Optimized FPGA Based Implementation of Particle Filter for Tracking Applications

Amin Jarrah, Mohsin M. Jamali, and Seyyed Soheil Sadat Hosseini
Department of Electrical Engineering & Computer Science
The University of Toledo
Toledo, Ohio, USA
amin.jarrah@rockets.utoledo.edu, mohsin.jamali@utoledo.edu, s.sadathosseini@gmail.com

***Abstract*— Particle filter has been proven to be a very effective method for identifying targets in non-linear and non-Gaussian environment. However, particle filter is computationally intensive. So, particle filter has been implemented on FPGA by exploiting parallel and pipelining approaches to reduce the computational burden. Our optimized FPGA implementation improves up to twelve times speed up. Also more speed ups are achieved with increasing number of particles.**

Keywords: *Particle filter, FPGA, High level synthesis tools, Tracking, VHDL.*

I. INTRODUCTION

Target Tracking [1, 2] can be defined as a sequential estimation of a variable or target of interest based on some observations over a period of time. Target tracking is performed by obtaining the position of the target. It is accomplished by performing position predictions and estimating the target positions in consecutive time scans. Different factors play important role in the efficiency of tracking process such as target parameters (position, velocity), target motion, and algorithm selection. However, the objective of the tracking process is to provide sequential prediction of the target using some observations. So, the tracking process can be divided into two stages; state stage which represents the values of target interest (prediction); and observation stage which represents some specified probabilistic relationship between observation and state (feature extraction).

Target tracking is very important for both military and civilian applications. Different versions of Kalman filters and particle filters are available in the literature [3]. Kalman filters are used when the system is linear and Gaussian whereas particle filters are popular when the system is non-linear and non-Gaussian. However, most of tracking methods are computationally intensive especially in Multi Target Tracking (MTT) radar systems. This leads to a heavy computation burden which prevents tracking to be performed in real-time. So, efficient hardware implementation will be required with the use of parallel processing platform. Hardware implementation should be efficient in terms of latency, area, power consumption, cost, and flexibility. Modern processors are multi-core and their parallel programming can be difficult and time consuming. They also have limited number of cores. One of the hardware implementation options is the use of Field Programmable Gate Arrays (FPGAs). Modern FPGAs have large amount of hardware resources including processing elements, memory and on-chip multipliers. They are reconfigurable and provide option of rapid prototyping. One drawback of FPGAs is that it requires programming using VHDL.

However, the design and implementation of complex applications using low level languages such as Verilog and VHDL is challenging. HDL is difficult to learn and far from traditional high level languages. The use of high level languages is easy to learn and modify. They also do not require deep knowledge of computer architecture. Therefore, High-Level Synthesis (HLS) tools [4-5] arise as an alternative to HDLs when using FPGAs. Meeus et. al [6] have compared twelve HLS tools based on many metrics and provided designers with a good overview of current HLS tools. Based on their evaluation, Vivado HLS tool [6] is used in this work since it is achieved the best performance based on their metrics evaluation.

The goal of this work is to implement particle filter on FPGA by exploiting parallel and pipelining architecture so it will be appropriate for real time applications. Inherent parallelism of particle filter is also explored. Optimization techniques such as pipelining, code in-lining, loop merging, and dataflow techniques are used for the concurrent execution of various operations. These techniques contribute to improve the throughput and latency. The FPGA implementation will then offer high speed computation with awareness of power, area, and cost.

The remainder of this paper is organized as follows: in Section II, a background on particle filter operation is presented. The analysis and optimization techniques for particle filter implementation are presented in Section III. Section IV provides particle filter simulation and synthesis results. Finally, Section V contains conclusions.

II. PARTICLE FILTER OPERATION

Particle filter [3, 7, 8, 9] offers following advantages:

- It is very effective in identifying the targets in an efficient and accurate manner.
- It can be useful in radar tracking applications with high cluttered environment.
- It is appropriate for tracking targets where the system is nonlinear and non-Gaussian.

Moreover, in the presence of multiple targets, tracking becomes difficult as the discrimination of target is inaccurate. Particle filters calculate the posterior density for different values of targets which is converted to likelihood functions

and helps to detect the number of targets. This approach can simultaneously handle processing, data association and tracking targets. Particle filter consists mainly of four steps as follows:

1. Prediction and measurement step.
2. Importance step.
3. Resampling step.
4. Output estimation.

Fig. 1 shows the computational steps to implement the particle filter. It shows data dependencies between various operations and their computational sequence.

