Accelerating Sequential Monte Carlo Method for Real-time Air Traffic Management

Thomas C.P. Chau[1], James S. Targett[1], Marlon Wijeyasinghe[2], Wayne Luk[1], Peter Y.K. Cheung[2], Benjamin Cope[3], Alison Eele[4], Jan Maciejowski[4]
[1]*Department of Computing,* [2]*Department of Electrical and Electronic Engineering*
Imperial College London, United Kingdom
[3]*Altera Europe Limited, United Kingdom*
[4]*Department of Engineering, University of Cambridge, United Kingdom*
email: {c.chau10, james.targett10, marlon.wijeyasinghe09, w.luk, p.cheung}@imperial.ac.uk, bcope@altera.com, {aje46, jmm1}@cam.ac.uk

ABSTRACT

This paper presents how field-programmable gate arrays (FPGAs) are used to accelerate the Sequential Monte Carlo method for air traffic management. A novel data structure is introduced for a particle stream that enables efficient evaluation of constraints and weights. A parallel implementation for this streaming data structure is designed, and an analytical model is provided for estimating the performance and resource usage of our implementation. We compare our design to implementations on CPU and GPU. We show 9.3 times speed up and 89 times improvement in energy efficiency over an Intel Core i7-950 CPU with 8 threads and demonstrate 1.3 times speed up and 13.5 times improvement in energy efficiency over an NVIDIA Tesla C2070 GPU with 448 cores. We also estimate the performance of FPGA in future scenario and show that FPGA is able to control 15 times and 2.8 times more aircraft than CPU and GPU in real-time respectively.

Keywords

Air Traffic Management, Sequential Monte Carlo

1. INTRODUCTION

Sequential Monte Carlo method (SMC), also known as Particle Filter, is a computationally intensive state estimation technique that is applied to solve dynamic problems involving non-Gaussianity and non-linearity. SMC keeps track of a large number of *particles*, where each contains information about how a system should evolve. The current state of a system is approximated by the collection of particles. The more complex the problem, the larger the number of particles that are needed. Each particle requires many floating-point operations. It is observed that time required to carry out SMC in a sequential manner is prohibitive for use in real-time systems. A detailed discussion of SMC can be found in [1].

Air Traffic Management (ATM) is concerned with the routing and scheduling of aircraft in regions of airspace. The main concerns lie with avoiding dangerous encounters around the terminal manoeuvring area through maintenance of safe separation between aircraft [2]. Nowadays the process is largely performed by humans but attempts are being made to automate this process [3]. The current ATM system is near the upper limit of traffic it can safely accommodate [4]. The level of anticipated growth in aviation travel is predicted to double in the next 20 years [5, 6]. SMC has been studied extensively in controlling air traffic [2, 7]. In [2], SMC is applied on scenarios with multiple aircraft flying under the effects of wind and additional uncertainty. It is observed that the time required to solve such problems is currently prohibitive, not to mention meeting future requirements [2]. Various attempts have been made to simultaneously process SMC using multi-threaded CPU or GPU, in applications such as object tracking [8], signal processing [9] and robot localisation [10]. However, due to complicated iterative nature of ATM, research in accelerating SMC for ATM through parallelisation is limited. The sequential implementation of ATM does not have sufficient speed for real-time practice [2].

This paper presents techniques for accelerating SMC in ATM through the use of FPGA. We also implement the application on multi-threaded CPU and GPU. The contributions of this paper include:

- We develop a novel data structure for a particle stream that enables efficient evaluation of constraints and weights.
- We provide a parallel implementation for this streaming data structure, and an analytical model for estimating the performance and resource usage of our implementation.
- Our FPGA design is 9.3 times faster and 88.5 times more energy efficient than an Intel Core i7-950 CPU with 8 threads, and 1.3 times faster and 13.4 times more energy efficient than an NVIDIA Tesla C2070 GPU with 448 cores.

2. BACKGROUND

The underlying concept of SMC is to approximate a sequence of states as a collection of particles. Each particle is weighted to reflect the quality of an approximation. The collection of particles varies with time as they are propagated iteratively using a sequential sampling and resampling mechanism. In this way a sample is drawn from the population of particles to act as an estimator of the current state.

