52nd IEEE Conference on Decision and Control
December 10-13, 2013. Florence, Italy

Parallelisation of Sequential Monte Carlo for Real-Time Control in Air Traffic Management

Alison Eele, Jan Maciejowski, Thomas Chau and Wayne Luk

Abstract— **This paper presents the parallelisation of a Sequential Monte Carlo algorithm, and the associated changes required when applied to the problem of conflict resolution and aircraft trajectory control in air traffic management. The target problem is non-linear, constrained, non-convex and multi-agent. The new method is shown to have a 98.5% computational time saving over that of a previous sequential implementation, with no degradation in path quality. The computation saving is enough to allow real-time implementation.**

I. Introduction

Air Traffic Management (ATM) is concerned with the routing, safety and scheduling of aircraft in regions of airspace. Currently this role is performed by Air Traffic Control (ATC) and is a very human-oriented process though steps towards autonomy have been made in recent years [1]. ATC is responsible for management of all stages of flight, from pre-flight plans, updates and additional instructions based on traffic flow to landing scheduling. The majority of ATC concern lies with the avoidance of dangerous encounters through maintenance of safe separation between aircraft. The role of ATC is further complicated by the introduction of uncertainty. This is introduced by the effect of the wind, incomplete knowledge of the physical coefficients of the aircraft and imprecision in the execution of ATC commands. Probabilistic models can be used to estimate the distribution of the future state of the aircraft given the current state.

There are many existing multi-aircraft trajectory optimisation methods, most focus purely on cruise-like conditions where trajectories can be approximated in two dimensions or movement in the third dimension is highly penalised for passenger comfort [2]–[4]. Methods such as mixed integer linear programming (MILP) require a linearisation of the problem and are too slow on centralised problems with large numbers of vehicles due to the number of binary variables [5]. Others use techniques like genetic algorithms to parameterise and optimise aircraft behaviour [6].

In model predictive control (MPC) an open loop optimal control problem is solved at each time step e.g. [7], [8]. This optimisation problem can be non-convex, mostly due to non-linear dynamics, non-convex constraints. Such optimisation problems are challenging to solve and can require sophisticated optimisation techniques. This paper is concerned with augmenting Sequential Monte Carlo optimisation [9] to solve the control problem at each time step. Stochastic optimisation allows for modelling the uncertainty present in ATC whilst also being able to cope with non-linear dynamics, constraints and an objective function.

Sequential Monte Carlo (SMC) optimisation is better known for its use in ‘particle filtering’ for model estimation [9]. It was observed that MPC optimisation shares similar features to that of model estimation and through this connection SMC was applied to MPC optimisation by [10]. Previous work by the authors [11] compared two stochastic methods along with a standard linearised method to give an indication of the quality of the stochastic solutions when given a limited computational timespan. The work of [10]–[12] includes examples of SMC applied to MPC of multiple vehicles considering two and three dimensional trajectories under the predicted effects of wind and additional uncertainty. These works also observed that the time required to solve such problems was currently prohibitive despite the positive features of the solutions and indicated that potential time savings could be achieved through parallelisation.

Parallelisation has become a popular topic of research due to improvements in the accessibility and availability of software and hardware tools. Graphics Processing Units (GPUs) are one of the key technologies to emerge into the mainstream. Originally developed as devices for real-time graphics rendering, they have entered into scientific computing circles through exploitation of their many multi-core processors. Monte Carlo methods when implemented on GPUs have a proven record of significant speed up for statistical, chemical and economic applications [13]–[15].

The paper is organised as follows: Section II and Section III reviews MPC and the SMC optimisation method whilst discussing the customisation for the considered application; Section IV explains the adaptations to the algorithm for parallelisation; Section V presents the results of the comparison simulations between the new GPU implementation of SMC and the older sequential implementation; Finally, Section VI presents conclusions.