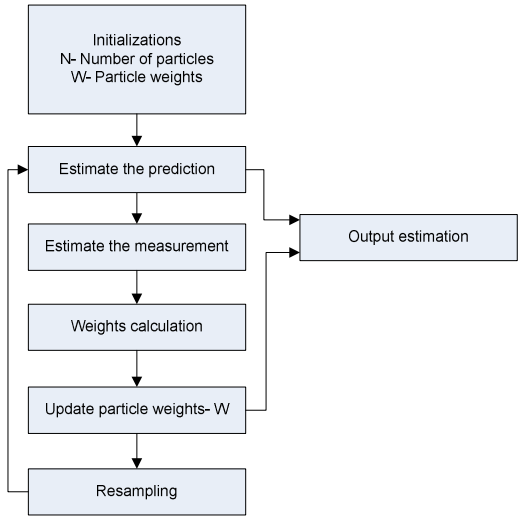

Fig. 1. Particle filter computational steps

The first step will initialize a nonlinear system prediction for assumed number of particles. Then the prediction and measurement of the new state of the target will be performed through the particles by considering a nonlinear system as follows:

$$x_k = f(x_{k-1}) + w_{k-1} \quad (1)$$

$$z_k = h(x_k) + v_k \quad (2)$$

Where x_{k-1} is the input target position, w_{k-1} and v_k are the process and measurement noise, f and h are nonlinear functions of process and observation vector, and z_k is the current observed measurement.

In order to show the robustness and effectiveness of particle filter for tracking purposes, a complex system with difficult state estimation is used in both the processes and measurements. It is expressed as [3]:

$$x_k = \frac{1}{2}x_{k-1} + \frac{25x_{k-1}}{1+x_{k-1}^2} + 8\cos[1.2(k-1)] + w_k \quad (3)$$

$$y_k = \frac{1}{20}x_k^2 + v_k \quad (4)$$

This highly nonlinear system is widely used for state estimation and comparison of efficiency and performance of new algorithms [10, 11].

The second step will involve the estimation and normalization of the particles weights [3] based on (5) and (6). This identifies the particles that have highest probability to represent the desired target. The weights of few particles will have large values as time progresses while the remaining weights of other particles will decrease in their values.

$$w_i = \frac{1}{(2\pi)^{m/2}|R|^{1/2}} \exp\left(\frac{-[z^* - h(x_{k,i}^-)]^T R^{-1}[z^* - h(x_{k,i}^-)]}{2}\right) \quad (5)$$

$$w_n^m = w_n^m / \sum_{m=1}^{M} w_n^m, \text{ for m=1,...,M.} \quad (6)$$

Resampling process in third step will remove small negligible weights particles and keep the largest one. This will improve the estimation of the future state by considering particles of higher posterior probability. This can be accomplished in different ways. One straightforward way can be performed in the following two steps:

- Generate a random number r [0,1].
- Accumulated the likelihoods w_n^m into a sum until the total sum is greater than r ($\sum_{m=1}^{j} w_m \geq r$). Then the new particle will be set to the old particle.

The final step will perform output calculations by multiplying the normalized weight by the predicted measurement of the particle as follows:

$$p(x_k | z_{1:k}) \approx \sum_{i=1}^{Ns} W_{ik} * x_k^i \quad (7)$$

III. ANALYSIS AND OPTIMIZATION TECHNIQUES

Particle filter is broken into set of regions as shown in Fig. 2 in order to exploit the parallel architecture of FPGA platform and the inherent parallelism of particle filter. However, Full parallelization of particle filter can't be achieved as particle filter is an iterative algorithm where the new particle prediction can't be performed until the resampled step is completed. Also, there is data dependency as the next computational step depends on the result of the current step. So, we need a way to arrange the operations of the algorithm to be performed in parallel without affecting its functionality. The implementation of particle filter can be improved by applying two optimization techniques:

- Merging technique: consecutive loops will be merged to reduce overall latency, increase sharing and optimization.
- Dataflow technique: allows sequential loops to be performed in a pipeline fashion to improve throughput and latency.

Merging technique allows the operations to be performed into one operation to reduce the additional overhead. For example, prediction step, measurement step, and weight calculation can be performed in one loop. This reduces the overhead from the unnecessary loops as additional N iteration loops for each step is removed. Also, weight normalization can be merged with resampling step to remove the N iteration loops of the normalization with little modification of resampling. Modified particle filter algorithm is as follows:

Merged Particle Filter

For i ⟵ 0 to N {
  Prediction step (Eqn. (3))
  Measurement step (Eqn. (4))
  Weight calculation (Eqn. (5)}
For j ⟵ 0 to N {
  Normalization step (Eqn. (6))
  Resampling step based on $\sum_{m=1}^{j} w_m \geq r$

So, the operations of the particle filter in this approach are merged instead of specified a separate loop of N iterations for each particle filter operation.

