Hardware/Software Co-design of Particle Filter and Its Application in Object Tracking

Shih-An Li
Department of Electrical Engineering
Tamkang University
Taipei, Taiwan
lishyhan@gmail.com

Chen-Chien Hsu
Department of Applied Electronics Technology
Nation Taiwan Normal University
Taipei, Taiwan
jhsu@ntnu.edu.tw

Wen-Ling Lin and Jui-Pin Wang
Department of Electrical Engineering
Tamkang University
Taipei, Taiwan
winnie741015@gmail.com

Abstract—**This paper presents a hardware/software co-design method for particle filter based on System On Program Chip (SOPC) technique. Considering both the execution speed and design flexibility, we use a NIOS II processor to calculate weight for each particle and a hardware accelerator to update particles. As a result, execution efficiency of the proposed hardware/software co-design method of particle filter is significantly improved while maintaining design flexibility for various applications. To demonstrate the performance of the proposed approach, a real-time object tracking system is established and presented in this paper. Experimental results have demonstrated the proposed method have satisfactory results in real-time tracking of objects in video sequences.**

Keywords- particle filter; FPGA; object tracking; SOPC; HW/SW co-design;

I. INTRODUCTION

As an effective tool for estimating system states, particle filters based on Bayesian approach [5] use authentic Information to establish posterior Probability Density Function (PDF) [6],[7] to estimate the state model [8]. Because of its effectiveness, the particle filter [3],[4] has been widely used for solving problems with high dimensions for non-linear systems. However, one of the disadvantages of the particle filter is the complexity involved, particularly, for real-time embedded applications because complex mathematical operations of the particle filter will inevitably involved during each iteration. When the quantity of the particles increases, memory usage and computational effort increase accordingly, resulting in poor performance of the particle filter.

As an attempt to solve the above-mentioned problem, this paper presents a hardware/software co-design [22] approach to realize the particle filter using a SOPC-based approach [20],[21]. Because of hardware acceleration with limited usage of resources, execution speed of the proposed approach is much faster than that of pure software realization of particle filter. Also, hardware does not have to undergo redesign for various applications because of design flexibility of the proposed approach. Based on a proposed framework of the hardware/software co-design method, weight of the particles is calculated on the Nios II processer in this paper, while a FPGA hardware circuit is designed as an accelerator to update particles. As a result, calculation burden of the Nios II processer can be significantly decreased. In conventional particle filter algorithm, resampling is a very important step, which affects the distribution of the particles in the next generation. Traditional particle filters generally use the Stochastic Universal Sampling method [27] as the resampling mechanism, where particles with high weight are statistically selected many times. This leads to a loss of diversity among the particles as the resultant sample will contain many repeated points. This problem, which is known as sample impoverishment, is severe in the case of small process noise [28]. To address this problem, we will introduce the concept of tournament selection [19] widely used in genetic algorithm as the resampling mechanism, where a tournament size can be used to determine the selectivity pressure to choose particles from the current population. By doing so, floating-point operation of weight calculation of the particles in the design of hardware circuit can be avoided, thus improving the performance of the particle filter.

To demonstrate the effectiveness of the proposed approach, a real-time object tracking system is proposed in this paper, where a multi master-slave system architecture is adopted to effectively improve the system performance in image capturing and display. Together with the hardware/software co-design with the use of a particle filter circuit, the proposed framework reduces the resources required during the computation while providing design flexibility for real-time object tracking in video sequences.

