Scalable and Efficient Associative Processor Solution to Guarantee Real-Time Requirements for Air Traffic Control Systems

Mike Yuan
Department of Computer Science
Kent State University
Kent, OH
Email: myuan@cs.kent.edu

Johnnie W. Baker
Department of Computer Science
Kent State University
Kent, OH
Email: jbaker@cs.kent.edu

Will Meilander(retired)
Department of Computer Science
Kent State University
Kent, OH
Email: wllcm@att.net

K. Schaffer
Department of Computer Science
Kent State University
Kent, OH
Email: kschaffe@cs.kent.edu

***Abstract*—This paper proposes a solution to air traffic control (ATC) using an enhanced SIMD machine model called an Associative Processor (AP). Our solution 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 supports accurate predictions of worst case execution times and guarantees all deadlines are met. Furthermore, the software developed based on the AP model is much simpler and smaller in size than the current corresponding ATC software. As the associative processor is built from SIMD hardware, it is considerably cheaper and simpler than the MIMD hardware currently used to support ATC. We have designed a prototype for eight ATC real-time tasks on ClearSpeed CSX600 accelerator that is used to emulate AP. Performance is evaluated in terms of execution time and predictability and is compared to the fastest host-only version implemented using OpenMP on an 8-core multiprocessor (MIMD). Our extensive experiments show that the AP implementation meets all deadlines that can be statically scheduled. To the contrary, some tasks miss their deadlines when implemented on MIMD. It is shown that the proposed AP solution will support accurate and meaningful predictions of worst case execution times and will guarantee that all deadlines are met.**

***Keywords*-Air Traffic Control (ATC); SIMD; MIMD; Associative Processor (AP); Processing Elements(PE); Conflict Detection and Resolution (CD&R); ClearSpeed CSX600 System; OpenMP; Federal Aviation Administration (FAA);**

I. Introduction

The Air Traffic Control (ATC) system is a real-time system that continuously monitors thousands of flights by processing large volumes of data that are dynamically changing due to reports by sensors 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. In the past, solutions to this problem have been implemented on MIMD systems. It is difficult for these MIMD implementations to satisfy reasonable predictability standards that are critical for meeting the strict certification standards needed to ensure safety critical software components.

Instead, we implement the ATC system on an enhanced SIMD hardware system called an Associative Processor (AP), where interactions are much simpler and more efficiently controlled by avoiding the need to coordinate the interactions of multiple instruction streams. The AP uses a single thread instruction stream (only one instruction can exist at any time). AP has well-known advantages over MIMD: low overhead synchronization and data exchange over cores. Previous papers [1], [2], [3], [4] have used AP to manage ATC computation. Similar SIMD parallel approaches have been used for collision avoidance algorithms between multiple agents for real-time simulations in S.Guy et. al. [5]. However, the assumed maximum number of aircraft being tracked is 4000 IFR (instrument flight rules) aircraft and 10000 VFR (visual flight rules) aircraft, for a total of 14000 aircraft [6]. Because the emulation tool CSX600 can only process a small number of aircraft, an ideal AP system would be a lot larger than the CSX600.

In [6], the report correlation and tracking task and the conflict detection and resolution (CD&R) task have been implemented on both the CSX600 board and a multicore server with 8 cores. However, only one task was executed repeatedly, without any competition from other tasks. Also, no adjustment was included for the greater combined computational speed of MIMD. In this paper, we not only implement 8 tasks in CSX600, which are: 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), but also schedule three tasks, report correlation and tracking, terrain avoidance and CD&R on both CSX600 and an 8-core MIMD with the fastest host-only version implemented using OpenMP [7]. Our results show that the MIMD implementation often misses deadlines, while the AP implementation meets all deadlines that can be statically scheduled. The results indicate that the AP implementation has much better scalability, efficiency and predictability than the MIMD implementation. Moreover, the proposed AP solution will support accurate and meaningful predictions of worst case execution times and will guarantee all deadlines are met. While SIMDs (and APs) and MIMDs each have their particular advantages and disadvantages, this paper establishes several important reasons that APs are much better suited than MIMDs for such real-time applications with hard deadlines as ATC.