II. Model Predictive Control

Model Predictive Control (MPC), also known as Receding Horizon Control, is an optimisation-based control strategy. A constrained, finite-horizon, optimal control problem is solved online at each iteration by means of the available state/measurements. This solution yields the controls for the duration of the finite horizon. The first control move of the sequence is then applied to the aircraft. The new states are measured and the optimisation loop begins again. For the

This work was supported by EPSRC (Engineering and Physical Sciences Research Council - UK) Grant No. EP/G066477/1

Faculty of Engineering University of Cambridge, Cambridge, UK (e-mail: {aje46,jmm1}@cam.ac.uk)

Department of Computing, Imperial College, London, UK (email: {c.chau10,w.luk}@imperial.ac.uk)

air traffic control problem this forms a closed loop control system, as depicted in Figure 1. Many alternative forms

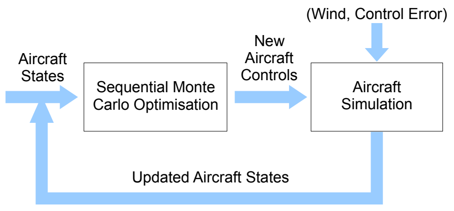

Fig. 1. Graphical Representation of MPC Update Step

of optimisation have been used in the first block of the diagram and often, in simpler systems, linear or quadratic programming can be applied. In this paper Sequential Monte Carlo optimisation is used as the method for solving for the applied control sequence. The remainder of this paper will deal with operations occurring in the SMC optimisation section of the update cycle.

III. Sequential Monte Carlo

The underlying concept of SMC is to approximate a sequence of distributions of interest as a collection of L discrete masses of the variables (more commonly referred to as particles). These particles are weighted by a collection of weights to reflect the shape of the distribution. As the distribution to be approximated can vary with time the weights and particles are propagated iteratively using a sequential importance sampling and resampling mechanism. This sampling and resampling mechanism uses the particles of iteration $J-1$ to obtain new particles at iteration J. In this way a population of particles is iterated upon until a final sample is drawn from the population to act as an estimator of the maximisers. Algorithm 1 summarises the implementation of SMC in the context of the current application. Each step of the algorithm will now be dealt with in further depth with specific reference to the application of ATM.

A. Particles

A particle represents a single instance of the problem with the data of all associated aircraft. The number of particles for optimisation is a design variable dependent on the complexity of the problem to be optimised. The more complex the problem, the larger the number of particles may be needed to adequately characterise the search space. Inside each particle there is: (i) An initial state for each of the N individual aircraft. These initial states for aircraft are the same across all particles; (ii) The control inputs for every step of the MPC horizon H for every aircraft. In this paper's formulation there are 3 controls for each aircraft (thrust, bank angle and pitch angle) totalling $3NH$ control inputs. These control inputs are different for every particle; and (iii) A separate weighting for each of the N individual aircraft in the scenario, used for resampling.

The controls inside particles are initialised as random samples from a bounded uniform distribution. After this initialisation, at line 5 the controls are updated following resampling and perturbation in lines 20–21. The perturbation moves particles locally in the search space, whilst resampling causes particles to cluster around areas of the search space with positive attributes as determined by the objective function. The weights of all aircraft are initialised as $1/L$ where L is the number of particles, and these are updated in line 14.

Algorithm 1 Sequential Monte Carlo

```
1:  $J \leftarrow 0$
2:  Define a SampleSchedule of length $J_{\max}$
3:  Clone all aircrafts' current states for each particle.
4:  Set the weights of all aircraft to $1/L$ where $L$ is the
    number of particles.
5:  For each particle and aircraft in the particle randomly
    generate controls over the entire horizon $H$
6:  **while** $J \leq J_{\max}$ **do**
7:      **for** each particle $l$ **do**
8:          $j \leftarrow 0$
9:          **while** $j \leq$ SampleSchedule(J) **do**
10:             Sample disturbance realisations for each aircraft
                and all time steps to $H$ steps.
11:             For each aircraft simulate its trajectory till $H$.
12:             For each aircraft check the constraints and set the
                aircraft's weight to 0 if it fails a constraint.
13:             Calculate each aircraft's cost.
14:             Scale each aircraft's weightings by their cost.
15:             $j \leftarrow j + 1$
16:         **end while**
17:     **end for**
18:     $J \leftarrow J + 1$
19:     **if** $J < J_{\max}$ **then**
20:         Resample all particles' controls
21:         Peturb all aircraft controls with Gaussian noise.
22:         Set the weights of all aircraft to $1/L$ where $L$ is
            the number of particles.
23:     **end if**
24: **end while**
25: Draw final sample from particle population.
```

