System Design and Algorithmic Development for Air Traffic Control Based on Associative Processor

Mike Yuan, *Student Member, IEEE,* Johnnie Baker, *Member, IEEE Computer Society,* Frank Drews, *Member, IEEE,* Lev Neiman, *Student Member, IEEE,* and Will Meilander, *Life Member, IEEE*

Abstract—This paper proposes a solution to air traffic control (ATC) using an enhanced SIMD machine model called an Associative Processor (AP). This differs from previous ATC systems that are designed for MIMD computers and have a great deal of difficulty meeting the predictability requirements for ATC, which are critical for meeting the strict certification standards required for safety critical software components. The proposed AP solution will support accurate and meaningful predictions of worst case execution times and will guarantee all deadlines are met. Also, the software will be much simpler and smaller in size than the current corresponding ATC software. An important consequence of these features is that the V&V (Validation and Verification) process will be considerably simpler than for current ATC software. Additionally, the associative processor is enhanced SIMD hardware and is considerably cheaper and simpler than the MIMD hardware currently used to support ATC. The ClearSpeed CSX600 accelerator is used to emulate the AP model. A preliminary implementation of the proposed method has been developed. Experimental results comparing MIMD and CSX600 approaches are presented, and show that our solution can guarantee 8 real-time ATC tasks to be finished within their hard deadlines for a large scale of aircraft. The performance of CSX600 has better scalability, efficiency, and predictability than that of MIMD.

Index Terms—Air Traffic Control (ATC), SIMD, MIMD, Associative Processor (AP), Conflict detection and resolution (CD&R), ClearSpeed CSX600, Validation and Verification(V&V), Federal Aviation Administration (FAA).

1 Introduction

The Air Traffic Control (ATC) system is a real-time system that continuously monitors, examines, and manages space conditions for thousands of flights by processing large volumes of data that are dynamically changing due to reports by sensors, pilots, and controllers, and gives the best estimate of position, speed and heading of every aircraft in the environment at all times. The ATC software consists of multiple real-time tasks that must be completed in time to meet their individual deadlines. The FAA has spent a great deal of effort on finding a predictable and reliable system to achieve *free flight* which would allow pilots to choose the best path to minimize fuel consumption and time delay rather than following pre-selected flight corridors [12], [22], [30]. Massive efforts have been devoted to finding an efficient MIMD solution to the ATC problems for many years. The AAS project was studied for two years and then had to be abandoned. Nevertheless the program was funded and canceled after about ten years work by a large team [20], [21].

The most critical issue of *free flight* is conflict detection and resolution. The performance of all CD&R algorithms available depends on aircraft state estimation according to the comprehensive survey of Kuchar and Yang [14]. The Kalman filter[1], [4] is the central algorithm for the majority of all modern tracking systems, known as $\alpha-\beta$, $\alpha-\beta-\gamma$ filters. The major problem with the single Kalman filter is that it does not predict well when the aircraft makes an unanticipated change of flight mode such as making a maneuver, accelerating etc [17]. Many adaptive state estimation algorithms have been proposed [18], [15], [2], [29]. The Interacting Multiple Model (IMM) algorithm [3], [16] runs two or more Kalman filters that are matched to different modes of the system in parallel. It uses a weighted sum of the estimates from the bank of Kalman filters to compute the state estimate. IMM and its variants have been applied to single and multiple aircraft tracking problems in [18]. However, it becomes inaccurate for tracking multiple aircraft as the number of aircraft increases. Current MIMD implementation of this algorithm is computationally very intensive. Hwang et. al. [9] propose that the mode likelihood function can be used to improve the estimation results of IMM algorithm. The likelihood function uses the mean of the residual produced by each Kalman filter. A heuristic algorithm that evaluates correlation error values has been shown to provide better results than the Kalman filter [17].

A comprehensive survey of the CD&R algorithms is presented in Kuchar and Yang [14]. In [13], Krozel et. al. propose one centralized strategy that is controller-oriented and two decentralized strategies that are user-

- *M. Yuan, J.Baker and W. Meilander are with the Department of Computer Science, Kent State University, Kent, OH, 44242.*
  E-mail: {myuan, jbaker}@cs.kent.edu, willcm@charter.net
- *F. Drews and L. Neiman are with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH.*
  E-mail: drews@ohio.edu, lev.neiman@gmail.com

oriented. In the centralized approach, a central agent analyzes the trajectories of the aircraft and determines resolutions. In the two decentralized strategies, each aircraft resolves its own conflicts as they are detected. In [30], Yang et. al. propose a conflict alerting logic based on sensor and trajectory uncertainties, with conflict probability based on Monte Carlo simulation. Chiang et. al. [5] propose CD&R algorithms from the perspective of computational geometry. Paielli et. al. [23] and Prandini et. al. [25] propose analytic algorithms for computing probability of conflict. Many of the algorithms consider only two aircraft. For example, Krozel et. al. [13] show that neither their centralized nor decentralized CD&R algorithms can guarantee safety for multiple aircraft when the number of aircraft is growing. Furthermore, many algorithms propose optimization schemes that are not guaranteed to be completed within real-time deadlines. Due to the increasing number of FAA problems, FAA is inviting proposals for new and efficient CD&R [34].

On the other hand, several papers [19], [20], [21], [31] have used an Associative Processor (AP) to manage ATC computation. The AP is an enhanced SIMD model which can execute several global operations in constant time, as will be explained in detail in Section 3. The assumed maximum number of aircraft being tracked by one air traffic control center is 4000 IFR (instrument flight rules) aircraft and 10000 VFR (visual flight rules) aircraft, for a total of 14000 aircraft [19], [20]. This involves maintaining flight information for all flights in the region controlled by one air traffic control region plus all flights in adjacent regions. There are 20 flight control centers in the contiguous U.S. states plus one each for Hawaii and Alaska.

An AP for ATC will have sufficient parallel processor memory so that the records for each aircraft can be stored permanently in the memory of a single processor, as the movement of large amounts of data is very time consuming. The memory size and speed for the parallel processors in a SIMD with upward tens of thousands of processors is typically small due to cost issues, thereby restricting the number of aircraft that each processor can manage. Also, if the maximum number of aircraft assigned to a single processor is large, this also severely impacts the AP's performance. For example, if a maximum of 100 aircraft are assigned to one processor, then the running time is increased by a factor of more than 100 over the running time achieved if each processor is assigned at most one aircraft.