II. PARTICLE FILTERS

Particle filter methods, particularly the sampling importance resampling (SIR) algorithm [2], are Monte Carlo (MC) methods that implement a recursive Bayesian filter by MC simulations. The key idea of particle filter methods is to represent the required posterior probability density function (pdf) by a set of random samples $X_k = \{x_k^1, x_k^2, ..., x_k^m\}$ with associated weights w_k^m to compute estimates based on these samples and weights, where *m* is the number of samples and *k* is the sample time. As the number of samples becomes very large, this MC characterization becomes an equivalent representation to the usual functional description of the posterior pdf [1], and the SIR filter approaches the optimal

Bayesian estimate. Essentially, a typical particle filter includes three computational steps:

A. Weight Calculating

Utilize measurement information z_k and particle x_k^m to calculate weight w_k^m associated with each particle x_k^m as a probability:

$$w_k^m = p(z_k \mid x_k^m) \tag{1}$$

These weights are then normalized by the sum over all particle weights to obtain the posterior distribution.

$$w_k^m = \frac{w_k^m}{\sum_{i=1}^{m} w_k^i} \tag{2}$$

B. Resampling

The tournament selection widely used in genetic algorithm is adopted in this paper to avoid the problem of sample impoverishment, where a tournament size "*K*" can be used to determine the selectivity pressure to choose particles from the current population. By doing so, we can draw particles from x_k^m according to the weight w_k^m.

C. Prediction

Draw samples x_k^m from state transition probability $p(x_k \mid u_k, x_{k-1}^m)$ conditioned on x_{k-1}^m at time k and control command u_k.

III. HARDWARE/SOFTWARE CO-DESIGN OF PARTICLE FILTER

Figure 1 shows a framework based on a hardware/software co-design approach for particle filters. We will use this framework to establish a real-time object tracking system.

As mentioned earlier in Introduction, most particle filters involve vast amount of computational complexity, thus resulting in challenges for real-time embedded applications. For example, particle filters implemented in embedded processor, such as Nios II processor, have very slow execution speed. As an attempt to provide design flexibility for various embedded applications and accelerate execution efficiency, a hardware/software co-design method using hardware description language Verilog is proposed in this paper, in which modular design is used to construct the major components, including a Particle Filter Accelerator and a Weight Calculating module. In what follows, we will describe these functional modules in more details.

A. Particle Filter Accelerator

The Particle Filter Accelerator module is a hardware circuit designed to accelerate the execution speed. As shown in Fig. 1, there are five important circuits in the Particle Filter Accelerator module, including (1) Initialization, (2) Particle Update, (3) Random Number Generator, (4) Particle Filter State Machine, and (5) On-chip Memory. Respective functionalities of these modules are described as follows:

1) Initialization module: The Initialization module is used to initialize particles required for the particle filiter. There are three components to achieve this objective, including (a) Boundary Comparison module, (b) State Controller module, and (c) FIFO module.

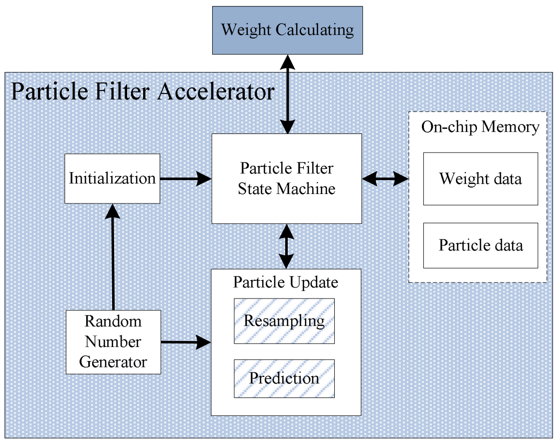

Figure 1. Hardware/software co-design of particle filter.

2) Particle Update module: This module is used to update particle data. This module is used to process the resampling and prediction steps to generate updated particle for the next gernation. There are three components in this module.

(a) Resampling module:

The Resampling module reads out the particle data and weight data from the Particle Filter State Machine module. Based on a tournament selection mechanism using random data #1, this module generate particles (Temp Particle Data) for use in the prediction step.

(b) Prediction module

The Prediction module generate New Particle Data by adding some random noises from Random Data #2 as perturbations to particles (Temp Particle Data) from the Resampling module. When all particles are generated, they will be trasnferred and saved into the On-chip memory through Particle Filter State Machine.