The remainder of this paper is organized as follows: Section II overviews CSX600 architecture and programming concepts. We discuss emulating the AP on the CSX600 in Section III. Section IV describes the algorithms for 8 key ATC real-time tasks implemented on CSX600, which can be mapped to AP. Experimental results are presented in Section V. In Section VI, the advantages of AP over MIMD are illustrated. Finally, Section VII concludes this paper.

II. Overview of ClearSpeed CSX600

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 [8].

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

Figure 1. CSX600 accelerator board

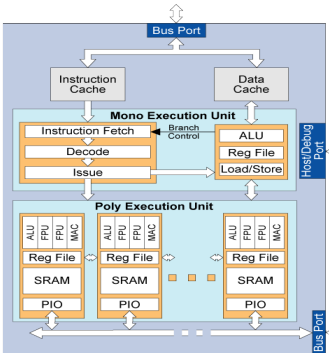

Figure 2. MTAP architecture of CSX600

PE and are processed by the PEs. Further information on the associated software can be found in the documentation [9], [10]. 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 [10], [11].

III. Emulating the AP on the ClearSpeed CSX600

An Associative Processor (AP) [1], [12] is a SIMD with several additional useful hardware enhancements that simplify supporting the real-time ATC system. A more complete and careful listing of the associative properties follows [13], [14]:

- 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 value of a poly variable across all active PEs whose attribute values match the search pattern. The active PEs whose attribute value in the poly variable matches the search pattern are called *responders*, and the active

Table I
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

PEs whose attribute value in the poly variable 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 efficiently, in both C^n language introduced in Section II and assembly language supported by CSX600. To evaluate their running time, we store 30 records in each PE and perform each associative operation once for each of these 30 records. The timing results in microsecond (μsec or 10^{-6} second) are shown in Table I. Although they are not constant time, they are very efficient, and additionally demonstrate that we can efficiently emulate an AP using CSX600.

IV. AN AP SOLUTION FOR ATC TASKS EMULATED ON CSX600

This section describes the algorithms of eight ATC 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 it is easy to scale up from 96 PEs to an AP with $14,000$ PEs by using the same algorithms.

A. Report Correlation and Tracking

The report correlation and tracking task is executed every 0.5 second. The input data are radar reports that are simulated in host. Track records are developed from reports 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. It is challenging because each report has to be compared with each track, and some aircraft are changing flight mode. We use the SIMD architecture to do the task. First, we create a box based on report and track errors for each report and each track and check whether they intersect. It is very efficient because of the features of AP. Second, increase the box sizes of tracks that have not correlated with any reports and then check again because some aircraft might accelerate or maneuver at that time. The details of this task are shown in Algorithm 1. This task is done both accurately and within deadline because of the SIMD features of AP.

Algorithm 1 Algorithm for Aircraft Tracking in CSX600

1: All radar reports are developed in host and then transferred from host to mono memory. Next, they are transferred from mono to PE memories, with each PE receiving an equal share of the reports.
2: Boxes of sides of length 1 nautical mile (nm) and altitude 1000 feet are created around each radar report and each track in each PE to accommodate report and track uncertainties.
3: Check the intersection of each report box with every track box in each PE. If there is an intersection, the radar report and the track are correlated.
4: The radar reports in each PE are transferred to the next PE using the $swazzle$ (i.e., ring) network. After 96 iterations, all reports have been compared with all tracks.
5: Double the box sizes of tracks that have not correlated with any reports to increase their probability to intersect a report box and repeat the steps 3 and 4 above for unmatched reports.
6: Triple the original box sizes of tracks that have not correlated and repeat the steps 3 and 4 above for unmatched reports.
7: If two tracks correlate to the same radar report, this report is discarded because of ambiguity.

B. Cockpit Display

First, the broadcast operation is used to broadcast the x, y and altitude coordinates of an aircraft. For each of its aircraft records, each processor computes the x-distance, y-distance and altitude distance between the location of its aircraft and the location of the aircraft broadcast. This step uses the SIMD architecture to parallelize the computation to improve its efficiency. Then select the aircraft that are in the proximity of this aircraft and transfer these selected aircraft's information to the server of CSX600, which plays the role of the cockpit display.

C. Controller Display Update

This task is similar to cockpit display. We transfer the updated flight identity, positions, altitude, speed, and heading, etc from PEs to the ClearSpeed server, which plays the role of the controller display in this simulation. It uses the SIMD architecture to speed up the display process.