A primary reason that an AP is ideal for the ATC problem is the speedup that it can achieve. If an AP has n processors, it can execute simultaneously n instances of the same task in essentially the same time that it requires MIMD to provide one instance of this task. The AP achieves near-optimal speedup. Although, assigning multiple aircraft to each processor is certainly feasible, the assignment of at most one aircraft to each processor is ideal. This will allow the maximum number of different ATC tasks to be processed and meet the individual deadlines for each task.

A second reason that the AP is ideal for ATC is that it has already been shown that the AP can handle this problem [19], [20], [21]. In fact, the STARAN AP was designed by Kenneth Batcher and built by Goodyear Aerospace explicitly for ATC. The ASPRO AP was a second generation STARAN and about 150 units were purchased by the NAVY and used for the related problems of air defense systems. These APs were considered to be natural database machines and intended for dynamic database applications like ATC. The deterministic architecture of a SIMD supported extremely accurate worst case estimates of running time, allowing a static schedule to be used instead of dynamic scheduling. Data is rarely moved, which substantially reduces the communication cost below the data movements costs that would be incurred in a MIMD system. The associative properties supported constant time execution of standard database activities [11]. As discussed in [20], [21], many time-consuming activities like dynamic scheduling, load balancing, shared resource management, assuring ACID properties for database transactions are either simple to handle or activities that are not needed for AP, due largely to its single instruction stream. That AP is a very different type of parallel machine is demonstrated by the fact that the proofs of most or all of the well-known NP-hard problems involving multiprocessors [6] do not apply to AP and, in fact, software solutions or approximate solutions to these types of problems are not needed in solutions of other problems.

Since the number of aircraft continues to increase rapidly and there is a preference at FAA to consolidate similar activities to minimize aircraft handoffs when possible and to perform multiple backup computations for redundancy purposes (e.g., for nearby regions), it is realistic to expect that ATC could fully utilize an AP with $100k$ or more PEs in the implementation of its current NextGen project.

In this paper, the ClearSpeed CSX600 is used to emulate the AP solution to the ATC problem. Our ATC algorithms use CSX600 SIMD features to process up to 96 instances of the same task synchronously. The architecture of our proposed AP is much smaller and simpler than previous and current ATC MIMD architectures. Our preliminary experimental results in Section 6 compare the performance and predictability of CSX600 implementation and two MIMD implementations of aircraft tracking algorithm, and give timings for 8 real-time tasks.

This paper is organized as follows. Section 2 overviews CSX600 architecture and programming concepts. We discuss emulating the AP on the CSX600 in Section 3. The overall system design and static scheduling is illustrated in Section 4. Section 5 presents our approaches for 8 key ATC tasks. Section 6 presents experimental results. Conclusions and future work are presented in Section 7.

Fig. 1. CSX600 accelerator board

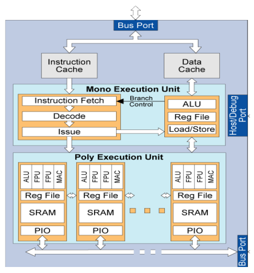

Fig. 2. MTAP architecture of CSX600

2 Overview of ClearSpeed Accelerator Board

The ClearSpeed accelerator board shown in Figure 1 is a PCI-X card equipped with two CSX600 coprocessors. The CSX600 board is a multi-core processor with two CSX600 coprocessors, each with 96 processing elements (PEs) connected in the form of a one-dimensional array. At present, we are only using one of the two coprocessors in order to obtain a more SIMD-like environment. This multi-core section is called a multi-threaded array processor (MTAP) core, and the architecture is shown in Figure 2. The programmer only has to provide a single instruction stream, and the instructions and data are dispatched to the execution units that have two parts: one is mono that functions as a control unit and processes non-parallel data, and the other is poly that has 96 PEs. At each step, all active PEs execute the same command synchronously on their individual data. Each PE has its own local memory of 6 Kbytes, a dual 64-bit FPU, its own ALU, integer MAC, registers and I/O. The PEs operate at a clock speed of 250 MHz. The aggregate bandwidth of all PEs is specified to be 96 Gbytes/s, which is for on-chip memory. Further information on the hardware architecture can be found in the documentation [35].

The ClearSpeed accelerator provides the C^n language as the programming interface for the CSX600 processors. It is very similar to the standard C programming language. The main difference is that it introduces two types of variables, namely mono and poly variables. The mono variables are equivalent to common C variables and used by the control unit. The poly values have an instance on each PE and are processed by the PEs. Further information on the associated software can be found in the documentation [36], [37]. There are three library functions for data transfer on the ClearSpeed. The first one is from mono to poly *memcpym2p*, second one is from poly to mono *memcpyp2m*, and third one is to exchange data with adjacent PEs using the *swazzle* network, which is a ring network connecting all the processors together. More details of usages of library functions can be found in the documentation [37], [7].

TABLE 1
Timings of Associative Functions

Associative functions	Timings in C^n	Timings in assembly
max	5.257	3.654
min	5.257	3.654
AND	7.024	NA
OR	7.364	NA
associative search	29.2	NA
any	0.281	NA
nany	15.147	8.245
get	13.116	7.876
next	13.032	7.816
broadcast	100	NA

3 Emulating the AP on the CSX600

An Associative Processor (AP) [19], [27] is a SIMD machine with additional hardware *I/O* enhancements. A more complete and careful listing of the associative properties follows [24], [11]:

- MAX and MIN: Global reduction operations of integers or real numbers that occur in each instance of the same record across all active PEs using maximum or minimum.
- AND and OR: Global reduction operations of Boolean values in each instance of the same record across all active PEs using AND or OR.
- Associative search: Finds all instances of the same record across all active PEs whose data values match the search pattern. The active PEs whose data value in the record matches the search pattern are called *responders*, and the active PEs whose data value in the record does not match the search pattern are called *non-responders*.
- Any-Responder: ANY is to determine if there is at least one *responder* after an associative search.
- Pick-One: Selects one responding *responder* from the set of responding PEs. It is implemented by ClearSpeed using GET and NEXT operations.
- Broadcast data or instructions from the control unit to PEs.

We have implemented these associative functions on the CSX600. To evaluate their running time, we store 10 records in each PE and perform each associative operation once for each of these 10 records. The timing results in microsecond (μsec or 10^{-6} second) are shown in Table 1. Although they are not constant time, they are very efficient, and additionally establish that we can efficiently emulate an AP using CSX600.