This work was presented in part at the fourth international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2013), Edinburgh, Scotland, UK, June 13-14, 2013.

The task of ATM is to control aircraft that appear concurrently around the terminal manoeuvring area. The number of aircraft is denoted as N_A. When applying SMC method on ATM, each particle contains information of all aircraft, in which each has a known state. The state consists of the current position in 3 dimensional space (x, y, a), heading angle χ, velocity V and mass M. From change in mass we can calculate the fuel usage. Each aircraft has a destination G.

The states of aircraft change with time. The state at time t is denotes as S_t. The SMC algorithm produces a good choice of controls for the aircraft for one time step.

When applying on ATM, the SMC algorithm is repeated to estimate the path of the aircraft further into the future. The algorithm consists of 5 main steps:

(1) Trajectory planning: ATM control is based on finite horizon optimization. SMC aims to find a good set of controls for each aircraft such that the resultant sequence of states makes good progress towards the goal. The method uses a set of particles, each of which estimates a state and a control sequence for all aircraft.

At time step t, for each particle, the current state S_t is sampled and a control strategy is computed for a horizon H in the future. During planning, the algorithm explores trajectories that emanate from the current state and finds a control strategy until time $t + H - 1$. Eventually, only the first step of the control strategy $C*_t^0$ has the chance to be committed. The control at time step t is denoted as $C_t^{0...H-1}$. The corresponding states across the horizon are denoted as $S*_t^{0,...,H-1}$, which are described as *exploring states*. For simplicity, we use *transition* to describe the movement between exploring states.

$$\begin{pmatrix} x' \\ y' \\ a' \\ V' \\ \chi' \\ M' \end{pmatrix} = \begin{pmatrix} x + \delta t V \cos(\chi') \cos(\tau) \\ y + \delta t V \sin(\chi') \cos(\tau) \\ a + \delta t V \sin(\tau) \\ \chi + \delta t L \sin(\phi)/(MV) \\ V + \delta t(\frac{T}{M} - \frac{D}{M} - g \sin(\tau)) \\ M - \eta \delta t T \end{pmatrix} \quad (1)$$

A state consists of a vector (x, y, a, V, χ, M), and a control is a 3-tuple: roll angle ϕ; pitch angle τ; and thrust T. $g, \eta, P, \rho, C_L, C_D, L, D$ are application-specific constants [2].

(2) Score calculation: SMC uses a score function to evaluate the quality of estimate by each particle. The score at one time step takes distance from destination G, fuel usage and altitude into account.

$$\begin{aligned} J(k) = \alpha_{distance}\, J_{distance}(k) + \\ \alpha_{fuel}\, J_{fuel}(k) + \alpha_{altitude}\, J_{altitude}(k) \end{aligned} \quad (2)$$

As an illustration, the score contributed by distance is shown below.

$$\begin{aligned} J_{distance}(k) = \Big(\sqrt{(x_0 - G_x)^2 + (y_0 - G_y)^2} + k\delta t V_{max} \\ - \sqrt{(x - G_x)^2 + (y - G_y)^2 + (a - G_a)^2)}\Big)/(k + 1) \end{aligned} \quad (3)$$

(3) Constraint handling: The algorithm makes sure there is no failed constraint such that all aircraft fly in a normal manner and are kept apart by minimum safe distances. If an aircraft fails any constraints relating to its own transition or its position relative to the other aircraft then its weight must be set to zero.

Equation 4 shows the single aircraft constraints which ensure that the aircraft only perform reasonable manoeuvres.

Inter-aircraft constraints ensure that aircraft maintain minimum safe distances. Each aircraft is surrounded by a conceptual cylinder of diameter $2P_R$ and height $2P_H$. If the cylinders of two aircraft intersect, the aircraft are considered to be in conflict and they fail the constraint. The constraint for aircraft i is thus expressed in Equation 5:

$$\begin{aligned} (\phi_{min} \le \phi \le \phi_{max}) \wedge (\tau_{min} \le \tau \le \tau_{max}) \wedge (T_{min} \le T \le T_{max}) \\ \wedge (V_{min} \le V \le V_{max}) \wedge (a_{min} \le a \le a_{max}) \wedge (M_{min} \le M) \end{aligned} \quad (4)$$

$$\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < 2P_R\right) \wedge (|a_i - a_j| < 2P_H) \wedge (j \ne i) \quad \forall j \in \{0, ..., N_A - 1\} \quad (5)$$