B. Trajectory Planning and Disturbance Realisations

In lines 10-11 each aircraft in each particle simulates its future trajectory given the initial state and controls (from the particle's data) and the disturbance samples. The discretised aircraft dynamics model is a standard model with 6 states and 3 inputs

$$x_i(k+1) = x_i(k) + \delta t v_{s,i}(k)\cos(\chi_i(k))\cos(\gamma_i(k)) \quad (1a)$$

$$y_i(k+1) = y_i(k) + \delta t v_{s,i}(k)\sin(\chi_i(k))\cos(\gamma_i(k)) \quad (1b)$$

$$z_i(k+1) = z_i(k) + \delta t v_{s,i}(k)\sin(\gamma_i(k)) \quad (1c)$$

$$v_{s,i}(k+1) = v_{s,i}(k) + \delta t\left(\frac{T_i(k) - D_i(k)}{m_i(k)} - g\sin(\gamma_i(k))\right) \quad (1d)$$

$$\chi_i(k+1) = \chi_i(k) + \delta t\left(\frac{L_i(k)\sin(\phi_i(k))}{m_i(k)v_{s,i}(k)}\right) \quad (1e)$$

$$m_i(k+1) = m_i(k) - \delta t \eta_i T_i(k) \quad (1f)$$

where: δt is the step length; x, y and z are the Cartesian coordinates of the aircraft (z acting as the altitude of the aircraft); m is the total mass of the aircraft; v_s is the true airspeed; χ is the heading angle; γ the climb angle and ϕ the bank angle. The subscript i denotes the i^{th} aircraft. The control variables are ϕ (bank), T (thrust) and γ (climb). Lift L and drag D are calculated using the standard aerodynamic relations with the assumption of steady flight mode. The fuel usage is controlled by the constant η_i. Each aircraft is given both an initial state $X_{0,i} = (x_{0,i}, y_{0,i}, z_{0,i}, \chi_{0,i}, v_{s,0,i}, m_{0,i})$ and goal $X_{t_f,i} = (x_{t_f,i}, y_{t_f,i}, z_{t_f,i})$ to reach by time t_f. The i^{th} aircraft is deemed to have finished its path once it has reached a terminal set defined by the following being satisfied for some k:

$$\begin{aligned}(x_{t_f,i} - x_i(k))^2 + (y_{t_f,i} - y_i(k))^2 \leq K_r^2 \\ \wedge |z_{t_f,i} - z_i(k)| \leq K_h\end{aligned}$$

The disturbance realisations added to the system are the primary method of simulating uncertainty within the system. This uncertainty can come from wind, sensor noise, controller noise and human factors. Within this paper we have limited the disturbances to Gaussian white noise on each of the control inputs and cartesian coordinates. The SMC method is by no means limited to Gaussian white noise disturbance rejection and has been demonstrated with full spatio-temporal wind models in [10]. Within the inner loop of optimisation (lines 9–16) each particle will have drawn $\text{SampleSchedule}(J)$ disturbance realisations applied them to the aircraft inside the particle and simulated that number of trajectories. This inner loop serves to simulate different disturbance scenarios on an aircraft to determine if the controls are valid with respect to constraints, and their fitness with respect to the objective function. This information is summarised in the aircraft's weight within the particle as described in the next two subsections. In applications with no uncertainty there would be no need for repeated planning and simulation of the aircraft and the inner loop would only be executed once. $\text{SampleSchedule}(J)$ is typically set as a monotonically increasing function such that the initial controls are tested sparingly to determine quickly which are clearly infeasible whilst controls after many resampling steps are tested more rigorously with many different disturbances to try and ensure a level of robustness.