4 ATC System Design

4.1 ATC Data Flow

The overall system design is shown in Figure 3, which is a modification of a figure in [27]. The executive box controls the single instruction stream of AP using static scheduling. All control paths are from ATC in the executive box to all of the tasks, e.g., report correlation and tracking etc. Controller input simulates sporadic requests, e.g., weather change, controller input, etc. Radar reports data are simulated by data from data lines to two modems and transferred from host to CSX600 PEs. Tracks are simulated from flight plans in PEs. The radar reports and tracks are used for report correlation and tracking task. The outputs of tracking task are used for cockpit display, controller display update, terrain avoidance, conflict detection and resolution (CD&R) and final optimization. The results of terrain avoidance, CD&R and final approach are used for cockpit display and controller display update. The results of terrain avoidance and CD&R are used for automatic voice advisory that transfers results to automatic voice advisory driver in host and produces voice output. The resolution advisories of CD&R task are sent to controllers.

4.2 Static Scheduling

The report correlation and tracking (1) is executed every 0.5 second, cockpit display (2), controller display update (3) and sporadic requests (4) are executed every one second, automatic voice advisory (5) is executed once every 4 seconds, terrain avoidance (6), conflict detection and resolution (7), and final approach (8) are executed every 8 seconds.

An 8 second period is split into 16 0.5 second periods, which is denoted as t. Tasks 1, 2 and 3 are executed in the first, third, fifth, seventh, ninth, $11th$, $13th$, $15th$ t. Tasks 1 and 4 are executed in the second, sixth and tenth t. Tasks 1, 4 and 5 are executed in the fourth and $12th$ t. Tasks 1, 4 and 6 are executed in the eighth t. Tasks 1, 4 and 7 are executed in the $14th$ t. Tasks 1, 4 and 8 are executed in the $16th$ t.

5 AP Solution for ATC Tasks Implemented on CSX600

This section describes the algorithms of eight real-time tasks, report correlation and tracking, cockpit display, controller display update, sporadic requests, automatic voice advisory, terrain avoidance, conflict detection and resolution (CD&R) and final approach (runway optimization). The solutions are implemented on CSX600 architecture, but all of them can be mapped on AP.

5.1 Report Correlation and Tracking

The report correlation and tracking algorithm is shown in Algorithm 1. The input data are radar reports that are simulated in host and track records in PEs. If total time consumed is considered, this is easily the ATC task that consumes the most time, as it is performed much more frequently than the other tasks.

Algorithm 1 Algorithm for Aircraft Tracking

```
1: Radar reports are transferred from host to mono memory using $CSAPI\_write\_mono\_memory$ function, then from mono to PEs using $memcpym2p$.
2: Boxes are created around each radar report and each track in each PE to accommodate report and track uncertainties.
3: Check intersection of each report box with every track box in each PE.
4: If there is an intersection, the radar report and the track are correlated. The $match\_count$ of this report is incremented, which indicates that it correlates one track, and its ID and positions are entered into the correlated track's record.
5: All radar reports in each PE are transferred to next PE using the $swazzle$ function, and steps 3 and 4 are repeated. If two tracks correlate to the same radar report, this report is dropped to avoid velocity errors.
6: After 96 iterations, all reports have been compared with all tracks. A track that is not produced by noise might not correlate to any reports because the aircraft that it corresponds to is maneuvering.
7: Double the box sizes of tracks that have not correlated with any reports to increase their probability to intersect a report box and repeat steps 3 to 6 to compare them with uncorrelated reports.
8: Triple the original box sizes of tracks that have not correlated yet, and run the algorithm again.
9: After 3 rounds, if there are still any uncorrelated reports, they are used to start new tracks.
```

5.2 Cockpit Display

First the associative operation PickOne is used to select one aircraft. Next, the broadcast operation is used to broadcast the x, y and altitude coordinates of the plane picked in step 1. All processors that handle each plane compute its x-distance, y-distance and altitude distance between its location and the location of the aircraft selected in step 1. Then select aircraft that are approaching this aircraft and within conflict distance in 2 minutes, i.e., using the conflict detection algorithm to find aircraft that will be within 2×2 nm in x and y and within 1000 feet in altitude in 2 minutes. Next, transfer these selected aircraft's identity, x, y positions, altitude, velocity, heading and conflict information etc to display server. In the end, use conflict resolution algorithm to get conflict avoidance advisory and transfer it to the server.

5.3 Controller Display Update

First, transfer the updated flight identity, positions, altitude, speed, and heading, etc from PEs to host. Next, if

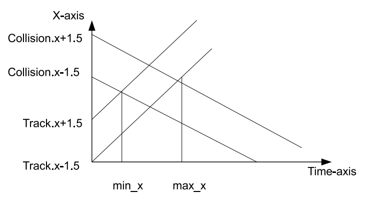

Fig. 4. Conflict detection

there are any advisories for conflict avoidance maneuver, transfer this information to the server.

5.4 Automatic Voice Advisory

Automatic Voice Advisory (AVA) automatically advises an uncontrolled flight (VFR) of near term conditions of other aircraft and terrain by voice. This task is simulated by printing advisories of conflict detection and resolution, and terrain avoidance tasks etc. For example, if there is an aircraft that is approaching the aircraft called, the message might be "aircraft at 4 miles, 4,500 feet, in 1 minute"; if the aircraft called is heading for a terrain, the message might be "terrain, 4 miles, 3,100 feet ahead".

5.5 Sporadic Requests

Sporadic requests include information requests or changes in data. For example, aircraft have to avoid an area that has bad weather, aircraft makes maneuver to avoid bad weather, or controllers make a request for runway usage, etc. This task is executed once every second. Although the requests are not processed immediately, they are processed very quickly. We simulate this task as follows. First, Use associative operation PickOne to select one aircraft. Next, change its heading to avoid bad weather, e.g., turn right one degree.

5.6 Conflict Detection and Resolution(CD&R)

This paper considers a conflict to occur when two aircraft are predicted to be within a distance of three nautical miles in x and y and within 1000 feet in altitude. We assume that all aircraft to fly at the same altitude for simplicity. To assure timely evaluation we let the detection cycle be eight seconds, and we want to determine the possibility of a future conflict between any pairs of aircraft within a five minute "look ahead" period (i.e., 300 seconds). The conflict detection algorithm is shown in Algorithm 2. The input data are the track records in PEs. We copy each *track*'s *ID*, 3D position X_t, Y_t and H_t, and velocity V_{xt} and V_{yt} to the following variables, respectively, *ID*, X_c, Y_c, H_c, V_{xc} and V_{yc}, to a poly structure *collision* in PEs. Initialize their $time\_till$ which is the aircraft's earliest collision time with other aircraft to 300.00.

