EFFICIENT IMPLEMENTATION OF AIR TRAFFIC CONTROL USING THE CLEARSPEED CSX600 SYSTEM

Mike Yuan,Johnnie W. Baker
Department or Computer Science
Kent State University
Kent, Ohio, USA
email: myuan,jbaker@cs.kent.edu

Frank Drews
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio, USA
Email: drews@ohio.edu

Will Meilander(retired)
Department or Computer Science
Kent State University
Kent, Ohio, USA
email: willcm@charter.net

ABSTRACT
Past approaches for air traffic control (ATC) problems use MIMD solutions and have severe problems meeting the requirements of ATC. We propose a new and efficient solution to the ATC problem using SIMD architecture ClearSpeed CSX620. This solution uses synchronous processing of jobs and proposes a new tracking algorithm that can estimate states accurately and a new conflict detection and resolution(CD&R) algorithm that can guarantee the qualities of safety and efficiency. A preliminary prototype of the proposed method has been developed on the ClearSpeed CSX620 and results are presented.

KEY WORDS
Air Traffic Control (ATC); conflict detection and resolution (CD&R); SIMD; ClearSpeed; MIMD.

1 Introduction

We consider the problem of automatic ATC in this paper. The ATC software consists of multiple real-time tasks that must be completed in time to meet the individual deadlines. Because of the continued growth of air traffic, it is very necessary to improve safety and efficiency. 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 rather than following preselected flight corridors to minimize fuel consumption and time delay[10, 19, 25]. Massive efforts have been devoted to finding an efficient MIMD solution to the ATC problems for many years. However, none of the results have been satisfactory thus far. One notable example is the ten-year effort to develop the Automated Air Traffic Control System (AAS) that was canceled in June 1994 after expenditure of several billion dollars[17]. It is currently widely accepted that large real-time problems such as ATC does not have a polynomial time MIMD solution[5, 23].

Conflict detection and resolution (CD&R) is the most critical issue for supporting *free flight*. According to the comprehensive survey of Kuchar and Yang[12], all CD&R algorithms are based on aircraft state estimation. Furthermore, all traffic advisories are based on the aircraft's current state estimates. The Kalman filter[1, 4, 22] is the central algorithm to 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, e.g, makes a maneuver, accelerates etc. Many adaptive state estimation algorithms have been proposed [2, 7, 8, 15, 24]. The Interacting Multiple Model (IMM) algorithm [3, 13] runs two or more Kalman filters in parallel, each using a different model for target motion or errors. Then the IMM model forms an optimal weighted sum of the outputs of all the filters and is able to adjust to target maneuvers. The IMM algorithm is computationally intensive in current MIMD implementations. Our approach first develops boxes around all tracks and radar reports, then checks each report box and against all track boxes. If a report box intersects with a track box, the report correlates with this track. If there are any uncorrelated tracks, double the sizes of their boxes and execute the correlation algorithm again. If there are still uncorrelated tracks, increase the sizes of their boxes to three times their original size and execute the algorithm. The correlated reports are used to get better prediction of tracks, this process is called track smoothing. The number of radar reports received each half-second is large (e.g., $6,000$), entering this data into the parallel processors reduces the run time of tracking algorithms. The ClearSpeed architecture can finish large scale jobs within deadlines using its features including simultaneous parallelism of instructions, efficient associative operations and data transfer etc.

There has been a lot of previous work on solving CD&R problems[6, 7, 11, 20, 25]. A comprehensive survey of the CD&R methods is presented in[12]. All of these algorithms are intended for MIMD implementation and have problems in guaranteeing both safety and efficiency at the same time[12]. Our detection algorithm compares each air-

Figure 1. CSX620 accelerator board

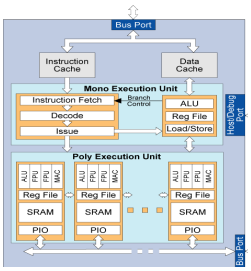

Figure 2. MTAP architecture of CSX620

craft with all the other ones to detect conflicts if their flight modes do not change within the look ahead time. Our resolution algorithm picks out the aircraft that will collide in the shortest time, computes its future flight path repeatedly if its heading is incremented from left to right 30 additional degrees, detects conflicts between each changed path and all the other aircraft, and find the one that can result in the latest collision time.

This paper is organized as follows: Section 2 reviews the architecture of CSX620 and its C^n programming language. In Sections 3, we provide an overview for our ATC system design. Section 4 presents implementations of various ATC tasks, including report correlation and tracking, track smoothing and updating, conflict detection and resolution etc. Section 5 presents experimental results. Conclusions and future work are presented in Section 6.

