforms. They have been designed for varying number of parameters (number of particles, features, target/object tracking). Few implementations [5, 13, 21, 29] targeted a scalable design of particle filter for GPU and FPGA platforms. But scalability issues have not been discussed in terms of object size, image size, number of particles and number of objects. To handle scalability issues of particle filter based visual object tracking algorithm, we propose an implementation based on a scalable NoC platform, that is to be realized on FPGA to track single or mutiple targets.

4.1 NoC as a scalable platform

As discussed, FPGAs have become a tangible implementation platform due to their low power consumption [11]. In FPGA based system design, custom computing modules/elements are designed and connected through point-to-point links or through system level interconnects (mostly shared or dedicated bus) along with external memory and other I/O devices. When system size grows, point-to-point communication architecture using exclusive connections do not provide scalable implementation, due to high implementation complexity. As an alternative, bus-based architecture can link a few hardware blocks or IP cores in an efficient way to reduce complexity in the design. But, they lack scalability when the communicating processing elements (PEs) are more, especially when PEs are executing in parallel.

One promising solution to tackle scalability with parallel computation on PEs is NoC, where communication between various PEs is managed by routing data packets through short links, routers and standard interfaces. NoC is described as a scalable interconnect for system-on-chip [2]. With a NoC, each computing module is made to run at its own rate. Thus, NoC provides flexibility to the designer by partitioning the hardware into stand-alone blocks, which improves predictability aspects along with adaptability according to algorithm modifications. As number of cores increases in the implementation, network bandwidth increases to provide a scalable communication platform.

The proposed FPGA implementation of particle filter based visual object tracking algorithm will benefit from the flexibility and rapid prototyping aspect of NoC. This implementation can support varying image sizes, object sizes and any number of particles and requires less modification in realizing a desired system with minimal intervention. We investigate the computation and communication demands (Section 5), while mapping various processing modules of the algorithm.

4.2 Particle filter based visual object tracking

Various steps in our implementation for particle filter based visual tracking are listed below.

- Read the first frame. The center $C_1 = (C_x, C_y)$ of the region of interest (ROI) for the object to be tracked is initially provided by the user. ROI of $(w \times h)$ pixels is considered around the given center C_1.
- *For first frame*
  - **Initialize**
    - Calculate 8-bin RGB reference histogram (q_u) (observation model mentioned in Section 2.2.1) for the selected ROI in first frame
- *For frames k: 2 to n*
  - **Prediction**

  - Generate samples $\{x_k^i\}_{i=1,..N}$ using Gaussian distribution with estimated center (initial center for first frame) as the mean and variance as 20 pixels.
- **Update**
  - Calculate 8-bin RGB histogram (p_i, where $i = 1, ..N$) for ROI around each particle
  - Calculate weights (Bhattacharyya coefficients) $\{w_k^i\}_{i=1,..N}$ using (12)
- **Estimate**
  - Multiply each Bhattacharyya coefficient (w_k^i) with the respective center values (x_k^i) to obtain $B[i] = w_k^i \times x_k^i$.
  - Calculate new center $NewC = \left\{ \frac{\sum_{i=1}^{N} B[i]}{\sum_{i=1}^{N} w_k^i} \right\})$

The particle filter algorithm, primarily comprises of these three steps (i) prediction (sampling stage) (ii) update (calculating importance weights), and (iii) estimate. In the context of implementation on a NoC, the first two steps can be implemented independently for each particle. Coordination and communication between each particles is required for calculating the single state estimate.

4.3 Proposed NoC based implementation

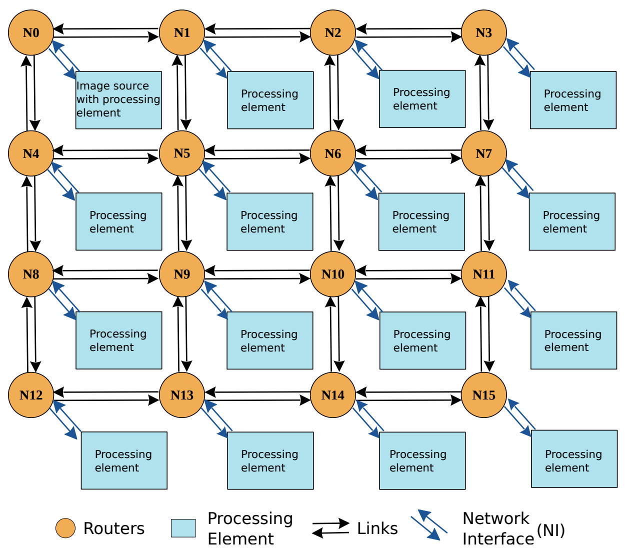


Fig. 1 Proposed 4×4 NoC network using CONNECT [25] for visual object tracking algorithm, primarily made up of sixteen processing elements (PEs) arranged in a Mesh topology. It is built by sixteen routers, each consisting of maximum five IN/OUT ports. The four IN/OUT ports of router connected with the other routers port for providing the routing to data transmission. The fifth IN/OUT port of the router is connected to the PE node.

For our implementation, we have used a freely available web-based synthesizable RTL generator for custom network-on-chip (NoC), named CONNECT (Configurable Network Creation Tool) [25]. CONNECT can be used for generating a NoC of arbitrary topology and supports a large variety of router and network configurations. Also, CONNECT incorporates a number of useful features fine-tuned for the FPGA platform. The proposed NoC implementation for PF based visual object tracking is shown in Figure 1. Here, each processing element (PE) is connected to network interface (NI) of a local router. Among 16 PEs connected to 4×4 NoC network, PE at node *N0* is considered as master (root) node of the network. Image source has been connected with the master node of the NoC to read the image pixel value from the image source module. The master node is used to transmit the image data (such as ROI block for computing histogram) to remaining 15 PEs (*N1,. . . ,N15* represented as *N#*) of the network.

For inter-node communication, packet containing information (flit), is sent from source node to NI of the local router. The flit is then sent to destination node via links connected between routers. Format of flit is shown in Figure 2. In flit, *valid* bit indicates that flit to

Valid (1)	Tail (1)	Destination Port (4)	VC (1)	Data (#N)

Fig. 2 Flit structure for CONNECT NoC.

be transmitted/received from port is valid and *VC* bit indicates that the virtual channel is enabled or disabled. Flit consists of information about the data transmitted from source to destination port. The flit is buffered and then sent to the corresponding port for computing.

4.4 Processing element realization and interfacing to NoC

Considering a PE is to be connected to the network interface in NoC, it can have a generalized structure as shown in Figure 3 [17]. It consists of three modules: *Data collector*, *Data processing* and *Data distributor*. *Data collector* and *Data distributor* modules are responsible for making

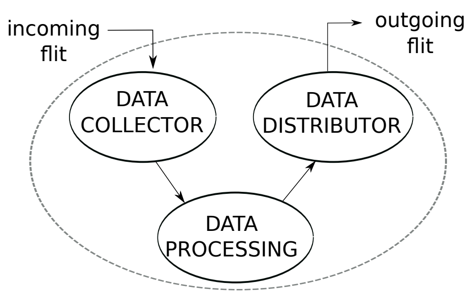


Fig. 3 General structure of processing element (PE) connected to NoC [17].

the external communication in PE. Flit (packet) data coming from router is received by *Data Collector* module. When all data are received, a *start* signal is sent to *Data processing* module. After completing the computation, the *Data processing* module emits *done* signal. Once the result is ready, it can be sent to network by the *Data distributor* module.

5 Implementation planning on NoC

There are different implementation possibilities of particle filter based visual object tracking algorithm on a 4×4 NoC network. For exploiting parallelism, computation of histogram and Bhattacharyya coefficient can be carried out at all 16 PEs and center estimation can be done at master node. Other design option for area optimization, may include only histogram computation at PEs and computation of Bhattacharyya coefficient and center estimation can be done at master node. There are other possibilities of mapping computation to different PEs. One of the possibility, is to design heterogeneous nodes at different location of PEs, where few PEs have capability to compute both histogram and Bhattacharyya coefficient and rest of PEs only computes histogram. In our implementation, we choose designs with homogeneous computing at every PEs. In this paper, we present two designs: (1) optimized for speed and (2) optimized for area. These NoC based implementations can be adapted to other design options with minimal effort.

