Mapping Adaptive Particle Filters to Heterogeneous Reconfigurable Systems

THOMAS C. P. CHAU and XINYU NIU, Department of Computing, Imperial College London, UK
ALISON EELE and JAN MACIEJOWSKI, Department of Engineering, University of Cambridge, UK
PETER Y. K. CHEUNG, Department of Electrical and Electronic Engineering, Imperial College London, UK
WAYNE LUK, Department of Computing, Imperial College London, UK

This article presents an approach for mapping real-time applications based on particle filters (PFs) to heterogeneous reconfigurable systems, which typically consist of multiple FPGAs and CPUs. A method is proposed to adapt the number of particles dynamically and to utilise runtime reconfigurability of FPGAs for reduced power and energy consumption. A data compression scheme is employed to reduce communication overhead between FPGAs and CPUs. A mobile robot localisation and tracking application is developed to illustrate our approach. Experimental results show that the proposed adaptive PF can reduce up to 99% of computation time. Using runtime reconfiguration, we achieve a 25% to 34% reduction in idle power. A 1U system with four FPGAs is up to 169 times faster than a single-core CPU and 41 times faster than a 1U CPU server with 12 cores. It is also estimated to be 3 times faster than a system with four GPUs.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Heterogeneous (hybrid) systems*; C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Particle filters, sequential Monte Carlo, reconfigurable systems, FPGAs, runtime reconfiguration

ACM Reference Format:
Thomas C. P. Chau, Xinyu Niu, Wayne Luk, Alison Eele, Jan Maciejowski, and Peter Y. K. Cheung. 2014. Mapping adaptive particle filters to heterogeneous reconfigurable systems. ACM Trans. Reconfig. Technol. Syst. 7, 4, Article 36 (December 2014), 17 pages.
DOI: http://dx.doi.org/10.1145/2629469

1. INTRODUCTION

Particle filter (PF), also known as the sequential Monte Carlo (SMC) method, is a statistical technique for dynamic systems involving nonlinear and non-Gaussian properties. PF has been studied in various application areas, including object tracking [Happe et al. 2011], robot localisation [Montemerlo et al. 2002], speech recognition [Vermaak et al. 2002], and air traffic management [Eele and Maciejowski 2011].

This work is supported in part by the European Union Seventh Framework Programme under grant agreement numbers 257906, 287804, and 318521; UK EPSRC grant numbers EP/L00058X/1, EP/I012036/1, and EP/G066477/1; Maxeler University Programme, Xilinx, and the Croucher Foundation.
Authors' addresses: T. C. P. Chau, X. Niu, and W. Luk, Department of Computing, Imperial College London; email: c.chau10@imperial.ac.uk; A. Eele and J. Maciejowski, Department of Engineering, University of Cambridge, and P. Y. K. Cheung, Department of Electrical and Electronic Engineering, Imperial College London.

PF keeps track of a large number of particles, and each contains information about how a system would evolve. The underlying concept is to approximate a sequence of states by a collection of particles. Each particle is weighted to reflect the quality of an approximation. The more complex the problem, the larger the number of particles needed. One drawback of PF is its long execution times, which limit its practical use.

This article presents an efficient solution to PF. We derive an adaptive algorithm that adjusts its computation complexity at runtime based on the quality of results. To map our algorithm to a heterogeneous reconfigurable system (HRS) consisting of multiple FPGAs and CPUs, we design a pipeline-friendly data structure to make effective use of the stream computing model. Moreover, we accelerate the algorithm with a data compression scheme and data control separation.

The key contributions of this work include:

(1) An adaptive PF algorithm that adapts the size of particle set at runtime. The algorithm is able to reduce computation workload while maintaining the quality of results.
(2) Mapping the proposed algorithm to a scalable and reconfigurable system by following the stream computing model. A novel data structure is designed to take advantage of the architecture and alleviate the data transfer bottleneck. The system uses the runtime reconfigurability of FPGA to switch between computation mode and low-power mode.
(3) An implementation of a robot localisation application targeting the proposed system. Compared to a nonadaptive and nonreconfigurable implementation, the idle power of our proposed system is reduced by 25% to 34% and the overall energy consumption decreases by 17% to 33%. Our system with four FPGAs is up to 169 times faster than a single-core CPU, 41 times faster than a 1U CPU server with 12 cores, and 3 times faster than a modelled four-GPU system.

2. BACKGROUND AND RELATED WORK

This section briefly outlines the PF algorithm. A more detailed description can be found in Doucet et al. [2001]. PF estimates the state of a system by a sampling-based approximation of the state probability density function. The state of a system in timestep t is denoted by X_t. The control and observation are denoted by U_t and Y_t, respectively. Three pieces of information about the system are known a priori:

—$p(X_0)$ is the probability of the initial state of the system.
—$p(X_t|X_{t-1}, U_{t-1})$ is the state transition probability of the system's current state given a previous state and control information.
—$p(Y_t|X_t)$ is the observation model describing the likelihood of observing the measurement at the 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 $\{w^{(i)}\}_{i=1}^P$. X_0 and U_0 are initialised. This computation consists of three iterative steps.

(1) *Sampling*: A new particle set $\{\widetilde{\chi}_t^{(i)}\}_{i=1}^P$ 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 weighting*: The 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 $\{\widetilde{\chi}_t^{(i)}\}_{i=1}^P$. Then each particle is assigned a weight $w^{(i)}$ with respect to the likelihood.
(3) *Resampling*: Particles with higher weights are replicated, and the number of particles with lower weights is reduced. With resampling, the particle set has a smaller

variance. The particle set is used in the next timestep to predict the posterior probability subsequently. The distribution of the resulting particles $\{\chi_t^{(i)}\}_{i=1}^{P}$ approximates $p(X_t|Y_{1:t})$.

The particles in PF are independent of each other; thus, the algorithm can be accelerated using specialised hardware with massive parallelism and pipelining. In Happe et al. [2011], an approach for PF on a hybrid CPU/FPGA platform is developed. Using a multithreaded programming model, computation is switched between hardware and software during runtime to react to performance requirements. Resampling algorithms and architectures for distributed PFs are proposed in Bolic et al. [2005].

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 prespecified threshold Koller and Fratkina [1998]. Kullback Leibler distance (KLD) sampling is proposed in Fox [2003], which offers better-quality results than the likelihood-based approach. KLD sampling is improved in Park et al. [2010] by adjusting the variance and gradient of data to generate particles near high-likelihood regions. The preceding methods introduce data dependencies in the sampling and importance weighting steps, so they are difficult to be parallelised. An adaptive PF is proposed in Bolic et al. [2002] that changes the number of particles dynamically based on estimation quality. In Chau et al. [2012], adaptive PF is extended to a multiprocessor system on FPGA. The number of particles and active processors change dynamically, but the performance is limited by soft-core processors. In Liu et al. [2007], a mechanism and a theoretical lower bound for adapting the sample size of particles are presented. Our previous work Chau et al. [2013a] presents a hardware-friendly adaptive PF. The algorithm is mapped to an accelerator system that consists of an FPGA and a CPU. However, the system suffers from a large communication overhead when the particles are transferred between the FPGA and CPU. Moreover, the scalability of the adaptive PF algorithm to multiple FPGAs is not covered. In this work, we extend our previous work to address the problems mentioned previously.