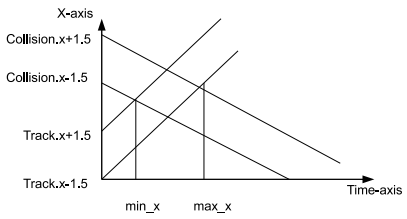

Figure 3. Conflict detection

D. Automatic Voice Advisory

Automatic Voice Advisory (AVA) automatically advises an uncontrolled flight (VFR) of proximity 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 advised, 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".

E. 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.

F. Conflict Detection and Resolution(CD&R)

1) Conflict Detection: This paper considers a conflict to occur when two aircraft are predicted to be within a distance of 3 nautical miles in x and y and within 1000 feet in altitude. We want to determine the possibility of a future conflict between any pairs of aircraft within a twenty minute "look ahead" period (i.e., 1200 seconds). The conflict detection algorithm is shown in Algorithm 2.

Formulas 1 to 6 are used in Algorithm 2.

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

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

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

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

Algorithm 2 Algorithm for Conflict Detection

1: Collision records are designed in each PE for copies of tracks records and conflict detection. Copy all track records to collision records, and for each collision and track record in each PE, check whether their flight *IDs* are different and their altitudes are within 1000 feet.
2: Project their positions 20 minutes into the future and add 1.5 nm to each x and y edge of the future position to provide a 3.0 nm minimal miss distance, as shown in Figure 3.
3: Calculate the min_x, max_x, min_y and max_y to obtain the minimum and maximum intersection times in x and y dimensions, as 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*.

$$time\_min = max\{min\_x, min\_y\} \quad (5)$$

$$time\_max = min\{max\_x, max\_y\} \quad (6)$$

2) Conflict Resolution: We use the SIMD architecture of CSX600 to do conflict resolution accurately and efficiently. First, we find which aircraft have shortest conflict time. This is the best or trial aircraft that will make the heading change. Second, each PE will have a trajectory where the trial aircraft makes a different heading change from left to right 3 degrees to evaluate numerous different possible paths for the trial aircraft in parallel. Third, use the conflict detection algorithm to find the furthest time when the trajectory collides with another aircraft. Next, find the maximum of the collision times of all trajectories. The corresponding trajectory is the best heading change that the trial aircraft will make.

G. Terrain Avoidance

The shapes that will be used to implement terrain avoidance will consist of a box or a sequence of boxes that contain the terrain feature, e.g., a TV tower is a 1.0 by 1.0 nm 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 3 and is similar to conflict detection algorithm 2.

Algorithm 3 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 into the future and add 1.5 nm to each x and y edge of the future positions in order to provide a 3.0 nm minimal miss distance. The terrains considered here are 1.0 by 1.0 nm boxes that contain towers.
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. Likewise, record $time\_max$ as the smaller 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.

H. Final Approach (Runway Optimization)

The final approach task is to optimize runway usage. Each flight has a flight plan that specifies its departure terminal and planned departure time, its destination terminal and planned arrival time. The runways that occur in the region being managed by an ATC system could be distributed among the processors. Then each processor would manage the information for the runways assigned to it. Here, we assume that there are 96 runways in the sector being managed by this ATC system and assign one runway to each processor. 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.

V. Experimental Results

This section describes the results of a set of preliminary experiment results 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 comparison between the CSX600-based implementation and the fastest host only version using OpenMP on a state-of-the-art multiprocessor server system featuring a total of 8 system core. Similar experiments using OpenMP have been used in image processing algorithms on GPUs [15]. Third, the experiments will provide some initial evidence for the claim that the proposed AP-based ATC system implementation exhibits greater efficiency and a huge increase in the 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.

