A PARAMETERIZED DESIGN FRAMEWORK FOR HARDWARE IMPLEMENTATION OF PARTICLE FILTERS

Sankalita Saha[1] *, Neal K. Bambha*[2] *and Shuvra S. Bhattacharyya*[1]

[1]Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, 20742.
[2]U.S. Army Research Laboratory,Adelphi, Maryland, 20783.
Email: {ssaha, ssb}@umd.edu, nbambha@arl.army.mil

ABSTRACT

Particle filtering methods provide powerful techniques for solving non-linear state-estimation problems, and are applied to a variety of application areas in signal processing. Because of their vast computational complexity, real-time hardware implementation of particle-filter-based systems is a challenging task. However, many particle filter applications share common characteristics, and the same system design can be reused with appropriate streamlining. To achieve this, a parameterized design framework for particle filters is proposed in this paper. In this framework, parameterization of system features that vary over specific implementations enables reuse of a generic design for a wide range of applications with minimal re-design effort. Using this framework, we explore different design options for implementing two different particle filtering applications on field-programmable gate arrays (FPGAs), and we present associated results on trade-offs between area (FPGA resource requirements) and execution speed.

Index Terms — Field programmable gate arrays, Parallel architectures, Recursive estimation.

1. INTRODUCTION

Particle filtering is a powerful, emerging methodology with a wide range of applications in science and engineering. Researchers from a variety of fields ranging from signal processing to statistics and econometrics use particle filters because of its potential for coping with difficult nonlinear and/or non-Gaussian noise problems. They are based on the idea of approximating the probability density functions (PDFs) of the state of a dynamic model by random samples (particles) with associated weights and propagating them across iterations based on a probabilistic model of the state update and the measurements. However, use of particle filters in real-time systems has been limited due to their computational complexity. A particle filter typically involves several complex mathematical operations that are invoked at every iteration of the filter, as well as a large a number of particles, which in turn results in high memory requirements. A possible solution for real-time implementation of such systems is parallelization with the use of multiprocessor systems [2]; but this is also restricted because of the presence of an unavoidable computing step — resampling — which is serial in nature, making it difficult to fully parallelize an implementation. Though various efforts have been made to derive distributed resampling algorithms ([3]), complete parallelization has not been possible. This suggests the exploration customized solutions.

Implementation of particle filter applications on hardware or hardware/software codesign platforms is further challenging due to the resource constraints on such platforms. Architectural design and memory management for hardware implementations explored before ([1]) mainly focus on one particle-filter-based application and optimize the various resulting subsystems. Design and implementation of a generic yet highly optimized architecture for all particle filter based systems is not possible because of the wide range of applications to which particle filtering techniques are applied currently and may be applied in the future. But, there are many applications that share similarities as far as particle filtering is concerned. A generic implementation framework that can be suitably and easily reconfigured for different applications would be of significant utility. A limited form of such a framework was explored by Hong et al., where two particle filtering algorithms are implemented on the same platform, and the system can be configured to use any one of them through switching mechanisms [5]. However, our methodology uses a much higher-level and hence more general approach. In particular, our methodology can be applied to any arbitrary set of particle filtering algorithms, rather than being constrained to a fixed, pre-defined set of algorithms.

In this paper a novel parameterized design framework to implement particle filter based applications on field programmable gate arrays (FPGAs) is proposed to enable comprehensive design space exploration of the whole system with attention to the interaction between the various subsystems and the different particle filtering parameter configurations that may be used across different applications. The class of applications towards which this work is geared is particle filters with underlying one-dimensional models. Extending such an architecture to multi-dimensional problems is also of great importance and is a useful direction for future work. Design and implementation details for experiments with two applications using the above framework are provided to demonstrate the application of the framework and illustrate its advantages.