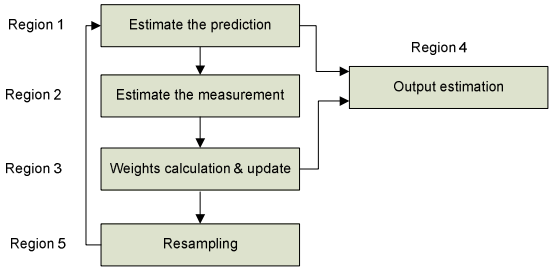

Fig. 2. Computational regions of particle filter

Moreover, it is not necessary for region 2 to wait until region 1 completes all its iterations. So, region 2 can start execution after the first iteration of region 1 is completed. This can be exploited by applying the dataflow technique between these regions where the data can flow asynchronously from the first region to the next one as shown in Fig. 3.

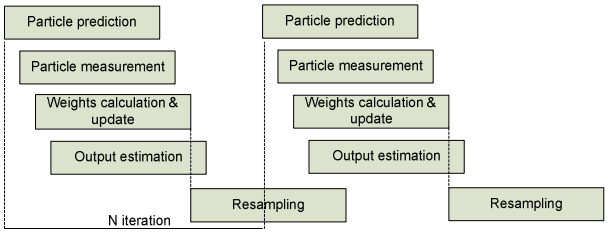

Fig. 3. Timing diagram for overlapping particle filter operations

So, regions 1 and 2 execute in a pipelined fashion until all iterations are completed. Region 1 forwards the value from current iteration to region 2 and begins with the next iteration at the same time region 2 can start execution. Similar approach is applied to other regions and is shown in the following pseudo code.

Parallelization Method

For each blip:
For k ⟵ 1 to N
$x_k = f(x_{k-1}) + w_{k-1}$ } This Loop can be executed in parallel.
End
⇩ Applying Loop Dataflow Pipelining
For k ⟵ 1 to Ns
$z_k = h*(x_k) + v_k$ } This Loop can be executed in parallel.
End
⇩ Applying Loop Dataflow Pipelining

For k ⟵ 1 to Ns
Wight calculation } This Loop can be executed in parallel.
End
⇩ Applying Loop Dataflow Pipelining
For k ⟵ 1 to Ns
$p(x_k \mid z_{1:k}) \approx \sum_{i=1}^{Ns} W_{ik} * x_k^i$ } This Loop can be executed in parallel.
End
⇩ Applying Loop Dataflow Pipelining
For k ⟵ 1 to Ns
Resampling based on $\sum_{m=1}^{j} w_m \geq r$.
End

Moreover, all these operations of particle filter are for only x-dimension. We need to implement the same operations for y-dimension. Fortunately, there are independent of each other and can be executed in parallel. Moreover, the steps of x-dimension and y-dimension can also be merged together to reduce overall latency, increase sharing and optimization.

IV. Experimental Results

Particle filter component is synthesized with the Xilinx ISE [12]. XA7A100TCSG324-1q FPGA device is used in this work. Table 1 lists the overall results in terms of hardware resources and power consumption.

Table 1. Resource utilization and overall implementation performance

Parameters	Resource Utilization
BRAM_18K	8
FFs	8724
LUTs	15644
Number of IOBs	230
Power Consumption (mW)	2434

The design of the control unit of a deep pipelined data-path that controls the scheduling has 101 stages. The number of clock cycles, throughput, clock period, and the execution

time from the start of execution until the final output is written for different number of vector size are shown in Table 2.

Table 2. Simulation time comparison of different number of particles

Parameter	Number of particles		
	250	500	1000
CLK Freq. (MHz)	145.34	145.34	145.34
Clock Period(ns)	6.88	6.88	6.88
Throughput (cycles)	410325	710773	1100800
Execution Time (ms)	2.823	4.89	7.57

The execution time of particle filter implementation before and after optimization is summarized in Table 3. The results show that the optimized FPGA implementation performs much better than un-optimized one. The superior performance of the optimized implementation is attributed to the exploitation of parallel architecture of FPGA and the parallelization of the particle filter. The result also shows that the optimized implementation achieves more speed-up with increasing number of particle which is attributed to the high parallelism and pipelining exploited in the array architecture.

