Heterogeneous Reconfigurable System for Adaptive Particle Filters in Real-Time Applications

Thomas C.P. Chau[1], Xinyu Niu[1], Alison Eele[3],
Wayne Luk[1], Peter Y.K. Cheung[2], and Jan Maciejowski[3]

[1] Department of Computing, Imperial College London, UK
{c.chau10, niu.xinyu10, w.luk}@imperial.ac.uk
[2] Department of Electrical and Electronic Engineering, Imperial College London, UK
p.cheung@imperial.ac.uk
[3] Department of Engineering, University of Cambridge, UK
{aje46, jmm1}@cam.ac.uk

Abstract. This paper presents a heterogeneous reconfigurable system for real-time applications applying particle filters. The system consists of an FPGA and a multi-threaded CPU. We propose a method to adapt the number of particles dynamically and utilise the run-time reconfigurability of the FPGA for reduced power and energy consumption. An application is developed which involves simultaneous mobile robot localisation and people tracking. It shows that the proposed adaptive particle filter can reduce up to 99% of computation time. Using run-time reconfiguration, we achieve 34% reduction in idle power and save 26-34% of system energy. Our proposed system is up to 7.39 times faster and 3.65 times more energy efficient than the Intel Xeon X5650 CPU with 12 threads, and 1.3 times faster and 2.13 times more energy efficient than an NVIDIA Tesla C2070 GPU.

1 Introduction

Particle filter (PF), also known as sequential Monte Carlo (SMC) method, is a statistical method for dealing with dynamic systems having nonlinear and non-Gaussian properties. PF has been applied to real-time applications including object tracking [1], robot localisation [2], speech recognition [3] and air traffic control [4]. However, PF operates on a large number of particles resulting in long execution times, which limits the application of PF to real-time systems.

In this paper, an adaptive algorithmic and architectural approach using reconfigurable hardware is proposed for PF in real-time applications. An adaptive PF algorithm is employed to dynamically adjust the size of particle set for reduced computation complexity. A heterogeneous reconfigurable system (HRS) consisting of a multi-core CPU and an FPGA is developed for the adaptive PF algorithm. As most of the PF operations can be performed independently, the algorithm suits ideally for implementation in FPGA which consists of thousands of customisable resources and dedicated digital signal processing units to exploit parallel processing. The challenge is to meet real-time requirement by organising the operations in streaming manner to maximise throughput and hide latency.

The contributions of this paper include:

1. Adaptive PF algorithm: the computational complexity of PF is reduced through adapting the size of particle set dynamically, and the algorithm is optimised for hardware acceleration (Section 3).
2. Heterogeneous architecture: fully pipelined computations of the PF are streamed through the FPGA kernel, while control-oriented computations are handled by the host CPU (Section 4).
3. Energy saving by run-time reconfiguration: FPGA is reconfigured to low-power mode dynamically (Section 4).
4. Prototype: a robot localisation application is implemented on an FPGA based on the proposed adaptive PF approach. Compared to a non-adaptive implementation, the idle power is reduced by 34% and the overall energy consumption decreases for 26-34% (Section 5).

2 Background and Related Work

This section briefly outlines the PF algorithm. A more detailed description can be found in [5]. PF estimates the state of a system by a sampled-based approximation of the state probability density function. The state of a system in time step t is denoted by X_t. Sequences of control information and observations are denoted by U_t and Y_t respectively. Three pieces of information about the system are known a-priori: a) $p(X_0)$ is the probability of the initial state of the system, b) $p(X_t|X_{t-1}, U_{t-1})$ is the state transition probability of the system's current state given a previous state and control information, c) $p(Y_t|X_t)$ is the observation model describing the likelihood of observing the measurement at current state.

PF approximates the desired posterior probability $p(X_t|Y_{1:t})$ using a set of P particles $\{\chi_t^i|i = 1, ..., P\}$ with their associated weights $\{w_t^i|i = 1, ..., P\}$. X_0 and U_0 are initialised. This computation consists of three iterative steps.