3. ADAPTIVE PARTICLE FILTER

This section introduces an adaptive PF algorithm that changes the number of particles at each timestep. The algorithm is inspired by Liu et al. [2007], and we transform it to a pipeline-friendly version for mapping to the stream computing architecture. This algorithm is shown in Algorithm 1, which consists of four stages.

3.1. Stage 1: Sampling and Importance Weighting (Line 8 to 9)

At the initial timestep ($t = 0$), the maximum number of particles are used—that is, $P_0 = P_{max}$. At the subsequent timesteps, the number of particles is denoted as P_t. Initially, the particle set $\{\chi_t^{(i)}\}_{i=1}^{P_t}$ is sampled to $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$. Then, a weight from $\{w^i\}_{i=1}^{P_t}$ is assigned to each particle. As a result, $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ and $\{w^{(i)}\}_{i=1}^{P_t}$ give an estimation of the next state.

During sampling and importance weighting, the computation of every particle is independent of each of the others. The mapping of computation to FPGAs will be described in Section 4.

3.2. Stage 2: Lower Bound Calculation (Line 10)

This stage derives the smallest number of particles that are needed in the next timestep to bound the approximation error. The adaptive algorithm seeks a value that is less than or equal to P_{max}. This number, denoted as $\widetilde{P}_{t+1}$, is referred to as the lower bound

ALGORITHM 1: Adaptive PF algorithm

```
1: $P_0 \leftarrow P_{max}$
2: $\{X_0^{(i)}\}_{i=1}^{P_0}$ ←random set of particles
3: $t = 1$
4: **for** each step $t$ **do**
5:    $r = 0$
6:    **while** $r \leq itl\_repeat$ **do**
7:       —On FPGAs—
8:       Sample a new state $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ from $\{\chi_t^{(i)}\}_{i=1}^{P_t}$
9:       Calculate unnormalised importance weights $\{\widetilde{w}^{(i)}\}_{i=1}^{P_t}$ and accumulate the weights as $w_{sum}$
10:      Calculate the lower bound of sample size $\widetilde{P}_{t+1}$ by Equation (1)
11:      —On CPUs—
12:      Sort $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ in descending $\{\widetilde{w}^{(i)}\}_{i=1}^{P_t}$
13:      **if** $\widetilde{P}_{t+1} < P_t$ **then**
14:         $P_{t+1} = max(\lceil \widetilde{P}_{t+1} \rceil, P_t/2)$
15:         Set $a = 2P_{t+1} - P_t$ and $b = P_{t+1}$
16:         –Do the following loop in parallel–
17:         **for** $i$ in $P_t - P_{t+1}$ **do**
18:            $\widetilde{\chi}_{t+1}^{(i)} = \frac{\chi_{t+1}^{(a)}\widetilde{w}^{(a)} + \chi_{t+1}^{(b)}\widetilde{w}^{(b)}}{\widetilde{w}^{(a)} + \widetilde{w}^{(b)}}$
19:            $\widetilde{w}^{(i)} = \widetilde{w}^{(a)} + \widetilde{w}^{(b)}$
20:            $a = a + 1$ and $b = b - 1$
21:         **end for**
22:      **else if** $\widetilde{P}_{t+1} \geq P_t$ **then**
23:         $a = 0$ and $b = 0$
24:         **for** $i$ in $P_{t+1} - P_t$ **do**
25:            **if** $\widetilde{w}^{(a)} < \widetilde{w}^{(a+1)}$ and $a < P_{t+1}$ **then**
26:               $a = a + 1$
27:            **end if**
28:            $\widetilde{\chi}_{t+1}^{(P_t+b)} = \widetilde{\chi}_{t+1}^{(a)}/2$
29:            $\widetilde{\chi}_{t+1}^{(a)} = \widetilde{\chi}_{t+1}^{(a)}/2$
30:            $\widetilde{w}^{(P_t+b)} = \widetilde{w}^{(a)}/2$
31:            $\widetilde{w}^{(a)} = \widetilde{w}^{(a)}/2$
32:            $b = b + 1$
33:         **end for**
34:      **end if**
35:      Resample $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ to $\{\chi_{t+1}^{(i)}\}_{i=1}^{P_{t+1}}$
36:      $r = r + 1$
37:   **end while**
38: **end for**
```

of sampling size. It is calculated by Equations (1) through (4):

$$\widetilde{P}_{t+1} = \sigma^2 \cdot \frac{P_{max}}{Var(\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t})} \tag{1}$$

$$\begin{aligned}\sigma^2 = &\sum_{i=1}^{P_t} \left(w^{(i)} \cdot \widetilde{\chi}_{t+1}^{(i)}\right)^2 - 2 \cdot E\left(\left\{\widetilde{\chi}_{t+1}^{(i)}\right\}_{i=1}^{P_t}\right) \cdot \sum_{i=1}^{P_t} \left(\left(w^{(i)}\right)^2 \cdot \widetilde{\chi}_{t+1}^{(i)}\right) \\ &+ \left(E\left(\left\{\widetilde{\chi}_{t+1}^{(i)}\right\}_{i=1}^{P_t}\right)\right)^2 \cdot \sum_{i=1}^{P_t} \left(w^{(i)}\right)^2\end{aligned} \tag{2}$$

$$Var\left(\left\{\widetilde{\chi}_{t+1}^{(i)}\right\}_{i=1}^{P_t}\right) = \sum_{i=1}^{P_t} \left(w^{(i)} \cdot \left(\widetilde{\chi}_{t+1}^{(i)}\right)^2\right) - \left(E\left(\left\{\widetilde{\chi}_{t+1}^{(i)}\right\}_{i=1}^{P_t}\right)\right)^2 \tag{3}$$

$$E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1}) = \sum_{i=1}^{P_t} w^{(i)} \cdot \widetilde{\chi}^{(i)}_{t+1} \tag{4}$$

As shown in Equations (2) through (4), $w^{(i)}$ is a normalised term. To calculate $w^{(i)}$, a traditional software-based approach is to iterate through the set of particles twice. The sum of weights w_{sum} and unnormalised weight $\widetilde{w}^{(i)}$ are calculated in the first iteration. Then, $w^{(i)}$ is obtained by dividing $\widetilde{w}^{(i)}$ by w_{sum} in the second iteration. However, this method is inefficient for FPGA implementation. Since $2P_t$ cycles are needed to process P_t pieces of data, the throughput is reduced to 50%.