The following formulas are used in Algorithm 2.

$$min\_x = \frac{|collision.X_c - track.X_t| - 3}{|collision.V_{xc} - track.V_{xt}|} \tag{1}$$

Algorithm 2 Algorithm for Conflict Detection

```
1: For each collision and track record in each PE, first check whether their flight IDs are different and altitudes are within 1000 feet.
2: Project their positions into 5 minutes, add 1.5 to each $x$ and $y$ edge of the future position to provide a 3.0 minimum miss distance in each dimension. The $x$ dimension is shown in Figure 4.
3: Calculate the $min\_x$, $max\_x$, $min\_y$ and $max\_y$ for minimum and maximum intersection times in $x$ and $y$ dimensions, which are illustrated in equations 1, 2, 3 and 4.
4: Find the largest minimum time $time\_min$ and smallest maximum time $time\_max$ across the two dimensions using equations 5 and 6.
5: If $time\_min$ is less than $time\_max$, there is a potential conflict between the aircraft whose ID is $collision.ID$ and another aircraft whose ID is $track.ID$.
6: If $time\_min$ is less than $collision.time\_till$, $collision.time\_till$ is updated to $time\_min$.
7: All collision records in each PE are passed to next PE by swazzle function and steps 1 to 6 are repeated.
8: After 96 iterations, all collisions have been compared with all tracks. The $time\_till$ of each collision is its soonest collision time with another track.
```

$$max\_x = \frac{|collision.X_c - track.X_t| + 3}{|collision.V_{xc} - track.V_{xt}|} \tag{2}$$

$$min\_y = \frac{|collision.Y_c - track.Y_t)| - 3}{|collision.V_{yc} - track.V_{yt}|} \tag{3}$$

$$max\_y = \frac{|collision.Y_c - track.Y_t)| + 3}{|collision.V_{yc} - track.V_{yt}|} \tag{4}$$

$$time\_min = max\{min\_x, min\_y\} \tag{5}$$

$$time\_max = min\{max\_x, max\_y\} \tag{6}$$

The algorithm for conflict resolution is described in the Algorithm 3. The input data are the *tracks* and *collision* records in PEs. Each PE can have 1 to 17 *tracks* and *collision* records.

5.7 Terrain Avoidance

Terrains are lines that make a box shape at a terrain height, e.g., a TV tower is a 1.0 by 1.0 nautical miles box with a height equal to 3,100 feet. All terrains and tracks are entered in each PE. The terrain avoidance algorithm is shown in Algorithm 4 that is similar to conflict detection algorithm 2.

Algorithm 3 Algorithm for Conflict Resolution

```
1: Find the minimum $time\_till$ of all collisions records sequentially in each PE.
2: Find the minimum $time\_till$ across PEs using associative operation $cs\_reduce\_min$.
3: Transfer the collision records from PEs to mono using associative operation $memcpyp2m$.
4: Search for the ID of the aircraft that has minimum $time\_till$ from the first aircraft to the end in mono sequentially because step 2 can only find the minimum $time\_till$ but cannot record its ID. This is the best or trial aircraft that will make the heading change.
5: Calculate that aircraft's velocity and angle using its velocity in $x$ and $y$ dimension in the mono memory.
6: We have $projectedpath[95]$ in mono, copy the best aircraft's $ID$ and positions to all of the records of $projectedpath[95]$, initialize all of their soonest collision time with other aircraft time_till to $1200.00$.
7: Each of the $projectedpath[i]$ represents a path where the best aircraft makes a different heading change from left to right 3 degrees to evaluate numerous different possible paths for the trial aircraft in parallel.
8: Transfer the $projectedpath[0], \cdots, projectedpath[95]$ from mono to PEs each of which has one $projectedpath[i]$ using associative operation $memcpym2p$.
9: Each projectedpath is compared to all collision records in each PE: if their flight IDs are different (so that the trial aircraft will not compare to itself) and altitudes are within $1000$ feet, calculate $min\_x$, $max\_x$, $min\_y$ and $max\_y$ for minimum and maximum intersection times in $x$ and $y$ dimension using the same equations 1, 2, 3 and 4.
10: Use equations 5 and 6 to get $time\_min$ and $time\_max$. If $time\_min$ is less than $time\_max$, there is a potential conflict between the aircraft whose ID is the collision ID and this path.
11: Check whether $time\_min$ is less than the $time\_till$ of this $projectedpath$, if so, this $projectedpath.time\_till$ is updated to $time\_min$.
12: The projectedpath records in each PE are then passed to the next PE using swazzle function to compare with collision records in the neighbor PE.
13: After $96$ iterations, all projectedpath records have been compared to all collision records, i.e., all other aircraft.
14: Find the maximum $time\_till$ using associative function $cs\_reduce\_max$ because there is only one $projectedpath.time\_till$ record in each PE.
15: Transfer projectedpath records from PEs to mono using associative function $memcpyp2m$.
16: Find which new path has the maximum $time\_till$. This path is the best scenario. Record the $ID$ of it as $minid$.
17: Transfer all track records in each PE to mono using associative function $memcpyp2m$.
18: The track whose $ID$ is $minid$ is the best aircraft, change its $x$ and $y$ velocity to the best scenario path's velocity.
19: Display the resolution advisory and change the flight plan in the host.
```

Algorithm 4 Algorithm for Terrain Avoidance

```
1: For each terrain and track in each PE, check whether the track's height is lower than the terrain's, if yes, go on to next step.
2: Project the track's position to $2$ minutes, add $1.5$ to each $x$ and $y$ edge of the future position to provide a $3.0$ minimum miss distance. The terrains are $1.0$ by $1.0$ nm boxes.
3: Calculate the minimum and maximum intersection times in both $x$ and $y$ dimensions.
4: Record $time\_min$ as the larger of the two minimum intersection times in both $x$ and $y$ dimensions in step $3$. Same, record $time\_max$ as the larger of the two maximum intersection times in both $x$ and $y$ dimensions in step $3$.
5: If $time\_min < time\_max$, there is a potential conflict between the track and the terrain.
6: All track records in each PE are passed to next PE by $swazzle$ function and steps $1$ to $5$ are repeated.
7: After $96$ iterations, all track records have been compared with all terrain records for terrain avoidance.
```

5.8 Final Approach (Runways)

The final approach task is to optimize runway usage. First, each flight has a flight plan that specifies its departure terminal and planned departure time, its destination terminal and planned arrival time. Second, we set 96 runways, one in each PE. Third, each runway collects departure and arrival time on it and sorts the time. Fourth, the flights will increase or decrease their speed to optimize runway usage and also optimize fuel cost. The last step is currently done by controllers manually.