A. Experimental Setup

We are creating a prototype solution since our implementation 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 three dimensional airspace of $1,024$ by $1,024$ nautical miles, $1,000$ to $10,000$ feet in altitude. The initial positions and realistic 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 simulation of the AP solution of ATC uses the ClearSpeed CSX600 accelerator board and is implemented using the C^n language as described in the previous Section II. The alternative implementation was executed on an Intel Dual Processor Xeon E5410 Quad Core 2.33 GHz system with 32 GB of main memory and 2×6Mb of L2 cache (for each CPU). This system has a total of 8 cores. The implementation was done in C using the gcc compiler version 4.1.3. A single threaded implementation on a single core of the multicore is called *STI*. We denote the multi-threaded implementation on the multicore by *MTI*. We have carefully tuned the OpenMP codes to achieve a good performance, e.g., avoid false sharing, maximize parallel regions, ensure the efficient use of cache and avoid poor load balance problems, etc.

It should be noted that we use only one CSX accelerator board, not multiple processors or boards. Our target architecture for the realistic size ATC is a modern AP that has $14,000$ PEs, not a larger CSX600 board or multiple CSX600 boards. Some of the "disadvantages" in MIMD systems may show up in a system with multiple CSX600 boards.

B. Experiment Results

1) Comparison of CSX600 and MIMD: We scheduled the tasks of report correlation and tracking, terrain avoidance and CD&R on both machines. The report correlation and tracking is executed every 0.5 second, the terrain avoidance every 1 second, and the CD&R task every 2 seconds. The deadline for each task occurs at the end of each of its period.

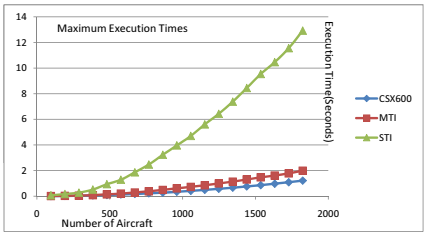

Figure 4. Timing of correlation task.

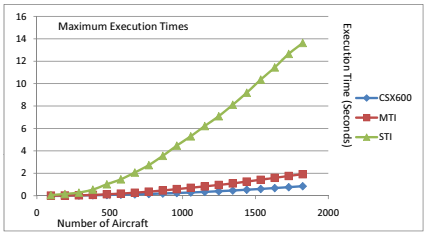

Figure 5. Timing of terrain avoidance task.

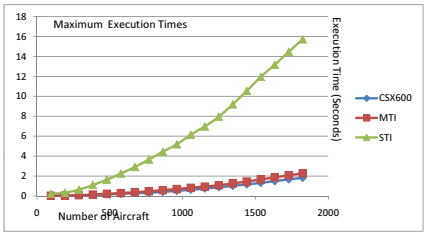

Figure 6. Timing of CD&R task.

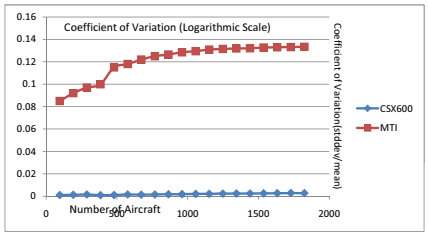

Figure 7. Predictability of execution times for report correlation and tracking task.

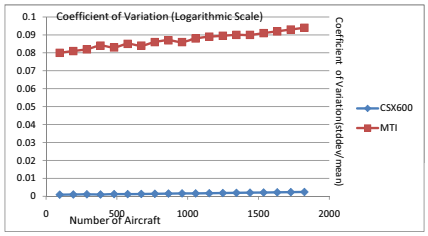

Figure 8. Predictability of execution times for terrain avoidance task.

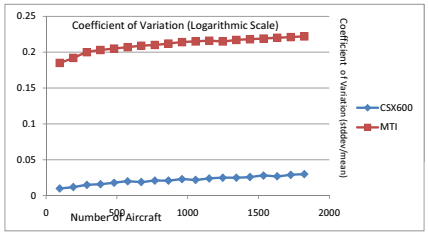

Figure 9. Predictability of execution times for conflict detection and resolution task.

None of the tasks can start their executions before their release times and all of them must complete their executions before their deadlines. Each approach was executed for a varying number of aircraft, ranging from 96 to 1824 in increments of 96. For each approach, and for each number of aircraft in the given range, each approach was executed for 100 iterations. The major period is 2 seconds and contains one CD&R period, two terrain avoidance periods, and four report correlation and tracking periods. If at least one of the three tasks miss its deadlines during a major period, we say that the execution has missed its deadline during this period.

