PARRALELIZATION OF NON-LINEAR & NON-GAUSSIAN BAYESIAN STATE ESTIMATORS (PARTICLE FILTERS)

Amin Jarrah, Mohsin M. Jamali, S. S. S. Hosseini
Dept. of Elect. Engg. and Computer Science
The University of Toledo, Toledo, Ohio, USA
mohsin.Jamali@utoledo.edu

Jaakko Astola and Moncef Gabbouj
Department of Signal Processing
Tampere University to Technology
Tampere, Finland

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 and may not achieve the real time requirements. So, it's desirable to implement it on parallel platforms by exploiting parallel and pipelining architecture to achieve its real time requirements. In this work, an efficient implementation of particle filter in both FPGA and GPU is proposed. Particle filter has also been implemented using MATLAB Parallel Computing Toolbox (PCT). Experimental results show that FPGA and GPU architectures can significantly outperform an equivalent sequential implementation. The results also show that FPGA implementation provides better performance than the GPU implementation. The achieved execution time on dual core and quad core Dell PC using PCT were higher than FPGAs and GPUs as was expected.

Keywords:

Index Terms— Field Programmable Gate Array (FPGA); Graphic Processing Unit (GPU); Parallel Architecture; Particle Filter; MATLAB Parallel Computing Toolbox (PCT)

1. INTRODUCTION

Target tracking has many applications. Multiple moving target tracking is needed when data is received from multiple sensors. The data may come from microphones, geophones, radars, sonars, video cameras, thermal and IR cameras. Therefore the problem is to deal with multiple sources with multiple moving targets in a much cluttered environment and their tracking in a real-time. Environment may be non-linear and noise can be non-Gaussian.

Tracking algorithms determine the location of sources and their motion [1-4]. They include source motion dynamics and array motion. Source motion is also estimated in the form of source velocity. Kalman filter has been used to predict the most likely position and velocity of a moving target. Kalman filter based approaches for target tracking are considered as linear and Gaussian in nature. Kalman filter provides a computationally efficient means to estimate the state of a process.

When the estimated and measurement processes have non-linear relationship then Extended Kalman Filter (EKF) approach may be used. An EKF linearizes the current mean and covariance when the system is considered as non-linear. It uses partial derivative of the process and measurement functions. Unscented Kalman Filter (UKF) which may be superior to EKF is also used and employs a set of parameterized points to model the non-linearity [1-4].

Bayesian approaches are useful for non-linear and non-Gaussian dynamic systems [1-5]. They can help in providing better performance and be able to associate estimates at different times. Bayesian methods use prior tracks of moving objects along with the likelihood of their current and future positions. Bayesian filters can also be very good to find true tracks in a very noisy environment but they are computationally expansive.

A Particle Filter (PF) falls under Bayesian system [1-5]. Particle filter assumes that the system is non-linear and non-Gaussian. It uses Monte Carlo simulation for handling non-linear systems. The particle filter algorithm will allow use of prior information and evaluate the likelihood function. It uses posterior distribution and weighted particles forming an independent hypothesis of the state of various tracks at certain time. Various particles are updated in time and weighted accordingly. Sometimes large number of particles is required to solve a given problem. This leads to a heavy computation burden which prevents tracking in real-time.

The goal of this work is to exploit parallel processing and implement particle filter on parallel architectures to achieve desired computation time appropriate for real-time applications. Parallel architectures can now be designed using multi-core processors, multi-core DSPs, GPUs and FPGAs. There are many parallel processing languages that are available but they are tied to certain type of platform. MATLAB also offers a Parallel Computing Toolbox (PCT).

This work is partially supported from Fulbright-Tampere University of Technology Scholar Award 2014-2015.

Particle filter has been implemented on FPGA and GPU platforms in this work. Parallel processing techniques are employed. We have also experimented with MATLAB Parallel Computing Toolbox (PCT) using two different machines and computed execution time for different number of particles. A performance comparison between FPGA, GPU and PCT is performed.