(4) Weight calculation and normalisation: The weight W of each particle is calculated as the sum of scores of the transitions across the horizon. The weight of particle with failed constraints will be zero.

$$W = \sum_{k=0}^{H-1} J(k) \quad (6)$$

(5) Resampling: The idea of resampling is to remove the particle trajectories with small weights and replicate the trajectories with large weights. The resampling step is to reduce the variance of the particle weights quickly. Otherwise, the inference would be degraded because very few normalized weights are substantial and only a small number of particles can be used for inference.

3. KERNEL DESIGN

Our FPGA design attempts to take advantage of the abundant logic resources of the FPGA to simultaneously process particles in two aspects: a) exploiting spatial parallelism, b) utilising the pipeline by arranging particles in stream. Figure 2 shows how a stream of N_P particles is organised in a data structure that enables efficient evaluation of constraints and weights. The importance of this data structure will emerge after the description of the flow illustrated in Figure 1.

In the *Initialisation* stage, the control of each particle is picked randomly within a permitted range. The weight W is initialised to $1/N_P$. The set of particles are iterated for $Schedule\_R$ times. At each iteration, random noises are added to the control which become $C*_t^{0...H-1}$.

In the *Sampling* stage, the effect of control in the trajectory of aircraft is reflected in states $S*_t^{0...H-1}$ using Equation 1. This stage is iterated for H times to compute all the *exploring states*. Then $S*_t^{0...H-1}$ are evaluated using the score function in Equation 2. Any control that violates constraints in Equation 4 and 5 is marked with a fail flag.

In the *Weighting* stage, a weight W is assigned to each particle as indicated in Equation 6. The sampling and weighting stage is iterated for $Schedule\_W(i)$ times, at each time random noises are added to the controls. W is updated and normalised in every iteration such that those estimations with sustainable good effects are assigned higher values. The value of $Schedule\_W(i)$ increases with the number of resamples performed, because the set of particles is reflecting a closer approximation after each resample.

In the *Resampling* stage, particles are either replicated or deleted according to their weights. In the *Update* stage,

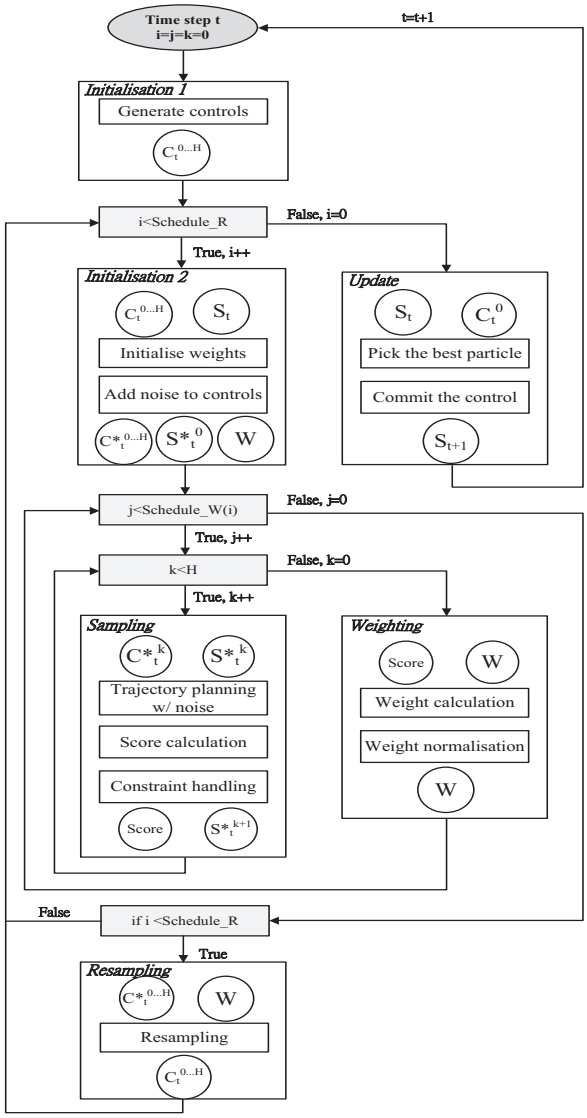


Figure 1: Control and data flow of SMC applied on ATM. Data are in circles; processes and controls are in rectangles.

the best particle in the set is selected and the corresponding control is committed to the aircraft.