6 Experimental Results

This section describes the results of a set of preliminary experiments that were conducted to achieve four different goals. First, the experiments provide a proof-of-concept for the proposed ATC system implementation based on the CSX600. Second, the performance and scalability of the proposed approach will be evaluated by performing a preliminary comparison between the CSX600-based implementation of the tracking and correlation task, and two alternative implementations, namely a single-threaded version and a multi-threaded version running on a state-of-the-art multiprocessor/multicore server system featuring a total of 8 system cores. Third, the experiments will provide some initial evidence for the claim that the proposed AP-based ATC system implementation exhibits a significantly larger degree of predictability than the MIMD-based solution. Fourth, we will show that our prototype can meet the deadlines for the hard real-time ATC tasks.

6.1 Experimental Setup

Since we are creating a prototype solution, our implementations cannot manage the number of aircraft in a real-world situation. In order to have information about flights that we can control, we simulate the real-world situation by generating aircraft flights in a two dimensional airspace of 1024 by 1024 nautical miles. The initial positions and velocities of the aircraft are generated randomly and trajectories of aircraft consist of a constant velocity mode and a coordinated turn mode. Some radar noise is randomly generated. Since we control this process, we can generate different numbers of aircraft to test algorithms and test the limits on the number of aircraft that can be processed fast enough to meet deadlines. Unlike live flight data, when two aircraft are on a collision course, we can alter the flight path of one aircraft to eliminate this problem.

The AP-based implementation is based on the ClearSpeed accelerator board and implemented in the C^n language as described in the previous section 2. Note that only one of the two CSX600 chips, i.e., one MTAP with a total of 96 PEs, has been used for the experiments. The reason for this is that with the CSX600, the two chips cooperate as two separate SIMDs. The latest SDK CSX600 software makes it easier to have the two chips operate as a single SIMD, but is still not as good a simulation of AP as obtained by using only one single chip.

The alternative implementations were all executed on an Intel Dual Processor Xeon E5410 Quad Core 2.33 GHz system with 32 GB of main memory and 2x6Mb of L2 cache (for each CPU). The systems feature a total of 8 cores. The operating system environment consists of Ubuntu 64-bit Linux kernel release 2.6.22-14-generic. The implementation was done in C using the gcc compiler version 4.1.3. We used the compiler optimization flags '-mfpmath=387,sse -msse3 -ffast-math -O3' to enable Intel's Streaming SIMD Extensions (SSE) for both the single-threaded as well as the multi-threaded version of the code. The single-threaded version is executed on a single core. The multi-threaded version is based on POSIX PThreads. It was carefully designed to minimize typical performance limiting effects such as *false sharing*, *cache-ping-pong*, and *high lock contention*. We specifically designed the multi-threaded code in such a way that locking was avoided whenever possible.

6.2 Experiment Results

6.2.1 Performance on CSX600

We measure the timings in CSX600 SIMD environment in this section. The experiment results are illustrated in Figures 5 and 6: the horizontal axis represents the number of tracks and the vertical axis represents the execution time of the tasks in seconds. The units of the rest of the graphs in this paper are the same. The experiment results show that the running time on CSX600 is almost constant when the maximum number of aircraft for each PE is 1, 2, 3, and 4, i.e., the numbers of aircraft as reported in the figures should actually be $1-96$, $97-192$, $193-288$ and $289-384$. Moreover, the gap between them slightly increases. The results show that CSX600 emulates the

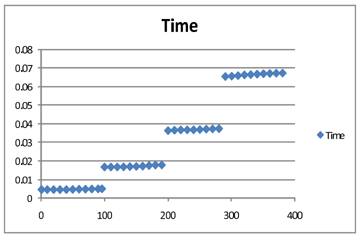

Fig. 5. Time of correlation for number of aircraft between 10 and 380.

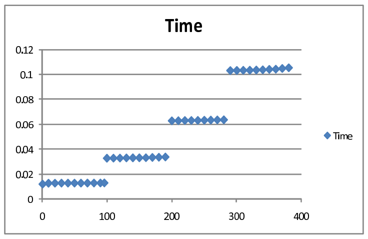

Fig. 6. Time of CD&R for number of aircraft between 10 and 380.

properties of the AP when the number of aircraft in each PE is 1. In the USN ASPRO the time for $2,000$ tracks using 0.45 microsec memory was 0.113 seconds [27].

Our current implementation allows a maximum of 17 tracks per PE, i.e., 1632 tracks in total. The results are shown in Figure 7 and Figure 8. We assess the scalability of the tracking and CD&R algorithms by evaluating their running time over a wide range of number of tracks and plot the time each algorithm takes on a graph. We can see that the CSX600 approach can finish all the three tasks within deadlines even when the number of tracks is scaled up to 1500.

The sequential complexity of both algorithms illustrated in Figures 7 and 8 are $O(n^2)$, because each report box is evaluated with each track box. The CSX600 can execute 96 instances of a task in essentially the same time as it requires to execute 1 instance of this task. However, 96 is not a constant with respect to $n = 1632$, as $\log 1632$ is about 10.7 and 1632 is roughly 96 raised to the 1.6 power. Since the sequential complexity of both algorithms is quadratic, the runtime of the CSX600 on both of these algorithms should be faster than quadratic but slower than linear.

Next, we show the speedup and efficiency of the system. The sequential program is simulated by programs that run on only one PE of CSX600, and the running time is recorded as t_s. The parallel running time is recorded as t_p, p is the number of processors. Here $p = 96$. One PE can only handle 100 tracks because it only has $6k$ memory. The comparison of t_s and t_p for report correlation and tracking task is shown in Figure 9. The speedup is

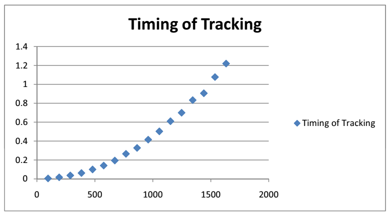

Fig. 7. Timing of Tracking Algorithm

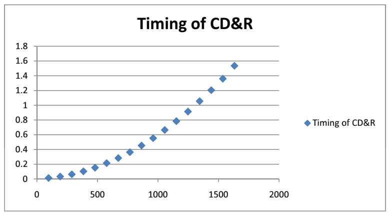

Fig. 8. Timing for CD&R Algorithm

$t_s/t_p = 95.48$, and efficiency is $t_s/(p \times t_p) = 0.95$, which is almost optimal. Because CSX600 emulate AP properties, the speedup and efficiency for AP should be optimal too.