2. SYSTEM DESIGN FRAMEWORK FOR PARTICLE FILTER IMPLEMENTATION

Particle filters provide a method for recursively estimating the unknown state X, from a collection of noisy observations. The state parameters to be estimated depend on the exact problem being considered. The state transition and observation models are

$$\textit{State Transition Model: } X_t = F(X_{t-1}, W_t), \text{ and} \quad (1)$$

$$\textit{Observation Model: } Y_t = G(X_t, V_t), \quad (2)$$

where W_t is the system noise and V_t is the observation noise. X_t represents the dynamically evolving state of the system, and Y_t is the observation vector of the system, which is corrupted by the measurement noise V_t at instant t. The particle filter estimates the state of the system X_t and updates it based on the received, corrupted observations.

A particle filter based system essentially consists of the following three computational steps: (1) *Sampling*: In this step, samples (particles) of the unknown state are generated based on the given sampling function which provide an estimate of the current state of the system and also propagate the particles from the previous time step to the current time; (2) *Weight Calculation*: Based on the observations, an importance weight is assigned to each particle and (3) *Resampling*: This step involves redrawing particles from the same probability density based on some function of the particle weights such that the weights of the new particles are approximately equal.

2.1 Overview

The architecture proposed in this paper for particle filters is based on the computational framework described above. The wide range of applications to which particle filtering techniques are applied prohibits the focus on a generic system architecture suitable for all applications. However, since there exist wide ranges of applications that use the same particle filtering algorithm with different state models, it is possible to develop a generic architecture for a subset of applications and streamline it for specific applications in this subset. The goal of this framework is to provide the user a systematic approach for such streamlining — with the ability to explore the various design tradeoffs between area and execution speed — and provide the capability to implement a wide range of applications with significantly reduced re-design effort.

To achieve this, a parameterized design framework is proposed, where the overall system is composed of small parameterized subsystems. Each such subsystem can be modified to the needs of a wide range of applications, as well as to final target constraints by setting appropriate parameters, such as the memory size, and the number of particles. An overview of a two-processing-element configuration of our architecture is given in figure 1. The framework essentially consists of an array of processing elements (PEs), and a resampling unit, along with a set of parameterized interfaces. A PE consists of three units, a PEcore, a weight calculation unit (WU), and a noise generator. Each of these units can operate independent of changes in functionality of the other units. However, the interaction between various units can change with the variation in the functionality of any one unit. These changes are handled by the interfaces so that the individual streamlined units need not be redesigned, which would require significant effort. The PEcores perform the sampling operation, while a separate weight calculation unit (WU) is used for calculating the weights. The PEcore as well as the WU interact with memory banks whose sizes are dependent on system parameters and hence they can be parameterized as well. The interfaces provide parameterized interaction between the various subsystems such as memory banks and the resampling interface and perform synchronization operations.

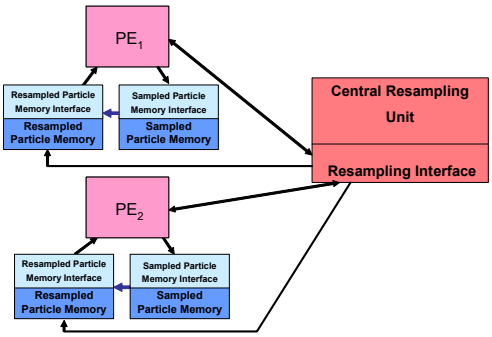

Fig. 1. Distributed particle filter architecture.

2.2 Design Framework

Figure 2 shows the overall design framework. We use Xilinx's System Generator for design and functional verification and the Xilinx ISE tool-set for synthesis. Xilinx System Generator provides a hardware library that consists of various architectural units, such as RAMs and adders, for modular design. It also allows the use of custom Verilog or VHDL modules for system design.

