# A Multithreaded Framework for Sequential Monte Carlo Methods on CPU/FPGA Platforms

Markus Happe, Enno Lübbers, and Marco Platzner

University of Paderborn, Germany
`(cyclash, enno.luebbers, platzner)@upb.de`

**Abstract.** Sequential Monte Carlo techniques are among the principal tools for the on-line estimation of the state of a non-linear dynamic system. We propose a framework for the multithreaded implementation of the widely popular *sampling importance resampling* (SIR) method on hybrid CPU/FPGA systems. The framework is based on the multithreaded reconfigurable operating system ReconOS which allows for an easy repartitioning of threads between hard- and software. We demonstrate the framework on a case study for visual object tracking and evaluate the performance of different hardware/software partitionings.

## 1 Introduction

Sequential Monte Carl (SMC) methods, also denoted as particle filters, have become a popular tool for on-line estimation of the state of a non-linear, dynamic system. Particle filters are iterative methods that track a number of possible state estimates, the so-called particles, across time and gauge their probability by comparing them to measurements. State estimation of non-linear dynamic systems is an important problem with applications in areas as diverse as object tracking, network packet processing, and sensor networks. Because the particles tracked by SMC methods are independent, many calculations can be parallelized and are, additionally, amenable to implementation in dedicated logic. For example, Athalye et al. [1] developed methods and architectures for accelerating the resampling stage of the SIR algorithm while at the same time reducing the memory requirements for hardware implementations. Our framework adapts their technique for parallelizing the resampling stage. At the same time, the sequential algorithm governing these computations is implemented more efficiently using general purpose CPUs. These facts make today's modern platform FPGAs, which integrate fine-grained reconfigurable logic with dedicated CPUs, a promising implementation target for particle filters in embedded systems.

The novel contribution of this paper is a framework for implementing SMC methods on hybrid CPU/FPGA platforms. The framework significantly simplifies the design of particle filters following the *sampling importance resampling* (SIR) algorithm. By building on top of our multithreaded reconfigurable operating system ReconOS [2] the framework handles the recurring tasks of particle data transfer and thread control, letting the designer focus on the application-specific details of an individual particle filter. Because the operating system transparently supports both software and hardware threads using one unifying programming model, the designer can quickly create different hardware/software partitionings to react to changing application and performance requirements. The work of Saha et al. [3], who developed a parameterizable framework for the hardware implementation of particle filters, bears some similarity to our approach in that it provides an interface for the model definition of a particle filter. However, their proposed framework targets a hardware-only implementation of the filter and thus significantly differs from our flexible, multithreaded hardware/software approach.

## 2 Sequential Monte Carlo Methods

We are considering a dynamic system with state $x_t$ at a given time $t$. As there can be uncertainty in the state information, we model the initial system state by its probability distribution $p(X_0)$, where $X_0$ is a random variable describing the state at time $t = 0$. The *system model* is a Markov process of first order. Thus, $p(X_t|X_{t-1})$ denotes the probability distribution of the system's current state given the system's previous state. We assume that the system state can only be tracked by measurements $y_t$, which may be influenced by noise. The relation between measurements and system states is described by the *measurement model.* The distribution $p(Y_t = y_t|X_t)$ describes the probability of the current measurement given the system's current state.

To predict the current state based on past measurements, $p(X_t|y_1, \cdots, y_{t-1})$, our framework closely follows the *sampling importance resampling* (SIR) algorithm. SIR is one of the most widely used sequential Monte Carlo methods, which allow system state estimates to be computed on-line while the state changes, as it is the case for tracking algorithms. For a more thorough discussion of the theoretical foundations of SMC methods we refer to [4].

