DISTRIBUTED ARCHITECTURE AND INTERCONNECTION SCHEME FOR MULTIPLE MODEL PARTICLE FILTERS

Akshay Athalye, Sangjin Hong and Petar M. Djurić

Stony Brook University, Stony Brook, NY 11794–2350
Email: {athalye, snjhong, djuric}@ece.sunysb.edu

ABSTRACT

In this paper, we present a hardware architecture for a Sampling Importance Resampling Filter (SIRF) applied to systems with multiple interacting models. This filter outperforms traditional filters in practical scenarios due to superior abilities of the SIRFs in dealing with nonlinear and/or non-Gaussian models. Compared to existing approaches, our method does not require knowledge of model transition probabilities and keeps a constant number of particles per model at all times. This allows for a regular hardware structure with deterministic execution time. A highly scalable, parallel architecture consisting of distributed processing elements and a central unit is described. We propose an interconnection scheme and data exchange protocol using the concept of distributed resampling that greatly speeds up filter execution and drastically reduces the required interconnect to a single bus without causing any communication bottleneck. The proposed architecture is evaluated on a Xilinx FPGA platform for a multiple model target tracking application and its efficiency and scalability is shown.

1. INTRODUCTION

Frequently, several systems require a multiple model formulation of the state dynamics (ex. position of a maneuvering target). In practice these models may be nonlinear and non-Gaussian. The interacting multiple model (IMM) methods have been extensively applied for state estimation in such scenarios [1]. These methods incorporate a bank of Kalman Filters or Extended Kalman Filters (EKFs) each tuned to a particular model, and use model transition probabilities to bring about interaction between individual filters and combine estimates. In case of nonlinear models, the traditional IMM filters perform poorly due to the shortcomings of the EKF. Alternative approaches for nonlinear and non-Gaussian IMMs include the unscented Kalman filter (UKF), [2], and the regularized particle filter (RPF) [3]. Some practical applications of the IMM filtering approach are: tracking of maneuvering mobile station in CDMA environment [4] and tracking of maneuvering vehicles for adaptive cruise control [2].

The SIRF is a Monte Carlo method for filtering in dynamic state space (DSS) models involving nonlinear functions and non-Gaussian noise [5]. SIRFs estimate the state based on the principle of Importance Sampling (IS), whereby, samples (particles) are drawn from a known density (Sample step) and assigned appropriate weights based on received observations using IS rules (Importance step). This weighted set of particles represents the posterior density of the state and is used to find various estimates of the state like MMSE or MAP. The posterior is then recursively updated in time as the observations become available. Another operation called Resampling is incorporated into this scheme to prevent weight degeneracy which causes the weights of many particles to be negligible after a few time instants. Resampling replicates those particles with large weights and discards those particles with small weights in a systematic manner. The above mentioned steps of sampling, importance computation, output (estimate) calculation and resampling form the traditional SIRF. The main advantage of the SIRF is that it can handle nonlinear and non-Gaussian states much more effectively than other traditional filters like the EKF and the UKF.

In this paper, we develop a parallel architecture for the efficient hardware implementation of the SIRF algorithm applied to systems with multiple interacting models thus enabling its use for real time processing. An SIRF based approach for MM systems has been previously presented in [6]. Compared to this method, our algorithm does not require knowledge of the model transition probabilities and keeps a constant number of particles per model at all times. This leads to a regular pipelined hardware structure and efficient interconnection scheme with negligible communication bottleneck. The architecture consists of distributed processing elements (PEs) and a central unit (CU). We apply the concept of distributed resampling [7] which drastically reduces the data exchange requirement between the PEs and the CU and also parallelizes the inherently sequential resampling process. This significantly reduces the resampling time thus increasing the speed of the filter. The required interconnect is reduced to a single bus and the communication bottleneck is eliminated by using a mechanism based on distributed particle storage and index addressing to bring about the data exchange. This architecture is highly scalable and also eliminates memory requirement in the CU. When evaluated on an FPGA platform, this architecture achieved speeds 100 times faster than a DSP implementation.