5.1 Design optimized for speed (Design I)

In the first design, we achieve speed optimization by computing histogram and Bhattacharyya coefficient at all 16 PEs and center estimation is done at master node.The flowchart of this design for visual object tracking using particle filter on NoC, in terms of computation and communication, is shown in Figure 4. As shown, the reference histogram is computed in master node (*N0*) and stored in local memory for the first frame. This reference histogram is sent to all other processing nodes. For consecutive frames, the ROI blocks ($w \times h$ pixels) are extracted from the image ($W \times H$ pixels) obtained from the image source at the master node (*N0*). The ROI blocks ($w \times h$ pixels) are transmitted to all processing nodes for computation of the histogram and Bhattacharyya coefficient with respect to the reference histogram. Computation at processing elements (*N0* and *N#*) starts when all ROI pixels are stored in local memory. After performing computation, Bhattacharyya coefficient is sent to the master node by all 15 processing nodes. As 16 processing elements are available, computation of histogram and Bhattacharyya coefficient are performed in sets of 16 particles, until calculation for all N particles is complete. After receiving all N Bhattacharyya coefficient values,

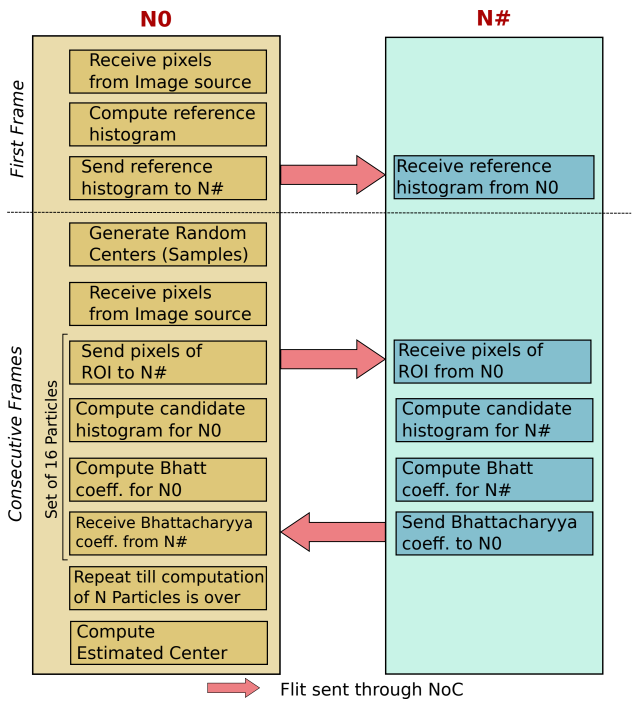

N0
N#
First Frame
Receive pixels from Image source
Compute reference histogram
Send reference histogram to N#
Receive reference histogram from N0
Consecutive Frames
Generate Random Centers (Samples)
Receive pixels from Image source
Set of 16 Particles
Send pixels of ROI to N#
Receive pixels of ROI from N0
Compute candidate histogram for N0
Compute candidate histogram for N#
Compute Bhatt coeff. for N0
Compute Bhatt coeff. for N#
Receive Bhattacharyya coeff. from N#
Send Bhattacharyya coeff. to N0
Repeat till computation of N Particles is over
Compute Estimated Center
Flit sent through NoC

Fig. 4 Flow chart showing computation and data movement in the proposed framework for visual object tracking using particle filter on NoC (Design I).

Valid	Tail	Destination Port	VC	Add_X	Add_Y	Pixel
(1)	(1)	(4)	(1)	(10)	(10)	(9)

(a) Flit for sending image pixel values from node *N0* to node *N#*

Valid	Tail	Destination Port	VC	Index	Hist_value
(1)	(1)	(4)	(1)	(5)	(24)

(b) Flit for sending reference histogram values from node *N0* to node *N#*

Valid	Tail	Destination Port	VC	0	Node_id	Bhatt_value
(1)	(1)	(4)	(1)	(1)	(4)	(24)

(c) Flit for sending Bhattacharyya coefficient values from node *N#* to node *N0*

Fig. 5 Flit formats used for communication between node *N0* and node *N#*

estimate of the target center is performed in the master node. The format of the message flits (packets) for the various communications are shown in Figure 5. Currently, proposed design is configured to support 29-bit data in NoC flit, to represent pixel location and intensity, histogram values and Bhattacharyya coefficient. As shown in Figure 5a, pixel intensity value is sent along with pixel location (Add_X, Add_Y) from node *N0* to node *N#*. *Add_X* and *Add_Y* are 10 bits long and pixel value is limited to 9 bit (3-bit per each channel of RGB image). As shown in Figure 5b, histogram values (24-bit) of reference histogram are sent to all nodes (*N#*) with corresponding index (5-bit) of histogram values. After computing Bhattacharyya coefficient, each of the node

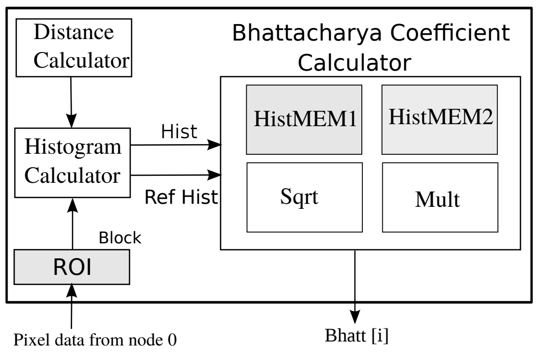


Fig. 6 Processing element (PE) at node *N0* and node *N#*. PE at node *N#* of the network, receives pixel data from master node through NI of the router. Processing element at node *N0* received data through FIFO.

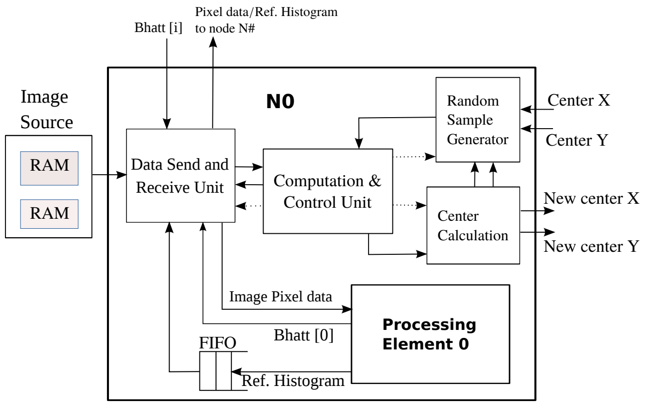


Fig. 7 Processing element at master node (*N0*) of the network.

sends this value (24-bit) to the master node ($N0$) using flit structure, shown in Figure 5c.

5.1.1 Internal architecture of processing element

Figure 6 shows the block diagram of processing element at the master node and other 15 nodes. Processing element at master node is also responsible for calculating reference histogram for the first frame. Processing elements at other nodes are active after the initial frame has been processed. Pixels received by processing elements are stored in ROI module and computation starts after all pixels in the ROI are received. Each of the submodules of the processing element is described next.

Region of Interest (ROI)

The input to this module is pixel data, which is extracted from the input flits (shown in Figure 5a). This module is a RAM with size $w \times h$ to store pixels in the ROI. As RGB image data is stored with only 3-bits (corresponding to 8-bin) from each channels, size of ROI memory is $9 \times w \times h$ bits.

Distance Calculator

Based on the observation model (10), larger weight should be given to pixels near the center compared to the pixels farther from the center. This module computes weights by calculating the distance of each pixel of the ROI to the center pixel according to (9).

Histogram Calculator

The inputs to this module are from the distance module and the ROI module. A distance weighted histogram of ROI is computed parallelly at every node of the NoC and stored in the histogram memory modules (*HistMEM1 and HistMEM2*) [9].