1. **Sampling**: A new particle set $\widetilde{\chi}_t^i$ is drawn from the distribution $p(X_t|X_{t-1}, U_{t-1})$, forming a prediction of the distribution of X_t.
2. **Importance**: Likelihood $p(Y_t|\widetilde{\chi}_t^i)$ of each particle is calculated. The likelihood indicates whether the current measurement Y_t matches the predicted state $\widetilde{\chi}_t^i$. Then a weight w^i is assigned to the particle. The higher the likelihood, the higher the weight.
3. **Resampling**: Particles with higher weights are replicated and the number of particles with lower weights are reduced. With resampling the particle set has a smaller variance. The particle set is then used in the next time step to predict the posterior probability subsequently. The distribution of the resulting particles χ_t^i approximates $p(X_t|Y_{1:t})$.

The parallelism of particles in PF means it can be accelerated using specialised hardware with massive parallelism and pipelining. In [1], an approach for PF on a hybrid CPU/FPGA platform is developed. Using a multi-threaded programming model, computation is switched between hardware and software during run-time to react to performance requirements. Resampling algorithms and architectures for distributed PFs are proposed in [6].

Adaptive PFs have been proposed to improve performance or quality of state estimation by controlling the number of particles dynamically. Likelihood-based adaptation controls the number of particles such that the sum of weights exceeds a pre-specified threshold [7]. Kullback Leibler distance (KLD) sampling is proposed in [8], which offers better quality results than likelihood-based approach. KLD sampling is improved in [9] by adjusting the variance and gradient of data to generate particles near high likelihood regions. The above methods introduce data dependencies in the sampling and importance steps, so they are difficult to be parallelised. An adaptive PF is proposed in [10] that changes the number of particles dynamically based on estimation quality. Our previous work [11] extends their techniques for multi-processor system on FPGA. The number of particles and active processors change dynamically but the performance is limited by soft-core processors. In [12], a mechanism and a theoretical lower bound for adapting the sample size of particles is presented.

3 Adaptive Particle Filter

This section introduces an adaptive PF algorithm where the size of the particle set is changed in each time step. The algorithm is inspired by [12], and we optimise the algorithm to exploit the capability of FPGAs to support streaming and deep pipelines.

Algorithm 1 shows the processing step of PF. The first part is performed on the FPGA because the operations can be scheduled sequentially to maximise throughput. The second part is done on the CPU because the operations involve non-sequential or random access of data, such as sorting and resampling, that cannot be mapped efficiently to FPGA's streaming architecture.

Sampling and Importance (line 4 to 5): In time step $t-1$, the number of particles is $P_{t-1} \leq P_{max}$. A particle has dimensions $d = 1, ..., dim$. Particle set $\{\chi^i_{d,t-1}\}$ is sampled to $\{\widetilde{\chi}^i_{d,t}\}$ and importance weight $\{\widetilde{w}^i\}$ is assigned, where $i = 1, ..., P_{t-1}$. For simplicity, we denote the set of $P_{d,t-1}$ particles $\{\widetilde{\chi}^i_{d,t}\}$ as a vector $\widetilde{X}_{d,t} = \{\widetilde{\chi}^1_{d,t}, \widetilde{\chi}^2_{d,t}, ...\widetilde{\chi}^{P_{t-1}}_{d,t}\}$. $\{\widetilde{X}_{d,t}\}$ and $\{\widetilde{w}^i\}$ give an estimation of the current system state at dimension d.

Calculate the lower bound of particle set size (line 6): This step derives the smallest number of particles that are needed to bound the approximation error. This number, denoted as $\widetilde{P}_t$, is referred to as the lower bound of sampling size. It is calculated by Equation 1 to 5.