Referring to Figure 1, three challenges are identified:

1. Data dependency: the control flow consists of nested loops and conditional branches. To minimise the impact of data dependency, the stream of particles needs to be buffered in an optimised order.

2. Iterative process: in the inner loop, each particle need to go through the sampling stage for $schedule\_R \times schedule\_W \times H$ times.

3. Randomness: the resampling process relies on randomness to replicate and eliminate particles. The ordering of particle is infeasible for streaming.

To address challenge 1, we introduce a novel data structure by organising particles as a stream as shown in Figure 2. A particle contains information for all aircraft $\{A_i | i = 0, ..., N_A - 1\}$. In the *Sampling* stage, data dependency exists as the current exploring state ($S*_t^k$) is used to compute the transition to $S*_t^{k+1}$. Therefore, a stream is divided into H parts, each of which represents all the particles in one step of the horizon. Meanwhile, as the detection of inter-aircraft constraints requires pairwise comparison between aircraft, the data of each aircraft are packed together within a particle such that the results are produced consecutively.

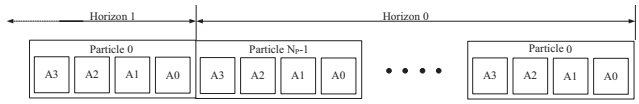


Figure 2: A stream of particles when $N_A = 4$

Challenge 2 implies that the inner loop retains most floating-point operations. Therefore, as illustrated in Figure 3, a customised kernel is designed with dedicated hardware architecture for accelerating the inner loop. Control decisions are kept outside the kernel such that the kernel is fully-pipelined. The kernel can be replicated as many times as FPGA resources allow such that the core computation can achieve the maximum speedup. The names of the blocks are mapped to the operations depicted in Figure 1.

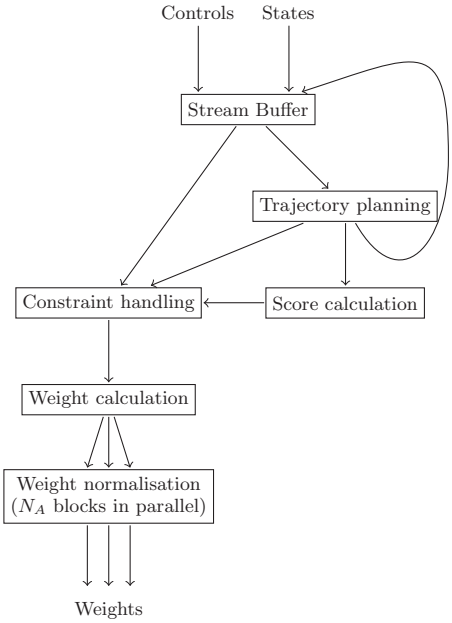


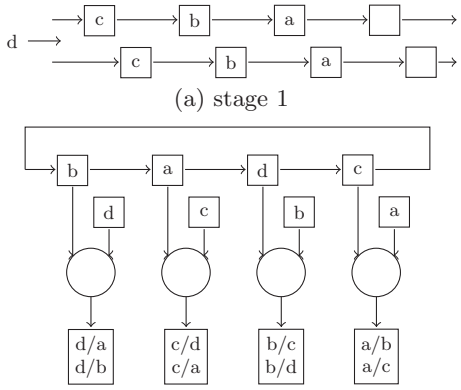
Figure 3: Kernel design

Stream buffer: It is the point where the data enter the kernel. Controls are accepted from outside the kernel for every sampling process. States are only accepted for the first transition ($k = 0$) because future states will be fed back from the *Trajectory planning* block. The *Stream Buffer* communicates with the sender to inform it when the kernel can accept new data. The data of the *Trajectory planning* block need not to be transferred outside the kernel, thus retaining the throughput.

Trajectory planning: On each clock cycle the *Stream Buffer* sends a set of state and control to the *Trajectory planning* block, which then calculates the new state. As the

algorithm finds a control strategy until time $t + H - 1$, the states across the horizon $\{S *_t^k | k = 0...H - 1\}$ need to be passed back to the *stream buffer* for $H - 1$ times. Meanwhile, the state is sent to the *Score calculation* block and the *Constraint handling* block.