To fully utilise deep pipelines targeting an FPGA, we perform function transformation. Given $w^{(i)} = \frac{\widetilde{w}^{(i)}}{w_{sum}}$, we extract w_{sum} out of Equations (2) through (4). By doing so, we obtain a transformed form as shown in Equations (5) through (7). w_{sum} and $\widetilde{w}^{(i)}$ are computed simultaneously in two separate data paths. At the last clock cycle of the particle stream, σ^2, $Var(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1})$ and $E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1})$ are obtained. The details of the FPGA kernel design will be explained in Section 4.

$$\begin{aligned}\sigma^2 = \frac{1}{(w_{sum})^2} \cdot \Bigg(&\sum_{i=1}^{P_t} \left(\widetilde{w}^{(i)} \cdot \widetilde{\chi}^{(i)}_{t+1}\right)^2 - 2 \cdot E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1}) \cdot \sum_{i=1}^{P_t} \left((\widetilde{w}^{(i)})^2 \cdot \widetilde{\chi}^{(i)}_{t+1}\right) \\ &+ \left(E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1})\right)^2 \cdot \sum_{i=1}^{P_t} (\widetilde{w}^{(i)})^2 \Bigg)\end{aligned} \tag{5}$$

$$Var(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_t} \left(\widetilde{w}^{(i)} \cdot (\widetilde{\chi}^{(i)}_{t+1})^2\right) - \left(E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1})\right)^2 \tag{6}$$

$$E(\{\widetilde{\chi}^{(i)}_{t+1}\}^{P_t}_{i=1}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_t} \widetilde{w}^{(i)} \cdot \widetilde{\chi}^{(i)}_{t+1} \tag{7}$$

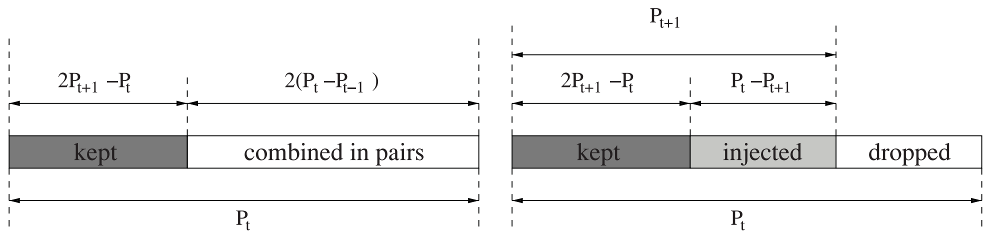
3.3. Stage 3: Particle Set Size Tuning (Lines 12 through 34)

The adaptive approach tunes the particle set size to fit the lower bound P_{t+1}. This stage is done on the CPUs because the operations involve nonsequential data access that cannot be mapped efficiently to FPGAs.

The particles are sorted in descending order according to their weights. As the new sample size can increase or decrease, there are two cases:

—**Case I: Particle set reduction when $\widetilde{P}_{t+1} < P_t$**
The lower bound P_{t+1} is set to $max(\lceil \widetilde{P}_{t+1} \rceil, P_t/2)$. Since the new size is smaller than the old one, some particles are combined to form a smaller particle set. Figure 1 illustrates the idea of particle reduction. The first $2P_{t+1} - P_t$ particles with higher weights are kept, and the remaining $2(P_t - P_{t+1})$ particles are combined in pairs. As a result, there are $P_t - P_{t+1}$ new particles injected to form the target particle set with P_{t+1} particles. We combine the particles deterministically to keep the statements in the loop independent of each of the others. As a result, loop unrolling is undertaken to execute the statements in parallel. The complexity of the loop is in $\mathcal{O}(\frac{P_t - P_{t+1}}{N_{parallel}})$, where $N_{parallel}$ indicates the level of parallelism.

—**Case II: Particle set expansion when $\widetilde{P}_{t+1} \geq P_t$**
The lower bound P_{t+1} is set to $\widetilde{P}_{t+1}$. Some particles are taken from the original set and are inserted to form a larger set. The particles with larger weight would have



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

Fig. 1. Particle set reduction.

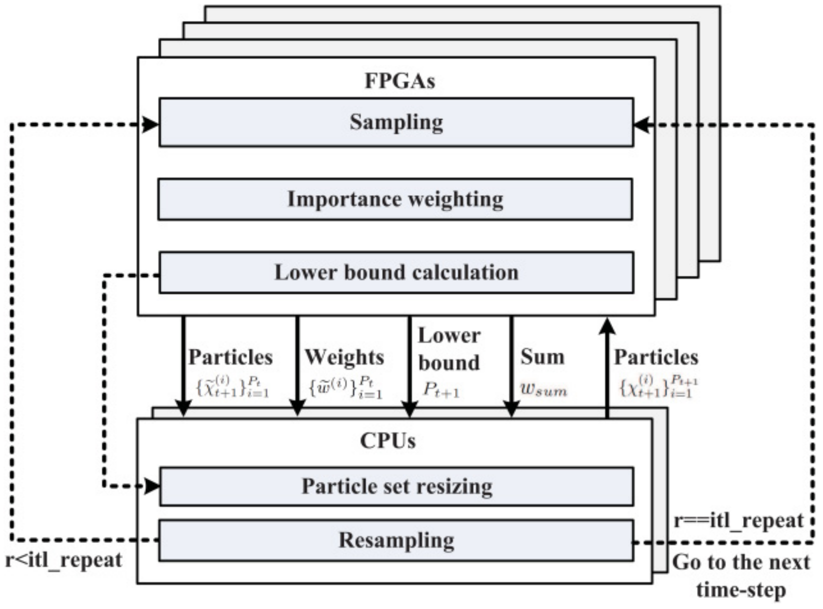


Fig. 2. Heterogeneous reconfigurable system (Solid lines: data paths; Dotted lines: control paths).

more descendants. As shown in lines 22 through 34, the process requires picking the particle with the largest weight at each iteration of particle incision. Since the particle set is presorted, the complexity of particle set expansion is $\mathcal{O}(P_{t+1} - P_t)$.

3.4. Stage 4: Resampling (Line 35)

Resampling is performed to pick P_{t+1} particles from $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ to form $\{\chi_{t+1}^{(i)}\}_{i=1}^{P_{t+1}}$. The process has a complexity of $\mathcal{O}(P_{t+1})$.

4. HETEROGENEOUS RECONFIGURABLE SYSTEM

This section describes the proposed HRS. It is scalable to cope with different FPGA devices and applications. HRS also takes advantage of the runtime reconfiguration feature for power and energy reduction.