An SIR filter usually manages a fixed number of possible system state hypotheses $x_t^i$, also called particles. Ideally, these particles approximate the distribution of the system state, $p(X_t)$. The SIR algorithm distinct stages iterated over discrete time steps. Figure 1 graphically represents one iteration. The individual stages are:

- **Sampling:** To follow the state during subsequent iterations ($t \leftarrow t + 1$), the system model is used to obtain a possible new state for every particle $\tilde{x}_t^i$ based on its last state $x_{t-1}^i$. Formally, this corresponds to drawing or *sampling* the new particle state from the distribution $p(X_t|X_{t-1} = x_{t-1}^i)$. Now, the set of particles $\tilde{x}_t^i$ forms a prediction of the distribution of $X_t$.
- **Importance:** The measurement model is evaluated for every particle and the current measurement to determine the *likelihood* that the current measurement $y_t$ matches the predicted state $\tilde{x}_t^i$ of the particle. Formally, this corresponds to evaluating $p(Y_t = y_t|X_t = \tilde{x}_t^i)$. The resulting likelihood is assigned as a weight $w_t^i$ to the particle and indicates the relative quality of the state estimation. In Figure 1, particles with higher weights are drawn as larger circles.

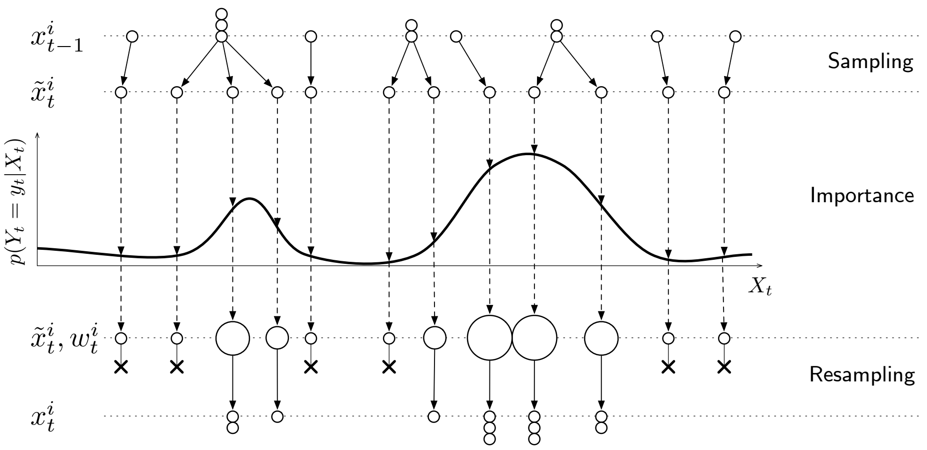


**Fig. 1.** Sampling importance resampling algorithm

– **Resampling:** Particles with comparatively high weights are duplicated and particles with low weights are eliminated. The distribution of the resulting particles $x_t^i$ approximates the distribution of the weighted particles before resampling.

## 3 Particle Filter Framework

All particle filters using the SIR algorithm rely on the same underlying algorithmic structure. Hence, a substantial part of the functionality, code, and – in the case of hybrid CPU/FPGA systems – hardware circuitry can be re-used, allowing for a framework-based design approach. Our particle filter framework takes care of common tasks shared by all SIR implementations, such as data transfer and control flow, and lets the designer focus on the application-specific tasks, such as system and measurement modeling.

The distinctive feature of our particle filter framework is the use of multithreaded programming across the hardware/software boundary. Because SIR filters are composed of a mixture of highly parallel, data-centric tasks and purely sequential, control-dominated tasks, hybrid CPU/FPGA systems appear as a natural choice for their implementation. Multithreading on such hybrid systems is enabled by the reconfigurable operating system ReconOS [2], which allows an application to be modularized into threads that are executed either in software on the system's CPU or in hardware in the FPGA's reconfigurable fabric. All threads communicate and synchronize using the same programming model primitives such as semaphores, message boxes, and shared memory.