Score calculation: This block uses the new states from the *Trajectory planning* block to calculate a score of H transitions.



(a) stage 1

(b) stage 2

Figure 4: Inter-aircraft constraints detection when $N_A = 4$

Constraint handling: This block uses the control from the *stream buffer* and the new state from the *Trajectory planning* block to calculate whether any constraint has been violated. The single-aircraft constrains are computed simply, however, the inter-aircraft constrains require the state of the other aircraft in the particle. An *Inter-aircraft constraints detection block* is designed to make use of the specialised data structure that the information of aircraft is arranged consecutively,

As shown in Figure 4, the block has two shift buffers to buffer a set of aircraft such that collisions can be tested in parallel. Each shift buffer stores N_A locations. In stage one (Figure 4(a)), we shift the buffered aircraft's locations into the shift buffer for N_A cycles. In stage 2 (Figure 4(b)), one buffer remains unchanged, whereas the other buffer continues shifting with the end value sent back into the start of the shift register. This stage process lasts for N_A cycles. For example, to detect collisions between the first aircraft, we check the output of the first element of the static buffer. In the second cycle of stage 2, the output shows whether the first aircraft collides with the second aircraft. If we take an *or* operation of all output values of the first element of the static buffer, we find the result of the collision constraint for the first aircraft. If any constraint fails, the score is replaced with a fail flag. The score of the transition for each aircraft is then sent to the *Weight calculation* block.

By this scheme, the collision detection of all N_A aircraft are done in parallel. However, this method has a bottleneck as it takes $2N_A$ cycles to process N_A cycles of input data. A naïve solution to compute using N_A cycles is to duplicate the entire process such that collision detection is done for two particles together. However, the blocks for evaluating the collision detection (marked as circles in Figure 4(b)) are only in use during stage 2. We only have to duplicate the shift registers and introduce a multiplexer to share the computation section. It is similar to the idea of double buffering. Therefore, when the set of aircraft from one particle is computed in stage 2, the set for the next particle in the stream is filling up the shift registers.

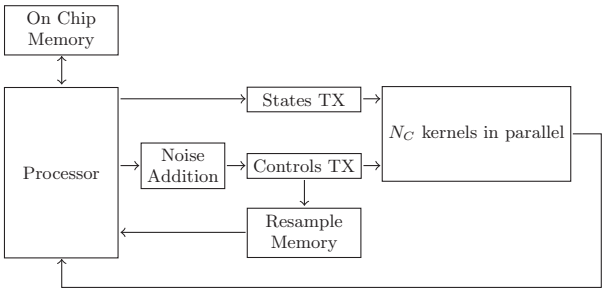


Figure 5: System design (TX - memory transmission blocks)

Weight calculation: The purpose of this block is to combine the scores of the transitions across the horizon. The operation is essentially accumulation. Therefore, for computation of the weight of a particle, this block consists of a FIFO long enough to store a score for each aircraft of each particle. Each score that is received by the block is added to the score at the head of the FIFO. For example, when the score of the first aircraft at $k = 2$ arrives, the head of the FIFO is storing the sum of the scores at $k = 0$ and 1 of the first aircraft. Eventually, each item of the FIFO is the sum of the scores it received for each particle and the scores are then streamed out as the weights.

Weight Normalisation: There is a separate *Weight Normalisation* block for each aircraft. The stream coming from the *Weight calculation* block is considered as being time-multiplexed on aircraft. Therefore, the stream is split to the separate blocks, each of which contains a FIFO long enough to store a weight for each particle. In the first stage, each weight received from the *weight calculation* block is multiplied by the head of the FIFO and stored back in the FIFO. Meanwhile, the value that was stored is added to an accumulator. At the end of each sequence of weights, each weight in the FIFO is the product of all the weights for that particle, and the accumulator value is the normalisation constant. In the second stage, values are read out of the FIFO and they are divided by the normalisation constant. The result of this stage is a stream of the normalised weights stored in the FIFO for the next iteration in the kernel or the resampling stage in the processor.