$$\widetilde{P}_t = \max_{d}^{dim} \widetilde{P}_{d,t} \tag{1}$$

Algorithm 1 Adaptive PF algorithm

```
1: $P_0 \leftarrow P_{max}$, $\{X_0\} \leftarrow$random set of particles, $t = 1$
2: for each step $t$ do
3:     —On FPGA—
4:     Sample a new state $\{\widetilde{\chi}_t^i\}$ from $\{\chi_{t-1}^i\}$ where $i = 1, ..., P_{t-1}$
5:     Calculate unnormalised importance weights $\widetilde{w}^i$ and accumulate the weights as $w_{sum}$
6:     Calculate the lower bound of sample size $\widetilde{P}_t$ by Equation 1 to 5
7:     —On CPU—
8:     Sort $\widetilde{\chi}_t$ in descending $w^i$
9:     if $\widetilde{P}_t < P_{t-1}$ then
10:        $P_t = max\left(\lceil \widetilde{P}_t \rceil, P_{t-1}/2\right)$
11:        Set $a = 2P_t - P_{t-1}$ and $b = P_t$
12:        –Do the following loop in parallel–
13:        for $i$ in $P'$ do
14:            $\widetilde{\chi}_t = \frac{\chi_t^a w^a + \chi_t^b w^b}{w^a + w^b}$
15:            $w^i = w^a + w^b$
16:            $a = a + 1$ and $b = b - 1$
17:        end for
18:    else if $\widetilde{P}_t \geq P_{t-1}$ then
19:        $a = 0$ and $b = 0$
20:        for $i$ in $P_t - P_{t-1}$ do
21:            if $w^a < w^{a+1}$ and $a < P_t$ then
22:                $a = a + 1$
23:            end if
24:            $\widetilde{\chi}_t^{P_{t-1}+b} = \widetilde{\chi}_t^a/2$
25:            $\widetilde{\chi}_t^a = \widetilde{\chi}_t^a/2$
26:            $w^{P_{t-1}+b} = w^a/2$
27:            $w^a = w^a/2$
28:            $b = b + 1$
29:        end for
30:    end if
31:    Resample $\{\widetilde{\chi}_t^i\}$ to $\{\chi_t^i\}$ where $i = 1, ..., P_t$
32: end for
```

$$\widetilde{P}_{d,t} = \sigma_d^2 \cdot \frac{P_{max}}{Var(\widetilde{X}_{d,t})} \tag{2}$$

$$\begin{aligned}\sigma_d^2 = \frac{1}{(w_{sum})^2} \cdot \sum_{i=1}^{P_{t-1}} \left(\widetilde{w}^i \cdot \widetilde{\chi}_{d,t}^i\right)^2 - 2 \cdot E(\widetilde{X}_{d,t}) \cdot \sum_{i=1}^{P_{t-1}} (\widetilde{w}^i)^2 \cdot \widetilde{\chi}_{d,t}^i \\ + \left(E(\widetilde{X}_{d,t})\right)^2 \cdot \sum_{i=1}^{P_{t-1}} \left(\widetilde{w}^i\right)^2\end{aligned} \tag{3}$$

$$Var(\widetilde{X}_{d,t}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_{t-1}} \widetilde{w}^i \cdot (\widetilde{\chi}_{d,t}^i)^2 - \left(E(\widetilde{X}_{d,t})\right)^2 \tag{4}$$

$$E(\widetilde{X}_{d,t}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_{t-1}} \widetilde{w}^i \cdot \widetilde{\chi}_{d,t}^i \tag{5}$$

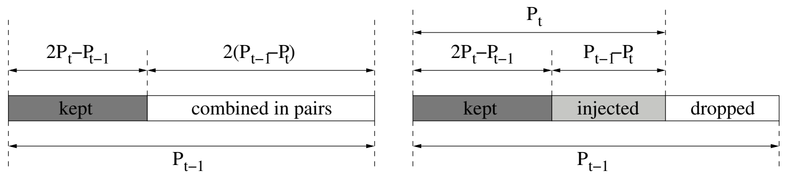
$\widetilde{w}^i$ is unnormalised. To calculate normalised weights w^i, a trivial approach is to stream data through the FPGA twice, one for accumulating the sum of weights w_{sum} and one for dividing the weights $\widetilde{w}^i$ by w_{sum}. This method is inefficient as it reduces the throughput of the FPGA by half. Without degrading the performance, our design computes w_{sum} and $\widetilde{w}^i$ simultaneously as data stream through the FPGA. As shown in Equation 3 to 5, correct values of σ_d^2, $Var(\widetilde{X}_{d,t})$ and $E(\widetilde{X}_{d,t})$ appear at the last cycle by dividing the last piece of data by w_{sum}.

Meanwhile, Equation 3 to 5 involve accumulation which requires feedback of data in the previous cycle. If accumulation is performed in floating-point representation, the feedback path would take multiple clock cycles and greatly reduce throughput. Therefore, we use fixed-point data path such that the delay of the feedback path is kept in one clock cycle. The precision is designed to ensure that no overflow or underflow occurs.