2. THE SIR MM ALGORITHM

The multiple model system is described using multiple DSS models as

$$\mathbf{x}_n = f_n^k(\mathbf{x}_{n-1}, \mathbf{q}_{n-1}) \quad (1)$$

$$\mathbf{z}_n = h_n^k(\mathbf{x}_n, \mathbf{v}_n) \quad (2)$$

where $\mathbf{x}_n$ describes the dynamically evolving state of the system at time n, f_n^k is the possibly nonlinear model dependent system transition function, $k = 1, 2...K$ represents the model index, where K is the total number of models used. The symbol $\mathbf{z}_n$ represents the observations of the system, h_n^k is the possibly nonlinear model dependent observation function, and $\mathbf{q}_n$ and $\mathbf{v}_n$ are state and observation noise processes, respectively, that may be non-Gaussian. Unlike other multiple model filtering approaches, we *do not* consider a model for the transition probabilities. The prior model probabilities and transition probabilities are respectively assumed constant while the posterior model probabilities are accounted for implicitly in the algorithm as shall be seen. The input to the filter at every sampling

This work has been supported by the NSF under Award CCR-0220011.

instant is the observation $\mathbf{z}_n$. The goal of the filter is to estimate the state using this observation and the multiple model state description.

In our algorithm, the traditional SIRF is extended to multiple models. In the sample step a set of particles is drawn from each model. This is represented as $\left\{\mathbf{x}_n^{(i),(k)}\right\}_{i=1}^{N_s}$ where N_s is the number of particles used in $each$ model and k is the model index number. The importance step in each model assigns a weight to each of its particles based on the received observations. Thus after these steps, we have a weighted set of particles from each model k, represented as $\left\{\left(\mathbf{x}_n^{(i),(k)}, w_n^{(i),(k)}\right)\right\}_{i=1}^{N_s}$. This set represents the individual model conditioned approximation to the state posterior.

We use the resampling step not only to avoid the generic weight degeneracy, but also to combine the weighted sets of particles from all the models. This resampled set of particles represents the combined posterior of the state over all models and can be used to compute the desired estimate. The resampling operation selects N_s of the total $K \times N_s$ sampled particles. This resampled set is denoted by $\left\{\widetilde{\mathbf{x}}_n^{(i)}\right\}_{i=1}^{N_s}$, where the particles are chosen such that

$$Pr(\widetilde{\mathbf{x}}_n^{(i)} = \mathbf{x}_n^{(j),(k)}) = w_n^{(j),(k)} \tag{3}$$

where $i, j = 1$ to N_s and $k = 1$ to K. The weight of each resampled particle is $1/N_s$ [5]. The posterior probability of each model is accounted for implicitly in this resampling step by the fact that the model with the largest weight will contribute the highest number of particles to the combined resampled set and hence to the final estimate. This *combined* set of N_s particles is then propagated to the sample step of each model to determine the sampled particles of the next time instant using (1). This brings about model interaction and makes the algorithm robust as the space of each model is explored at every instant.

The above algorithm was applied to the problem of tracking a moving vehicle with two models described as in [2]. A constant velocity model describes straight line motion and an almost-constant speed turn model covers vehicle maneuvers like U turns and rotaries. The results of the tracking, with N_s = 1000 particles per model, averaged over 100 Monte Carlo runs are presented in Fig. 1. The above example with N_s = 1000 and K = 2, when evaluated on a DSP platform (TI-TMS320C67x), gave a processing speed of about 200Hz. Hence design of efficient hardware is essential to meet real time processing requirements, particularly in situations where a larger K and larger N_s are needed.

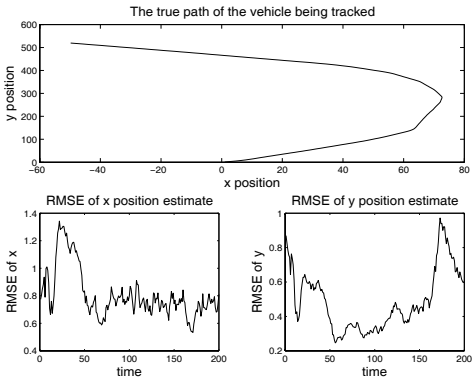


Fig. 1. Performance of the algorithm on a tracking example.