4. SYSTEM DESIGN

To address the iterative processing and the randomness issue mentioned in challenges 2 and 3 of Section 3, we partition the system into three parts: a) kernels, b) soft-processor, c) custom interface that connects kernels and processor together. The system design is outlined in Figure 5.

Soft-processor and on-chip memory: The soft-processor controls the iterative invocation of kernels, and the streaming of data between each iteration. The processor also allows random addressing of particle data during the resampling process. Resampling is not the bottleneck of computation, so it is performed on the soft-processor in the current design. It is possible to move this stage to hardware in the future.

There are two considerations in organising memory and IO access for the streaming of data into the kernels:

1. The processor covers multiple kernels running in paral-

lel. However, the processor can only write data serially through a bus with limited data width while each kernel can accept a much larger data width. Streaming data into the kernels serially would represent a large overhead.

2. Each kernel receives the same set of data many times.

Therefore, we design a custom interface that can make good use of the kernels and prevent the performance from being constrained by the bandwidth between the kernels and processor.

Custom interface: To achieve maximum performance, data should be streamed into the kernels in parallel. Since the processor is only capable of transmitting data in serial, we introduce two customised memory transmission (TX) blocks as intermediaries between the processor and the kernels. One is used for the states (*States TX*) and the other is used for the controls (*Control TX*). The TX blocks do the serial to parallel conversion, and hence the processor writes data in serial in just once.

For the first iteration of loop *Schedule_W*, the processor writes data into the TX blocks. Then through control registers, the processor instructs *States TX* and *Control TX* to stream data to all the kernels multiple times in parallel. The performance gain is obtained from the fact that in subsequent iterations, the processor does not have to repeatedly transfer large amounts of data. Data about state and control of the aircraft are transferred in burst through Direct Memory Access (DMA). It is not necessary to replicate data across the on-chip memory and the TX blocks, and hence we can reduce the size of on-chip memory.

For every control that is sent to the kernels, it has random noise added to it. The *Noise addition* block intercepts the stream to *Controls TX* and adds the noise.

The resampling step takes the control data and resamples them according to weights that are received from the kernels. During the process, the controls are updated, but at the same time, the original controls are needed throughout the process. Therefore, a *Resampling memory* is introduced to store the original controls so they are not overwritten as *Resampling memory* has the same size as the *Control TX* block. The controls are copied from the *Control TX* block to the *Resampling memory* via DMA such that the copy process occurs in parallel with the kernel operation.

Performance model: Performance changes with a number of design parameters: the number of aircraft (N_A), the number of particles (N_P), the length of horizon (H) and the number of kernels processing data in parallel (N_C). We derive an analytical model for performance estimation as shown in Equations 7 to 9. The model will be used in Section 5 to describe the scalability of the system design.

$$\texttt{Completion time} = \frac{\gamma_1 \frac{N_{Amax} N_P H}{N_C} + \gamma_2 N_A N_P H + \gamma_3 N_P + \gamma_4}{f_{\texttt{Max}}} \tag{7}$$

N_{Amax} is a compile time parameter that accounts for the maximum number of aircraft the design can accommodate. γ_1 to γ_4 are constants calibrated empirically. The γ_1 component is the time spent in the kernels. The γ_2 component is the time taken for the processor to stream and read controls, and to complete the resampling stage. The γ_3 component is the time used for the particles to pass through the *Weight normalisation* step. γ_4 summerise the time used in various initialisation steps and control steps.

$$\texttt{LUT/DSP usage} = \beta_1 N_A + \beta_2 N_A N_C + \beta_3 N_C + \beta_4 \tag{8}$$

Similarly, β_1 to β_4 are constants calibrated empirically. The β_1 component is due to the *Weight normalisation* blocks which has fixed size but duplicated with the number of aircraft. The β_2 component accounts the *Inter-aircraft constraints detection* block as N_A checks happen concurrently in each kernel. The β_3 component summerises the resources used in the rest of the each kernel. The β_4 component is the resources consumed by the soft core processor.

$$\begin{aligned}\texttt{Memory usage} = {} & \alpha_1 N_A H + \alpha_2 N_A + \alpha_3 N_A N_P \\ & + \alpha_4 N_A N_P N_C + \alpha_5 N_C + \alpha_6\end{aligned} \tag{9}$$

The α_1 component is the *Control TX*. The α_2 component is the *State TX*. The α_3 component is the buffer stored in the normalisation blocks. The α_4 component is the stream buffer in each kernel. The α_5 component is register bits used through each kernel for pipelining. The α_6 component is the memory used in the processor and to hold the program and stack that the processor runs.