Particle set size tuning (line 8 to 30): The particle sample size is tuned to fit the lower bound P_t.

First, the particles are sorted in descending order according to their weight. Then, as the new sample size can increase or decrease, there are two cases.

- **Case I: Particle set reduction when $\widetilde{P}_t < P_{t-1}$**
  The lower bound P_t is set to $max\left(\lceil\widetilde{P}_t\rceil, P_{t-1}/2\right)$. Since the new size is smaller than the old one, some particles are combined to form a smaller particle set. Fig. 1 illustrates the idea of particle reduction. The first $2P_t - P_{t-1}$ particles with higher weights are kept and the remaining $2(P_{t-1} - P_t)$ particles are combined in pairs. As a result, there are $P_{t-1} - P_t$ new particles injected to form the target particle set with P_t particles. To remove loop dependency, we restrict that particles are combined deterministically. Therefore, each iteration of the loop can be processed independently and accelerated using multiple threads. The complexity of the loop is in $\mathcal{O}\left(\frac{P_{t-1}-P_t}{N_{parallel}}\right)$, where $N_{parallel}$ indicates the level of parallelism.



(a) Combining the last $2(P_{t-1} - P_t)$ particles with lower weights (b) P_t new particles are formed

Fig. 1. Particle set reduction

- **Case II: Particle set expansion when $\widetilde{P}_t \geq P_{t-1}$**
  The lower bound P_t is set to $\widetilde{P}_t$. Some particles are taken from the original set and are incised to form a larger set. The particles with larger weight would have more descendants. As shown in line 18 to 30, the process requires picking the particle with the largest weight at each iteration of particle incision.

Since the particle set is pre-sorted, the complexity of particle set expansion is $\mathcal{O}(P_t - P_{t-1})$.

Resampling (line 31): Resampling is performed to pick P_t particle from $\widetilde{X}_t$ to form X_t. The process has a complexity of $\mathcal{O}(P_t)$.

4 Heterogeneous Reconfigurable System

This section describes the proposed heterogeneous reconfigurable system (HRS) which makes use of run-time reconfiguration for power and energy reduction. The architectural diagram of HRS is shown in Fig. 2. The system consists of an FPGA board and a multi-threaded host CPU. The FPGA resources are customised to a deeply pipelined structure and the CPU performs coordination of particles. The FPGA has its own on board dynamic random-access memory (DRAM) because the amount of data is too large to be stored on-chip.

For the sampling and importance processes, the computation is independent for every particle. Therefore, particles are organised in a stream that is fed to the FPGA. In every clock cycle, one particle is taken from the FPGA's onboard DRAM. The FPGA kernel has a long pipeline that is filled with particles, and therefore many particles are processed at once. Fixed-point data representation is customised at each pipeline stage to reduce bit-widths and hence FPGA resource usage. One particle is written back to the DRAM in every clock cycle.

For lower bound calculation, particle set size tuning and resampling processes, the host CPU gathers all the particle data from the FPGA via PCI Express.

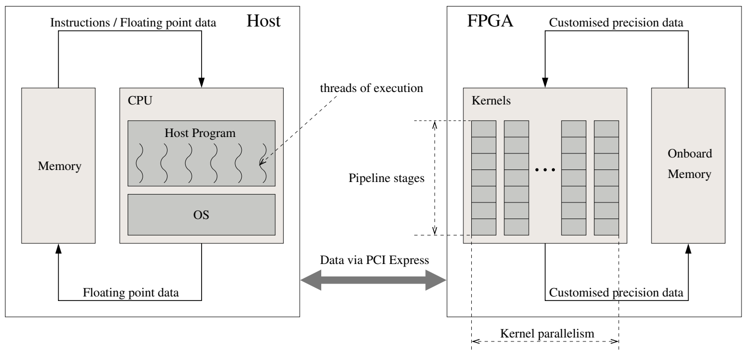


Fig. 2. Heterogeneous reconfigurable system

We derive a model to analyse the total computation time of the proposed system. The model helps us to design a configuration that can satisfy the real-time bound and, if necessary, amend the real-time application's specification. The model is validated by experiments in Section 5.