Figure 2 shows the basic structure of an SIR implementation using our framework. The particles cycle through the three stages sampling, importance, and resampling. Each of these stages can have an arbitrary number of software and

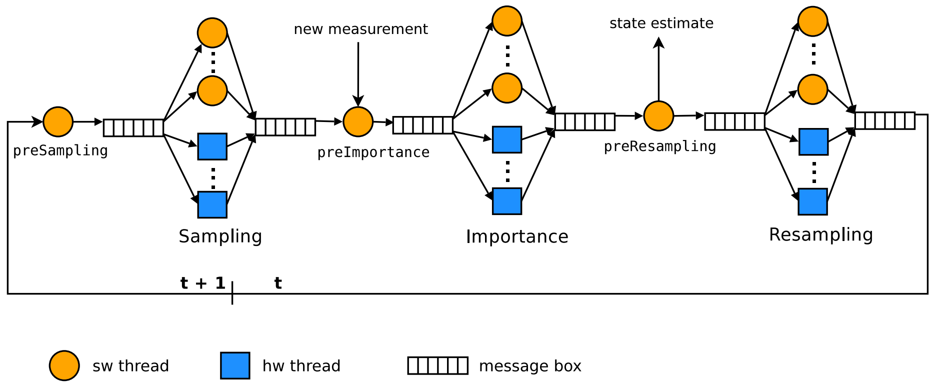


**Fig. 2.** Structure of an SIR filter implementation

hardware threads. This execution structure is created and initialized by a software thread, which also sets the initial number of threads for each stage.

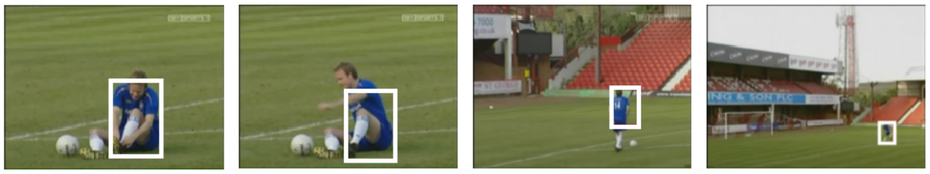
Generally, the number of HW and SW threads for each of the stages will depend on the availability of computing resources, i.e. CPU utilization factors and logic area. The other threads of the framework are mainly control-dominated or show limited potential for parallelism and are therefore implemented in software. Access to the needed data, the control flow, as well as necessary operating system services for communication and synchronization are completely managed by the framework. Due to the fully transparent communication and synchronization across the hardware/software boundary provided by ReconOS, the designer can easily change the hardware/software partitioning by instantiating a different number of hardware and software threads in order to adapt to changing performance requirements and resource constraints.

## 4 Case Study

To demonstrate our framework, we have implemented a prototype system for visually tracking moving objects in a video sequence. The application uses the software implementation by Hess [5] as a template and reference. Given an initial tracking target, the particle filter estimates the object's position and size in each subsequent frame of the video sequence. An example of the desired tracking behavior can be seen in Figure 3.

We have implemented two prototypes on a Virtex-II Pro XC2VP30 FPGA and a Virtex-4 XC4VFX100 FPGA. On both systems, an embedded PowerPC 405 CPU, clocked at 300 MHz, runs the SW threads and the OS kernel, while the remaining system including buses and HW threads is clocked at 100 MHz.

To illustrate how our framework assists the designer in resolving the rather involved problem of selecting the appropriate HW/SW partitioning, the performance of our object tracking prototype has been evaluated using the following



(a) Frame 5 (b) Frame 90 (c) Frame 150 (d) Frame 260

**Fig. 3.** Object tracking in a video sequence (video sequence soccer)

five different hardware/software partitionings: all threads run in software (*sw*), only one thread executes in hardware ($hw_s$: sampling, $hw_i$: importance, $hw_r$: resampling), or all threads of the SIR stages execute in hardware ($hw_a$), while the remaining threads run in software. Figure 4 shows the performance of the individual partitionings, as well as the reference implementation on a PC equipped with an Intel Pentium 4 HT 630 processor (template), measured in clock cycles per frame. The measurements were performed on the soccer video sequence displayed in Figure 3.