5. RESULTS

Light air traffic: To simulate a realistic scenario, we apply the flight plan from [2]. The terminal manoeuvring area is considered as a circle of radius 30km. The real-time requirement is 30s (thirty seconds). The plan also sets $N_P = 1024$ and $H = 6$ for decent trajectory planning results. As an example, we study a 4-aircraft scenario, i.e. $N_A = 4$.

A single-kernel FPGA design is implemented on an Altera DE4 development board. Single precision floating-point data type is being used. The board accommodates a Stratix IV EP4SGX530 FPGA with 424,960 ALUTs, 1,024 embedded multipliers and 21.2Mb of block memory. The soft-processor is implemented using a Nios II/f core. Multiple clock domains are employed such that the kernel is clocked at 170MHz and the soft-processor is clocked at 195MHz. The design occupies 24% of ALUT, 88% of embedded multipliers and 56% of block memory. Due to resource limitation, we can only implement one kernel on the DE4 board.

If more embedded multipliers are available on the FPGA, we can replicate more kernels to process data in parallel. We target our design to a Stratix V 5SGSD8 FPGA with 524,800 ALUTs, 1,963 DSP blocks and 52.6 Mbit of block memory. Figure 6 shows the resource utilisation for different number of kernels (N_C). The data are obtained from post placement and routing report of the design tool. The system frequency is targeted at 200MHz.

We observe that resource utilisation scales linearly with the number of kernels. The resource of Stratix V 5SGSD8 FPGA can support up to 5 kernels. The results match the analytical model in Equation 8 to 9. We also notice that the system frequency decreases from 200MHz for 1 kernel to 150 MHz for 5 kernels. The main reason is that the routing is more congested as there are more kernels occupying the resources on the FPGA.

Table 1 shows the speed and power performance of our FPGA designs compared with CPU and GPU. Our designs on FPGA, even for one kernel setting on Stratix IV, are able to meet the 30s real-time requirement. The FPGA design

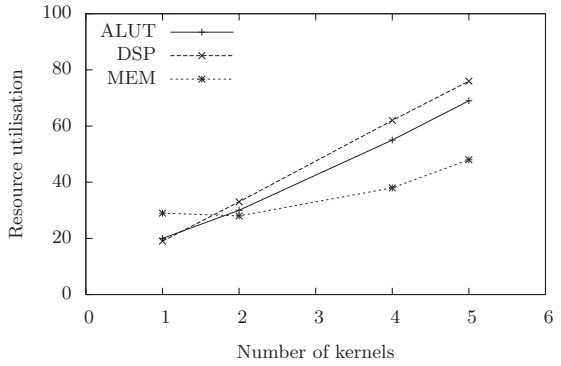


Figure 6: Resource utilisation vs. number of kernels when $N_{Amax} = 4$

Table 1: Performance comparison when $N_{Amax} = 4$

	CPU [a]	GPU [b]	FPGA1 [c]	FPGA2 [d]
No. cores	8	448	1	5
Comp. Time (s)	36.2	5.1	3.9	2.2 [e]
Time eff.	1x	7.1x	9.3x	16.5x
Active power (W)	247	265	26	-
Idle power (W)	133	153	19	-
Energy (J)	8940	1350	101	-
Energy eff.	1x	6.6x	89x	-

[a] CPU: Intel Core i7-950 @3.06GHz with 8 threads, optimised by Intel Compiler.
[b] GPU: Nvidia Tesla C2070.
[c] FPGA1: Altera Stratix IV EP4SGX530.
[d] FPGA2: Altera Stratix V 5SGSD8.
[e] Estimated by Equation 7.

has over 13 times improvement in energy consumption compared with the GPU implementation. Meanwhile, the CPU is not able to meet the real-time requirement and it is almost 90 times less energy efficient than the FPGA design.

Heavy and future air traffic: In addition, we apply the heavy traffic scenario from [2].