The total computation time of the system (T_{comp}) consists of three components.

$$T_{comp} = T_{kernel} + T_{host} + T_{io} \tag{6}$$

Kernel time: It describes the time spent on the FPGA kernel for the sampling and importance processes. P_t denotes the number of particles at current time step and $freq_{kernel}$ denotes the clock frequency of the FPGA kernel. The sampling and importance processes can be repeated for N_{si} times before going to the resampling process. N_{kernel} denotes the level of parallelism as multiple kernels can be instantiated in the FPGA. L is the latency due to pipelining.

$$T_{kernel} = \frac{P_t \cdot N_{si}}{freq_{kernel} \cdot N_{kernel}} + L - 1 \tag{7}$$

Host time: It describes the time spent on the host CPU. The clock frequency and number of threads of the host CPU are represented by $freq_{host}$ and N_{thread} respectively. *par* is an algorithm-specific parameter in the range of $[0, 1]$ which represents the ratio of CPU instructions that are parallelisable, and α is a scaling constant derived empirically.

$$T_{host} = \alpha \cdot \frac{P_t}{freq_{host}} \cdot \left(1 - par + \frac{par}{N_{thread}}\right) \tag{8}$$

IO time: It describes the time of moving particle data between the FPGA's onboard DRAM and host memory. *dim* is the number of dimensions of a particle, e.g. if a particle represents coordinate (x, y) and weight, $dim = 3$. *bw* is the bit-width to represent one dimension. $freq_{pcie}$ is the clock frequency of PCI Express bus and *lane* is the number of PCI Express lanes. Since PCI Express encodes data during transfer, the effective data is denoted by eff (in PCI Express gen2 the value is 8/10). The constant 2 accounts for the movement of data in and out of the FPGA.

$$T_{io} = \frac{2 \cdot P_t \cdot dim \cdot bw}{freq_{pcie} \cdot lane \cdot eff} \tag{9}$$

In real-time applications, the time for each step is fixed and is known as real-time bound T_{rt}. The derived model helps system designers to ensure that the computation time T_{comp} is shorter than T_{rt}. An idle time T_{idle} is introduced to represent the gap between finish time of computation and real-time bound.

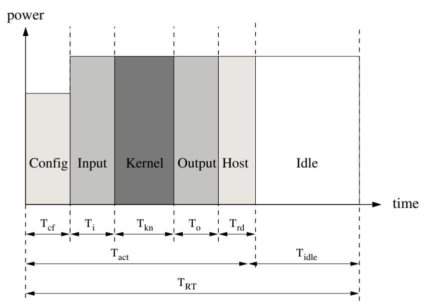
$$T_{idle} = T_{rt} - T_{comp} \tag{10}$$

Fig. 3(a) shows the timing of system operations. It illustrates that the FPGA is still drawing power after the computation finishes. We propose a method to reduce dynamic power by using run-time reconfiguration of FPGA. During idle time, the FPGA is loaded with a low-power configuration which has minimal active resources and runs at a very low clock frequency. The idea is shown in Fig. 3(b). Equation 11 describes the sleep time when the FPGA is idle and being loaded with the low-power configuration. If the sleep time is positive, it is always beneficial to load the low-power configuration.

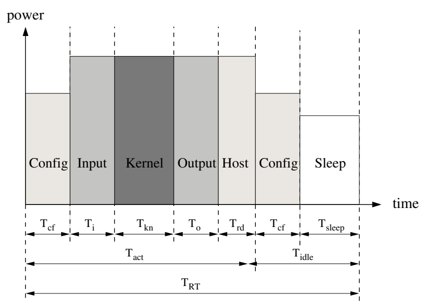
$$T_{sleep} = T_{idle} - T_{config} \tag{11}$$

Configuration time (T_{config}): It describes the time needed to download a configuration bit-stream to the FPGA. $size_{bs}$ represents the size of bitstream in bits. $freq_{config}$ is the configuration clock frequency in Hz and $port_{config}$ is the configuration port width in bits.