Bhattacharyya coefficient calculation

The inputs to this module are taken from the histogram memory modules (*HistMEM1 and HistMEM2*). It has two inputs: i^{th} candidate histogram and reference histogram. The calculation of Bhattacharyya coefficient requires multiplication, square root and addition. Bhattacharyya coefficient is calculated as,

$$B[i] = \rho[\hat{p}, \hat{q}] = \sum_{j=1}^{8\times8\times8} \sqrt{\hat{p}_i(j)\hat{q}(j)} \tag{14}$$

where, $\hat{q}$ is the reference histogram stored in memory module at every node of NoC, $\hat{p}_i$ is the calculated histogram at every node of NoC for i^{th} particle and $8 \times 8 \times 8$ is the number of bins used in the histogram calculation for ROI of the RGB Image. The Bhattacharyya coefficients are calculated for each particle and denoted as $B[i]$, where i varies from 0 to $N-1$ for N particles. All calculated Bhattacharyya coefficients are sent to the processing element of the node *N0* for estimating the target center using the flit structure shown in Figure 5c.

5.1.2 Internal architecture of master node

Figure 7 shows the block diagram of the processing element at the master node. Master node is responsible for reading the pixel data from the image source. It also contains one of the processing element, which is used for computation of reference histogram. This module also contains random number generator module for generating samples (particles). It also estimates the new target center value by a weighted mean calculation. Each of the submodules are described next.

Image Source

This block plays the role of frame buffer for two images at the same instance. As it is preferred to fit the whole design on the FPGA board, we use a 320×240 sized image in our design. For saving memory, upper 3-bits (corresponding to 8 bins) of pixel values for three (R,G,B) channels, is stored, which requires the size of each RAM to be $9 \times 320 \times 240$. When one RAM is used for the computation of the current frame, the other RAM is used for storing the pixels from next frame. Hence, the next frame is available for processing, as soon as the computation of the current frame is done.

Random Sample Generator

This module provides N random Gaussian samples with mean as the center (C_x, C_y) and variance of 20 pixels. Gaussian distributed random numbers are generated using Ziggurat algorithm [8].

Data Send and Receive Unit

In this module, the sixteen ROI blocks $(w \times h)$ around the randomly generated centers $(x_k^i = \{C_x[i], C_y[i]\})$ by the *Random Sample Generator* are sent to the other nodes of the NoC network. The inputs to this module are from the image storage RAM memory and center x_k^i for current sample i from the *Random Sample Generator* module. As shown in Figure 5a, the image data (pixel value) with address (pixel location) is sent to the chosen node by the master node. It takes $w \times h$ clock cycles for sending selected ROI block.

Target Center Estimation

New target center estimate is obtained by weighted mean of samples (center values), where weight is decided by Bhattacharyya coefficient. As shown in Figure 7, the Bhattacharyya coefficient $(B[i])$ is received from the i^{th} node through network interface of node *N0*. Received values $B[i]$ are multiplied with their relative center values. They are denoted as $B_x[i]$ and $B_y[i]$, and computed as,

$$\{B_x[i], B_y[i]\} = B[i] * \{C_x[i], C_y[i]\} \tag{15}$$

The new center $(NewC_x,\ NewC_y)$ is estimated by,

$$\{NewC_x, NewC_y\} = \left\{ \frac{\sum_{i=1}^{N} B_x[i]}{\sum_{i=1}^{N} B[i]}, \frac{\sum_{i=1}^{N} B_y[i]}{\sum_{i=1}^{N} B[i]} \right\} \tag{16}$$

This computation is done with a finite word length division, with 40-bits to represent numerator and 30-bits for denominator, to estimate the target center.

5.1.3 Execution flow

Execution flow of *Design I* is shown in Figure 8. *Computation and Control unit (C)* sends ROI pixels to all PEs at nodes *N0 to N15*. Nodes send Bhattacharyya coefficient values to *Computation and Control unit (C)*. This process is repeated till computation of N particles for all objects is over. Center estimate is finally done at node N0.

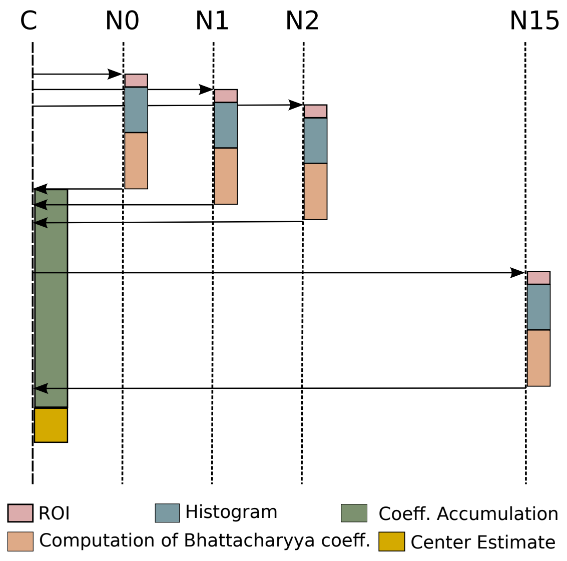


Fig. 8 Execution flow of Design I. *Computation and Control unit (C)* sends ROI pixels to all nodes *N0 to N15* and receives Bhattacharyya coefficient values from nodes till computation of N particles for all objects is completed

.

5.2 Design optimized for area (Design II)

Flowchart of design with area optimization for particle filter based object tracking on NoC is shown in Figure 9. Processing element at node *N#* receives pixel information

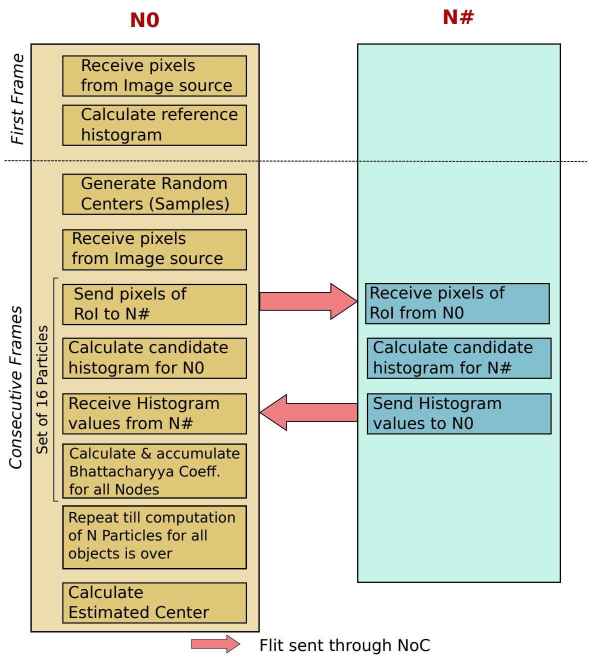

N0
N#
First Frame
Receive pixels from Image source
Calculate reference histogram
Consecutive Frames
Generate Random Centers (Samples)
Receive pixels from Image source
Set of 16 Particles
Send pixels of RoI to N#
Calculate candidate histogram for N0
Receive Histogram values from N#
Calculate & accumulate Bhattacharyya Coeff. for all Nodes
Repeat till computation of N Particles for all objects is over
Calculate Estimated Center
Receive pixels of RoI from N0
Calculate candidate histogram for N#
Send Histogram values to N0
Flit sent through NoC

Fig. 9 Flow chart showing computation and data movement in the proposed framework for visual object tracking using particle filter on NoC (*Design II*).

for calculating candidate histograms. After computation is over, candidate histogram is sent to master node for Bhattacharyya coefficient computation. The flit format of the design, is shown in Figure 10. The pseudocode

Valid (1)	Tail (1)	Destination Port (4)	VC (1)	Add_X (10)	Add_Y (10)	Pixel (9)

(a) Flit for sending image pixel values from node *N0* to node *N#*

Valid (1)	Tail (1)	Destination Port (4)	VC (1)	PE_id (4)	Hist_addr (9)	Hist_value (16)

(b) Flit for sending candidate histogram values from node *N#* to node *N0*

Fig. 10 Flit formats of optimized design, used for communication between node *N0* and node *N#*.

of algorithm to be implemented on the master node is shown in Algorithm 1. Pseudocode of other computation carried out at master node and other processing elements for optimized design are presented in Appendix A.