The remainder of this paper is organized as follows: Section two provides an overview of related work. Section three discusses various parallel processing platforms. Section four provides parallel algorithms for particle filter. Simulation results are presented in Section five. Conclusions are given in Section six.

2. RELATED WORK

Particle filters have been widely studied and many papers are available in literature for number of years. Their hardware implementations are very limited. Only few of many references are provided in this paper [6-13]. In [6], embedded implementation of the particle filter was performed, but their work does not consider any parallel implementation strategy or optimization techniques. In [7], hardware implementation was performed where some modifications of particle filter were considered. However, complicated data exchange patterns are required between processing elements and the control unit for their modifications. In [8], architecture to reduce memory usage was proposed by applying dual-port memory, but this prevents full parallelization of all the particles. Fixed precision implementation on performance was analyzed in [9].

A recent study applies a GPU-accelerated particle filter for accelerating the image processing steps of visual tracking application [10]. However, the particle filter is only partially executed on the GPU while most of the particle filter steps were performed on the host CPU. Another recent study proposed threshold-based resampling for high-speed particle filtering, but the hardware implementation of the particle filter is complex and needs more efforts where their implementation causes performance degradation [11]. A GPU based implementation of particle filter is described in [12]. They also show parallelization of the particle filter on GPU. This work extends to experimentation and implementation with FPGAs and MATLAB PCT.

3. PARALLEL PROCESSING PLATFORMS

Enormous computing and storage power can now be carried in one's hand. So far computer industry was following Moore's law with doubling of transistors every 18 months. This doubling of transistors in a single core processor came to an abrupt halt some times in 2004. The performance improvement is running out of steam as clock rates has topped out for a uniprocessor system. Power is growing at a faster rate than the clock frequency. Therefore, power dissipation and various thermal constraints are hampering the future growth. Power consumption has started to go up. Heat dissipation is becoming a challenge and reliability is also an issue. These factors of power consumption, clock speed, heat dissipation and reliability are favoring laws of diminishing results.

Semiconductor manufacturers have realized that performance increases can only come from increasing the number of cores. Multi-core is providing performance without increased power consumption and increased clock frequency. This work focusses on FPGAs and GPUs due to ease of their availability and software infrastructure. The idea is to achieve the real time requirements by exploiting parallel processing. This work is performed to minimize computation time using parallel processing schemes on available parallel platforms such as FPGAs and GPUs.

3.1 Field Programmable Gate Arrays (FPGAs)

FPGAs have large amount of logic slices, memory, interconnection and other resources that can be programmed and re-configured for a given task. FPGAs are programmed using standard Hardware Description Languages (HDL) such as VHDL or Verilog. FPGAs offer memory storage in the form of LUTs and block memory. External memory can also be interfaced with FPGAs via Double Data Rate (DDR3) memory. An efficient synthesizer can choose any of these memories as storage as part of the computational process. This work uses Xilinx's Vivado synthesizer [14-15] which allows incorporation of various optimization such as data flow, loop merging, loop unrolling and pipelining. It also eliminates programming in VHDL as it is perceived to be difficult to program by many university students (personally talking to students for many years). The developed parallel algorithm will be suitable for FPGA implementation which should be efficient in terms of latency, area, power consumption, cost, and flexibility.

3.2 Graphic Processing Units (GPUs)

The GPUs have multi-core architecture consisting of hundreds of cores. Each core contains a grids and each grid contains threads. There are threads, thread blocks, and grids of thread blocks that all differentiate themselves based on memory access and kernel execution. A thread block is a group of threads that have the ability to cooperate with each other and communicate via the per-Block shared memory. This type of architecture is attractive for offloading numerically intensive computations. The combination of high-bandwidth memories and hardware that performs floating point arithmetic at significantly higher rates than conventional CPUs makes graphic processors attractive targets for computational intensive algorithms.