As mentioned in section 2.1, multiple processing elements (PEs) for the sampling and weight calculation step are used. Within a given PE, further pipelining can generally be used, but the degree to which pipelining can be employed is strongly dependent on the characteristics of the targeted application. The sampling and weight calculation operations involve complex mathematical operations, and thus impose restrictions on the number of PEs that can be implemented. The number of particles handled by each PE is

$$p = \lceil P/N \rceil, \tag{3}$$

where $\lceil x \rceil$ denotes the smallest integer that is greater than or equal to the real number x; P is the number of particles; and N is the number of PEs. A straightforward memory management scheme for particle storage and updating is used. Three memory banks or buffers are used for each PE for storing (1) sampled particles, (2) particle weights, and (3) resampled particles. Since the number of memory banks that are available on a given platform is limited, we have

$$N = M/3, \tag{4}$$

where M is the number of memory banks available on the targeted FPGA board. The area consumed by the associated memory banks directly depends on P and N. The observation data is stored in a shared memory between clusters of PEs. The memory interface for this buffer handles the read requests from the PEs. The reading

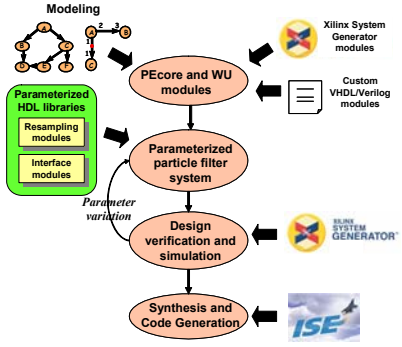

Fig. 2. Parameterized design framework.

from this memory for the i th operation can be overlapped with either the resampling step of the $(i-1)$ th operation, the sampling step of the i th operation, or both. However, if the system throughput is greater than or equal to the observation input rate, this interface becomes trivial as only a single buffer is required.

There are seven main interfaces corresponding to the operations of (1) observation data reading, (2) sampled particle memory interfacing, (3) resampled particle memory interfacing, (4) particle weight memory interfacing and (5) resampling unit interfacing. Among these, the reading of observation data is not dependent on N or P, while the rest are dependent on N and P. The resampling unit varies based on the resampling scheme being used and is functionally independent from the rest of the units. It is triggered when all the P particles have been processed for a given iteration. The resampling interface consists of a global address generator and a local address generator. The global address generator generates addresses for P particles and depends on P. These addresses are routed to individual PEs by the local address generator, which, thus, depends on both P and N. In this framework, systematic resampling has been used. However, this can be easily replaced with other sequential resampling mechanisms. Systematic resampling is often a preferred method due to its computational simplicity and good empirical performance. A library of these parameterized interfaces and resampling schemes are created using a combination of Xilinx System Generator hardware components and custom HDL modules.

The execution time for resampling directly depends on P and is constant over all iterations. Thus, the total execution time (in terms of clock cycles) for one iteration is,

$$T = T_{PECore} + L_{resampling} + L_{WU}. \quad (5)$$

where $L_{resampling}$ is the latency due to the resampling unit, L_{WU} is the latency induced by the WU unit, and T_{PEcore} is the execution time of PEcore, which for a fully pipelined PEcore is given as

$$T_{PEcore} = L_{PEcore} + \lceil P/N \rceil. \quad (6)$$

For systematic resampling, the latency is given by [1]:

$$L_{resampling} = 2 \times P - 1. \quad (7)$$

This signifies that the latency of the resampling unit increases directly with an increase in number of particles, and thus the latency will generally become a bottleneck for applications requiring very high P. For the first processing iteration, any initial latency that exists should be added to the latency model of (equation 6). Such initial latency may exist, for example, because of initial latency of the noise generator. For further details the reader is referred to [7].

3. EXPERIMENTS AND RESULTS

In this section, implementations for two different particle filter problems using the proposed framework are demonstrated along with corresponding experimental results. The two systems were designed and synthesized using Xilinx System Generator 9.1 and Xilinx ISE 9.1. The target device family was the Xilinx Virtex 4 FPGA. Although the FPGA board used in the experiments could support a clock frequency of 500 MHz, this frequency could not be attained in most cases. By varying the parameters appropriately, different implementations were obtained and various design options were explored.