2 Overview of ClearSpeed CSX620

2.1 Architecture of CSX620

The ClearSpeed accelerator board is a PCI-X card equipped with two CSX620 coprocessors. A view of the board is shown in Figure 1. The CSX620 is a multi-core processor with 96 processing elements (PEs) connected in the form of a one-dimensional array. This multi-core section is called a multi-threaded array processor (MTAP) core, and the architecture is shown in Figure 2. The PEs operate at a clock speed of 250 MHz. 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 that operate parallel data. Each PE has its own local memory of 6 Kbytes that provides high bandwidth to frequently used data.

2.2 Programming Concept

The ClearSpeed accelerator provides the C^n language as the programming interface for the CSX620 processors. It is very similar to the standard C programming language. The main difference is that it introduces mono and poly variables that are held in mono and poly execution unit separately. Values defined as mono are scalar (i.e., non-vector) values as in Standard C. Variables defined as poly have an instance on each PE and enable parallel data processing. These can also be viewed as vector values. Some standard libraries are provided, including memory and print operations, in addition to specialized math libraries that exploit the SIMD functionality of the accelerator. Library functions for data transfer between mono and poly memories are provided, for example:

$$memcpym2p(\&pdst, \&msrc, 4 * \text{sizeof(struct track)});$$

This function transfers 4 track records from the address pointed to by *msrc* in the mono to *pdst* in the poly. Data transfer from poly to mono involves the library function $memcpyp2m$.

The swazzle network is a circular (or ring) network connecting all 96 PEs. Functions for shifting or exchanging data with adjacent PEs via *swazzle* path are also provided:

$$track.X_t(t_k) = circular\_swazzle\_down(track.X_t(t_k));$$

This function sends the value $track.X_t(t_k)$ to the left neighbor (PE 0 will send to PE 95) and overwrite the $track.X_t(t_k)$ on it.

3 Overview of ATC Design

The flow chart of the whole ATC system is shown in Figure 3. We simulate radar reports by using flight plans in the host (or server) and adding some random noise. Next, we transfer them from the host to mono memory, then to PEs, and correlate them with tracks that are predicted in PEs to estimate aircraft states. Then execute tasks such as conflict detection and avoidance etc. There are more than one track in each PE, which is different from the assumption in [16] that there is only one track in each PE.

3.1 ATC Component Overview

There are two basic components: radar reports in the host, and tracks in the PEs. Their data structures are shown in Table 1 and Table 2, in which positions of next period are predicted by smoothed current positions and velocities.

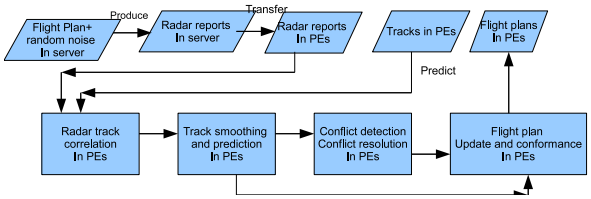

Figure 3. ATC system data flow

Table 1. Data Structures for *Radar Reports*

Attribute	Type	Comments
$report\_id$	int	Report identity
r	float	Report box size
$X_r(t_k)$	float	X position
$Y_r(t_k)$	float	Y position
$H_r(t_k)$	int	Altitude
$Match\_count$	int	Number of correlated tracks
$Match\_id$	int	ID of the correlated track

Table 2. Data Structures for *Tracks*

Attribute	Type	Comments
ID	int	Flight identity
Q	int	Track state(seven values)
C	int	Error measures(3 values)
j	float	Track box size
$report\_id$	int	ID of correlated report
$X_t(t_k)$	float	Current X position
$Y_t(t_k)$	float	Current Y position
H_t	int	Current altitude
$X_t(t_{k+1})$	float	Predicted X position
$Y_t(t_{k+1})$	float	Predicted Y position
$V_x(t_k)$	float	Current X velocity
$V_y(t_k)$	float	Current Y velocity
$V_x(t_{k+1})$	float	Predicted X velocity
$V_y(t_{k+1})$	float	Predicted Y velocity
$X_r(t_k)$	float	X position of correlated report
$Y_r(t_k)$	float	Y position of correlated report

3.2 Host Board Communication

Because the CSX620 board does not have a clock itself and we can only use function $get\_cycles()$ to count cycles on PEs, the task scheduler is implemented on the host. There

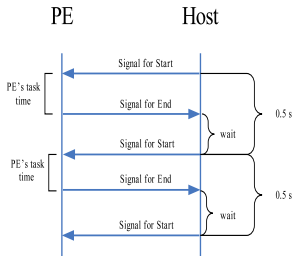

Figure 4. Host board communication

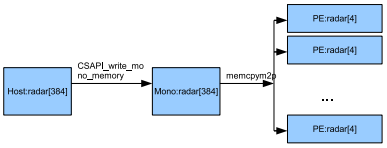

Figure 5. Illustration of data transfer

are two semaphores named sem_start and sem_end between the host and PEs. The host first signals sem_start, and the PEs wait for this signal and then execute. The host records the start time, waits for sem_end that is sent by PEs indicating that this task is finished, and records the time. The host next calculates how much time is spent in this task, then waits for the remaining time in the cycle and then starts the process over again, Figure 4 illustrates the process.