4.1. Mapping Adaptive PF to HRS

The system design of HRS is shown in Figure 2. A heterogeneous structure is employed to make use of multiple FPGAs and CPUs. FPGAs and CPUs communicate through

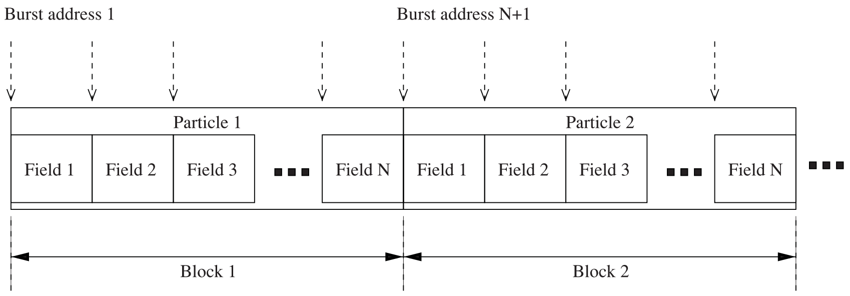


Fig. 3. A particle stream.

high-bandwidth buses. FPGAs are responsible for (1) sampling, (2) importance weighting, and (3) lower bound calculation. The data paths on the FPGAs are fully pipelined. Each FPGA has its own onboard dynamic random-access memory (DRAM) to store the large amount of particle data. On the other hand, the CPUs gather all of the particles from FPGAs to perform particle set size tuning and resampling.

4.2. FPGA Kernel Design

Sampling, importance weighting, and lower bound calculation are the most computation-intensive stages. In each timestep, these three stages are iterated for *itl_repeat* times. An FPGA kernel is designed to enable acceleration of them.

For sampling and importance weighting, the computation of each particle is independent of each of the others. Particles are fed to the FPGAs as a stream shown in Figure 3. Each block of the particle stream consists of a number of data fields that store information of a particle. The number of data fields is application dependent. In every clock cycle, one piece of data is transferred from the onboard memory to an FPGA data path. Each FPGA data path has a long pipeline where each stage is filled with a piece of data, and therefore many particles are processed simultaneously. Fixed-point data representation is customised at each pipeline stage to reduce the resource usage.

Figure 4 shows the components of the FPGA kernel. The kernel is fully pipelined to achieve one output per clock cycle. It can also be replicated as many times as FPGA resource allow, and the replications can be split across multiple FPGA boards. The kernel takes three inputs from the CPUs or onboard DRAM: (1) states, (2) controls, and (3) seeds. Application-specific parameters are stored in ROMs. Three building blocks correspond to the sampling, importance weighting and lower bound calculation stages as described in Section 3.

Meanwhile, the accumulation of w_{sum} introduces a feedback loop. A new weight comes along every cycle that is more quickly than the floating-point unit to perform addition of the previous weight. To achieve one result per clock cycle, fixed-point data path is implemented while ensuring that no overflow or underflow occurs.

4.3. Timing Model for Runtime Reconfiguration

We derive a model to analyse the computation time of HRS. The model helps us to design a configuration schedule that satisfies the real-time requirement and, if necessary, amend the application's specification. The model will be validated by experiments in Section 6.

The computation time (T_{comp}) of HRS consists of three components: (1) data path time $T_{datapath}$, (2) CPU time T_{CPU}, and (3) data transfer time T_{tran}. The sampling, importance weighting, and resampling processes are repeated for *itl_repeat* times in

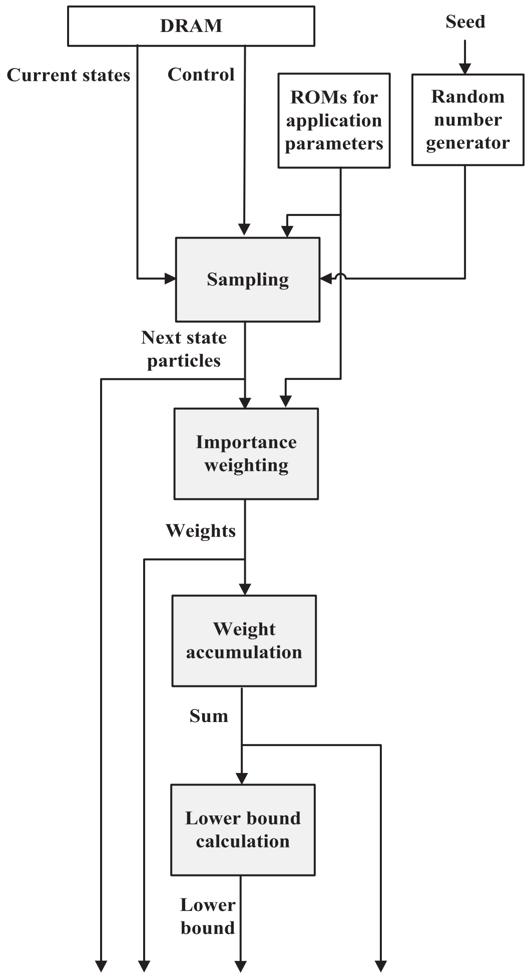


Fig. 4. FPGA kernel design.

every timestep.

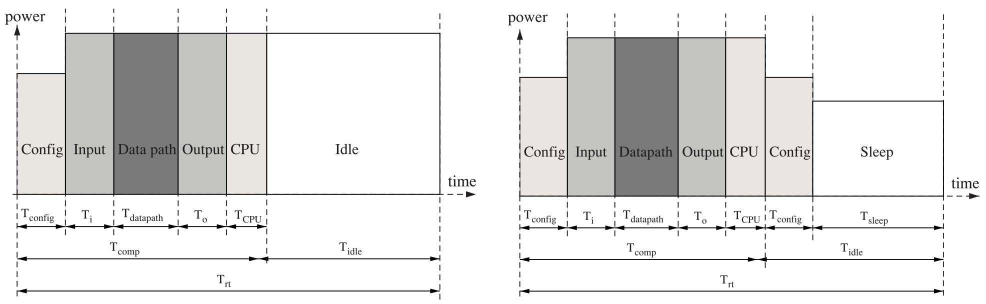
$$T_{comp} = itl\_repeat \cdot (T_{datapath} + T_{CPU} + T_{tran}) \tag{8}$$

Data path time, $T_{datapath}$, denotes the time spent on the FPGAs. P_t denotes the number of particles at the current timestep, and f_{FPGA} denotes the clock frequency of the FPGAs. L is the length of the pipeline. $N_{datapath}$ denotes the number of data paths on one FPGA board. N_{FPGA} is the number of FPGA boards in the system.

$$T_{datapath} = \left(\frac{P_t}{f_{FPGA} \cdot N_{datapath}} + L - 1 \right) \frac{1}{N_{FPGA}} \tag{9}$$

CPU time, T_{CPU}, denotes the time spent on the CPUs. The clock frequency and number of threads of the CPUs are represented by f_{CPU} and N_{thread}, respectively. par is an application-specific parameter in the range of [0, 1] representing the ratio of CPU instructions that are parallelisable, and α is a scaling constant derived empirically.

$$T_{CPU} = \alpha \cdot \frac{P_t}{f_{CPU}} \cdot \left(1 - par + \frac{par}{N_{thread}} \right) \tag{10}$$



(a) Without reconfiguration

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

Fig. 5. Power consumption of the HRS over time.