6.2.2 Comparison of CSX600 and MIMD

We execute the report correlation and tracking task under all three approaches. In the following we will also refer to the CSX600-based ATC implementation as $SIMD$, to the single-threaded implementation as STI, and to the multi-threaded implementation as MTI. Each approach was executed for a varying number of planes, ranging from 4000 to 14000 in increments of 1000. For each approach, and for each number of planes in the given range, each approach was executed for 50 iterations. One iteration corresponds to one radar refresh cycle. Hence, a total of 50 sets of incoming radar reports had to be completed. We measured the execution time for each iteration. Note that since none of these approaches ever idles to wait for new incoming radar reports, the number of iterations divided by the total execution time represents the highest-possible radar update rate that can be sustained by each approach.

Figure 10 shows the maximum value of the execution times for each of the experiments and for each of the three approaches. From these results we can see that STI always takes the most time and increases the quickest. MTI takes a little less time than $SIMD$ from 4000 to 7000 aircraft, their time is similar between 7000 and 8000 aircraft, and from 8000 aircraft and on, MTI takes more time and increases quicker than $SIMD$. In addition, the performance results on the CSX600 portrays a linear time per aircraft shown in Figure 10 because 1632 tracks are processed each time, so it takes nine iterations to process 14688 tracks as our worst case, and the data transfer takes very short time while computation takes most of the time.

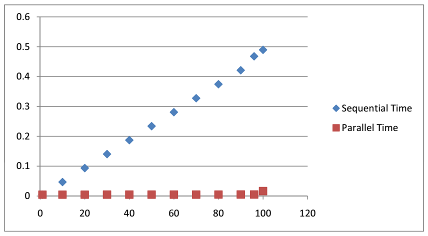

Fig. 9. Speedup and Efficiency of Tracking.

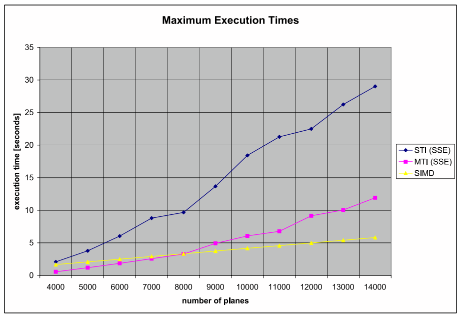

Fig. 10. Performance of STI, MTI, and $SIMD$

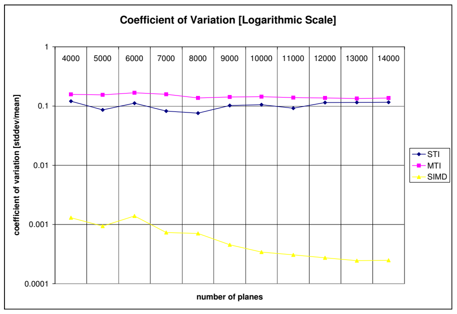

Fig. 11. Predictability of Execution Times for STI, MTI, and $SIMD$

While the results in Figure 10 demonstrate that the performance of CSX600 is better than that of STI and MTI, Figure 11 focuses on the predictability of the execution times, which is an important factor in guaranteeing that all the ATC processing can be performed within the specified time bounds. The result in Figure 11 measure the *Coefficient of Variation* (COV), which is a common normalized measure of dispersion, and is defined as the ratio of the standard deviation to the mean. Unlike the standard deviation, the COV is dimensionless. Note that the y-axis in Figure 11 uses a logarithmic scale. The results clearly show that the COV values for $SIMD$ are several orders of magnitude below the ones for STI and MTI. We attribute the fluctuations in the execution times of both STI and MTI to both operating system and hardware interference.

6.2.3 Timings for 8 Tasks

In this section we will show that our prototype can meet the deadlines for the hard real-time ATC tasks. Table 2 shows the performance of one flight per PE, i.e., 96 flights. The execution time (secs) is the time that is spent for this task once. The processing time (secs) is the total time that is spent for this task in an 8 second period. We can see that all tasks can be done within their deadlines. The total used time is 0.25508 seconds, which is only 3.19% of available time 8 second.

The maximum number of flights per PE is 7 to guarantee real-time requirements of the tasks on our CSX600 prototype. The performance is shown in Table 3. In the 14th 0.5 second cycle, the 3 tasks, aircraft tracking, sporadic requests and CD&R take 0.498 second. All the other cycles have more unused time. The used time is 6.67717 seconds, which is 83.46% of available time 8 second. We can put more tasks into some slots where there is unused time. Furthermore, the timings are very predictable, i.e., the execution times of the tasks are always stable, and the conflict detection and resolution task always misses its deadline, while other tasks never miss their deadlines. According to the experiment results of Section 6.2.2, the execution times of the tasks running on MIMD are not stable. Therefore, the predictability of ATC system based on CSX600 or AP is better than the one based on MIMD.

7 Conclusions and Future Work

In this paper, we propose an efficient and scalable solution for Air Traffic Control problems using the associative processor system that is emulated on the CSX600 system. The contributions of this paper are summarized as follows. First, the AP system proposed can not only execute the 8 ATC tasks accurately, but also has demonstrated performing the 8 tasks with $7,500$ aircraft within deadlines. Because the emulation tool CSX600 can only process maximum 672 aircraft within deadlines in Section 6.2.3, an ideal AP system would have to be a lot larger than the CSX600. As discussed in Section 1, the ideal AP should have at least 14000 processors in order to provide reasonable flexibility in the number of distinct ATC tasks executed and to meet all worst case execution deadlines for each of these tasks. Since all PEs would be assigned at most one aircraft, this system should provide roughly a linear speedup over sequential performance. Moreover, the worst case running time for the proposed single thread instruction stream *AP* system could be precisely predicted. The number of instructions for the AP demonstrated in 1972 was only $3,493$ [27]. In contrast, the MIMD-based systems are evaluated only on average running time and have highly unpredictable worst case running times. These two contributions can provide major help in meeting the goals of FAA's NextGen Plan: fly more aircraft, more safely, more precisely, more efficiently and use less fuel [33]. Third, the software used by AP will be substantially simpler and smaller in size than the current corresponding ATC software. Fourth, the V&V (Validation and Verification) process is simpler than that for current ATC software. Fifth, the hardware architecture of AP is simpler than the current ATC MIMD hardware. Moreover, considering the AP's advantages of simple programming style and simple hardware implementations, it obviously deserves more attention and utilization if we want to obtain satisfactory solutions to numerous critical real-time problems with hard deadlines including ATC.