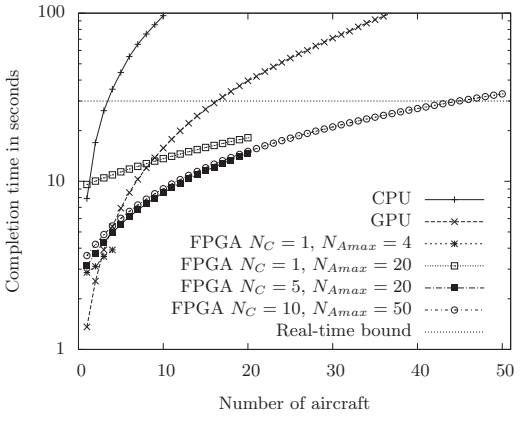


Figure 7: Completion time vs. number of aircraft

Refer to Figure 7, the curve *'$N_C = 1$, $N_{Amax} = 20$'* illustrates the performance of the FPGA designed to accommodate a maximum of 20 aircraft. The kernel processing time is fixed, but the resampling time increases as the aircraft processed varies from 1 to 20. On the other hand, the curve *'$N_C = 5$, $N_{Amax} = 20$'* shows the situation when the FPGA design has 5 kernels. The kernel processing time decreases by 80%, but the resampling time is not improved.

We also study a future scenario which has 50 aircraft circulating together in the terminal manoeuvring area. Though the current FPGA does not have enough resources for such design, we apply the experimental results of *'$N_C = 1, N_{Amax} = 4$'* to calibrate the model mentioned in Section 4. The curve *'$N_C = 10$, $N_{Amax} = 50$'* predicts the performance of an FPGA design that aims to accommodate 50 aircraft's traffic. It shows that the FPGA can meet the 30s real-time requirement upto 45 aircraft and outperforms CPU and GPU which can only handle 3 and 16 aircraft respectively. The FPGA design scales better than CPU and GPU in dealing with extremely heavy air traffic.

6. CONCLUSION

This paper has demonstrated how SMC is accelerated using FPGA technology, which provides a promising solution for large-scale air traffic management. The FPGA design is shown to be faster and more energy efficient than those implemented on CPU and GPU. Future work includes moving the resampling stage to the kernel block such that the FPGA design can scale better for future air traffic. A larger FPGA or a multi-FPGA platform would allow more aircraft to be processed in real-time.

Acknowledgment. This work is supported in part by the European Union Seventh Framework Programme under grant agreement number 257906, 287804 and 318521, by UK EPSRC, by the Croucher Foundation, and by Altera.

7. REFERENCES

[1] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[2] A. Eele and J. Maciejowski, "Comparison of stochastic methods for control in air traffic management," in *Proc. IFAC World Congress*, 2011.

[3] J. A. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "On-line decision support for take-off runway scheduling with uncertain taxi times at London Heathrow airport," *Journal Scheduling*, vol. 11, no. 5, pp. 323–346, Oct. 2008.

[4] European Commission, "Single European sky: report of the high-level group," 2001.

[5] Federal Aviation Authority, "FAA aerospace forecasts for years 2009-2025," 2009.

[6] Eurocontrol Air Traffic Statistics and Forecast Service, "Long-term forecast of air traffic (20082030)," 2009.

[7] I. Lymperopoulos and J. Lygeros, "Sequential monte carlo methods for multi-aircraft trajectory prediction in air traffic management," *Int. Journal Adaptive Control and Signal Processing*, vol. 24, no. 10, pp. 830–849, 2010.

[8] M. Happe, E. Lübbers, and M. Platzner, "A multithreaded framework for sequential monte carlo methods on CPU/FPGA platforms," in *Proc. Int. Workshop Reconfigurable Computing: Architectures, Tools and Applications*, 2009, pp. 380–385.

[9] G. Hendeby, J. Hol, R. Karlsson, and F. Gustafsson, "A graphics processing unit implementation of the particle filter," in *Proc. European Signal Processing Conf.*, 2007, pp. 1639–1643.

[10] T. Chau, X. Niu, A. Eele, W. Luk, P. Cheung, and J. Maciejowski, "Heterogeneous reconfigurable system for adaptive particle filters in real-time applications," in *Proc. Int. Symp. Applied Reconfigurable Computing*, 2013, pp. 1–12.