Algorithm 1: Master Node: ObjectTracking ($Im1$, $Im2$, $NumObj$, N, Cx, Cy, $NewCx$, $NewCy$)

```
Input  : $Im1, Im2$ - frames stored in RAM1,RAM2 of $Image\,Source$ block, $NumObj$ - Number
         of objects to be tracked, $N$ - Number of particles, ($Cx$, $Cy$) - Arrays of initial center
         values
Output : ($NewCx$, $NewCy$)-Arrays of estimated center values

For first frame;
// Calculate ref. historam for all objects
for $j \leftarrow 1$ to $NumObj$ do
    HMO($j$)← RefHistogram($Im1$);
end
For consecutive frames;
repeat
    // Generate Samples (Particles)
    S ←GenSamples();
    sCount ←0;
    sbh ←0;
    Read frame from $Im2$;
    repeat// with a batch of 16 PEs
        // Extract and Send ROI from $Im2$
        ExtractSendROI ($Im2$, $NumObj$, S, $Cx$, $Cy$);
        // Receive histograms from nodes
        for $i \leftarrow 1$to 16 do
            HM($i$)←ReceiveHistogram($Node(i)$);
        end
        // Compute and accumulate Bhattacharyya coeff.
        CompAccBhatt ($NumObj$, HMO, HM, bh, sbh, sbhx, sCount);
    until sCount $<= N$;
    // Estimate centers
    EstCenters ($NumObj$, sbh, sbhx, $NewCx$, $NewCy$);
until all frames are processed;
```

5.2.1 Processing element

In Design-I (Figure 6), pixel data sent by the master node is collected by all processing elements and stored in memory module (ROI) of $w \times h \times 9$ bits for candidate histogram calculation. Like stream processing, pixel data can be directly used to compute histogram in this module. This will alleviate the memory requirements of the processing nodes by eliminating the RAM used for ROI. Despite this modification, the processing element consumes comparatively more logic resources due to computation of multiplication and square root. Additionally, histogram memory blocks (*HistMEM1, HistMEM2*) also consume more block RAM resources of the FPGA. With 16 processing elements, complete design may not fit into a single FPGA. Processing element can be further optimized by reducing the various computations performed at each processing element. Among all computations, Bhattacharyya coefficient computation is one of the most resource-consuming computation, which can be off-loaded from all processing elements. Such optimized processing element is shown in Figure 11. However, more clock-cycles would be used at the master node to compute all the Bhattacharyya coefficients.

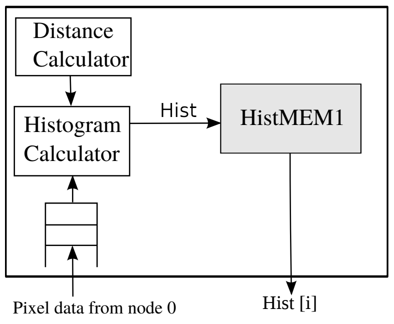


Fig. 11 Design of processing element for *Design-2*

5.2.2 Internal architecture of master node

As computation of Bhattacharyya coefficients is now in the master node, memory requirement in the master node is increased to store histograms of all processing elements. Modified architecture of the master node is shown in Figure 12. Candidate histograms are stored in memory modules (*HM1-HM16*). Reference histogram is also available at the master node. Maximum four reference histograms (*HM01-HM04*) can be stored related to four target objects.

5.2.3 Modified histogram module

Reference histogram is calculated at the master node (node *N0*) to be sent to all processing elements for calculating Bhattacharyya coefficient. In the original design

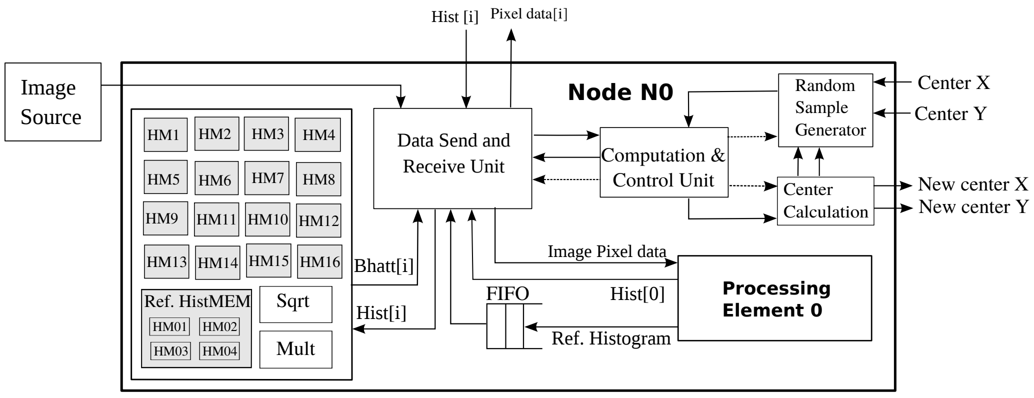


Fig. 12 Master node for Design-II (Figure 6), supporting storage of four reference histograms.

(Figure 6) of processing element, histogram is calculated using simple array of counters similar to [24], with the difference that distance value is added to bin values instead of incrementing it by 1. With streaming data, histogram computation is also to be implemented using a stream processing model. Parallel histogram computation introduced in [31], is a good solution for stored image, which may not be suitable for streaming data. Therefore, we have implemented distance based histogram module using a dual-port memory module as shown in Figure 13. Here,

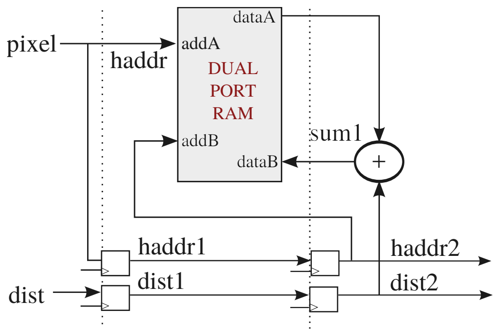


Fig. 13 Histogram computation using Dual port RAM supporting streaming data.

pixel information is used as an address (*addA*) of dual port RAM for getting current histogram value, where address (*haddr*) will also be propogated to pipeline stage for updating the histogram value later via *addB* of dual port RAM. Histogram value, available through *dataA* is added with distance value (*dist2*) related to the location of the pixel and resultant sum (*sum1*) would be written to the dual port RAM through *dataB*.

5.2.4 Communication optimization

Performance and power consumption of the NoC based system also depends on the amount of data transmitted in the network. As shown in the flow chart of the original design (Figure 4), three types of packets (flits) are transmitted : (1) image pixel data of ROI from master node to other processing nodes (*one-to-all*) (2) reference histogram data from master node to other processing nodes (*one-to-all*) (3) Bhattacharyya coefficient from all processing nodes to master node (*all-to-one*). As observed from the flow chart of optimized design (Figure 9), two types of packet are transmitted through network links: (1) image pixel data of ROI from the master node to other processing nodes (*one-to-all*) (2) candidate histogram data from all the processing nodes to the master node (*all-to-one*).

In NoC, flow of data from master node to all the processing nodes (*one-to-all)* is managed through credit provided to router links by destination node upon receipt of flit. In case of data transfer from all processing nodes to the master node *(all-to-one)*, there are chances of congestion due to heavy flit traffic towards a single node.

Significant traffic can be reduced, by sending only non-zero entries of histogram values to the master node as reflected in Table 1. This enables real-time processing and reduced power consumption. The proportion of non-zero entries in the RGB histogram was obtained (using Matlab) for various image sequences with $8 \times 8 \times 8$ bins and is shown in Table 2. It is found that number of non-zero bin entries for image sequences ranges from $5 - 10\%$, leading to a reduction of $\approx 80 - 90\%$ traffic (due to two flits per single histogram entry) in the NoC.