Data transfer time, T_{tran}, denotes the time of moving a particle stream between the FPGAs and the CPUs. df is the number of data fields of a particle. For example, if a particle contains the information of coordinates (x, y) and heading h, $df = 3$. Given that the constant 1 represents the weight and the constant 2 accounts for the movement of data in and out of the FPGAs, and bw_{data} is the bit-width of one data field, the expression $(2 \cdot df + 1) \cdot bw_{data}$ is regarded as the size of a particle.

f_{bus} is the clock frequency of the bus connecting the CPUs to FPGAs, and $lane$ is the number of bus lanes connected to one FPGA. Since many buses, such as the PCI Express bus, encode data during transfer, the effective data are denoted by eff (in PCI Express Gen2, the value is 8/10). In our previous work [Chau et al. 2013a], the data transfer time has a significant performance impact on HRS. To reduced the data transfer overhead, we introduce a data compression technique that will be described in Section 5.

$$T_{tran} = \frac{(2 \cdot df + 1) \cdot bw_{data} \cdot P_t}{f_{bus} \cdot lane \cdot eff \cdot N_{FPGA}} \tag{11}$$

In real-time applications, each timestep is fixed and is known as the 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 time gap between the computation time and real-time bound.

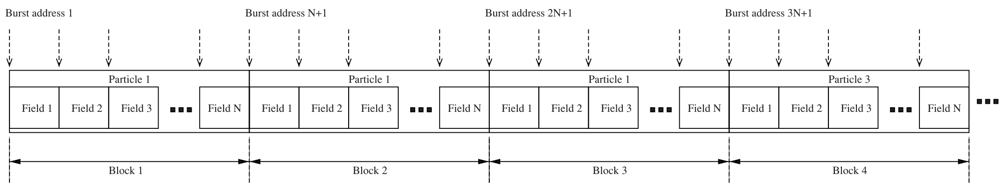
$$T_{idle} = T_{rt} - T_{comp} \tag{12}$$

Figure 5(a) illustrates the power consumption of an HRS without runtime reconfiguration. It shows that the FPGAs are still drawing power after the computation finishes. By exploiting runtime reconfiguration as shown in Figure 5(b), the FPGAs are loaded with a low-power configuration during the idle period. Such configuration minimises the amount of active resources and clock frequency. Equation (13) describes the sleep time when the FPGAs are idle and being loaded with the low-power configuration. If the sleep time is positive, reconfiguration would be helpful in these situations.

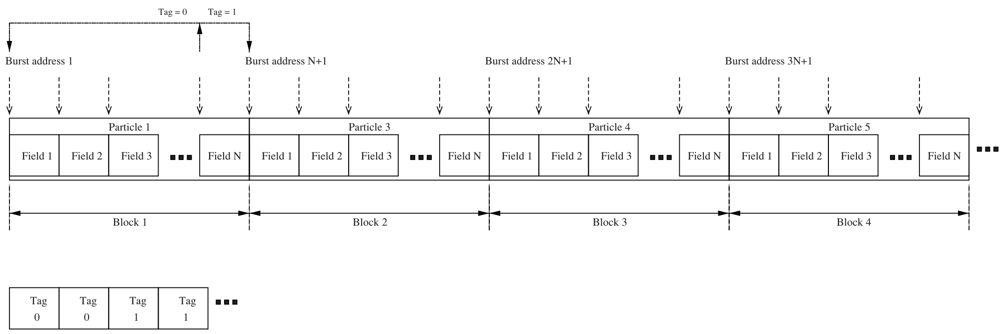
$$T_{sleep} = T_{idle} - T_{config} \tag{13}$$

Configuration time, T_{config}, denotes the time needed to download a configuration bit-stream to the FPGAs. $size_{bs}$ represents the size of bitstream in bits. f_{config} is the configuration clock frequency in Hertz, and bw_{config} is the width of the configuration port.

$$T_{config} = \frac{size_{bs}}{f_{config} \cdot bw_{config}} \tag{14}$$



(a) Particle stream before compression



(b) Compressed particle stream

Fig. 6. After the resampling process, some particles are eliminated and the remaining particles are replicated. Data compression is applied so that every particle is stored and transferred once only.

5. OPTIMISING TRANSFER OF PARTICLE STREAM

In Section 4, the data transfer time depends on the number of particles and the bus bandwidth between the CPUs and FPGAs. It can be a major performance bottleneck as depicted in Chau et al. [2013a]. Refer to Figure 6(a); each block stores the data of a particle. When the CPUs finish processing, all data are transferred from the CPUs to the FPGAs. The data transfer time cannot be reduced by implementing more FPGA data paths or increasing the FPGAs' clock frequency because the bottleneck is at the bus connecting the CPUs and FPGAs.

To improve the data transfer performance, we design a data structure that facilitates compression of particles. The idea comes from an observation of the resampling process—some particles are eliminated, and the vacancies are filled by replicating noneliminated particles. Replication means that data redundancy exists. For example, in the original data structure shown in Figure 6(a), particle 1 has three replicates and particle 2 is eliminated; therefore, particle 1 is stored and transferred for three times.

By using the data structure in Figure 6(b), data redundancy is eliminated by storing every particle once. Each particle is also transferred once. As a result, the data transfer time and memory space are reduced.

An HRS often contains DRAM that transfers data in burst to maximise the memory bandwidth. This works fine with the original data structure where the data are organised as a sequence from the lower address space to the upper. However, using the new data structure, the data access pattern is not sequential anymore, and the address can go back and forth. The DRAM controller needs to be modified so that the transfer throughput would not be affected by the change of data access pattern. As illustrated in Figure 6(b), a tag sequence is used to indicate the address of the next block. For example, after reading the data of particle 1, the burst address is at N. If the tag is one, the next burst address will point to the address of the next block at $N + 1$. Otherwise, the burst address will point to the start address of the current block (which is 1). The data are still addressed in burst, so the performance is not degraded.

Following is the data transfer time with compression. *Rep* is the average number of replication of the particles, and therefore the size of the resampled particle stream is reduced by a ratio of *Rep*. The range of *Rep* is from 1 to P_t, depending on the distribution of particles after the resampling process. The effect of *Rep* on data transfer time will be evaluated in the next section.

$$T_{tran} = \frac{(\frac{df}{Rep} + df + 1) \cdot bw_{data} \cdot P_t}{f_{bus} \cdot lane \cdot eff \cdot N_{FPGA}} \tag{15}$$

6. EXPERIMENTAL RESULTS

To evaluate the performance of the HRS and make comparison with the other systems, we implement an application that uses PF for localisation and tracking of a mobile robot. The application is proposed in Montemerlo et al. [2002] to track location of moving objects conditioned upon the robot poses over time. Given an a priori learned map, a robot receives sensor values and moves at regular time intervals. Meanwhile, M moving objects are tracked by the robot. The states of the robot and objects at time t are represented by a state vector X_t:

$$X_t = \{R_t, H_{t,1}, H_{t,2}, \ldots, H_{t,M}\}. \tag{16}$$

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

The following equation is used to represent the posterior of the robot's location:

$$p(X_t|Y_t, U_t) = p(R_t|Y_t, U_t) \prod_{m=1}^{M} p(H_{t,m}|R_t, Y_t, U_t). \tag{17}$$

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 an object's location $p(H_{t,m}|R_t, Y_t, U_t)$ is represented by a set of object particles, where each object–particle set is attached to one particular robot particle. In other words, if there are P_r robot particles representing the posterior over the robot path, there are P_r object–particle sets and each has P_h particles.