We execute the three tasks under all three approaches, *CSX600*, *MTI* and *STI*. The comparison of maximum execution times are shown in Figures 4, 5 and 6 (the average execution times are not shown because of page limit and they do not make much difference). From these results we can see that *MTI* takes more time than the CSX600 and the additional time required for the *MTI* implementation increases more rapidly than the CSX600 as the number of aircraft increases.

While the results above demonstrate that the performance of CSX600 is better than that of *STI* and *MTI*, we next focus 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. We 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. Because unlike the standard deviation, the COV is dimensionless, so we think that it is better to compare predictability than standard deviation. The COV of the three tasks are shown in Figures 7, 8 and 9. Note that the y-axis in these figures uses a logarithmic scale. The results clearly show that the COV values for CSX600 are several orders of magnitude below the ones for *MTI*.

Last, we compare the number of major periods that have missed their deadlines for both CSX600 and *MTI*. The scheduling was described in the beginning of this section. The results are shown in Figure 10. When the number of aircraft increases, the number of deadlines missed by the *MTI* execution increases dramatically. However, the CSX600 does not miss any deadline during this period.

2) 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 II shows the performance of one flight per

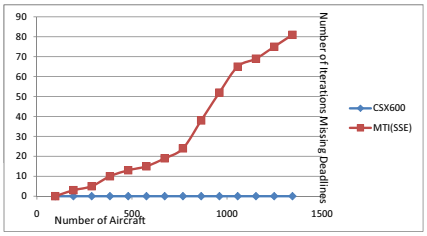

Figure 10. Number of iterations missing deadlines when scheduling three tasks.

Table II
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

PE, which is the closest scenario to the AP. The execution time (secs) is the time that is spent to execute 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.

Table III lists the worst case timings of eight tasks for 10 aircraft per PE running on the CSX600. The performance results are relatively good and demonstrate the scalability of this implementation on the CSX600. We can conclude that the AP implementation can meet all deadlines that can be statically scheduled.

VI. ADVANTAGES OF AP OVER MIMD FOR ATC

Since there is only one instruction stream (IS) or control unit, control-type communication between processors is completely eliminated. Communication between the IS and processors occurs in constant time using a broadcast/reduction network. Data communication between processors is completely deterministic and has a tight upper bound for the worst case. As a result, a numerical worst case time required for communications can be accurately calculated. In the mean time, multiprocessors are described in [16] as parallel computers that communicate using either message passing or shared memory thus usually average case is considered for MIMD. Also some additional problems encountered by multiprocessors are also avoided by APs. The problems that are avoided include the following: dynamic scheduling, load balancing, sorting and indexing, multi-tasking and multi-thread software, shared resource management, coherency management (memory and cache), preemption management, synchronization, and priority inversion handling etc. The details have been explained in the paper [3].

Table III
PERFORMANCE OF EIGHT TASKS FOR TEN FLIGHTS/PE

Tasks	Exec Time	Proc Time
Report Correlation & Tracking	0.3409	5.4544
Cockpit Display	0.1705	1.364
Controller Display Update	0.1825	1.46
Sporadic Requests	0.0987	0.7896
Automatic Voice Advisory	0.1586	0.3172
Terrain Avoidance	0.2386	0.2386
Conflict Detection & Resolution	0.5182	0.5182
Final Approach(96 runways)	0.2598	0.2598
Total		10.4018

Additionally, APs have well-known advantages over multiprocessor: low overhead synchronization and data exchange over PEs. In addition, the AP has much wider memory to processor bandwidth, much shorter programs typically of length at most that of a sequential program for the same task, predictable worst case execution times, etc. Additionally, we encounter the following problems that affect the performance of our multiprocessors which do not occur with APs: dynamic scheduling, load balancing, overlap computation and communication, false sharing, and cache ping pong effect etc. The paper [17] has identified similar difficulties in scheduling periodic real-time tasks on MIMD. However, our AP solution does not have these problems. Our CSX600 prototype can guarantee all tasks to be finished within their deadlines.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we describe an efficient and scalable solution for ATC systems using the AP system that is emulated on the CSX600 system. The contributions of this paper are summarized as follows. First, the proposed AP solution will support accurate and meaningful predictions of worst case execution times and will guarantee all deadlines are met. Second, the AP implementation has much better scalability and efficiency than the MIMD implementation. Third, the execution time of the AP implementation is extremely predictable while the MIMD implementation is very unpredictable. These three 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 [18]. Fourth, the software used by the AP is much simpler and smaller in size than the current corresponding ATC software, as the MIMD software must also support additional activities such as dynamic scheduling and load balancing etc, which are not needed by the AP. Also the AP hardware is considerably cheaper, simpler, and easier to build than the MIMD hardware currently used to support ATC. As observed in Section I, SIMDs (and APs) and MIMDs each have their particular advantages