Table 1 Data transfer in NoC

Data	Transfer Type	Initial design (Figure 4)	Optimized design (Figure 9)
Image pixel	*one-to-all*	$N \times 17 \times 17$	$N \times 17 \times 17$
Reference histogram	*one-to-all*	$(N-1) \times 512$	-
Candidate histogram	*all-to-one*	-	$(N-1) \times (2 \times M)$
Bhattacharyya coefficient	*all-to-one*	$(N-1)$	-

where M is the average number of non-zero entries in histogram, $2 \times M$ is due to two flits to be sent for single histogram entry

Table 2 Distribution of non-zero entries in RGB images from various datasets.

Image sequence	Image Size	non-zero bin entries	%
Ball	136 × 136	66	12.9%
Coca-cola	640 × 480	51	10%
iLids (cars) [3]	720 × 576	20	3.9%
Hand	384 × 288	18	3.5%
PETS2006 [27]	720 × 576	14	2.7%
Heli	320 × 240	38	7.4%
CAVIAR(Set1) [10]	384 × 288	24	4.7%
CAVIAR(Set2)[10]	384 × 288	24	4.7%
CAVIAR(Set3)[10]	384 × 288	21	4.1%

5.2.5 Execution flow

Execution flow of *Design II* is shown in Figure 14. *Computation and Control unit (C)* sends ROI pixels to all PEs at nodes *N0 to N15*. Nodes send histograms to *Computation and Control unit (C)*. Bhattacharyya coefficient computation for all nodes are carried out at node *N0*. This process is repeated till computation of N particles for all objects is completed. Finally, center estimation is done at node *N0*.

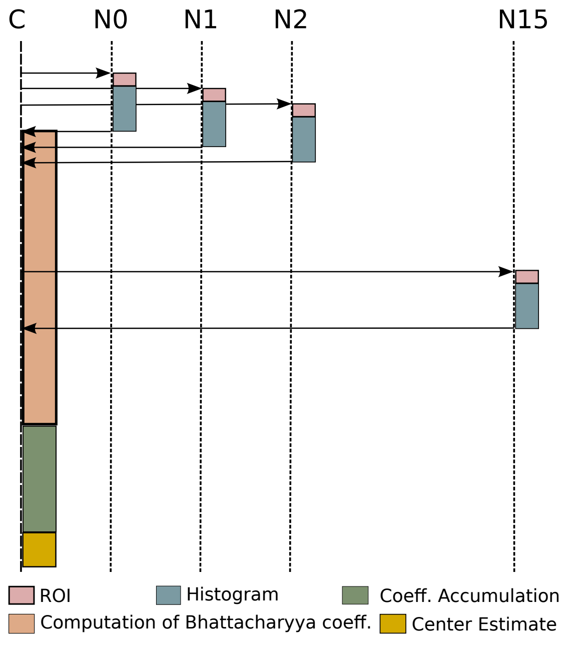


Fig. 14 Execution flow of Design II. *Computation and Control unit (C)* sends ROI pixels to all nodes *N0 to N15* and receives histogram values from nodes

6 Analysis of the proposed architecture

Proposed architecture (*Design II*) was analysed for scalability to support varying image sizes, object sizes, number of particles and number of objects. Timing analysis is also performed to estimate maximum frame rate possible for different FPGA platforms.

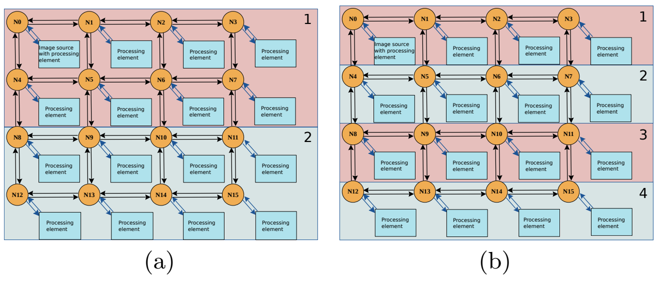


Fig. 15 PE allocation for multi-object tracking (a) allocation of 8 PEs each for 2 objects (b) allocation of 4 PEs each for 4 objects.

6.1 Scalability

The proposed implementation can support different image sizes, object sizes, more number of particles and track multiple objects.

Image size

Proposed design supports various image sizes by changing the size of RAM in *Image source* block, typically restricted by available BRAM sources in FPGA. Although, flit for sending image pixel data currently supports image size upto 1024 × 1024 pixels.

Object size

As histogram computation is carried out with streaming data, any object size is supported by processing element with modified histogram module (Section 5.2.3).

Number of objects

Designed master node can support four reference histograms for four different objects to be tracked simultaneously (Figure 12). Additionally, multi-object tracking (for 2 or 4 objects) is seamlessly supported by allocating groups of PEs for computation related to each object in modified design as shown in Figure 15.

Number of particles

Proposed design supports more number of particles (in multiple of 16), by repeating the computation in batch of 16/8/4 PEs for 1/2/4 number of objects (Figure 9).All these changes can be easily achieved by changing configuration parameters (statically) or passing values to processing elements (dynamically).

Table 3 Resource utilization for both designs

Design		**Design I**	**Design II**		**Design II**	
Image Size (in pixels)		320 × 240	320 × 240	320 × 240	320 × 240	
Object Size (in pixels)		17 × 17	17 × 17	33 × 33	17 × 17	
FPGA	**Zynq XC7Z020 (Zedboard)**				**XC4VFX100** [14]	
	Available	**Used(%)**	**Used(%)**	**Used(%)**	**Available**	**Used(%)**
Slice Registers	106400	87101 (**82%**)	27127 (**25%**)	28330 (**26%**)	42176	41686 (**98%**)
Slice LUTs	53200	123376 (**232%**)	45018 (**84%**)	46570 (**87%**)	84352	33236 (**39%**)
Block RAM	140	150 (**107%**)	120 (**85%**)	128 (**91%**)	376	217 (**57%**)
DSP48E	220	30 (**13%**)	30 (**13%**)	30 (**13%**)	160	26 (**16%**)
Max. Frequency			69.7 MHz	69.7 MHz	37.3 MHz	
Power consumption			0.594 W	0.6 W	1.223 W	

7 Implementation and Results

7.1 Implementation

Among the two designs, we have implemented *Design II* on FPGA. Our implementation (*Design II*) was synthesized for two FPGA boards (Zedboard and Virtex-4 FPGA[13, 14]) for image size of 320 × 240, different object sizes (17 × 17, 33 × 33) and different number of particles (128, 256, 512 and 1024). Hardware results match with the simulation results for image sequences. Resource utilization of the proposed design and power consumption (estimated by Xilinx Xpower analyzer with maximum frequency) for the two FPGA boards, is shown in Table 3. The proposed design (*Design II*) fits well in these FPGA platforms.

7.2 Results

The proposed scalable implementation of particle filter was validated on a FPGA platform (Zedboard) using the setup shown in Figure 16. Petalinux is installed on the ARM processor of Zedboard. Image frames from the SD card are sent to the FPGA fabric through ARM AXI interface. Frame buffer at the master node is used to store image data for the computation related to the particle filter based visual tracking algorithm. Once computation is over, the estimated target center is sent back to the ARM processor for generating results.

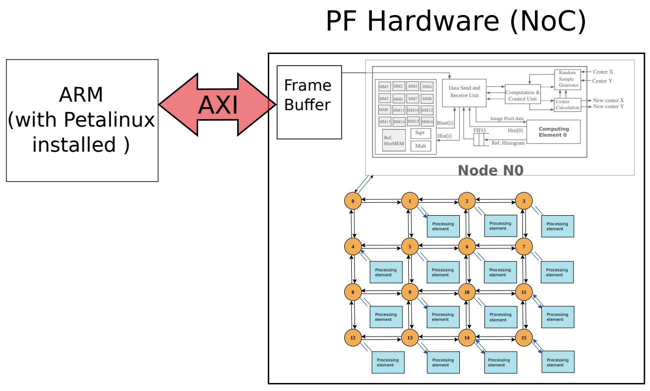
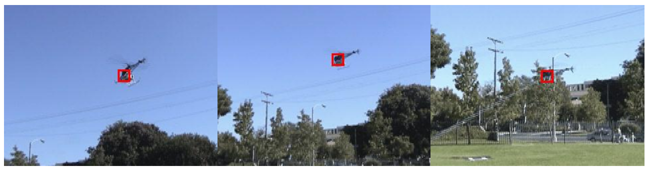
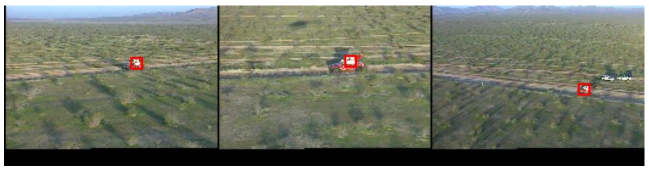


Fig. 16 Validation of the proposed NoC based visual object tracking, by having input from a ARM based system and sending back updates to the system with computation performed on the FPGA.