Table 3. Execution time of different implementation

Implementation	Number of particles		
	250	500	1000
Before Optimization (ms)	21.93	43.94	87.5
Optimization with FPGA (ms)	2.823	4.89	7.57

In order to examine and verify particle filter method, it must be tested against highly non-linear and non-Gaussian data. So, complex system with difficult state estimation is considered in our work in both the process and measurements based on Eqns. (3) and (4). Figs. 4 and 5 show the particle filter estimation performance and the error rate over the true state respectively. It shows that the estimating state is close to the true states which validate the efficiency of particle filter operation.

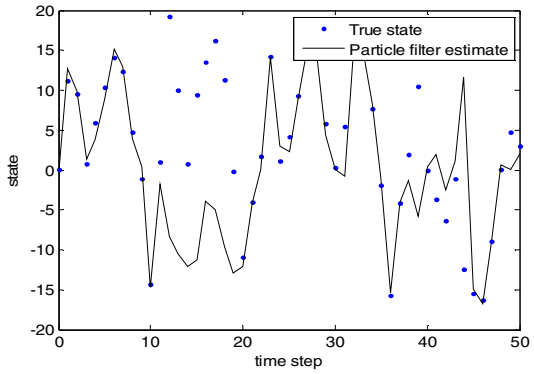

Fig. 4. Particle filter estimation performance

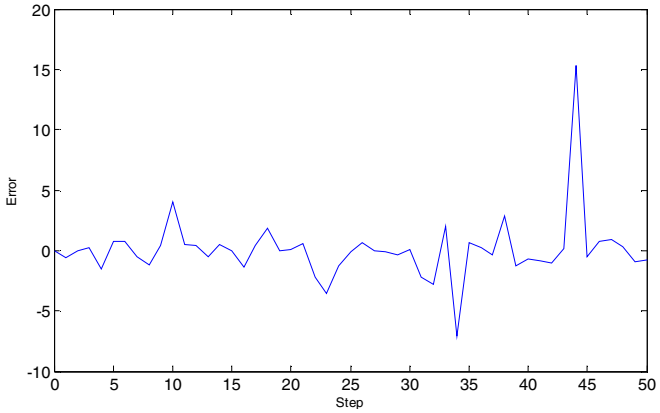

Fig. 5. Error rates of 100 particles over 50 time step

V. CONCLUSION

Particle filter has been implemented on FPGA by exploiting parallel and pipelining approaches to reduce the burden computation. The achieved speed-up of our optimized FPGA implementation over un-optimized one is improved up to 12 times. The result also shows that our optimized implementation achieves more speed-ups with increasing number of particles.

REFERENCES

[1] Y. Bar-Shalom and W. D. Blair, Multitarget-Multisensor Tracking: Applications and Advances, vol. III, Archtech House, Norwood, MS, 2000.

[2] S. Blackman and R. Popoli, Design and Analysis of Modern Tracking Systems, Artech House, Norwood, MA, 1999.

[3] http://pskla.kpi.ua/lib/2009/Simon.pdf

[4] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," IEEE Design & Test of Computers, vol. 26, no. 4, 2009.

[5] J. Cong, "A new generation of C-base synthesis tool and domain specific computing," in IEEE International SOC Conference, 2008, pp. 386-386.

[6] Wim Meeus, Kristof Van Beeck, Toon Goedemé , Jan Meel, and Dirk Stroobandt; "An overview of today's high-level synthesis tools"; Springer Science+Business Media; 12 August 2012.

[7] Y. Boers and J.N. Driesses, Multitarget Particle Filter Track Before Detect Application, IEEE Proceedings Radar, Sonar and Navigation, 2003, Vol. 151: p. 351-357.

[8] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon and Tim Clapp, A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking, IEEE Transactios on Signal Processing, Feb. 2002, Vol. 50(2): p.174-188.

[9] Niclas Bergman, Recursive Bayesian Estimation: Navigation and Tracking Applications, 1999, http://www.control.isy.liu.se/research/reports/Ph.D.Thesis/PhD579.pdf

[10] G. Kitagawa, "Non-Gaussian state-space modelling of non-stationary time series (with discussion)," Journal of the American Statastical Association, 82(400), pp. 1032-1063 (December 1987).

[11] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," IEEE Proceedings-F, 140(2), pp. 107-113 (April 1993).

[12] Artix7 FPGAs from Xilinx, Inc. (www.xilinx.com).