$$T_{config} = \frac{size_{bs}}{freq_{config} \cdot port_{config}} \tag{12}$$



(a) Without reconfiguration



(b) With reconfiguration to low-power mode during idle

Fig. 3. Timing of the system's operations

5 Result and Evaluation

To evaluate the HRS and make comparison with other systems, a simultaneous robot localisation and people-tracking application is implemented. Given a priori learned map, a robot receives sensor values and moves at regular time intervals. In each time step, M people are tracked by the robot. The state of the whole system of robot and people is represented by a state vector X_t:

$$X_t = \{R_t, H_{t,1}, H_{t,2}, ..., H_{t,M}\} \tag{13}$$

R_t denotes the robot's pose at time t, and $H_{t,1}, H_{t,2}, ..., H_{t,M}$ denote the locations of the M people.

The PF uses the following equation to represent the posterior of robot and people locations:

$$p(X_t|Y_t, U_t) = p(R_t|Y_t, U_t) \prod_{m=1}^{M} p(H_{t,m}|R_t, Y_t, U_t) \tag{14}$$

Y_t is the sensor measurement and U_t is the control of the robot at time t. The robot path posterior $p(R_t|Y_t, U_t)$ is represented by a set of robot particles. The distribution of a person's location $p(H_{t,m}|R_t, Y_t, U_t)$ is represented by a set of people particles, where each people particle set is attached to one particular robot particle. Therefore, if there are P_r robot particles representing the posterior over robot path, there are P_r people particle sets, each has P_h particles.

In the application, the map has an area of 12m*18m. The robot makes a movement of 0.5m every 5s, i.e. $T_{rt} = 5s$. The robot can track 8 people at the same time. The system supports a maximum of 8192 particles for robot-tracking and each robot particle is attached with 1024 particles for people-tracking. Therefore, the maximum number of kernel cycles is 8*8192*1024=67M.

Experiment settings: For the evaluation of HRS, we use the MaxWorkstation reconfigurable accelerator system from Maxeler Technologies. It has an FPGA board equipped with a Xilinx Virtex-6 XC6VSX475T FPGA which has 297,600 lookup tables (LUTs), 595,200 flip-flops (FFs), 2,016 digital signal processors (DSPs) and 1,064 block RAMs. The FPGA board is connected to an Intel i7-870 quad-core CPU clocked at 2.93GHz through a PCI Express link with a bandwidth of 2 GB/s. We develop the FPGA kernels using the MaxCompiler, which adopts a streaming programming model. At each pipeline stage, the fixed-point calculations are customised to different mantissa bit-widths.

There are two FPGA configurations: a) *sampling and importance configuration* is clocked at 100MHz, with 115961 LUTs (39%), 169188 FFs (28%), 967 DSPs (48%) and 257 block RAMs (24%) per kernel. b) *Low-power configuration* is clocked at 10MHz, with 5962 LUTs (2%), 6943 FFs (1%) and 12 block RAMs (1%). It uses the minimum amount of resources just to maintain communication between the FPGA and CPU.

The CPU performance results are obtained from an Intel Xeon X5650 CPU clocked at 2.66GHz. It is optimised by ICC with SSE4.2 and flag *-fast* enabled. OpenMP is used to utilise 6 physical cores (12 threads) of the CPU.

For the GPU performance results, we use NVIDIA Tesla C2070 GPU which has 448 cores running at 1.15GHz and has a peak performance at 1288GFlops.

Adaptive vs. non-adaptive: Table 1 shows the breakdown of computation time using our model and experimental data. Initially, the maximum number of particles are instantiated for global localisation. For the non-adaptive scheme, the particle set size does not change. The total computation time is estimated to be 1.957s which is within the real-time bound. The remaining idle time is enough to reconfigure to sleep mode, since $T_{sleep} = (5 - 1.957 - 0.87)s = 2.173s$.

For the adaptive scheme, the number of particles varies from 70 to 8192, and the computation time scales linearly with the number of particles. Fig. 4 shows how the number of particles varies versus time. Both the model and experiment show 99% reduction in computation time.

Fig. 5 illustrates the localisation error of the mobile robot. The error is the highest during initial global localisation and it is reduced when the robot moves. It is observed that the localisation error is not adversely affected by reducing the number of particles.