3.3 Data Transfer

The CSX620 does not support transferring data from outside devices into PEs directly, so the radar reports should be read into the host first, transferred from the host to the mono memory, then transferred to PEs. The process is illustrated in Figure 5. We use $CSAPI\_write\_mono\_memory$ function to transfer radar reports from the host to mono memory, then use $memcpym2p$ to transfer them from mono to PEs.

4 ATC Tasks Implementation

4.1 Report Correlation and Tracking

In this section, we consider the problem of correlating radar reports with the predicted positions of established tracks for aircraft under observation. Our current implementation does not include height information that can be used to produce a better correlation. However, the technique used in this 2D algorithm can be easily extended to create a similar 3D algorithm.

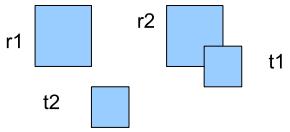

Figure 6. Track/report intersect

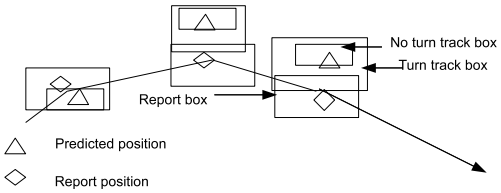

Figure 7. Search box size for flight maneuver

First boxes are created around each radar report and each track to accommodate report and track uncertainties. These boxes are $(X_r(t_k) \pm r, Y_r(t_k) \pm r)$ for each report and $(X_t(t_k) \pm j, Y_t(t_k) \pm j)$ for each track, where $r > 0$ is based on uncertainties in the radar report, and $j > 0$ is based on uncertainties of each track.

Each report box is compared with every track box in each PE. If the box around a radar report intersects the box around a track, they are said to be correlated.As seenn in Figure 6, $r1$ and $t2$ do not intersect and $r2$ and $t1$ intersect. If the boxes intersect, the $match\_count$ of this report is incremented, its ID is entered into the correlated track's $report\_id$, and the ID of the track is entered into the radar report's $match\_id$. We calculate the distance between them, which is the track's $shortest\_distance$. This report's positions $X_r(t_k)$ and $Y_r(t_k)$ are entered into the track's $X_r(t_k)$ and $Y_r(t_k)$. Then the radar reports in each PE are transferred to next PE using the $swazzle$ function, and this process is repeated. If two tracks correlate to the same radar report, i.e., the radar report's $match\_count > 1$, then this radar report is discarded. If a second radar report correlates with the same track, calculate the distance between the track and radar report and call it the track's $current\_distance$; if it is less than $shortest\_distance$, update the $shortest\_distance$ to be this distance, and record this report's position as a candidate report position for this track. After the above procedure is repeated 96 iterations, all report boxes have been compared with all track boxes, set error measure $C = 1$ for the tracks that have correlated reports. The use of error measure will be explained in the next section.

A flight that is not produced by noise might not correlate to any reports because the flight that it corresponds to is maneuvering. We increase track box size for any track that has not correlated with a radar report to increase its probability to intersect a radar report box as shown in Figure 7. First we double the box sizes of tracks that do not correlate any reports, i.e, $j = j \times 2$. Next, we apply the same algorithm to compare them with uncorrelated reports whose $match\_count$ are 0. The error measures C are set to 2 for tracks that correlate reports in this round. We then triple the original box sizes of tracks that have not correlated any reports yet, and run the algorithm again. The error measures C are set to 3 for tracks that correlate reports in this round. If a track has not correlated with any radar reports after the third round, it is discarded as noise. If a radar report has not correlated with any tracks after the third round, it may correspond to a new plane so we create a new track for this possible new flight.