In the application, the area of the map is 12m by 18m. The robot makes a movement of 0.5m every 5 seconds—that is, $T_{rt} = 5$. The robot can track eight moving objects at the same time. A maximum of 8,192 particles are used for robot tracking, and each robot particle is associated with 1,024 object particles. Therefore, the maximum number of data path cycles is 8*8192*1024 = 67,108,864. Each particle being streamed into the FPGAs contains coordinates (x,y) and heading h, which are represented by three single precision floating-point numbers. For the particle being streamed out of the FPGAs, it also contains a weight in addition to the coordinates. From Equation (11), the size of a particle is $(2 \cdot 3 + 1) \cdot 32$ bits $= 224$ bits.

6.1. System Settings

HRS: Two reconfigurable accelerator systems from Maxeler Technologies are used. The system is developed using MaxCompiler, which adopts a stream computing model.

—*MaxWorkstation* is a microATX form factor system that is equipped with one Xilinx Virtex-6 XC6VSX475T FPGA. The FPGA 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 CPU (four physical cores, eight threads in total, clocked at 2.93GHz) via a PCI Express Gen2×8 bus. The maximum bandwidth

Table I. Comparison of Adaptive and Nonadaptive PF on HRS (MaxWorkstation with One FPGA; No Data Compression Is Applied)

	Nonadaptive PF		Adaptive PF	
	Model	Exp.	Model	Exp.
Number of Particles	67M		573k	
Data path time $T_{datapath}$ (s)	0.336	0.336	0.003	0.003
CPU time T_{CPU} (s)	0.117	0.117	0.001	0.001
Data time T_{tran} (s)	0.875	1.432	0.007	0.012
Total comp. time T_{comp} (s)	1.328	1.885	0.011	0.016
Comp. speedup (higher is better)	1×	1×	120.7×	117.8×

of the PCI Express bus is 2GB/s according to the specification provided by Maxeler Technologies.

—*MPC-C500* is a 1U server accommodating four FPGA boards, each of which has a Xilinx Virtex-6 XC6VSX475T FPGA. Each FPGA board is connected to two Intel Xeon X5650 CPUs (12 physical cores, 24 threads in total, clocked at 2.66GHz) via a PCI Express Gen2×8 bus.

To support runtime reconfigurability, there are two FPGA configurations:

—*Sampling and importance weighting configuration* is clocked at 100MHz. Two data paths are implemented on one FPGA to process particles in parallel. The total resource usage is 231,922 LUTs (78%), 338,376 FFs (56%), 1,934 DSPs (96%), and 514 block RAMs (48%).

—*Low-power configuration* is clocked at 10MHz, with 5,962 LUTs (2%), 6,943 FFs, (1%) and 12 block RAMs (1%). It uses minimal resources just to maintain communication between the FPGAs and CPUs.

CPU: The CPU performance results are obtained from a 1U server that hosts two Intel Xeon X5650 CPUs. Each CPU is clocked at 2.66GHz. The program is written in C language and optimised by Intel Compiler with SSE4.2 and flag *-fast* enabled. OpenMP is used to utilise all of the processor cores.

GPU: An NVIDIA Tesla C2070 GPU is hosted inside a 4U server. It has 448 cores running at 1.15GHz and has a peak performance by 1,288 GFlops. The program is written in C for CUDA and optimised to use all available cores. To get more comprehensive results for comparison, we also estimate the performance of multiple GPUs. The estimation is based on the fact that the first three stages (sampling, importance weighting, lower bound calculation) can be evenly distributed to every GPU and be computed independently, so the data path and data transfer speedup scales linearly with the number of GPUs. On the other hand, the last two stages (particle set resizing and resampling) are computed on the CPU no matter how many GPUs are used; therefore, the CPU time does not scale with the number of GPUs.

6.2. Adaptive PF Versus Nonadaptive PF

The comparison of adaptive and nonadaptive PF is shown in Table I. Both model estimation and experimental results are listed. Initially, the maximum number of particles are instantiated for global localisation. For the nonadaptive scheme, the particle set size does not change. The total computation time estimated and measured are 1.328 seconds and 1.885 seconds, respectively. The difference is due to the difference between the effective and maximum bandwidth of the PCI Express bus.

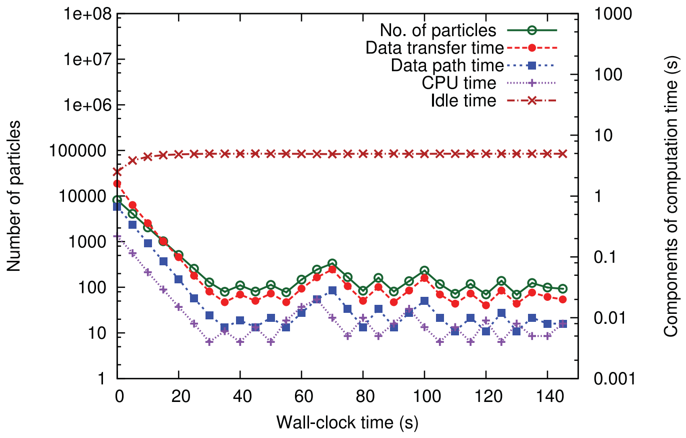


Fig. 7. Number of particles and components of total computation time versus wall-clock time.

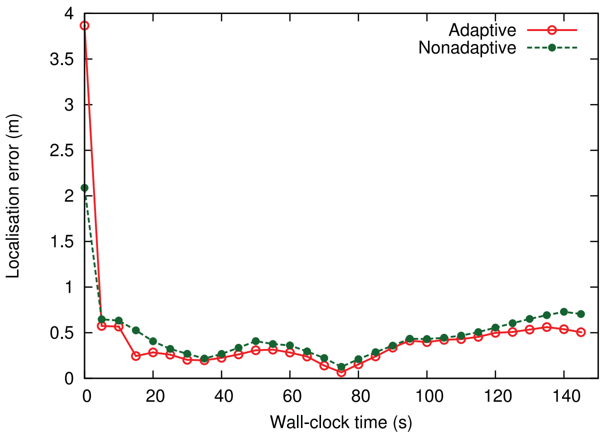


Fig. 8. Localisation error versus wall-clock time.

For the adaptive scheme, the number of particles varies from 573k to 67M, and the computation time scales linearly with the number of particles. From Table I, both the model and experiment show 99% reduction in computation time.