Object tracking results of our implementation with object size of 17 × 17 using the validation platform is shown in Figure 17. It took an average ≈ 197917 *cycles* to compute the center, achieving 348 *fps* with operating frequency of 69 MHz for Zedboard (Xilinx SoC XC7Z020), for the image sequence (Heli) with object size of 17 × 17 and 128 particles. Results for various image sequences with object size of 33 × 33 is shown in Figure 18, for different object sizes and different number of particles.

The proposed implementation was also validated for multi-object tracking with a synthetic video, where four objects are being tracked. Object tracking results of our


(a) Image sequence-Heli with number of particles: 128 (Frame nos: 1, 159, 251)


(b) Image sequence-RedTeam [28] with number of particles: 512 (Frame nos: 1, 378, 924)

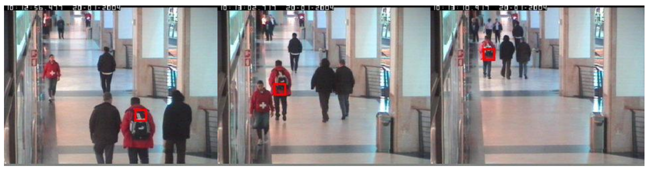
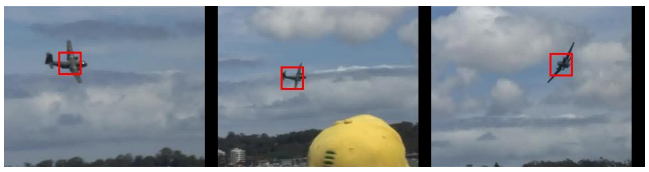

(c) Image sequence-CAVIAR1 [10] with number of particles: 1024 (Frame nos: 1, 162, 366)

Fig. 17 Tracking result obtained with the proposed framework on various image sequences with object size: 17 × 17.

(a) Image sequence-Heli with number of particles: 128 (Frame nos: 1, 159, 251)

(b) Image sequence-Aircraft1 with number of particles: 1024 (Frame nos: 1, 80, 269)


(c) Image sequence-Aircraft2 with number of particles: 1024 (Frame nos: 1, 83, 292)

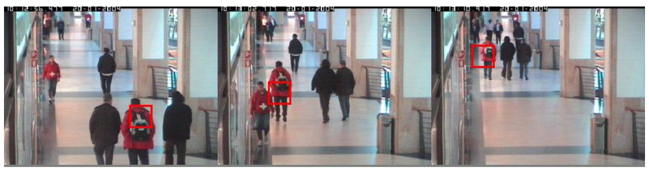

(d) Image sequence-CAVIAR1 [10] with number of particles: 1024 (Frame nos: 1, 162, 366)

Fig. 18 Tracking result obtained with the proposed framework on various image sequences with object size: 33 × 33.

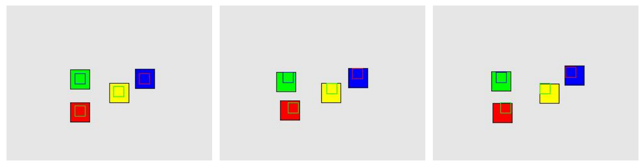


Fig. 19 Tracking result on a synthetic image sequence (multi-object) with image size: 320×240, object size: 17×17, number of particles: 128 (Frame nos: 1, 116, 213).

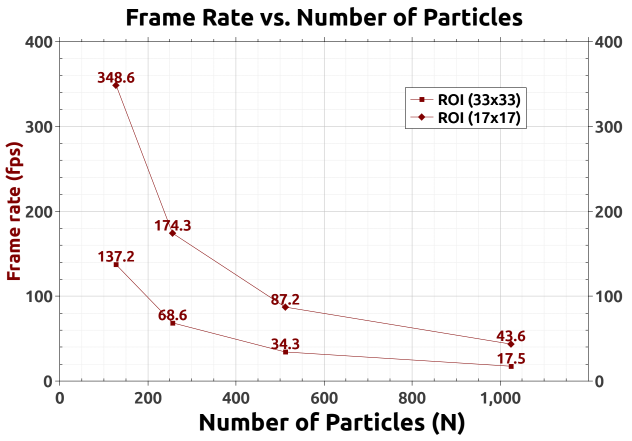


Fig. 20 Frame rate achieved using the proposed architecture for varying number of particles (128, 256, 512, 1024).

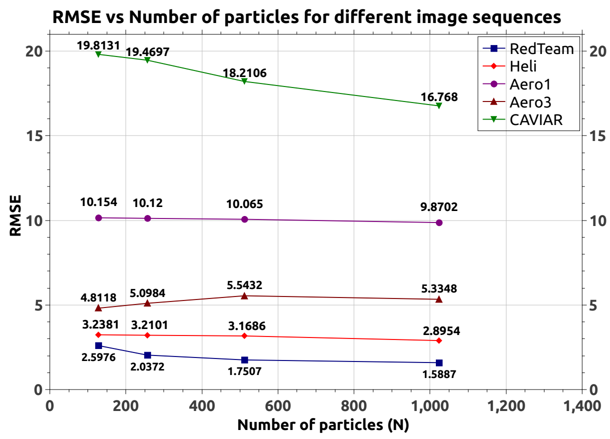


Fig. 21 Root-mean square error (RMSE) in object position estimates for varying number of particles (128, 256, 512, 1024).

implementation for tracking four objects simultaneously is shown in Figure 19 with object size of 17×17 and 128 particles. Frame rate of 87 fps is achieved for tracking four objects with operating frequency of 69 MHz on Zedboard (Xilinx SoC XC7Z020).

Figure 20 shows that the frame rate decreases with increasing number of particles as well as with different object sizes (17×17, 33×33). Tracking accuracy in terms of root-mean-square error (RMSE) is also shown in Figure 21 with different number of particles for various image sequences. The RMSE of predicted centers $(\hat{x}_t, \hat{y}_t)$ for times t is computed for n different frames as the square root of the mean of the squares of the deviations:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}((\hat{x}_t - x_t)^2 + (\hat{y}_t - y_t)^2)}{n}} \qquad (17)$$

As observed, accuracy of tracking improves with increasing number of particles.

Comparison of the proposed implementation with existing work is shown in Table 4. All implementation are scalable for respective platforms. Proposed implementation is pure hardware (HW) implementation compared to pure software (SW) [22] and SW/HW [14, 29] implementation. Proposed implementation is validated for different object sizes (17×17, 33×33) and varying number of particles (128/256/512/1024) for various image sequences. As observed, it is possible to achieve low power consumption, approximately 1.7 mW/fps and 3.84 mW/fps with object size of 17×17 and 128 particles for Zynq XC7Z020 (Zedboard) and XC4VFX100 [14], respectively. This makes our implementation a good candidate for low-power, visual object tracking using FPGA. Hence,

Table 4 Comparison of proposed implementation (*Design II*) with other HW or HW/SW implementation of PF based visual object tracking. Provided results are from respective publications.