C. Constraint Handling

There are two types of constraint handled by line 12. The first is unary constraints which concern only one aircraft at a time. Examples of this include flight envelope constraints and the minimum aircraft mass constraint. The flight envelope constraints provide upper and lower bounds for both the state and control variables. The minimum aircraft mass constraint enforces that the total aircraft mass must always remain above the mass of the aircraft alone without fuel $m(t) \geq m_{\text{Aircraft}}$. This makes it infeasible for the optimisation to ask an aircraft to use more fuel than it is carrying.

The second type of constraint is the binary constraint. These are constraints between pairs of aircraft in the same particle, such as in conflict avoidance. In this paper the protection zone around each aircraft is modelled as a cylinder with horizontal radius P_r and altitude separation of P_h. Two aircraft i and j avoid each other if they satisfy the following constraint for every time step in the MPC horizon:

$$\begin{aligned}(x_i(k) - x_j(k))^2 + (y_i(k) - y_j(k))^2 \geq 2P_r^2 \\ \vee |z_i(k) - z_j(k)| \geq 2P_h \\ \forall k \in \{1, ..., H\}, \forall i, j : i \neq j.\end{aligned}$$

If an aircraft fails any of its unary constraints then its weight stored in the particle is set to 0. If a pair of aircraft fail a binary constraint then both aircraft have their weights set to 0. Any aircraft with a weighting of 0 will not be resampled in the resampling phase of the algorithm.

The constraint-handling method described in this section is a departure from the usual formulation for a SMC optimisation. In a usual formulation a weight is linked to a single particle which would contain the entire simulation of all N aircraft.Then if a single aircraft fails any constraint the entire simulation would be weighted as 0 and all controls from that particle discarded when the particles are resampled in the next iteration. Conversely, by using our implementation, aircraft are weighted individually inside a particle. If a single aircraft fails a constraint the non-failing aircraft weights would be non-zero. This allows their controls to continue to the next iteration however impacts on how the final control sample can be drawn whilst remaining valid for collision avoidance. The application under consideration is highly constrained and this heuristic modification gives greater flexibility in keeping valid control solutions for aircraft which would otherwise have been discarded. This in turn allows a smaller number of particles, as the distribution they are required to model is less complex than in the case of a multi-aircraft weighted particle.

D. Particle Costing

The SMC method maximises a non-negative objective function G_T. In our case this is the sum of the weighted objectives for minimising horizontal distance to goal, altitude difference to goal and minimum fuel usage. The altitude difference is specifically separated out to both discourage the aircraft losing height to reduce fuel usage and to allow for target altitudes to be maintained. Objective functions are obtained by negating the original minimisation function then normalising.

$$G_{1,i}(k:k+H) = \frac{-A_i(k:k+H) + \sup A_i(k:k+H)}{\sup A_i(k:k+H) - \inf A_i(k:k+H)} \tag{2a}$$

$$G_{2,i}(k:k+H) = \frac{(m_i(k) - m_i(k+H)) + \delta t N T_{\max} \eta_i}{\delta t H T_{\max} \eta} \tag{2b}$$

$$G_{3,i}(k:k+H) = \frac{-B_i(k:k+H) + \sup B_i(k:k+H)}{\sup B_i(k:k+H) - \inf B_i(k:k+H)} \tag{2c}$$

$$G_{T,i}(k:k+H) = \sum_{j=1}^{3} \frac{\alpha_j G_{j,i}(k:k+H)}{H} \tag{2d}$$

where $T_{\max}$ is the maximum thrust, the notation $k:k+H$ refers to the summation of costs of each time step to horizon and:

$$A_i(k:k+H) = \sum_{j=k}^{k+H} \sqrt{(x_{t_f,i} - x_i(j))^2 + (y_{t_f,i} - y_i(j))^2}$$