Figure 7 shows how the number of particles and the components of total computation time vary over the wall-clock time (passage of time from the start to the completion of the application). Although the number of particles is reduced in the proposed design, the results in Figure 8 show that the localisation error is not adversely affected. The error is the highest during initial global localisation and is reduced when the robot moves.

6.3. Data Compression

Figure 9 shows the reduction in data transfer time after applying data compression. A higher number of replications means a lower data transfer time. The data transfer time has a lower bound of 0.212 seconds because the data from the FPGAs to the CPUs are not compressible. Only the particle stream after the resampling process is compressed when it is transferred from the CPUs to the FPGAs.

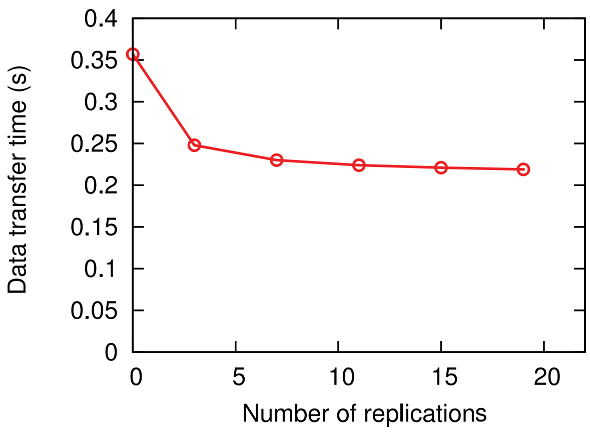


Fig. 9. Effect on the data transfer time by particle stream compression.

6.4. Performance Comparison of HRS, CPUs, and GPUs

Table II shows the performance comparison of the CPUs, GPUs and HRS.

Data path time: Considering the time spent on the data paths only, HRS is up to 328 times faster than a single-core CPU and 76 times faster than a 12-core CPU system with 24 threads. In addition, it is 12 times and 3 times faster than one GPU and four GPUs, respectively.

Data transfer time: The data transfer time of HRS is shown in three rows. The first row shows the situation when the PCI Express bandwidth is 2GB/s. The second row shows the performance when PCI Express gen3 x8 (7.88GB/s) is used such that the bandwidth is comparable with that of the GPU system. When multiple FPGA boards are used, the data transfer time decreases because multiple PCI Express buses are utilised simultaneously. The third row shows the performance when data compression is applied, and it is assumed that each particle is replicated an average of 20 times.

CPU time: The CPU time of HRS is shorter than that of the CPU and GPU systems because part of the resampling process of object particles is performed on the FPGA using the Independent Metropolis-Hastings (IMH) resampling algorithm [Miao et al. 2011]. The IMH resampling algorithm is optimised for the deep pipeline architecture where each particle occupies a single stage of the pipeline. On the CPUs and GPU, the computation of the particles are shared by threads, and therefore IMH resampling algorithm is not applicable.

Total computation time: Considering the overall system performance, HRS is up to 169 times faster than a single-core CPU and 41 times faster than a 12-core CPU system. In addition, it is 9 times faster than one GPU and 3 times faster than four GPUs. Notice that the CPUs violate the real-time constraint of 5 seconds.

Power and energy consumption: In real-time applications, we are interested in the energy consumption per timestep. Figure 10 shows the power consumption of HRS, CPUs, and GPU over a period of 10 seconds (two timesteps). The system power is measured using a power meter that is connected directly between the power source and the system. All curves of HRS show peaks when HRS is at the computation mode and troughs when it is at the low power mode. The power during the configuration period lies between the two modes. On the HRS with one FPGA, runtime reconfiguration reduces the idle power consumption by 34% from 145W to 95W. In other words, over a 5-second timestep, the energy consumption is reduced by up to 33%. On the HRS with

Table II. Performance Comparison of HRS, CPUs and GPU

	CPU(1)[a]	CPU(2)[a]	GPU(1)[b]	GPU(2)[b]	GPU(3)[b]	HRS(1)[c]	HRS(2)[d]	HRS(3)[d]
Clock freq. (MHz)	2660	2660	1150	1150	1150	100	100	100
Precision	single	single	single	single	single	single + custom	single + custom	single + custom
Level of parallelism	1	24	448	896	1792	2+8[e]	4+24[e]	8+24[e]
Data path time (s)	27.530	6.363	1.000	0.500	0.250	0.336	0.168	0.084
Data path speedup	1×	4.3×	27.5×	55.1×	110.1×	81.9×	163.9×	327.7×
Data tran. time (s)	0	0	0.360	0.180	0.090	1.432[f]	0.716[f]	0.358[f]
						0.363[g]	0.182[g]	0.091[g]
						0.223[h]	0.111[h]	0.056[h]
CPU time (s)	0.420	0.334	0.117	0.117	0.117	0.030	0.025	0.025
Total comp. time (s)	27.95	6.697	1.477	0.797	0.457	0.589	0.304	0.165
Overall speedup	1×	4.2×	18.9×	35.1×	61.2×	47.5×	91.9×	169.4×
Comp. power (W)	183	279	287	424	698	145	420	480
Comp. power eff.	1×	0.7×	0.6×	0.4×	0.3×	1.3×	0.4×	0.4×
Idle power (W)	133	133	208	266	382	95	360	360
Idle power eff.	1×	1×	0.6×	0.5×	0.4×	1.4×	0.4×	0.4×
Energy.(J)[i]	677/5115	673/1868	1041/1157	1331/1456	1911/2054	489/595	1896/1914	1994/2012
Energy eff.	1×	1×/2.7×	0.7×/4.4×	0.5×/3.5×	0.4×/2.5×	1.4×/8.6×	0.4×/2.7×	0.3×/2.5×

a 2 Intel Xeon X5650 CPUs @2.66 GHz (12 cores supporting 24 threads).
b 1/2/4 NVIDIA Tesla C2070 GPUs and 1 Intel Core i7-950 CPU @3.07 GHz (4 cores supporting 8 threads).
c 1 Xilinx XC6VSX475T FPGA and 1 Intel Core i7-870 CPU @2.93 GHz (4 cores supporting 8 threads).
d 4 Xilinx XC6VSX475T FPGAs and 2 Intel Xeon X5650 CPUs @2.66 GHz (12 cores supporting 24 threads).
e Number of FPGA data paths and number of CPU threads.
f Each FPGA communicates with CPUs via a PCI Express bus with 2 GB/s bandwidth.
g Each FPGA communicates with CPUs via a PCI Express Gen3×8 bus with 7.88 GB/s bandwidth.
h Each FPGA communicates with CPUs via a PCI Express Gen3×8 bus with data compression.
i Cases for 573k and 67M particles in a 5-second interval.

four FPGAs, the idle power consumption is reduced by 25% from 480W to 360W, and hence the energy consumption is decreased by up to 17%.

The runtime reconfiguration methodology is not limited to the Maxeler systems; it can be applied to other FPGA platforms as well. The resource management software of our system (MaxelerOS) simplifies the effort of performing runtime reconfiguration, and hence we can focus on studying the impact of runtime reconfiguration on energy saving.