and disadvantages. However, this paper establishes several important reasons that APs are much better suited for real-time applications with hard deadlines than MIMDs.

A possibly important extension to our current research would be to consider a GPU implementation using CUDA, which 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. Another potential project is to investigate implementing our prototype on other parallel systems, e.g., a Cray Systems, such as their vector processor, IBM's Cell processor, and Convex with FPGA reconfigurable hardware. The ATC problem has similar requirements to most embedded real-time problems with periodic tasks, e.g., command and control problems in military, etc. As a result, this research is relevant not only to the ATC problem, but also to other important applications that involve real-time problems with hard deadlines.

REFERENCES

[1] W. Meilander, M. Jin, and J. Baker, "Tractable real-time air traffic control automation," in *Proc. of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Cambridge, MA, Nov. 2002, pp. 483–488.

[2] W. Meilander, J. Baker, and M. Jin, "Predictable real-time scheduling for air traffic control," in *Fifteenth International Conference on Systems Engineering*, Aug. 2002, pp. 533–539.

[3] ——, "Importance of simd computation reconsidered," in *Proc. of the 17th International Parallel and Distributed Processing Symposium (IEEE Workshop on Massively Parallel Processing)*, Nice, France, Apr. 2003.

[4] M.Yuan, J.W.Baker, F.Drews, and W.Meilander, "Efficient implementation of air traffic control (atc) using the clearspeed csx620 system," in *Proc. of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Cambridge, MA, November 2009, pp. 353–360.

[5] S. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "Clearpath: highly parallel collision avoidance for multi-agent simulation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation(SCA)*. ACM, August 2009, pp. 177–187.

[6] M.Yuan, J.W.Baker, F.Drews, L.Neiman, and W.Meilander, "An efficient associative processor solution to an air traffic control problem," in *Large Scale Parallel Processing IEEE Workshop at the International Parallel and Distributed Processing Symposium (IPDPS2010)*, Atlanta, GA, Apr. 2010.

[7] "Openmp website," 2010, http://openmp.org/wp/.

[8] "Clearspeed technology plc. clearspeed whitepaper: Csx processor architecture," 2007, http://www.clearspeed.com/docs/resources/.

[9] (2007) Clearspeed technology plc. clearspeed whitepaper: Clearspeed software description. [Online]. Available: https://support.clearspeed.com/documents/

[10] "Clearspeed technology plc. csx600 runtime software user guide," 2007, https://support.clearspeed.com/documents/.

[11] J. Gustafson and B. Greer, "Clearspeed whitepaper: Accelerating the intel math kernel library," 2007, http://www.clearspeed.com/docs/resources/ClearSpeed Intel Whitepaper Feb07.pdf.

[12] J. A. Rudolph, "A production implementation of an associative array processor - staran," in *The Fall Joint Computer Conference (FJCC)*, Los Angeles, CA, Dec. 1972.

[13] 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.

[14] M. Jin, J. Baker, and K. Batcher, "Timings for associative operations on the masc model," in *Proc. of the 15th International Parallel and Distributed Processing Symposium (IEEE Workshop on Massively Parallel Processing)*, San Francisco, CA, Apr. 2001, pp. 193–200.

[15] K. Park, N. Singhal, M. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 91–104, January 2011.

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

[17] K. Lakshmanan, S. Kato, and R. Rajkumer, "Scheduling parallel real-time tasks on multi-core processors," in *Proc. of the 31st IEEE Real-Time Systems Symposium (RTSS'10)*, San Diego, CA, December 2010, pp. 259–268.

[18] (2009) Faa's nextgen implementation plan. [Online]. Available: http://www.faa.gov/about/initiatives/nextgen/media/ngip.pdf