Parameters	**Montemayor et. al.** [22]	**S. Saha et. al.** [29]	**Happe et. al.** [14]	**Proposed**	
Implementation	SW	SW/HW	SW/HW	HW	
Platform	GPU	FPGA	CPU/FPGA	FPGA	
Device	GPU	FPGA	XC4VFX100	Zynq XC7Z020	XC4VFX100
Image type	Color	Gray	Color	Color	
Resolution (in pixels)	320×240	-	-	320×240	
Object Size (in pixels)	16×16	-	Variable	17×17, 33×33	
Number of particles	-	50	100	128/256/512/1024	
Frame Rate	300 fps	27 fps	8 fps	348 fps[*]/137 fps[+]	318 fps[**]/168 fps[++]
Power Consumption	-	-	-	0.594 W/0.6 W	1.223 W
Scalable	Yes	Yes	Yes	Yes	Yes
Support for multi-FPGA	-	No	No	Yes	Yes

[*] Hardware Implementation with 128 Particles, $T_{\text{FIFO}} = 2$, $T_{\text{BRAM}} = 2$, $L_{\text{initFlit}} = 2$, $w = 17$, $h = 17$
[+] Hardware Implementation with 128 Particles, $T_{\text{FIFO}} = 2$, $T_{\text{BRAM}} = 2$, $L_{\text{initFlit}} = 2$, $w = 33$, $h = 33$
[**] Estimated values with 128 Particles, $T_{\text{FIFO}} = 2$, $T_{\text{BRAM}} = 2$, $L_{\text{initFlit}} = 2$, $w = 17$, $h = 17$
[++] Estimated values with 128 Particles, $T_{\text{FIFO}} = 2$, $T_{\text{BRAM}} = 2$, $L_{\text{initFlit}} = 2$, $w = 33$, $h = 33$

where, T_{BRAM}- overhead due to read/write of BRAM, T_{FIFO}-overhead due to read/write of FIFO, w is width of ROI, h is height of ROI, $L_{initFlit}$ is initial latency while sending the flit

proposed implementation can be used in low-power smart camera applications.

8 Conclusions

In this paper, we proposed a novel scalable NoC implementation of particle filter based visual tracking algorithm. Optimization at different levels were performed to achieve particle filter based visual tracking on the NoC. Optimization of NoC traffic was also done by sending only non-zero pixels through router links. We achieved a frame rate of 348 fps and 137 fps for single object tracking with object size of 17×17 and 33×33 pixels, respectively. Also, multi-target tracking was achieved by the scaling procedure, achieving a frame rate of 87 fps for multi-object tracking of four objects. We validated the particle filter based visual tracking with video feed from a Petalinux based system. We obtained very less power consumption, approximately 1.7 mW/fps with object size of 17×17 and 128 particles for Zynq XC7Z020 (Zedboard), operating at 69 MHz. This makes our implementation a good candidate for low-power, visual object tracking using FPGA. The proposed scalable architecture can also be easily adapted to other video processing applications.

In the proposed implementation, tracking accuracy is based on a simple state-space and observation model to represent the object states. Future work will consider complex state space models (such as [1, 4]) and features for robust visual object tracking. This implementation can be extended to support fusion of multiple features as discussed in [20, 23]. To support more objects and higher number of particles, proposed design can be partitioned across multi-FPGA using framework proposed in [17].

References

1. Abd El-Halym HA, Mahmoud II, Habib S (2012) Proposed hardware architectures of particle filter for object tracking. EURASIP Journal on Advances in Signal Processing 17(1):17
2. Ali M, Welzl M, Zwicknagl M (2008) Networks on chips: Scalable interconnects for future systems on chips. In: 4th European Conference on Circuits and Systems for Communications, 2008 (ECCSC-2008), pp 240–245
3. AVSS (2007) i-Lids dataset for AVSS-2007. ftp://motinas.elec.qmul.ac.uk/pub/iLids/
4. Brown J, Capson D (2012) A framework for 3D model-based visual tracking using a GPU-accelerated particle filter. IEEE Transactions on Visualization and Computer Graphics 18(1):68–80
5. Chitchian M, van Amesfoort A, Simonetto A, Keviczky T, Sips H (2013) Adapting particle filter algorithms to many-core architectures. In: 27th IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp 427–438
6. Cho JU, Jin SH, Pham XD, Jeon JW, Byun JE, Kang H (2006) A real-time object tracking system using a particle filter. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 2822–2827
7. Comaniciu D, Ramesh V, Meer P (2003) Kernel-based object tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence 25(5):564–577
8. Doornik J (2005) An improved Ziggurat method to generate normal random samples. Nuffield College, University of Oxford
9. Engineer P, Velmurugan R, Patkar SB (2015) Parameterizable FPGA framework for particle filter based object tracking in video. In: 28th International Conference on VLSI Design (VLSID-2015), pp 35–40

10. Fisher RB (2004) CAVIAR (Context Aware Vision using Image-based Active Recognition) test case scenarios. http://homepages.inf.ed.ac.uk/rbf/CAVIAR/, online; October 2004
11. Fowers J, Brown G, Cooke P, Stitt G (2012) A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '12, pp 47–56
12. Fresse V, Aubert A, Bochard N (2007) A predictive NoC architecture for vision systems dedicated to image analysis. EURASIP Journal on Embedded Systems 2007(1):36–36
13. Happe M, Lübbers E, Platzner M (2009) A multithreaded framework for Sequential Monte Carlo methods on CPU/FPGA platforms. In: Proceedings of 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC-2009), Springer Berlin Heidelberg, pp 380–385
14. Happe M, Lübbers E, Platzner M (2013) A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. Journal of Real-Time Image Processing 8(1):95–110
15. Isard M, Blake A (1998) Condensation - conditional density propagation for visual tracking. International Journal of Computer Vision 29:5–28
16. Kim D, Kim K, Kim JY, Lee S, Lee SJ, Yoo HJ (2009) 81.6 GOPS object recognition processor based on a memory-centric NoC. IEEE Transactions on VLSI Systems 17(3):370–383
17. Kumar VBY, Engineer P, Datar M, Turakhia Y, Agarwal S, Diwale S, Patkar SB (2015) Framework for application mapping over packet-switched network of FPGAs: Case studies. In: 2nd International Workshop on FPGAs for Software Programmers (FSP 2015), London, United Kingdom
18. Li SA, Hsu CC, Lin WL, Wang JP (2011) Hardware/software co-design of particle filter and its application in object tracking. In: Proceedings of International Conference on System Science and Engineering-2011, pp 87–91
19. Lübbers E, Platzner M (2009) ReconOS: Multithreaded programming for reconfigurable computers. ACM Transactions on Embedded Computing Systems 9(1):8:1–8:33
20. Maggio E, Smerladi F, Cavallaro A (2007) Adaptive multifeature tracking in a particle filtering framework. IEEE Transactions on Circuits and Systems for Video Technology 17(10):1348–1359
21. Montemayor AS, Pantrigo JJ, Sánchez A, Fernández F (2004) Particle filter on GPUs for real-time tracking. In: ACM SIGGRAPH 2004 Posters, pp 94–
22. Montemayor AS, Payne BR, Pantrigo JJ, Cabido R, Sánchez A, Fernández F (2006) Improving GPU particle filter with shader model 3.0 for visual tracking. In: ACM SIGGRAPH 2006 Research Posters
23. Moreno-Noguer F, Sanfeliu A (2005) A framework to integrate particle filters for robust tracking in non-stationary environments. In: Proceedings of Second Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA), Lecture Notes in Computer Science, pp 93–101
24. Muller S (1995) A new programmable VLSI architecture for histogram and statistics computation in different windows. In: International Conference on Image Processing, 1995, vol 1, pp 73–76
25. Papamichael MK, Hoe JC (2012) CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In: Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, pp 37–46
26. Pérez P, Hue C, Vermaak J, Gangnet M (2002) Color-based probabilistic tracking. In: Proceedings of the 7th European Conference on Computer Vision-Part I, Springer-Verlag, ECCV '02, pp 661–675
27. PETS (2006) Benchmark data for PETS-2006. http://www.cvg.reading.ac.uk/PETS2006/data.html
28. RedTeam (2005) Aerial footage of CMU red team unmanned ground vehicle. http://vision.cse.psu.edu/data/vividEval/datasets/PETS2005/RedTeam/index.html
29. Saha S, Bambha NK, Bhattacharyya SS (2010) Design and implementation of embedded computer vision systems based on particle filters. Computer Vision and Image Understanding 114(11):1203–1214
30. Saponara S, Fanucci L, Petri E (2013) A multi-processor NoC-based architecture for real-time image/video enhancement. Journal of Real-Time Image Processing 8(1):111–125
31. Shahbahrami A, Hur J, Juurlink B, Wong S (2008) FPGA implementation of parallel histogram computation. In: Proc. 2nd HiPEAC Workshop on Reconfigurable Computing, Gteborg, Sweden, pp 63–72
32. Sileshi BG (2014) Hardware/software co-design of particle filter in grid based fast-SLAM algorithm. In: International Conference on Embedded Systems and Applications (ESA),24–27 July, 2014
33. Sileshi BG, Oliver J, Ferrer C (2016) Accelerating particle filter on FPGA. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 591–594
34. Singh S, Shekhar C, Vohra A (2017) Real-time FPGA-based object tracker with automatic pan-tilt features for smart video surveillance systems. Journal of Imaging 3(2)
35. Szwoch G (2016) Performance evaluation of the parallel object tracking algorithm employing the particle filter. In: 2016 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), pp 119–124
36. Venkatrayappa D, Sidibé D, Meriaudeau F, Montesinos P (2014) Adaptive feature selection for object tracking with particle filter. In: Proceedings of 11th International Conference of Image Analysis and