We will implement all of the 8 ATC real-time tasks on the same MIMD system that had been used in Section 6 in the future and compare the performance of CSX600 and MIMD prototypes. The purpose of the comparison is to show that the MIMD-based systems are evaluated only on average running time and have highly unpredictable worst case running times so that the AP solution is superior. A possibly important extension to our current research would be to consider a GPU implementation using CUDA. Nvidia has many SIMD PE groups on its chips. The Nvidia technology including the latest FERMI chip has a lot in common with the MTAP approach of ClearSpeed, and implementing the CSX600 ATC algorithms on this architecture may provide another useful platform to use in this project.

References

[1] Y. Bar-Shalom and T.E. Fortmann, *Tracking and Data Association*, Academic Press, 1988.

[2] Y. Bar-Shalom and X.R. Li, *Estimation and Tracking: Principles, Techniques and Software*, Artech House, Boston, 1993.

[3] H. Blom and Y. Bar-Sharlom, "The Interacting Multiple Model Algorithm for Systems with Markovian Switching Coefficients", *IEEE Transactions on Automatic Control*, vol. 33, no. 8, pp.780-783, August 1988.

[4] H.A.P. Blom, R.A. Hogendoorn, and B.A. van Doorn, "Design of a multisensor tracking system for advanced air traffic control". In Y. Bar-Shalom, editor, *Multitarget-Multisensor Tracking: Application and Advances*, volume 2, pages 31-63, Artech House, 1990.

[5] Y-J. Chiang, J. T. Klosowski, C. Lee and J. S. B. Mitchell, "Geometric Algorithms for Conflict Detection/Resolution in Air Traffic Management", *36th IEEE Conference on Decision and Control*, pp. 1835-1840, San Diego, CA, December 1997.

[6] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*. W.H. Freeman, 65-66, 238-240, New York, 1979.

[7] J.L. Gustafson and B.S. Greer, "ClearSpeed Whitepaper: Accelerating the Intel Math Kernel Library," *http://www.clearspeed.com/docs/resources/ClearSpeed Intel Whitepaper Feb07.pdf*, 2007.

[8] I. Hwang and C. Tomlin, "Protocol-Based Conflict Resolution for Finite Information Horizon," *Proceedings of the AACC American Control Conference*, Anchorage, May 2002.

[9] I. Hwang, J. Hwang, and C. Tomlin, "Flight-Mode-Based Aircraft Conflict Detection Using a Residual-Mean Interacting Multiple Model Algorithm," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Austin Texas, August 2003.

[10] I. Hwang, H. Balakrishnan, K. Roy, and C. Tomlin, "Multiple-Target Tracking and Identity Management in Clutter for Air Traffic Control," *Proceedings of the AACC American Control Conference*, Boston, MA, June 2004.

[11] M. Jin, J. Baker and K. Batcher, "Timings for Associative Operations on the MASC Model," *Proc. of the 15th International Parallel and Distributed Processing Symposium (IEEE Workshop on Massively Parallel Processing)*, San Francisco, CA, 2001.

[12] S. Kahne and I. Frolow, "Air Traffic Management: Evolution with Technology," *IEEE Control Systems Magazine*, vol. 16, no. 4, pp. 12-21, 1996.

[13] J. Krozel, M. Peters, K.D. Bilimoria, C. Lee, and J.S.B. Mitchell, "System Performance Characteristics of Centralized and Decentralized Air Traffic Separation Strategies," *Fourth USA/Europe Air Traffic Management Research and Development Seminar*, 2001.

[14] J.K. Kuchar and L.C. Yang, "A Review of Conflict Detection and Resolution Modeling Methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179-189, 2000.

TABLE 2
Performance of One Flight/PE

Tasks	Exec Time	Proc Time
Report Correlation & Tracking	0.00552	0.08832
Cockpit Display	0.00272	0.02177
Controller Display Update	0.00276	0.02209
Sporadic Requests	0.00155	0.01244
Automatic Voice Advisory	0.00544	0.01088
Terrain Avoidance	0.00782	0.0782
Conflict Detection & Resolution	0.01301	0.01301
Final Approach(96 runways)	0.00837	0.00837
Total		0.25508

TABLE 3
Performance of Seven Flights/PE

Tasks	Exec Time	Proc Time
Report Correlation & Tracking	0.20828	3.33261
Cockpit Display	0.10414	0.83315
Controller Display Update	0.11479	0.91832
Sporadic Requests	0.06887	0.55099
Automatic Voice Advisory	0.16663	0.33326
Terrain Avoidance	0.25761	0.25761
Conflict Detection & Resolution	0.29011	0.29011
Final Approach(96 runways)	0.16112	0.16112
Total		6.67717

[15] D.G. Lainiotis, ”Partitioning: A Unifying Framework for Adaptive Systems I: Estimation,” *Proceedings of the IEEE*, vol. 64, pp. 1126-1142, August 1976.

[16] X.R. Li and Y. Bar-Shalom, ”Design of an Interacting Multiple Model Algorithm for Air Traffic Control Tracking,” *IEEE Transactions on Control Systems Technology*, vol. 1, no. 3, pp. 186-194, September 1993.

[17] K.M. Liu, ”Composition of Kalman and Heuristic Tracking Algorithms for Air Traffic Control,” Master Thesis, Kent State University, August 1999.

[18] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan, ”Interacting Multiple Model Methods in Tracking: A survey,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 1, pp. 103-123, 1998.

[19] W. Meilander, M. Jin, and J. Baker, ”Tractable Real-Time Air Traffic Control Automation,” *Proc. of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 483-488, Cambridge, MA, 2002.

[20] W. Meilander, J. Baker, M. Jin, ”Predictable Real-Time Scheduling for Air Traffic Control,” *Fifteenth International Conference on Systems Engineering*, pp. 533-539, August 2002.

[21] W. Meilander, J. Baker, and M. Jin, ”Importance of SIMD Computation Reconsidered,” *Proc. of the 17th International Parallel and Distributed Processing Symposium (IEEE Workshop on Massively Parallel Processing)*, Nice, France, April 2003.

[22] M.S. Nolan, *Fundamentals of Air Traffic Control*, Brooks/Cole, Wadsworth, 3rd Edition, 1998.

[23] R.A. Paielli and H. Erzberger, ”Conflict Probability Estimation for Free Flight,” *Journal of Guidance, Control, andDynamics*, vol. 20, no. 3, pp. 588-596, 1997.