(c) State Controller #2 module:

This State Controller #2 module is used to control and coordinate the Particle Update module. First, the Resampling module will transfer the partilce data and weight data from Particle Filter State Machine module. And next, the State Controller #2 will wait the Prediction module to complete all particle prediciton. Finally, the State Controller #2 will send the new particle data to the On-chip memory through Particle Filter State Machine.

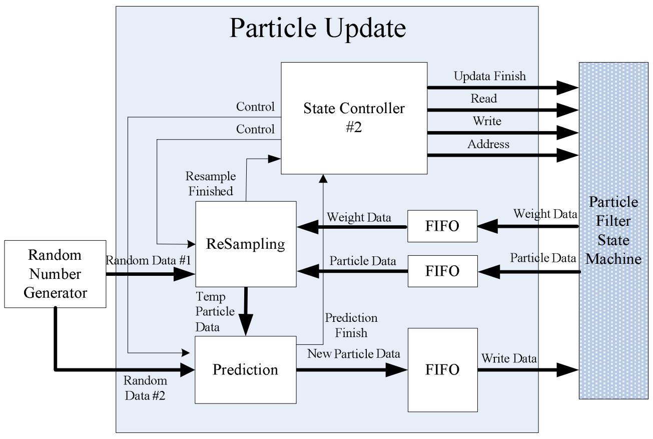

Figure 2. Particle update module.

3) Random Number Generator module*:* A Keep-It-Simple--Stupid (K.I.S.S) random number generator (RNG) algorithm is implemented [26] in this paper. The idea is to use simple, fast, individually promising generators to get a composite that will be fast, easy to code.

4) Particle Filter State Machine module: The Particle Filter State Machine module is designed using state machine to change state for access RAM data. There are three states in this state machine, including the initialization state, weight calculating state, and particle update state.

B. Weight Calculating

Fig. 3 shows the components of the proposed weight calculating module, including a Nios II processor for software weight assignment, a Hardware Weight Assignment Master module, and a Weight Slave. Because various applications need to be dealt with, the weight calculating module should accommodate the diversities of problems, and is therefore preferably implemented by a software module for evaluating weight functions in the first place.

The weight calculating module is designed with multi master-slave architecture, as shown in Fig. 3. Both Nios II processor and Hardware Weight Assignment Master can transfer the weights to Particle Filter Accelerator module by using Weight Slave module. Because Nios II processor can calculate weight by software code, therefore the use of Nios II processor has better design flexibility for various embedded applications. For a particular problem, we can use hardware implementation to speed up execution speed once prototype testing is completed.

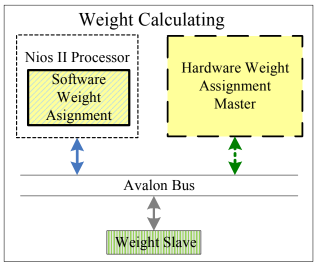

Figure 3. Weight calculating module.

IV. Object Tracking via the Proposed Particle Filter

Based on a multi master-slave [23] architecture, this paper proposed a SOPC design method to accelerate the processes of image capturing, image display, and object tracking via the particle filiter method. The architecture of the proposed proposed real-time object tracking system is shown in Fig. 4.

A. Image capture process

The CMOS Capture module will transfer image data to both the SSRAM and the SDRAM storage via an Avalon bus. The image data in SDRAM will be used to calculate the weight of the particles. The image data stored in SSRAM will be used to display the CMOS original image.

B. Image display process

The LTM Display module will transfer image data from SSRAM to LTM module and mark the object coordinate by a red rectangle on the screen.

C. Image tracking process

For real-time object tracking, each particle represents a coordinate in the image plane. In this paper, a Nios II processor is used to calculate weight associated with a particle. Because image data is very large, which requires significant computation, therefore a hardware circuit to calculate the weight for particles is desired. The weight calculating results will be transferred to the Particle Filter Accelerator module via the Weight Slave module. Finally, the particle with the highest weight value will be chosen and sent to the LTM module to display the object coordinate.