To identify the speed and energy trade-off, we produce a graph as shown in Figure 11. Each data point represents the computation time versus energy consumption of a system setting. Among all systems, the HRS with one FPGA has the computation speed that satisfies the real-time requirement while at the same time consumes the

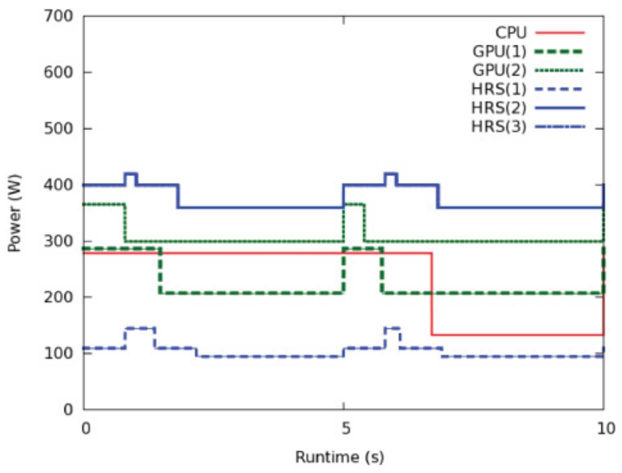


Fig. 10. Power consumption of HRS, CPU, and GPU in one timestep. Notice that the computation time of the CPU system exceeds the 5-second real-time requirement (the lines of HRS(2) and HRS(3) overlap).

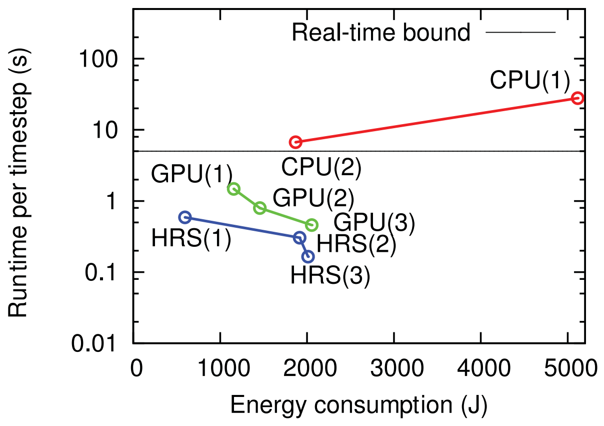


Fig. 11. Runtime versus energy consumption of HRS, CPUs, and GPUs (5-second timestep, 67M particles; refer to Table II for system settings).

smallest amount of energy. All configurations of CPU system cannot meet the real-time requirement. HRS(3)—the HRS with four FPGAs—is the fastest among all systems in comparison; therefore, it is able to handle larger problems and more complex applications.

7. CONCLUSION

This article presents an approach for accelerating adaptive PF for real-time applications. The proposed HRS demonstrates a significant reduction in power and energy consumption compared to CPU and GPU. The adaptive algorithm reduces computation time while maintaining the quality of results. The approach is scalable to systems with multiple FPGAs. A data compression technique is used to mitigate the data transfer overhead between the FPGAs and CPUs.

In the future, HRS will be developed for various PFs that are more compute intensive and have more stringent real-time requirements than the ones described previously. Air traffic management [Chau et al. 2013b] and traffic estimation [Mihaylova et al. 2007] are example applications that can substantially benefit from the proposed approach in meeting current and future requirements. Further work will also be required to automate the optimisation of designs targeting HRS.

ACKNOWLEDGMENTS

The authors thank Oliver Pell at Maxeler Technologies for comments on the article.

REFERENCES

Miodrag Bolic, Petar M. Djuric, and Sangjin Hong. 2005. Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing* 53, 7, 2442–2450.

Miodrag Bolic, Sangjin Hong, and Petar M. Djuric. 2002. Performance and complexity analysis of adaptive particle filtering for tracking applications. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Vol. 1. 853–857.

Thomas C. P. Chau, Wayne Luk, Peter Y. K. Cheung, Alison Eele, and Jan Maciejowski. 2012. Adaptive sequential Monte Carlo approach for real-time applications. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 527–530.

Thomas C. P. Chau, Xinyu Niu, Alison Eele, Wayne Luk, Peter Y. K. Cheung, and Jan Maciejowski. 2013a. Heterogeneous reconfigurable system for adaptive particle filters in real-time applications. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*. 1–12.

Thomas C. P. Chau, James S. Targett, Marlon Wijeyasinghe, Wayne Luk, Peter Y. K. Cheung, Benjamin Cope, Alison Eele, and Jan M. Maciejowski. 2013b. Accelerating sequential Monte Carlo method for real-time air traffic Management. In *Proceedings of the International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*.

Arnaud Doucet, Nando de Freitas, and Neil Gordon. 2001. *Sequential Monte Carlo methods in practice*. Springer.

Alison Eele and Jan M. Maciejowski. 2011. Comparison of stochastic optimisation methods for control in air traffic management. In *Proceedings of the IFAC World Congress*.

Dieter Fox. 2003. Adapting the sample size in particle filters through KLD-sampling. *IEEE Transactions on Robotics* 22, 12, 985–1003.

Markus Happe, Enno Lübbers, and Marco Platzner. 2011. A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. *Journal of Real-Time Image Processing* 8, 1, 1–16.

Daphne Koller and Raya Fratkina. 1998. Using learning for approximation in stochastic processes. In *Proceedings of the International Conference on Machine Learning*. 287–295.

Zhibin Liu, Zongying Shi, Mingguo Zhao, and Wenli Xu. 2007. Mobile robots global localization using adaptive dynamic clustered particle filters. In *Proceedings of the International Conference on Intelligent Robots and Systems*. 1059–1064.

Lifeng Miao, Jun Jason Zhang, Chaitali Chakrabarti, and Antonia Papandreou-Suppappola. 2011. Algorithm and parallel implementation of particle filtering and its use in waveform-agile sensing. *Journal of Signal Processing Systems* 65, 2, 211–227.

Lyudmila Mihaylova, Rene Boel, and Andreas Hegyi. 2007. Freeway traffic estimation within particle filtering framework. *Automatica* 43, 2, 290–300.

Michael Montemerlo, Sebastian Thrun, and William Whittaker. 2002. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings of the International Conference on Robotics and Automation*. 695–701.

Sang-Hyuk Park, Young-Joong Kim, and Myo-Taeg Lim. 2010. Novel adaptive particle filter using adjusted variance and its application. *International Journal on Control, Automation, and Systems* 8, 4, 801–807.

Jaco Vermaak, Christophe Andrieu, Arnaud Doucet, and Simon John Godsill. 2002. Particle methods for Bayesian modeling and enhancement of speech signals. *IEEE Transactions on Speech and Audio Processing* 10, 3, 173–185.

Received June 2013; revised February 2014; accepted March 2014