[24] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, and C. Asthagiri, ”ASC: An Associative-Computing Paradigm,” *Computer*, vol. 27, no. 11, pp. 19-25, 1994.

[25] M. Prandini, J. Hu, J. Lygeros and S. Sastry, ”A Probabilistic Approach to Aircraft Conflict Detection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp.199-219, December 2000.

[26] S. Reddaway, W. Meilander, J. Baker, and J. Kidman, ”Overview of Air Traffic Control Using an SIMD COTS System,” *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, CO, April 2005.

[27] J. A. Rudolph, ”A Production Implementation of an Associative Array Processor - STARAN”, *The Fall Joint Computer Conference (FJCC)*, Los Angeles, CA, December 1972.

[28] J. Stankovic, M. Spuri, M. Natale and G. Buttazzo, ”Implications of Classical Scheduling Results for Real-Time Systems,” *IEEE Computer*, pp. 16-25, June 1995.

[29] D.D. Sworder and J.E. Boyd, *Estimation Problems in Hybrid Systems*, Cambridge University Press, 1999.

[30] L.C. Yang and J.K. Kuchar, ”Prototype Conflict Alerting System for Free Flight,” *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 4, July-August 1997.

[31] M.Yuan, J.W.Baker, F.Drews and W.Meilander, ”Efficient Implementation of Air Traffic Control Using the ClearSpeed CSX620 System,” *Proc. of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pp. 353-360, Cambridge, MA, November, 2009.

[32] M.Yuan, J.W.Baker, F.Drews, L.Neiman and W.Meilander, ”An Efficient Associative Processor Solution to an Air Traffic Control Problem,” *Large Scale Parallel Processing IEEE Workshop at the International Parallel and Distributed Processing Symposium (IPDPS2010)*, 8 pages, Atlanta, GA, April, 2010.

[33] FAA's NextGen Implementation Plan, http://www.faa.gov/about/initiatives/nextgen/media/ngip.pdf, 2009.

[34] FAA Grants for Aviation Research Program Solicitation, http://www.tc.faa.gov/logistics/grants/, 2009.

[35] ClearSpeed Technology plc. ClearSpeed whitepaper: CSX processor architecture, http://www.clearspeed.com/docs/resources/, 2007.

[36] ClearSpeed Technology plc. ClearSpeed whitepaper: ClearSpeed Software Description, https://support.clearspeed.com/documents/, 2007.

[37] ClearSpeed Technology plc. CSX600 Runtime Software User Guide, https://support.clearspeed.com/documents/, 2007.

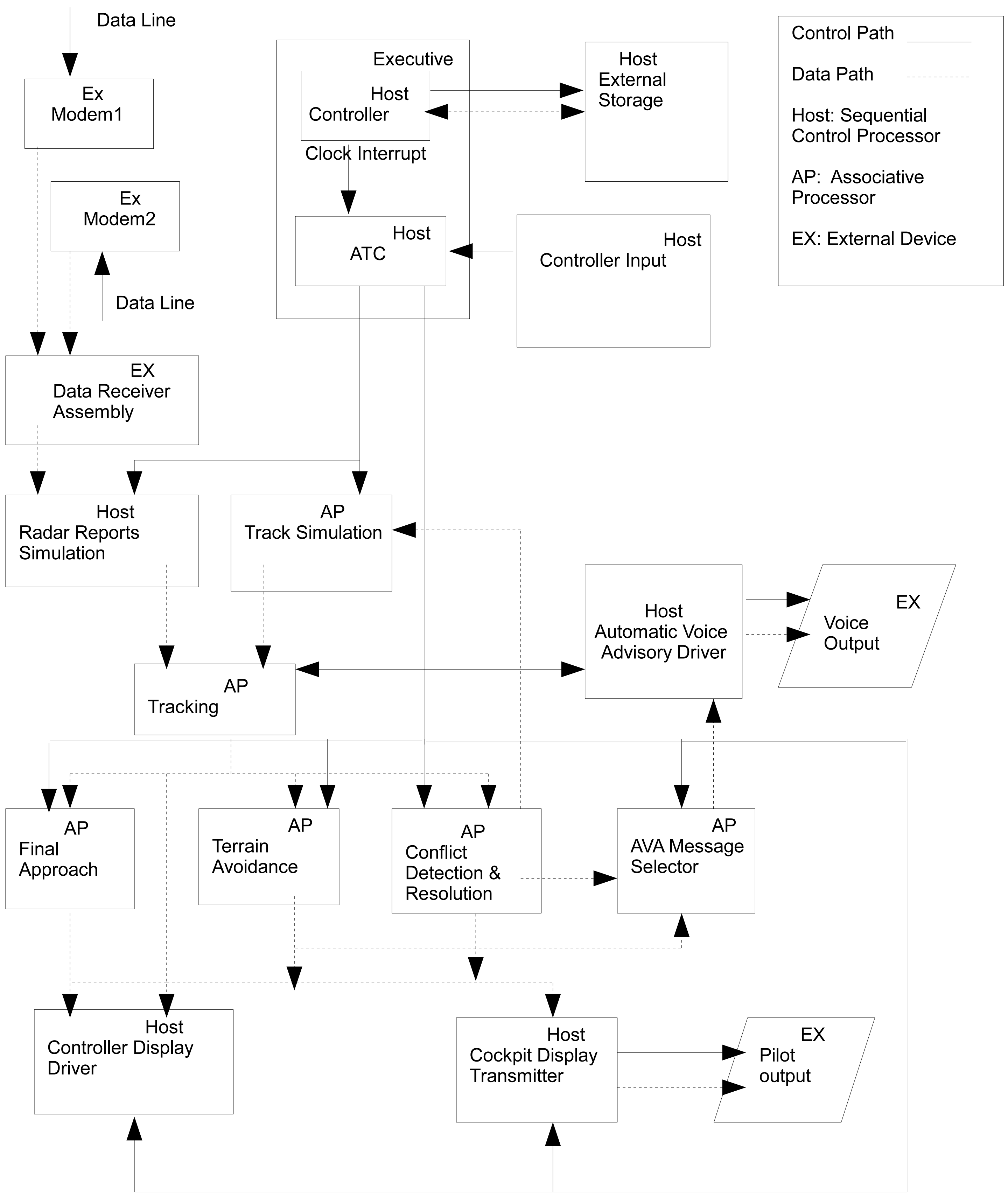

Fig. 3. Overall ATC System Design.