3. PARALLEL ARCHITECTURE FRAMEWORK

Fig. 2 shows a basic parallel framework for the implementation of the multiple model SIRF algorithm along with operations assigned

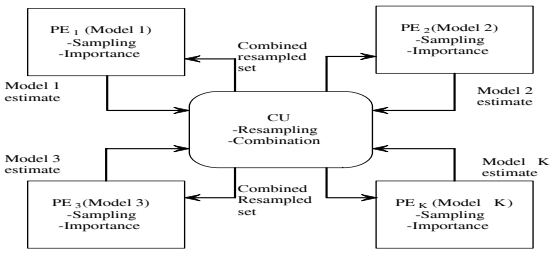


Fig. 2. Parallel architecture model for the algorithm.

to each unit. The PEs are tuned to different models and operate in parallel while the CU performs resampling and computes the estimate. Resampling is a centralized operation which operates on particles and weights from *all* the models (PEs). All traditional resampling algorithms are inherently sequential in operation. Accordingly, use of the traditional systematic resampling [5] would require all the $K \times N_s$ particles (N_s per PE) along with their weights to be sent to and stored in the CU. Resampling would then need to systematically choose N_s out of the total $K \times N_s$ particles. From results presented in [8], this resampling would take $((K+1) \cdot N_s - 1)$ cycles. Table 1 summarizes the amount of communication needed between the PEs and the CU if centralized resampling is used, where b_p and b_w are the fixed point bit widths used for representing the particles and weights respectively.

Data Transferred	Direction of Transfer	Amount (bits)
Weights	From *each PE* to CU	$K \times N_s \times b_w$
Sampled Particles	From *each PE* to CU	$K \times N_s \times b_p$
Resampled Particles	From CU to *each PE*	$N_s \times b_p$

Table 1. Data exchange requirement between PEs and CU if centralized resampling is used.

It is reasonable to assume b_p and b_w to be of the order of 16 and N_s of the order of 1000 or more. Thus the traditional centralized resampling approach has a very high data communication requirement, introduces a high resampling latency and also poses a significant memory requirement in the CU. This affects the scalability of the architecture since increasing the number of models or PEs K, increases the data exchange requirement, resampling time and CU memory requirement by a factor of N_s. This scheme also poses a serious bottleneck since resampling cannot start until all particles and weights are sent to the CU and the sample step of the next instant cannot start till the resampling step of the previous instant, which requires $(K+1) \cdot N_s - 1$ cycles, is completed.

To alleviate the problems of centralized resampling, we use the method of *distributed resampling* introduced for the single model distributed SIRF in [7]. The amount that each model (PE) contributes to the set of N_s resampled particles depends upon the weight of the PE, i.e., sum of weights of all particles in that PE. Distributed resampling makes use of this fact to split up resampling into a two stage hierarchical process. Initially, only the *sum of weights of all particles* of each PE, is sent to the CU. This is denoted by W^k for PE^k. The CU then performs the first stage of resampling which determines the *number of particles* that each PE will contribute to the final resampled set based on its sum of weights. We call this value as the PE (model) replication factor and denote it as R^k for PE^k where $\sum_{k=1}^{K} R^k = N_s$ and $0 \leq R^k \leq N_s$. This operation is done using the method of Residual Systematic Resampling (RSR). An efficient hardware implementation of RSR is presented in [8]. This computation requires K cycles.

The R^k values are then sent to the respective PEs. Each PE^k

performs resampling on its N_s sampled particles to produce R^k particles simultaneously with the other PEs. This resampling is still sequential, but has reduced execution time since only N_s particles are processed simultaneously in each PE as opposed to $K \times N_s$ in the CU if centralized resampling is performed. In distributed resampling, the communication between PEs and the CU consists only of K values of the sum of weights W^k and K replication factors R^k. This is a great reduction in comparison to the values presented in Table 1. The sample step in each PE requires *all* the N_s resampled particles of the previous time instant. Hence each PE^k needs to obtain $N_s - R^k$ particles from other PEs. We shall see in the next sections how this exchange is carried out efficiently without causing a communication bottleneck.