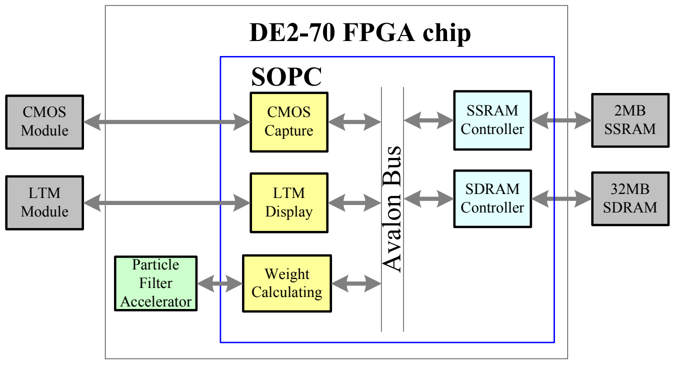

Figure 4. Architecture of the real-time object tracking system.

V. Experimental Results

In this paper, a DE2-70 development board by Altera [25] is used as an experiment platform for evaluating the performance of the proposed approach. We will compare the performance of particle filter implemented by both hardware and software design methods, followed by experiments of the proposed object tracking system, where a red ball is the object to be tracked. A set of 32 particles is used in the particle filter.

A. Performance of the Particle Filter Accelerator

TABLE I shows the time elapsed for two major computational steps of the particle filter via hardware and software implementation to search and track a red object. It is clear that hardware circuit has far better performance than software implementation. A speedup up to 7000 times can be obtained by using the Particle Filter Accelerator. However, hardware circuit design is more difficult than software realization. Therefore, we can use hardware/software co-design method to accelerate execution speed while maintaining design flexibility.

B. Performance of the object tracking system

TABLE II shows the performance of the real-time object tracking system. Two mask size of 32×32 and 64×64 are respectively used to validate the system performance of the particle filter, where weight calculation are implemented by either software and hardware implementation. As demonstrated in this table, a speedup of approximately 4 times can be achieved by the proposed hardware/software co-design method.

VI. Conclusions

In this paper, a SOPC-based framework for particle filters is proposed. Both Nios II processor as sofeware weight assignment or hardware weight assignment master as a hardware accelerator can be used to calculate weights for the particles. As a result, a balance between execution efficiency and design flexibility of the particle filter can be maintained. In the practical experiment, a multi master-slave architecture is proposed to demonstare image capturing, display, and object tracking in a video sequences. As demonstrated in the experimental esults, the proposed hardware/software code-design method has satisfactory performance for real-time object tracking.

TABLE I. EXECUTION PERFORMANCE OF PARTICLE FILTER

Computational step	**Time elapsed (µs)**		**Times**
	SW	**HW**	
Initialization	589.568	0.08325	7081.898
Re-sampling and prediction	4511.552	1.7925	2516.9

TABLE II. PERFORMANCE OF REAL-TIME OBJECT TRACKING VIA HW/SW CO-DESIGN

Method	**Mask size**	**Module**	**Time elapsed (ms)**	**Iteration (times/s)**
HW/SW co-design	32×32	Weight calculation (**SW**)	19.77598	50
		Particle update (**HW**)	0.0017925	
	64×64	Weight calculation **(SW)**	78.323196	13
		Particle update (**HW**)	0.001785	

Acknowledgments

This work was partially supported by the National Science Council, Taiwan, under Grant 99-2221-E-003-025-MY2.

References

[1] M. Isard and A. Blake, "Condensation--conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5-28, 1998.

[2] Y. Rui and Y. Chen, "Better proposal distributions: object tracking using unscented particle filter," *IEEE Computer Vision and Pattern Recognition*, pp. 786-793, 2001.

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

[4] P. M. Djuric, T. Lu, and M. F. Bugallo, "Multiple particle filtering," IEEE *International Conference on Acoustics Speech and Signal Processing'07*, pp. III-1181-III-1184, 2007.

[5] Y. C. Ho, and R. C. K. Lee, "A Bayesian approach to problems in stochastic estimation and control," *IEEE Transactions on Automatic Control*, vol. AC-9, pp. 333-339, 1964.