3.1 Uni-variate Non-stationary Growth Model

The first application explored is an example of a one-dimensional non-linear system (typically studied in the context of stochastic systems) [3]. The model is as follows

$$X_t = 0.5 \times X_{t-1} + \frac{25 \times X_{t-1}}{1 + X^2_{t-1}} + 8 \times \cos(1.2 \times (t-1)) + W_t, \quad (8)$$

$$Y_t = X_t^2/20 + V_t, \quad (9)$$

where W_t and V_t, are zero-mean Gaussian white noise with variances 10 and 1, respectively. The execution of the PEs and the resampling units is fully pipelined. The above equations were mapped to appropriate Xilinx System Generator computation blocks to build the PEcore and the WU. The noise generation was performed using Xilinx's Gaussian white noise generator. This noise generator needs only periodic resetting to provide continuous output, thus the PE interface did not have to send requests for data. However, the initial latency of the generator is 10 cycles, which is present only for the first iteration. Additionally, Xilinx's lookup-table-based cosine generators were used. These are both fast and inexpensive (area-efficient) compared to standard CORDIC [8] cosine generators. We employed fully-pipelined multipliers and dividers. In the design, the WU uses an exponential calculation unit that uses a combination of a look-up table and a polynomial approximation method. Uniform random number generation for resampling is done using multiple-bit, leap-forward linear feedback shift registers (LFSRs) [4]. Parameterized interfaces were used to build the interconnections between the various subsystems.

3.2 Uni-dimensional Failure Prognosis Model

This practical particle filtering application is adapted from [6], where particle filtering is used to track crack faults in the blades of a turbine engine. The fault growth model is given by

$$X_t = X_{t-1} + \frac{1}{(X^5_{t-1} + X^4_{t-1} + X^3_{t-1} + X^2_{t-1} + 1)} + W_t, \quad (10)$$

$$Y_t = X_t + V_t, \quad (11)$$

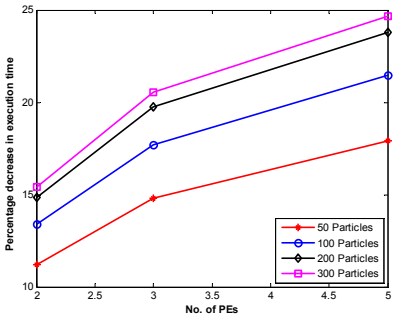

Fig. 3. Percentage decrease in execution time (1 iteration) for uni-variate non-stationary growth model implementation.

where W_t and V_t are zero-mean, Gaussian white noise with variances 10 and 1, respectively. The PEcore comprised of fully pipelined multipliers and a divider, all of which are fully pipelined cores from Xilinx. The Gaussian white noise generator, exponential calculation unit, and uniform random number generator used in section 3.1 are reused again. The resampling unit and interfaces were selected from the library and design space exploration is done by varying P and N.

3.3 Results

The percentage decreases in execution times compared to serial execution are shown in figures 3 and 4 for the various design cases while the tracking results are shown in figure 5. The results shown are for one iteration at steady state — i.e., not the first iteration, where there is additional latency due to the Gaussian white noise generator. The resource utilizations of the various implementations (200 particles) are shown in table 1 (the DSP48 utilization is same for both the applications due to their similar nature). The block RAM (BRAM) memory banks available for the Virtex 4 device family are each of size 18Kb, which is much higher than what is required for any of the implementations. Increasing P affects only the required memory bank sizes, thus the resource utilization remains the same for different numbers of particles. However, for applications with larger memory requirements, this would not be the case. Note that the execution times for both of the applications are similar because the latencies of the PEs are relatively small compared to the latency induced by P.