Table 1. Comparison of adaptive and non-adaptive PF on HRS (configuration with 1 FPGA kernel is used)

	Non-adaptive		Adaptive	
	Model	Exp.	Model	Exp.
No. of particles	8192		70-8192	
Kernel time T_{kernel} (s)	0.671	0.671	0.006-0.671	0.006-0.671
Host time T_{host} (s)	0.212	0.212	0.002-0.212	0.002-0.212
IO time T_{io} (s)	1.074	1.600	0.009-1.074	0.014-1.600
Total comp. time T_{comp}	1.957	2.483	0.017-1.957	0.022-2.483
Comp. speedup (higher is better)	1x	1x	1-115.12x	1-112.86x

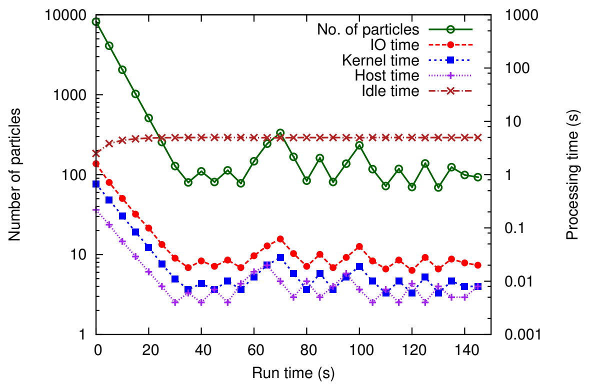


Fig. 4. Variation of particle set size and computation time

Performance comparison: Table 2 shows the performance comparison of CPU, GPU and HRS. Considering the kernel computation only, which ignores the IO time and host time, the HRS is up to 19.94 times faster than the CPU, and is 2.65 times faster than the GPU. If the overall system performance is considered, the HRS is up to 3.26 times faster than the CPU, and is 1.74 times slower than the GPU. Meanwhile, the CPU needs 7.002s to process a step, so the real-time constraint of 5s is violated. Currently the performance of HRS is limited by the PCI Express bus between the FPGA and CPU, which has a bandwidth of 2GB/s. If PCI Express gen3 x8 (7.88GB/s) is used, which has comparable bandwidth as that on the GPU, the overall system performance of the HRS is 7.39 times faster than the CPU, and is 1.3 times faster than the GPU.

In real-time applications, we need to consider the energy consumption within the real-time bound. Fig. 6 shows the power consumption varies between computation and idle time, and a significant amount of energy is consumed during idle time. Run-time reconfiguration reduces the idle power consumption of the HRS by 34%, from 135W to 95W. In other word, the energy consumption is reduced by 26-34%. For the case of 8192 particles, the HRS is up to 3.65 times more

Table 2. Comparison of using CPU, GPU and HRS

	CPU [a]	GPU [b]	HRS(1) [c]	HRS(2) [c]
Clock freq. (MHz)	2660	1150	100	100
Number of threads	12	448	1+8 [d]	2+8 [d]
Kernel time (s) [e]	0.058-6.780	0.008-0.892	0.006-0.671	**0.003-0.336**
Kernel speedup	1x	7.53x	10.11x	**19.94x**
Comp. time (s) [e]	0.060-7.002 [f]	0.011-1.236	0.021-2.483 [g] 0.011-1.283 [h]	0.018-2.148 [g] **0.008-0.948** [h]
Overall speedup	1x	5.67x	2.82x [g] 5.46x [h]	3.26x [g] **7.39x** [h]
Comp. power (W)	279	287	**135**	145
Comp. power eff.	1x	0.97x	**2.07x**	1.92x
Idle power (W)	133	208	**95**	**95**
Idle power eff.	1x	0.64x	**1.40x**	**1.40x**
Energy. (J) [e]	674-1954	1041-1138	489-587 [g] 489-539 [h]	489-595 [g] **489-535** [h]
Energy eff.	1x	0.64-1.72x	1.37-3.33x [g] 1.37-3.62x [h]	1.37-3.28x [g] **1.37-3.65x** [h]