This work uses NVIDIA GForce GTX 260 [16]. It contains 192 cores with graphic and processor clock frequency of 576 MHz and 1242 MHz respectively. It is programmed with Compute Unified Device Architecture platform (CUDA) [17]. A part of the program can be run either on CPU or GPU. Therefore GPU acts as coprocessor operating in Single Instruction Multiple Data (SIMD)

manner. A parallelized version of the program will run sequential code on CPU and its parallelized code on GPU. Generally programs are written in C and any part of the program that needs to be run on GPU can be specified as an extension under CUDA. There are libraries that can be used to facilitate programming in CUDA.

3.3 MATLAB Parallel Computing Toolbox (PCT)

MATLAB offers Parallel Computing Toolbox (PCT) [18]. It allows user to experiment with parallel processing using their parfor command. PCT offers parallel processing at loop level using their parfor command. Any of for-loops in the MATLAB code can be converted into parallel for loop with following restrictions [18-19].

- No nested parfor loops are allowed. You can have only one parfor loop. You can have either top level parfor loop or any other loop inside it.
- Execution of the loop should be independent of each other. There should be no data dependencies.
- A parallel for-loop "parfor" can also be inside a for loop.
- No deletion of variables is allowed.
- Different loops can be executed in different order.
- Printing and saving of data is allowed inside the parfor loop.

4. PARALLEL ALGORITHM (PARTICLE FILTER)

Particle filter consists of various computational steps such as prediction, measurement, weight calculation, resampling and output estimation [1-7]. It offers opportunities for exploiting parallel processing but also has data dependencies due to its iterative or updating nature. Its sequential version with data dependencies is as follows:

1. Predict the new state of the target.
2. Measure the new state of the target.
3. Estimate and update/normalize weights
4. Resample particles after removing particles with negligible weights.
5. Estimate the output.

Steps one and two predict and measure the new state of the target. This will be performed through the particles by using the given non-linear system. Therefore these steps can be parallelized with size of the loop equivalent to number of particles. Step three involves the estimation and normalization of the particle weights [1]. Its weight estimation can be parallelized. The step identifies the particles that have the 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.

Resampling process in step four will remove small negligible weights particles and keep the larger one. This will improve the estimation of the future state by considering particles of higher posterior probability. Step five performs output calculations by multiplying the normalized weight by the predicted measurement of the particle.

4.1 Parallel implementation on FPGAs

Xilinx FPGAs and their Vivado synthesizer tool are used for implementation of particle filter. Full parallelization of particle filter can't be achieved since there is a data dependency between its computational steps. So, particle filter is broken into a set of regions to exploit the opportunity of executing particle filter in parallel on FPGAs. Details of parallelization of particle filter on FPGA can be found in our previous work in [13].

The implementation of particle filter can be improved by applying two optimization techniques:

- Loop merging technique.
- Dataflow technique.

The loop merging technique allows the operations to be performed in one operation and reduces 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. The modified particle filter algorithm is as follows:

Merged Particle Filter

For i ⟵ 0 to N {
Prediction step
Measurement step
Weight calculation}
For j ⟵ 0 to N {
Normalization step
Resampling step based on $\sum_{m=1}^{j} w_m \geq r$ }

Moreover, it is not necessary for step 2 to wait until step 1 completes all its iterations. So, step 2 can start execution after the first iteration of step 1 is completed. This can be exploited by applying the dataflow technique between these steps where the data can flow asynchronously from the first step to the next one. Parallelization of particle filter on Xilinx FPGA was successfully completed and results are shown in Section five.