Recognition (ICIAR 2014), Springer International Publishing, pp 395–402

37. Wang Q, Chen F, Xu W, Yang MH (2011) An experimental comparison of online object-tracking algorithms. Proceedings of SPIE - The International Society for Optical Engineering pp 8138–8138–11
38. Wang Y, Pan Y, Long Y, Yan X, Huan R (2012) An on-line reconfigurable four-ary tree-based network on chip for distributed particle filters. In: 2nd International Conference on Computer Science and Network Technology (ICCSNT), 2012, pp 2102–2106
39. Yilmaz A, Javed O, Shah M (2006) Object tracking: A survey. ACM Computing Surveys 38(4)
40. Zhang L, Fresse V, Khalid M, Houzet D, Legrand AC (2009) Evaluation and design space exploration of a time-division multiplexed NoC on FPGA for image analysis applications. EURASIP Journal on Embedded Systems 2009(1):1–15
41. Zhao Y, Pei H (2013) Object tracking based on particle filter with discriminative features. Journal of Control Theory and Applications 11(1):42–53

Appendix A pseudocode for computation done at master node and other processing elements

Main computation at the master node is described in Algorithm 1 (Section 5.2). Algorithm 2, 3 and 4 describe subsections of the main computation carried out at the master node. Algorithm 5 describes the computations at all processing elements.

Algorithm 2: ExtractSendROI ($Im2$, $NumObj$, S, Cx, Cy)

```
/* Extract and send ROI to differnt PEs through
   network interface (NI) of NoC)                     */
Input  : $Im2$ - Current frame,
         $NumObj$ - Number of objects,
         $N$ - Number of particles,
         ($Cx$, $Cy$) - Initial center values
Output : Flit sent through NI of the router

if $NumObj = 2$ then
    for $i \leftarrow 1$ to 8 do
        SendROI ($Cx(1)$, $Cy(1)$, $S(i)$, $Node(i)$);
        SendROI ($Cx(2)$, $Cy(2)$, $S(i)$, $Node(i + 8)$);
    end
else if $NumObj = 4$ then
    for $i \leftarrow 1$ to 4 do
        SendROI ($Cx(1)$, $Cy(1)$, $S(i)$, $Node(i)$);
        SendROI ($Cx(2)$, $Cy(2)$, $S(i)$, $Node(i + 4)$);
        SendROI ($Cx(3)$, $Cy(3)$, $S(i)$, $Node(i + 8)$);
        SendROI ($Cx(4)$, $Cy(4)$, $S(i)$, $Node(i + 12)$);
    end
else // default: NumObj = 1
    for $i \leftarrow 1$ to 16 do
        SendROI ($Cx(1)$, $Cy(1)$, $S(i)$, $Node(i)$);
    end
end
```

Algorithm 3: CompAccBhatt ($NumObj$, HMO, HM, bh, sbh, sbhx, sCount)

```
/* Compute Bhatacharyya coefficients)                 */
Input  : $NumObj$ - Number of objects, HMO-
         Reference histograms , HM- Candidate
         histogram, bh- Bhatt. coeff
Output : sbh- sum of Bhattacharyya coefficients, sbhx-
         sum of multiplication of Bhattacharyya
         coefficient with center values, sCount-
         Sample count
if $NumObj = 2$ then
    for $i \leftarrow 1$ to 8 do
        bh($1$,$i$)←CmpBht(HMO($1$),HM($i$));
        bh($2$,$i$)←CmpBht(HMO($2$),HM($i$+$8$));
        sCount ←sCount +1;
        for $j \leftarrow 1$ to 2 do
            sbh($j$,$i$)←sbh($j$,$i$)+bh($j$,$i$);
            sbhx($j$,$i$)←sbhx($j$,$i$) + S [j,i]×bh($j$,$i$);
        end
    end
else if $NumObj = 4$ then
    for $i \leftarrow 1$ to 4 do
        bh($1$,$i$)←CmpBht(HMO($1$),HM($i$));
        bh($2$,$i$)←CmpBht(HMO($2$),HM($i$+$4$));
        bh($3$,$i$)←CmpBht(HMO($3$),HM($i$+$8$));
        bh($4$,$i$)←CmpBht(HMO($4$),HM($i$+$12$));
        sCount ←sCount +1;
        for $j \leftarrow 1$ to 4 do
            sbh($j$,$i$)←sbh($j$,$i$)+bh($j$,$i$);
            sbhx($j$,$i$)←sbhx($j$,$i$) + S [j,i]×bh($j$,$i$)
        end
    end
else // default: NumObj = 1
    for $i \leftarrow 1$ to 16 do
        bh($i$)←CmpBht(HMO($1$),HM($i$));
        sCount ←sCount +1;
        sbh($j$,$i$)←sbh($j$,$i$)+bh($j$,$i$);
        sbhx($j$,$i$)←sbhx($j$,$i$) + S [j,i]×bh($j$,$i$)
    end
end
```

Algorithm 4: EstCenters ($NumObj$, sbh, sbhx, $NewCx$, $NewCy$)

```
/* Estimate the center                                */
Input  : $NumObj$ - Number of objects to be tracked,
         sbh- sum of Bhattacharyya coefficients, sbhx-
         sum of multiplication of Bhattacharyya
         coefficient with center values
Output : ($NewCx$, $NewCy$)-Arrays of estimated
         center values
if $NumObj = 2$ then
    for $j \leftarrow 1$ to 2 do
        ($NewCx(j)$,$NewCy(j)$)← sbhx($j$)/sbh($j$);
    end
else if $NumObj = 4$ then
    for $j \leftarrow 1$ to 4 do
        ($NewCx(j)$,$NewCy(j)$)← sbhx($j$)/sbh($j$);
    end
else // default: NumObj = 1
    ($NewCx(1)$,$NewCy(1)$)← sbhx($1$)/sbh($1$);
end
```

Algorithm 5: Processing Element: CalcHistogram (*DataIn*, *h*, *w*)

```
/* Calculate Histogram and send histogram
   values to master through network interface
   (NI) of NoC)                                  */
Input  : $DataIn$ - Data coming to PE through NI,
         $h\,w$ - Height and width of the Object
Output : Flit sent through NI of the router

// Calculate distance values for object size of
   $w \times h$ to calculate distance weighted
   histogram
Dist ←CalcDist ($h$,$w$);
for $i \leftarrow 1$ to $w \times h$ do
    ROI($i$)← ReceiveROI ($DataIn$);
end
Hist ← CalcHistogram (ROI, Dist);
for $i \leftarrow 1$ to bins ×bins ×bins do
    if Hist($i$) $\neq 0$ then
        SendHistogram (Hist($i$), $Node(0)$)
    end
end
```