[6] J. H. Kotecha, and P. M. Djuric, "Gaussian Particle Filtering," *IEEE Transactions on Signal Processing*, vol. 51, no. 10, pp. 2592-2601, 2003.

[7] N. de Freitas, "Rao-Blackwellised particle filtering for fault diagnosis," *IEEE Aerospace Conference Proceedings*, vol. 4, pp. 4-1767-4-1772, 2002.

[8] Fujun Pei, Pingyuan Cui, and Yangzhou Chen, "Adaptive MCMC Particle Filter for Nonlinear and Non-Gaussian State Estimation," *Innovative Computing Information and Control*, pp. 494-494. 2008.

[9] G. Kitagawa and W. Gersch, *Smootness Priors Analysis of Time series*, New York: Springer-Verlag, 1996.

[10] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp, “A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp.174-188, 2002.

[11] H. Y. Cheng, “Object tracking using particle filter”, *CCL Technical Journal*, vol.112, pp.107-112, 2005.

[12] J. U. Cho, S. H. Jin, X. D. Pham, J. W. Jeon, J. E. Byun, and H. Kang, “A real-time object tracking system using a particle filter,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 2822-2827, 2006.

[13] J. U. Cho, S. H. Jin, X. D. Pham, and J. W. Jeon, “ Object tracking circuit using particle filter with multiple features,” *SICE ICASE International Joint Conference*, pp. 1431-1436, 2006.

[14] K. Nummiaro, E. Koller-Meier, and L. Van Gool, “An adaptive color-based particle filter,” *Image and Vision Computing*, vol. 21, Iss. 1, pp. 99-110, 10 January 2003.

[15] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, “Novel approach to nonlinear and non-gaussian bayesian state estimation,” *IEE Proceedings F -- Radar and Signal Processing,* vol. 140, no. 2, pp. 107-113, 1993.

[16] R. Niu, P. K. Varshney, M. Alford, A. Bubalo, E. Jones, and M. Scalzo, “Curvature nonlinearity measure and filter divergence detector for nonlinear tracking problems,” *8th International conference on Information Fusion*, pp. 1-8, 2008.

[17] Z. Cai and Z. Lin, “Fuzzy particle filter used for tracking of leukocytes,” *International Symposium on Intelligent Information Technology Application Workshop*, pp. 562-565, 2008.

[18] S. V. U. Ha and J. W. Jeon, “Combine kalman filter and particle filter to improve color tracking algorithm,” *International Conference on Control, Automation and Systems 2007*, pp. 558-561, 2007.

[19] B. L. Miller and D. E. Goldberg, “Genetic algorithms, tournament selection, and the effects of noise,” *Illinois Genetic Algorithms Laboratory Report*, no. 95006, July 1995.

[20] Altera Corporation, *SOPC Builder User Guide*, 2003.

[21] Altera Corporation, Designing With Nios & SOPC Builder, 2003.

[22] S.A. Li, C.C. Hsu, C.C. Wong, and C.J. Yu, “Hardware/Software Co-design for Particle Swarm Optimization Algorithm,” *Information Sciences*, doi:10.1016/j.ins.2010.07.017.

[23] Altera Corporation, *Avalon Interface Specifications*, March 2008.

[24] J. U. Cho, S. H. Jin, X. D. Pham, and J. W. Jeon, “Multiple objects tracking circuit using particle filters with multiple features,” *IEEE International Conference on Roboicts and Automation*, pp. 10-14, 2007.

[25] Terasic Corporation, URL: http://www.terasic.com.tw/en/

[26] G. Marsaglia, W.W. Tsang, “The Monty Python method for generating random variables,” *ACM Transactions on Mathematical Software*, vol. 24, no. 3, 341–350, 1998.

[27] James E. Baker, “Reducing Bias and Inefficiency in the Selection Algorithm,” *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pp. 14-21, 1987.

[28] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp, “A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp.174-188, 2002.