4.2 Parallel implementation on GPUs

Particle filter requires initialization of particle positions. This step can be fully parallelized by N threads where each thread is assigned to each particle as shown in the following code:

Parallel Particle Positions Initialization

int i ⟵ Thread index;

{Xpart[i] = x + sqrt (Q) * random(); //Q is process noise covariance

Steps one and two require predicting and measuring the state of each particle, respectively. These steps can be fully parallelized by N threads where each thread is assigned to each particle as shown in the following code:

Parallelization of Particle Prediction State

```
int i⟵ Thread index;
    {Xpartminus[i] = Xpart[i]+sqrt (Q) * random(); }
```

Parallelization of Particle Measurement State

```
int i⟵Thread index;
    {XXpart[i] = H * Xpartminus[i];} //H is the measurement transition matrix}
```

Moreover, particle filter requires repeating the basic calculation of random function generator for all the particles in each iteration. This typically involves a significant amount of calculation which represents a small fraction of the total computational effort. So, random number generation is also parallelized where each thread generates one random value instead of a large number of values.

Step four requires calculating particle weights. This step can be fully parallelized by N threads as shown in the following code:

Parallelization of Particle Prediction State

```
int i⟵Thread index;
  {Vhat = xmeasured[k] - XXpart[i];
 q[i] = (1 / sqrt (r) / sqrt (π) * exp(-vhat^2 / 2 / R);}  //R is measurement noise covariance
```

Step five requires the normalization of the particle weights which needs summing all the particle weights. In order to parallelize this operation efficiently, we have divided the particle weights by multiple threads where each thread sums a group of weights elements into its local variable. Then the global summation will be performed by adding these local variables. However, this technique requires a lock and barrier synchronizations to ensure correct results since the global summation variable is shared by all the threads. Following code is used to find the summation value of particle weights.

Parallelization of Particle Weights Summation

```
int index⟵  Thread index;
Tile_Size=Weights_Vector/# of threads.
x=index*Tile_Size.
y =(index+1)*Tile_Size.
For i⟵ x to y {
Local_sum_X= Local_sum_X + W_x[i];
__syncthreads();
Global_sum_X+=Local_sum_X;
Unlock_suncronization();
```

Normalization step requires dividing each particle weight by the total weights summation. This can also be fully parallelized by N threads as shown in the following code:

Parallelization of Particle Weights Normalization

```
int i⟵Thread index;
  {q[i] = q[i] / Global_sum_X;}
```

4.2 Parallel implementation using PCT

MATLAB code for particle filter was parallelized using PCT. Excellent information given in tutorial [19] was used to restructure the program. All possible for-loops were converted into parfor loops. A simplified parallel algorithm for the particle filter is as follows:

```
*/
Initialize the particle filter.
for k = 1 : tf  (No. of points)
        parfor  i = 1 : N (No. of particles)
        Compute predicted next state
        Compute measurement values
        Compute weights
        end
 Normalize the likelihood of each a priori estimate.
Accumulate all values of weights
        parfor  i = 1 : N
        Perform normalization
        end
        parfor  i = 1 : N
        Perform resampling
        end
end
*/
```

Developed parallel algorithm was executed on a dual core Dell Optiplex 960 with MATLAB and PCT. Number of cores were initialized to two. System used in this case, offered limited parallel processing opportunity due to use of only two parallel loops. Simulation results were higher than FPGA and GPU results as expected. Execution time for parallel version with dual core was also higher than running without use of parallel processing in traditional MATLAB sequential fashion. This is due to limited number of cores and large overhead in establishing workers (parallel operations). Execution time somewhat improved when program was run on a newer machine (Intel(R) core(TM) i5-2500 CPU) with four cores, 64-bit operating system under Windows 7. Execution times are given in the next section.

5. SIMULATION RESULTS

In order to examine and verify particle filter method, it must be tested against highly non-linear and non-Gaussian data. A commonly used and very popular complex system given in [5] is considered in our work in both the process and measurements. Plot not shown in this paper indicates

that the estimating state is close to the true states which validate the efficiency of particle filter operation [13].

The particle filter component is synthesized with the Xilinx ISE [13]. XA7A100TCSG324-1q FPGA device is used in this work. Number of used FF, LUTs and I/O blocks are 8724, 15644 and 230 respectively. Power consumption was 2.434 W. The design of the control unit of a deep pipelined data-path that controls the scheduling has 101 stages. The execution time from the start of execution until the final output is written for different number of vector sizes are shown in Table 1.