4. CONCLUSIONS.

In this paper, a new methodology for design, modeling and exploration for particle filters on reconfigurable system-on-chips (SoCs) has been presented. Our methodology uses the notion of parameterization to provide a useful tool for evaluating multiple design alternatives, and exploring the associated trade-offs in an efficient and intuitive manner. It also provides scope for implementing a wide range of applications with minimal redesign effort between different applications. For both of the applications that we examined, the execution speed was determined mainly by the number of particles, and thus, the latency of the resampling unit played a significant role in determining the overall execution time. This stresses the need to look further into methods for optimizing this unit. Although multiple expensive (area-consuming) computational units were used, the area constraint imposed by the target platform was met. The large underutilization of the block RAMs

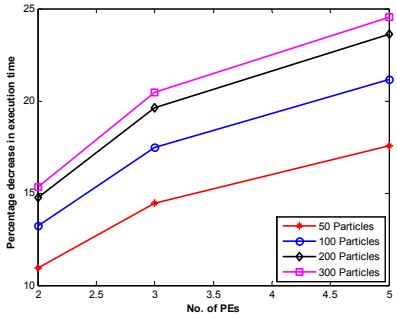

Fig. 4. Percentage decrease in execution time (1 iteration) for uni-dimensional failure prognosis model implementation.

Table 1. FPGA Resource utilization for uni-variate non-stationary growth model (Applcn 1) and unidmensional failure prognosis model (Applcn 2) implementation.

No. of PEs	Slices		Slice Flip-Flops		4 input LUTs		DSP48s	BRAMs
	Applcn1	Applcn2	Applcn1	Applcn2	Applcn1	Applcn2		
2	29.17%	19.81%	8.54%	6.13%	19.91%	14.27%	43.23%	15.63%
3	42.25%	28.39%	12.56%	8.94%	28.6%	20.18%	60.93%	23.44%
5	68.97%	45.57%	20.61%	14.57%	45.73%	31.98%	96.35%	39.06%

indicates potential for extending this architecture to systems with higher memory requirements, and this is an interesting direction for future work.

5. REFERENCES

[1] A. Athalye, M. Bolic, S. Hong and P. M. Djuric, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP Journal on Applied Signal Processing*, Vol. 2005 (2005), Issue 17, pp.2888-2902.

[2] A. S. Bashi, V. P. Jilkov, X. R. Li and H. Chen, "Distributed implementations of particle filters," *Proc. of the Sixth Intl. Conf. of Information Fusion*, 2003, Vol 2, pp. 1164-1171.

[3] B. P Carlin, N. G. Polson and D. S Stoffer, "A Monte-Carlo approach to nonnormal and nonlinear state space modelling," *Journal of American Statistical Association*. 1992, pp. 493-500.

[4] Chu, P., P., Jones, R., E., "Design Techniques of FPGA Based Random Number Generator," *Military and Aerospace Applications of Programmable Devices and Technologies Conf.*, The Johns Hopkins University- Applied Physics Laboratory, Sept. 1999.

[5] S. Hong, X. Liang, P. M. Djuric, "Reconfigurable particle filter design using dataflow structure translation," *IEEE Wkshp. on Signal Processing Systems*, 2004, pp. 325-330.

[6] M.Orchard, B. Wu and G. Vachtsevanos, "A Particle Filter Framework for Failure Prognosis," *Proc. of WTC2005 World Tribology Congress III*. Washington D.C., USA, Sept., 2005.

[7] S. Saha, "Design Methodology for Embedded Computer Vision Systems," *PhD thesis*, Deptt. of Electrical and Computer Engineering, University of Maryland, College Park, 2007.

[8] Jack E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. on Electronic Computers*, Sep. 1959.

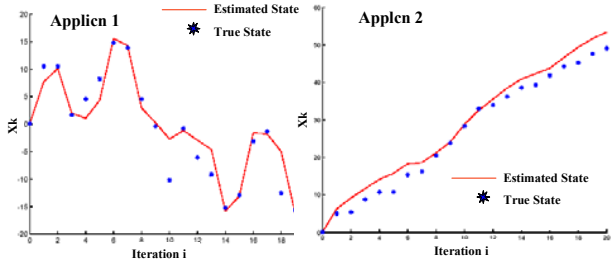

Fig. 5. Tracking results for uni-variate non-stationary growtl model (Applcn 1) and unidmensional failure prognosis model (Applcn 2) implementation.