4. ARCHITECTURE DESCRIPTION

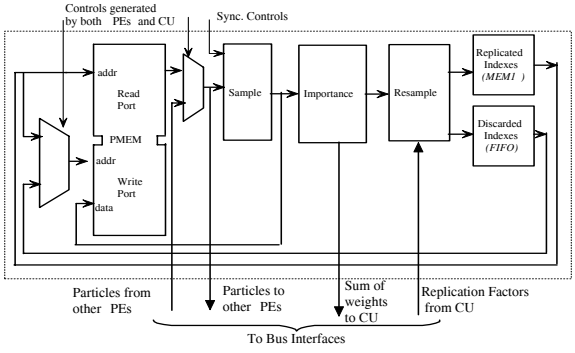


Fig. 3. Architecture of a single PE.

• **Structure a of PE:** Fig. 3 shows the basic architecture of each PE based on the memory schemes proposed in [8]. The sample step produces N_s particles in each PE using the combined resampled particles from the previous time instant. Weights of the particles are calculated by the importance step and the sum of weights W^k is sent to the CU. After receiving the number R^k from the CU resampling is done in parallel in each PE to determine which R^k of the N_s particles of PE^k will be present in the final resampled set. The process returns the addresses or indexes of the R^k resampled particles in $PMEM$. They are written to the replicated index memory $MEM1$ while the other indexes are written to the discarded index FIFO. The discarded particles indexes are used to appropriately write the sampled particles to $PMEM$ [8].

•**Inter-PE Communication:** The sample step requires the propagation of resampled particles in each PE to every other PE. Often, only one of the models contributes significantly to the resampled set. Due to this, having a dedicated bus between each pair of PEs would lead to most of the interconnect being idle. The utilization is maximized when a single bus is used for the particle exchange. Particle distribution is then sequential and takes $N_s \times t$ cycles where t is the number of cycles needed to transfer a word (particle) over the bus. This depends upon the bus latency and the width of the bus B_w with respect to the b_p, the bit width used for representation of particles. Using the architecture of Fig. 3, this particle distribution is pipelined with the sample and importance steps. Hence a communication bottleneck can be completely avoided if $t = 1$ cycle and $B_w = b_p$, i.e., one particle is available at the input of the sample step of each PE per cycle. If $t > 1$ cycle, then a larger bus width will need to be chosen to prevent the bottleneck. For our FPGA implementation we design with $t = 1$ and $B_w = b_p$.

•**Communication between the PEs and the CU:** The required communication between the PEs and the CU in each recursion is only $2 \cdot K$ words, K values of W^k from PEs to CU and K values of R^k from CU to PEs. Typical values of K are between $2 - 6$. The RSR process in the CU has a computation time of K cycles. The PE-CU data exchange does not overlap with inter-PE communication. Hence we use the same bus of width b_p, for this communication . Though fixed point widths for various quantities depend upon the application, we have arrived at the following general relations by applying the statistical analysis method of [9]

$$b_p \approx b_w \quad (4)$$

$$b_{W^k} \approx 2 \times b_w = 2 \times b_p \quad (5)$$

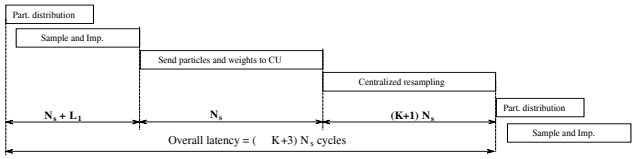
$$b_{R^k} = log_2 N_s << b_{W^k} \quad (6)$$

where b_{W^k}, b_w and b_{R^k} are the widths used in the fixed point representation of the sum of weights, individual weights and the replication factors, respectively.

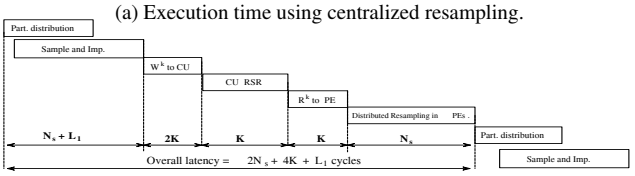
Using a bus of width b_p causes a communication latency of $K\lceil \frac{b_{W^k}}{b_p} \rceil$ in sending the sum of weights to the CU and a latency of K cycles in obtaining the replication factors. Due to the small value of K, this latency is of the order of $15 - 20$ cycles which is negligible compared to the overall execution time of the filter.