[a] Intel Xeon X5650 @2.66GHz with 12 threads.
[b] NVIDIA Tesla C2070 and Intel Core i7-950 @3.07GHz with 8 threads.
[c] Xilinx XC6VSX475T and Intel Core i7-870 @2.93GHz with 8 threads. HRS(1) has one FPGA kernel while HRS(2) has two.
[d] Number of FPGA kernels and number of threads in the CPU.
[e] Cases for 70 and 8192 robot particles.
[f] Real-time bound is violated as the constraint is 5s.
[g] On our platform, the FPGA and CPU communicate through PCI Express with bandwidth 2GB/s.
[h] Assume the FPGA and CPU communicate through PCI Express gen3 x8, bandwidth 7.88GB/s.

energy efficient than the CPU, and is 2.13 times more energy efficient than the GPU. For the case of 70 particles, the HRS is 1.37 times more energy efficient than the CPU, and is 2.13 times more energy efficient than the GPU.

6 Conclusion

This paper presents an adaptive particle filter for real-time applications. The proposed heterogeneous reconfigurable system demonstrates a significant reduction in power and energy consumption compared with the CPU and the GPU. The adaptive particle filter reduces computation time while maintaining quality of results. Ongoing and future work includes applying the adaptive approach to larger systems with multiple FPGAs. Distributed resampling and data compression techniques are explored to mitigate the data transfer overhead between the FPGA and the CPU. More compute-intensive applications using PF would be of interest on the extended heterogeneous reconfigurable system.

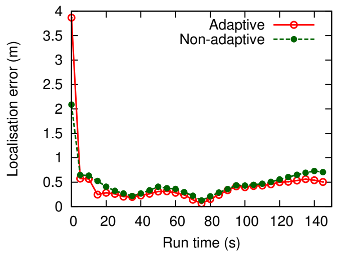


Fig. 5. Localisation error

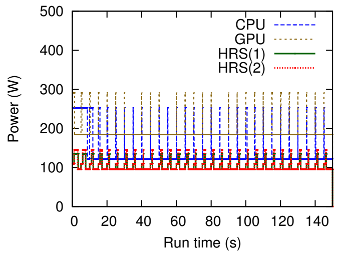


Fig. 6. Power consumption

Acknowledgment

This work is supported in part by the European Union FP7 under grant agreement number 248976, 257906 and 287804, by UK EPSRC, by Maxeler University Programme, by Xilinx, and by the Croucher Foundation. The author would like to thank Oliver Pell at Maxeler Technologies for comments on the paper.

References

1. Happe, M., et al.: A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. Journal Real-Time Image Processing (2011) 1–16
2. Montemerlo, M., et al.: Conditional particle filters for simultaneous mobile robot localization and people-tracking. In: Proc. Int. Conf. Robotics and Automation. (2002) 695–701
3. Vermaak, J., et al.: Particle methods for bayesian modeling and enhancement of speech signals. IEEE Trans. Speech and Audio Processing **10**(3) (2002) 173–185
4. Eele, A., Maciejowski, J.: Comparison of stochastic optimisation methods for control in air traffic management. In: Proc. IFAC World Congress. (2011)
5. Doucet, A., et al.: Sequential Monte Carlo methods in practice. Springer (2001)
6. Bolic, M., et al.: Resampling algorithms and architectures for distributed particle filters. IEEE Trans. Signal Processing **53**(7) (2005) 2442–2450
7. Koller, D., et al.: Using learning for approximation in stochastic processes. In: Proc. Int. Conf. Machine Learning. (1998) 287–295
8. Fox, D.: Adapting the sample size in particle filters through KLD-sampling. Int. Trans. Robotics **22**(12) (2003) 985–1003
9. Park, S.H., et al.: Novel adaptive particle filter using adjusted variance and its application. Int. Journal Control, Automation and Systems **8**(4) (2010) 801–807
10. Bolic, M., et al.: Performance and complexity analysis of adaptive particle filtering for tracking applications. In: Proc. Asilomar Conf. Signals, Systems, and Computers. Volume 1. (2002) 853–857
11. Chau, T.C., et al.: Adaptive sequential monte carlo approach for real-time applications. In: Proc. Int. Conf. Field Programmable Logic and Applications. (2012) 527–530
12. Liu, Z., et al.: Mobile robots global localization using adaptive dynamic clustered particle filters. In: Proc. Int. Conf. Intelligent Robots and Systems. (2007) 1059–1064