$$B_i(k:k+H) = \sum_{j=k}^{k+H} \sqrt{(z_{t_f,i} - z_i(j))^2} \qquad (3)$$

A is the distance travelled horizontally and B is the distance travelled vertically. Each aircraft in a particle has its own objective function. The cost function is designed such that the total cost J_T and all its constituent costs are normalised between 0 and 1, where 1 represents the best possible path an aircraft could take. The weightings on the terms of the objective function α are subject to priorities of the desired trajectory and determined by the designer empirically:

$$0 \leq G_{T,i}(k:k+H) \leq 1, \quad \sum_{j=1}^{3} \alpha_j = 1 \qquad (4)$$

Cost normalisation is used to bound the numerical behaviour when multiplying the cost by the weighting.

E. Weight Scaling

As previously mentioned in subsection III-A each aircraft in a particle has a weight associated with it. This weight records the degree of success an aircraft has in the simulations with various noise realisations. To do this in line 14 the cost of aircraft i in particle l is multiplied by the weight of aircraft i in particle l each time the inner loop 9–16 is executed. An aircraft only needs to violate 1 constraint in any execution of the inner loop to have its weighting set to 0. This weighting would remain as 0 until resampling (described in the next section) removes that aircraft's proposed controls from the population.

Ideally at each iteration of the inner loop the sum across all particles L of weights associated with aircraft i would be normalised to 1. This would reduce numerical issues as the inner loop progresses. Without such a normalisation step the weight would always be non-increasing

$$W_{i,l}^{j+1} = W_{i,l}^{j} J_{T,i,l}^{j}, \quad W_{i,l}^{0} = 1/L, \quad 0 \leq G_{T,i,l}^{j} \leq 1 \qquad (5)$$

where W_i^j represents the weight and $G_{T,i,l}^j$ the objective function value of aircraft i in particle l at inner loop iteration j and L is the number of particles. The cost is always between 0 and 1 and the weight starts at less than 1. Normalisation of the weight turns out to be a key topic in parallelisation of the method and is discussed in further depth in section IV

F. Particle Resampling and Perturbation

In lines 20 - 21 the aircraft are individually resampled to generate a new population of particles. These new particles have their aircraft controls perturbed by Gaussian white noise to separate particles with similar controls across the local search space. This perturbation also allows particles to explore away from points which were in the original randomly sampled population of controls.

Resampling is done on the basis of sequential importance sampling using a method such as Kitagawa resampling [16]. The higher a weight aircraft i has in particle l compared to the weight of aircraft i in all other particles, the more likely it is to have its controls resampled into the new population.

As the aircraft are resampled separately to form the new particles no conflict avoidance guarantees can hold once resampling has taken place. This arises from the fact that the controls from aircraft i from particle l could now be in a new particle with the controls from aircraft j from particle m. This again arises from the departure from the usual method of one weight per particle mentioned in section III-C, justified by the significant simplification of dimensionality of the search space and the effect on both computation time and particle populations needed. In the previous incarnation [11] when a particle is resampled, then prior to any Gaussian perturbation being added, the guarantee of conflict avoidance holds from the previous iteration. However by separating aircraft and sampling from each aircraft to generate particles individually these same avoidance constraints cannot be assumed to be held until the newly generated particle is tested with disturbance realisations. This resampling alteration therefore must have an effect on drawing the final sample from each of the aircraft distributions with regards to the binary avoidance constraints.

G. Final Sample Selection

This is the final step of the SMC optimisation (line 25) before it hands back to the MPC update process. This step determines the controls to be used by all aircraft for the next update step. In many generic SMC applications this final sample is taken as the mean of the values of the particles. In multi modal distributions this would give an inaccurate estimation of the global optimisers (consider the mean of a control where half of the population steers left around an obstacle and the other half right). An alternative procedure would be to select the mode of the final distribution. However the presence of binary constraints across aircraft control distributions could lead to constraint violation. Therefore throughout this paper we select the best performing particle in the final iteration as our estimate of the maximiser. All particles in the final iteration will have been assessed through the maximum number of noise realisations prior to the final sample being drawn, thus conflict avoidance guarantees will hold at this point.