The overall drop of frame processing times in the first 250 frames of the sequence is due to the soccer player retreating into the background, which reduces the amount of image data to be considered per particle in the importance stage. Figure 4 also shows that partitionings which compute the importance stage in hardware, $hw_i$ and $hw_a$, generally show the best performance. This indicates that, in our prototype, the importance stage is the computationally most expensive part of the application. However, the attainable speedup of any hardware-implemented stage is considerably data dependent – a partitioning that performs better than another during one part of the video can easily be worse during another part. This is demonstrated by the $hw_r$ partitioning at frame 200 or 275. Also, computing more than one stage in hardware does not necessarily lead to better performance, as can be seen when comparing the $hw_i$ and $hw_a$ partitionings between frames 100 and 250.

Overall, the experimental results serve to highlight the advantage of a framework that allows for an easy HW/SW repartitioning. Using our framework, an application designer can quickly explore the design space to determine the suitable partitioning. The framework even allows for changing the partitioning during runtime driven by characteristics of the input data. Adaptive repartitioning approaches are part of our ongoing research.

The resource requirements of the object tracking prototype as implemented on both target FPGAs are given in slices in the top table of Figure 4. It can be seen that the area used for the different partitionings is about the same on both target FPGAs. The most resource-consuming partitioning, $hw_a$, could only be mapped to the Virtex-4 board. The bottom table of Figure 4 shows FPGA resource utilization after synthesis for the three hardware threads of the framework, divided into slices for the functions provided by the framework, the user functions, and the ReconOS interface core.

|  | **Virtex2-Pro** | **Virtex-4** |
|---|---|---|
| sw | 5305 | 5355 |
| $\mathrm{hw}_s$ | 9449 | 9577 |
| $\mathrm{hw}_i$ | 8794 | 8955 |
| $\mathrm{hw}_r$ | 8314 | 8430 |
| $\mathrm{hw}_a$ | — | 15958 |

| **component** | **S** | **I** | **R** |
|---|---|---|---|
| **framework** | 1086 | 918 | 1227 |
| **user** | 1033 | 481(*) | - |
| **OS** | 1353 | 1353 | 1353 |
| **total** | 3479 | 2873(*) | 2580 |

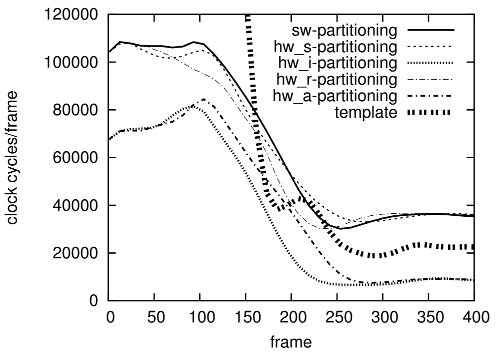


**Fig. 4.** Resource usage (slices) and performance ($\frac{\text{cycles}}{\text{frame}}$) of different partitionings

## 5 Conclusion

In this work, we have presented a multithreaded framework for the implementation of SIR filters on hybrid CPU/FPGA systems, and demonstrated it using a case study for visual object tracking which uses the repartitioning capabilities of the framework to quickly explore the design space with regard to hardware/software partitioning of the filter's threads.

Our ongoing research focuses on two areas. First, we are working on refining and optimizing the structure of the framework to allow for a more efficient usage of hardware resources. Second, we are working on approaches to adapt the partitioning during runtime. Here, we are trying to exploit the target FPGA's partial reconfiguration capabilities for better utilization of the logic resources.

## Acknowledgement

This work was supported by the German Research Foundation under project number PL471/2-1.

## References

1. Athalye, A., Bolić, M., Hong, S., Djuric, P.M.: Generic Hardware Architectures for Sampling and Resampling in Particle Filters. EURASIP Journal on Applied Signal Processing (2005)
2. Lübbers, E., Platzner, M.: ReconOS: An RTOS supporting Hard- and Software Threads. IEEE Int. Conf. on Field Programmable Logic and Applications (2007)
3. Saha, S., Bambha, N.K., Bhattacharyya, S.S.: A Parameterized Design Framework for Hardware Implementation of Particle Filters. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (2008)
4. Doucet, A., de Freitas, N., Gordon, N.: Sequential Monte Carlo Methods in Practice. Springer (2001)
5. Hess, R.: Particle Filter Object Tracking (2006) `http://web.engr.oregonstate.edu/~hess`.