• **Arbitration:** The values of R^k are determined in the CU and these values decide the number of cycles that each PE should have write access to the bus. The CU incorporates a controller which consists of a counter and comparators to generate control signals that grant access of the bus to each PE^k for R^k cycles. During this time the sample step of this PE^k reads particles from the local memory while the sample step of the other PEs get their input particles over the bus. Due to the pipelining of the particle distribution with sample and importance steps, the time from the start of a recursion till all the weights of particles within a PE are calculated is $N_s + L_1$ cycles where L_1 is the start up latency of the PE datapath. The resampling in the PE takes $2 \times N_s - 1$ cycles if systematic resampling is used. This information is incorporated into the central controller to generate control signals for PE to CU transfer of sum of weights and CU to PE transfer of replication factors.

5. EVALUATION



(a) Execution time using centralized resampling.



(b) Execution time of the proposed architecture.

Fig. 4. Timing of multiple model SIRF.

The proposed architecture drastically reduces the communication bottleneck and resampling time thus greatly speeding up the filter execution. Fig. 4 compares the timings of the filters using a centralized resampling approach and the proposed architecture with distributed resampling. The overall execution time of the filter for the two cases is given by

$$T_{cent} = (K + 3) \cdot N_s + L_1 \ (cycles) \quad (7)$$

$$T_{dist} = 2 \cdot N_s + 4 \cdot K + L_1 \ (cycles). \quad (8)$$

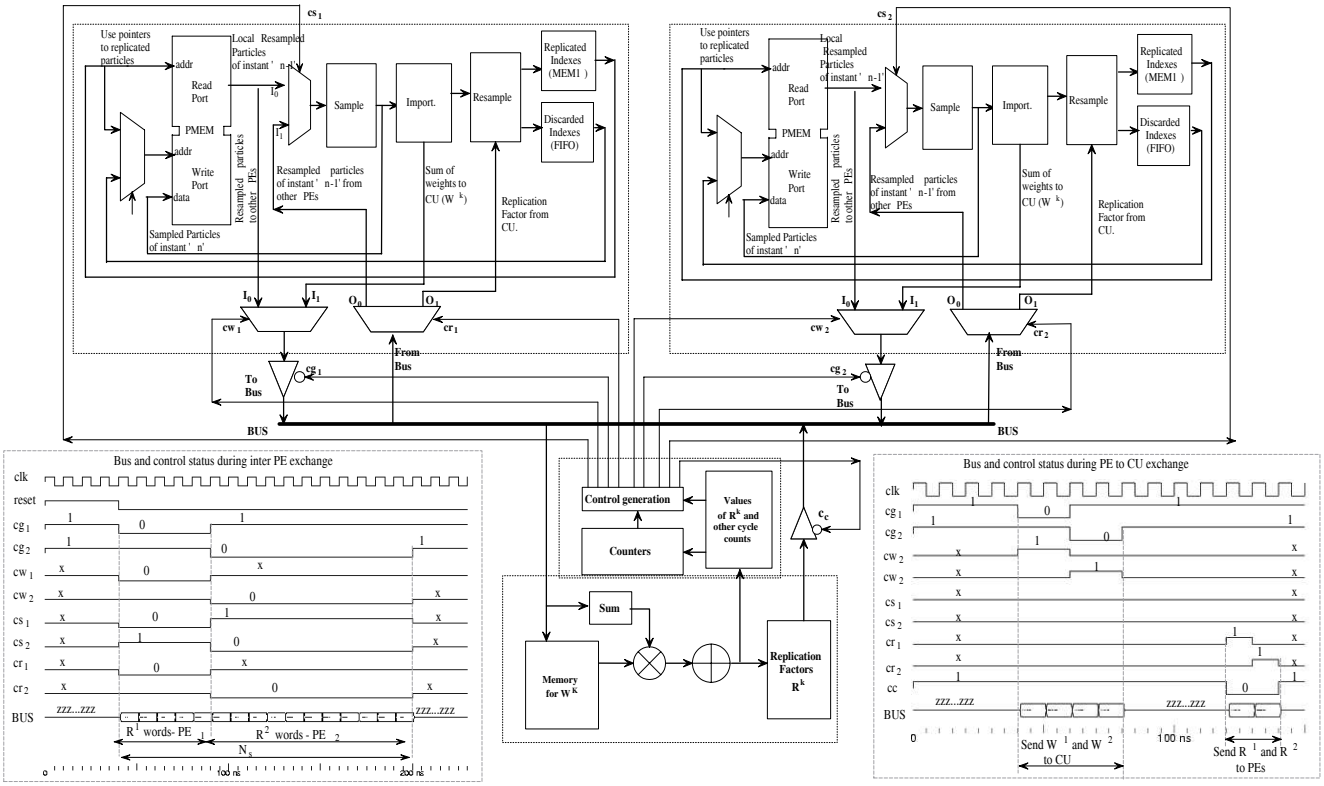


Fig. 5. Full Architecture with 2 PEs.

These expressions show that the architecture is highly scalable in that, increase in execution time by inclusion of more PEs and more particles per PE, is extremely small as compared to a centralized resampling approach.

Fig 5 shows the overall architecture of the filter with two PEs along with timing diagrams showing the status of the bus and associated controls during the inter-PE and CU-PE communication. On the FPGA platform the single bus is realized using macros which utilize long routing lines and tri-state buffers. Using the above architecture, the MM SIRF was implemented on a Xilinx Virtex II device for the tracking application [2] discussed in Section 2 with $K = 2$ and $N_s = 1000$. All the memory required in the PEs is realized using block RAMs and the necessary trigonometric and exponential functions are implemented using CORDIC units.

Unit	Slice	Slice.Reg	LUT	B. RAM	Mult	TBUF
PE_1	2,807	4,133	3,873	17	7	52
PE_2	4,597	6,872	5,423	17	10	52
CU	520	550	610	0	1	10