The execution times of particle filter implementation for the un-optimized, FPGA, GPU and PCT implementations are summarized in Table 1. The results show that the optimized FPGA and GPU implementations perform much better than un-optimized one. The superior performance of the optimized implementation is attributed to the exploitation of parallel architecture of the FPGA and the parallelization of the particle filter. The result also shows that the optimized implementation achieves more speed-ups with increasing number of particles which is attributed to the high parallelism and pipelining exploited in the array architecture.

The execution time for PCT was much higher due to use of only two workers (cores). Moreover only for-loops were converted into parfor loops. There is system overhead and in establishment of two workers that contributed to higher execution time. MATLAB PCT offered a quick way to see how parallel processing can be employed in a simplified fashion. Execution time somewhat reduces with the use of quad-core and newer machine.

Table 1. Execution time using different platforms

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
Optimization with GPU (ms)	3.07	6.48	11.86
MATLAB PCT on 2-core (ms)	11464	12336	19001
MATLAB PCT on 4-core (ms)	8169	5394	7771

6. CONCLUSIONS

An optimized FPGA and GPU implementations of particle filter are developed. This helps in minimizing the execution time to reach the real time requirements. The experimental result shows that the FPGA platform provides lower execution time when compared with GPU implementations. Execution time with PCT was much higher as expected as it offered limited number of cores and had lot of system overheads.

7. REFERENCES

[1] Dan Simon, "Optimal State Estimation," Wiley InterScience, 2006.

[2] 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.

[3] 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.

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

[5] N. J. Gordon, D. J. Salmond, A. F. M. Smith,"Novel Approach to nonlinear/non-Gaussian Bayesian state estimation," IEE Proceedings-F, Vol. 140, No. 2, pp. 107-113, April 1993.

[6] FLECK, S., AND STRASSER, W. Adaptive probabilistic tracking embedded in a smart camera. Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on 3 (2005), 134–134.

[7] M. Bolic, A. Athalye, P. M. Djuric, S. Hong, "Algorithmic Modification of Particle Filters for Hardware Implementation," Proc. European Signal Processing Conference, Vienna, Austria, 2004.

[8] 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, Vo. 17, pp. 2888 – 2902, 2005.

[9] M. Bolic, S. Hong, and P. M. Djuric, "Finite Precision Effect on Performance and Complexity of Particle Filters for Bearing-Only Tracking", Signals, Systems and Computers, Vol.1, pp. 838 – 842, 2002.

[10] J. Brown and D. Capson, "A framework for 3-D model-based visual tracking using a GPU-accelerated particle filter," IEEE Trans. Vis. Comput. Graph., vol. 18, no. 1, pp. 68–80, Jan. 2012.

[11] Z. Shi, Y. Zheng, X. Bian, and Z. Yu, "Threshold-based resampling for high-speed particle PHD filter," Progr. Electromagn. Res., vol.36, pp. 369–383, 2013.

[12] G. Hendeby, J. D. Hol, R. Karlson, F. Gustafson,"A Graphic Processing Unit implementation of the Particle Filters," EURASIP Conference, Poznan, 26007. Pp. 1639-1643.

[13] Amin Jarrah, M. M. Jamali, Soheil Hosseini "Optimized FPGA Based Implementation of Particle Filter for Tracking Applications," National Aerospace Conference (NAECON), Dayton, Ohio, June 2014.

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

[15] High-Level Synthesis Vivado Simulator from Xilinx, http://www.xilinx.com

[16] NVidia Corporation, "NVIDIA GeForce GPU Architecture Overview, Technical Brief", 2012.

[17] NVidia Corporation, CUDA Software Development Kit 5.0. 2012; Available from: https://developer.nvidia.com/cuda-downloads.

[18] MATLAB Parallel Computing Toolbox (PCT) www.mathworks.com

[19] Van Te Chow,"Beginners' Guide to MATLAB Parallel Computing," http://vtchl.uiuc.edu/node/537