To find the best performing particle the weights of all aircraft in the particle are multiplied together and the particle with the greatest overall weight is chosen $\max_l \prod_{i=1}^{N} W_{i,l}$. Any particle where an aircraft has failed any constraint in simulation has a weight of 0 associated with the failing aircraft and thus can not be the best performing particle.

IV. Parallelisation Implementation

As previously observed by other work and implemented in alternative applications there is a large degree of speed up that can be exploited by parallelising the SMC algorithm [13]–[15]. This work focuses on implementation on a graphics processing unit (GPU) using the CUDA programming language provided by NVIDIA. CUDA is a scalable parallel programming language similar to C and C++ with the libraries and utility to design kernels to implement on a GPU. These kernels are repeatable functions representing the code that the user wishes to be executed in parallel, using different data. Whereas GPUs can use thousands of threads at the same time, a CPU will only use a few. The execution and near instantaneous switching between threads is how a GPU achieves efficiency on parallel applications. A kernel is executed by an array of threads. Threads are bundled together into blocks for cooperation purposes. If threads need to share information they can use a small but very fast shared memory resource accessible by all threads in the same block. Therefore the primary decisions when implementing an existing algorithm in parallel are: firstly identifying the code to parallelise inside a kernel; and deciding on any use of shared memory.

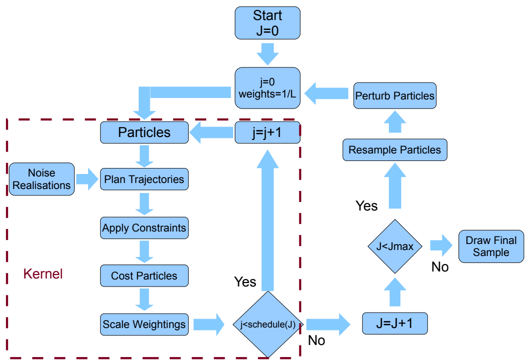

Fig. 2. Graphical Representation of SMC Algorithm with Kernel Highlighted

Figure 2 shows a simplified graphical representation of the SMC algorithm focusing on the operations taking place in a single computation step of the MPC. There is also a boxed region of the flowchart delineating the proposed GPU kernel. This kernel incorporates the inner loop of simulation, specifically lines 9-16 of the algorithm. This code is a bottleneck in a sequential implementation as it must be executed multiple times for each particle, the only difference between each execution being the particle's individual data. This makes it a perfect candidate for parallelisation with one particle per thread, as there is minimal thread interaction.

The normalisation step within the inner loop is non-essential for the smooth running of the algorithm. The resampling step can also operate without the weights being pre-normalised before receiving them. Presuming we remove the normalisation step from the inner loop and therefore any need for any use of shared memory or thread interaction we must instead consider the effects. Weights are initialised as $1/L$ and all costs are normalised between 0 and 1 (with 1 being the best possible cost). Therefore without normalisation the weights will get smaller with each individual step of the inner loop. This now imposes a precision limit on our design as the smallest positive value that a double precision floating point number can represent is $2.2250738585072014 \times 10^{-308}$. To reduce the chance of reaching this precision limit in practice and to effectively double the amount of space available to store these values the weights are premultiplied by the largest number representable in double precision. This scaling is the same for all particles and all weights and thus has no effect on the resampling. Henceforth our implementation of the algorithm no longer includes the normalisation step.

With normalisation removed from the GPU kernel there is no need for the threads to synchronise or communicate and thus the problem can be considered as 'embarrassingly parallelisable'. The remaining implementation challenge is to select an appropriate random number generator for the Gaussian disturbances. In this paper's implementation we have focused on using the NVIDIA provided library CURAND, specifically the XORWOW (Xor Shift added with Weyl sequence generator [17]) which seeds each thread and maintains the state of each random generator between kernel calls. This generator was used for its superior speed compared to that of the others included in CURAND.