Table 2. Resource Utilization on a XC2V4000 device.

TABLE 2 summarizes the resource utilization of various units on the target platform. Post place and route timing analysis determines that the maximum clock rate for this design is 60MHz. This means that for our example, the filter can process input observations at 20KHz.

6. CONCLUSION

In this paper, we have developed a parallel architecture for efficient hardware implementation of an SIRF algorithm with multiple models. Compared to traditional approaches, this algorithm does not require knowledge of transition probabilities and handles nonlinear and non-Gaussian models more efficiently. The proposed architecture is based on a distributed resampling mechanism which greatly speeds up resampling and drastically reduces the data communication requirement. An efficient communication scheme is proposed which minimizes communication bottleneck and interconnect requirement. The architecture was used to implement the MM SIRF for a practical tracking example on the Xilinx Virtex II platform. The filter achieved speeds of up to 20KHz which is about 100 times faster than the implementation on a TMS320C67x DSP platform.

7. REFERENCES

[1] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dyan, "Interacting multiple model methods in target tracking: A survey," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 1, pp. 103-123, 1998.

[2] Y. Kim and K. Hong, "An IMM algorithm for tracking maneuvering vehicles in an adaptive cruise control environment," *International Journal of Control, Automation and Systems*, vol. 2, no. 3, pp. 310-318, 2004.

[3] Y. Boers and J. Driessen, "IMM particle filter," *IEE Proc. on Radar Sonar and Navigation*, vol. 150, no. 5, pp. 344-349, 2003.

[4] J. Lee and H. Ko, "Effective tracking for maneuvering mobile station via IMM filter in CDMA environment," *IEICE Transactions on Communications*, vol. E86-B, no. 11, pp. 3336-3339, Nov. 2003.

[5] A.Doucet, N. DeFeritas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*, Springer, 2001.

[6] S. McGinnity and G. Irwin, "Multiple model bootstrap filter for maneuvering target tracking," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 3, pp. 1006-1012, 2000.

[7] M. Bolić, *Architectures for the efficient implementation of particle filters*, Ph.D. thesis, Stony Brook University, 2004.

[8] A. Athalye, M.Bolić, S. Hong, and P. Djurić, "Generic architectures for sampling and resampling in particle filters," *EURASIP Journal of Applied Signal Processing*, vol. 17, pp. 2888-2902, Sept. 2005.

[9] S. Kim, K. Kum, and W. Sung, "Fixed point optimization utility for C and C++ based digital signal processing," *IEEE Transactions on Circuits and Systems - II.*, vol. 45, no. 11, pp. 1455–1464, Nov. 1998.