V. Results

The parallelised SMC method was compared to the non parallelised implementation on test cases across a range of aircraft numbers. Both implementations are deterministic in computation time for each MPC time step and thus all comparisons in this section will be by time step. The computation time is linked to: the number of aircraft; sample schedule length; sample schedule function; the MPC horizon length; the number of particles used (even if all were implemented in parallel some overheads for memory writes remained); and thread layout. The noise experienced by both implementations was drawn from the same distribution however the samples differed between the implementations. Each test case had 2–20 aircraft. These aircraft were randomly given a starting point and target on the perimeter of a cylinder of height $6\,\text{km}$ and radius $20\,\text{km}$. Aircraft were given a starting heading directly towards the centre of the cylinder. No aircraft started off in conflict and no terminal points were in conflict. Both methods were given a horizon length of 6 steps to work with and the aircraft were constrained with the same maxima and minima for the bounding constraints. The objective function was weighted by a mix of importance and relative size using the empirically determined $\alpha_1 = 0.5, \alpha_2 = 0.1, \alpha_3 = 0.4$. MPC time steps were 10 seconds in length ($\delta t = 10$). The sequential implementation was written in C and compiled with the gcc compiler on Linux. The comparison tests were done on a range of computers with 96 GB memory and 2.67 GHz speed processors with no multi-core threading. The parallelised implementation was performed on an NVIDIA GeForce 580, which has 512 cores on the Fermi architecture (32 cores per streaming multiprocessor (SM) and 16 SMs). Both implementations used a $J_{\max} = 100$, with a schedule function of $\text{SampleSchedule}(J) = \lfloor(3 + 5e^{0.05J})\rfloor$ and $L = 1024$ particles (which have been naively arranged as 32 blocks of 32 threads for the GPU implementation). The two implementations used the same underlying code with only adaptations to allow running on a GPU for the parallel version. Code optimisation through compilers has been used but this paper has not considered the effects of varying the numerical precision.

Figure 3 shows the average computational time required to solve for 1 MPC time step for each of the implementations

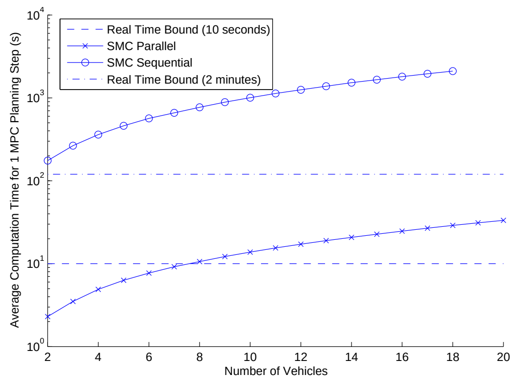

Fig. 3. Comparison of average computation time taken to solve a single MPC planning step for both implementations with differing aircraft numbers

and how this varies with the number of aircraft in the problem. There is approximately a fixed 98.5% computation time saving when implemented in a parallel, compared with a sequential implementation. Also of interest is the number of points which lie below the dashed 10 seconds per step line (Figure 3). This line represents the point at which the method can be considered 'realtime' with one step lookahead. The largest number of aircraft tested on both implementations was 20. This took 33.3 seconds to complete an MPC planning step in the parallel formulation and would require another 70% speed up to be practical in real time. Conversely the value of δt used in this paper can be considered very conservative with regards to safety and the need for regular updates. Were this system to be extended and applied as an advisory system for ATM there would be no need for a 10 second update sequence as this would arguably cause information overload for both the air traffic controllers and pilots. A more reasonable update time would be in the region of thirty seconds to two minutes. Thus the method would be fast enough for 20 vehicle problems given the current results.

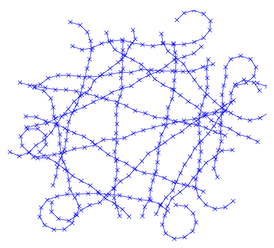

Fig. 4. Twenty Aircraft problem solved by SMC viewed from above

Since the only difference between the two implementations is the parallelisation, the paths generated are the same with allowance for the stochastic nature of the SMC algorithm. Figure 4 displays an overhead view of a set of paths generated by the method for a 20-aircraft case.

VI. Conclusions

In this paper we customised the stochastic optimisation method Sequential Monte Carlo (SMC) for non-linear, non-convex air traffic management (ATM) problems with multiple aircraft within a model predictive control (MPC) setup. This method was then parallelised using a Graphics Processing Unit (GPU) and the CUDA programming environment. The sequential and parallelised implementations were then compared on a series of test cases with between 2-20 aircraft.

The new parallelised method has a 98.5% computational time saving over that of a previous sequential implementation with no degradation in path quality between sequential and parallel implementations. This computation time saving is enough to treat the method as real time for update cycles shorter than two minutes. The underlying optimisation algorithm is not dependant on the specific aircraft dynamics model, cost functions, probability distributions or spectra of disturbances. Therefore all of these could be varied if desired, without invalidating our approach to the ATM problem.

References

[1] J. A. D. 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 of Scheduling*, vol. 11, no. 5, pp. 323–346, 2008.

[2] J. Hu, M. Pradini, and S. Sastry, "Optimal coordinated maneuvers for three dimensional aircraft conflict resolution," *AIAA Journal of Guidance, Control and Dynamics*, vol. 25, pp. 888–900, 2002.

[3] R. Ghosh and C. Tomlin, "Maneuver design for multiple aircraft conflict resolution," in *Proceedings of the American Control Conference*, (Chicago, Illinois), pp. 672–676, June 2000.

[4] A. Bicchi and L. Pallottino, "On optimal cooperative conflict resolution for air traffic management systems," *IEEE Transactions on Intelligent Transporation Systems*, vol. 1, no. 4, pp. 221–232, 2000.

[5] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the American Control Conference*, vol. 3, (Anchorage, Alaska), pp. 1936 – 1941, May 2002.

[6] D. Delahaye, N. Durand, J.-M. Alliot, and M. Schoenauer, "Genetic algorithms for air traffic control system," in *14th IFORS Triennal Conference*, 1996.

[7] J. Maciejowski, *Predictive Control with Constraints*. Pearson, Prentice Hall, 2002.

[8] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

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

[10] N. Kantas, J. Maciejowski, and A. Lecchini-Visintini, "Sequential Monte Carlo for model predictive control," in *International Workshop on Assessment and Future Directions of NMPC*, (Pavia, Italy), September 2008.

[11] A. Eele and J. Maciejowski, "Comparison of stochastic methods for control in air traffic management," in *International Federation of Automatic Control (IFAC) World Congress*, (Milan), September 2011.

[12] J. P. Villiers, S. Godsill, and S. Singh, "Particle predictive control," *Journal of Statistical Planning and Inference*, vol. 141, pp. 1753–1763, 2011.

[13] A. Lee, C. Yau, M. Giles, A. Doucet, and C. Holmes, "On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods," *Journal of Computational & Graphical Statistics*, vol. 19:4, 2010.

[14] J. Rosenthal, "Parallel computing and Monte Carlo algorithms," *Far East Journal of Theoretical Statistics*, vol. 4, pp. 207–236, 2000.

[15] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, "Accelerating molecular modeling applications with graphics processors.," *Journal of Computational Chemistry*, vol. 28, no. 16, 2007.

[16] G. Kitagawa, "Monte Carlo filter and smoother for non-gaussian non-linear state space models," *Journal of Computational and Graphical Statistics*, vol. 5, pp. 1–25, 1996.

[17] G. Marsaglia, "Xorshift random number generators," *Journal of Statistical Software*, vol. 8, no. 14, 2003.