Table 3. Track State Table

Q-Track state	C-Error measure		
	1	2	3
1	2	2	2
2	3	3	2
3	6	4	3
4	6	4	3
5	6	4	4
6	7	5	4
7	7	6	5

Table 4. Pre-Computed Weight Table

Track state Q	Position weight W_p	Velocity weight W_v
1	0.99	0.95
2	0.8333	0.5
3	0.7	0.3
4	0.66	0.24
5	0.524	0.143
6	0.464	0.107
7	0.417	0.083

4.2 Track Smoothing and Prediction

After a correlated report for a track is found at time t_k, it is necessary to smooth the position and velocity to predict the next position and velocity at time t_{k+1}. The error measure C (three values) is used to select the next quality Q of track state (seven values) that is used for smoothing. Table 3 shows how to compute next Q by current Q and C. For example, if current track state Q is 4, error measure C is 3, then next track state Q is 3. Table 4 shows the smoothing constants for each track coordinate by Q, which is used for the smoothing. The following equations 1 and 2 are to smooth positions of a track using $X_r(t_k)$ and $Y_r(t_k)$ that are correlated report positions of this track[14]. The

Table 5. Data Structures for *Collision*

Attribute	Type	Comments
ID	int	Flight's ID
X_c	float	Current X position
Y_c	float	Current Y position
H_c	int	Current altitude
V_{xc}	float	X velocity
V_{yc}	float	Y velocity
$time\_till$	float	Soonest collision time
$collide\_ID$	int	Collision track ID

equations 3 and 4 are to smooth velocities, ΔT is the time interval between measurements.

$$X_t(t_k) = X_t(t_k) + (X_r(t_k) - X_t(t_k)) \times WX_p \quad (1)$$

$$Y_t(t_k) = Y_t(t_k) + (Y_r(t_k) - Y_t(t_k)) \times WY_p \quad (2)$$

$$V_x(t_k) = V_x(t_k) + WX_v \times (X_r(t_k) - X_t(t_k))/\Delta T \quad (3)$$

$$V_y(t_k) = V_y(t_k) + WY_v \times (Y_r(t_k) - Y_t(t_k))/\Delta T \quad (4)$$

The estimated positions of x and y at time t_{k+1} $X_t(t_{k+1})$ and $Y_t(t_{k+1})$ are shown in equations 5 and 6. The estimated velocities of x and y at time t_{k+1} $V_x(t_{k+1})$ and $V_y(t_{k+1})$ are shown in equations 7 and 8.

$$X_t(t_{k+1}) = X_t(t_k) + V_x(t_k) \times \Delta T \quad (5)$$

$$Y_t(t_{k+1}) = Y_t(t_k) + V_y(t_k) \times \Delta T \quad (6)$$

$$V_x(t_{k+1}) = V_x(t_k) \quad (7)$$

$$V_y(t_{k+1}) = V_y(t_k) \quad (8)$$

The results of equations 5, 6, 7 and 8 will be used in equations 1, 2, 3 and 4 as previously estimated data at the next time.

4.3 Conflict Detection

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 twenty minute period, i.e, 1200 seconds. We have a poly structure *collision*, whose data structures are shown in Table 5.

First we copy each *track*'s *ID*, $X_t(t_{k+1})$, $Y_t(t_{k+1})$, H_t, $V_x(t_{k+1})$ and $V_y(t_{k+1})$ to a *collision*'s *ID*, X_c, Y_c, H_c, V_{xc} and V_{yc}, and initialize $time\_till$ to 1200.00. For each *collision* and *track* in each PE, first check whether their flight *IDs* are different and altitudes are within 1000 feet, i.e., $|collision.H_c - track.H_t| < 1000$. Then project their positions as envelopes into the future time, add 1.5 to each x and y edge of the future position to provide a 3.0 minimum miss distance, which is shown in Figure 8. We

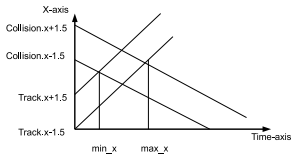

Figure 8. Conflict detection

calculate the min_x, max_x, min_y and max_y for minimum and maximum intersection times in x and y dimensions:

$$min\_x = \frac{|collision.X_c - track.X_t(t_{k+1})| - 3}{|collision.V_{xc} - track.V_x(t_{k+1})|} \quad (9)$$

$$max\_x = \frac{|collision.X_c - track.X_t(t_{k+1})| + 3}{|collision.V_{xc} - track.V_x(t_{k+1})|} \quad (10)$$

$$min\_y = \frac{|collision.Y_c - track.Y_t(t_{k+1})| - 3}{|collision.V_{yc} - track.V_y(t_{k+1})|} \quad (11)$$

$$max\_y = \frac{|collision.Y_c - track.Y_t(t_{k+1})| + 3}{|collision.V_{yc} - track.V_y(t_{k+1})|} \quad (12)$$

Find the biggest minimum time $time\_min$ and smallest maximum time $time\_max$ across the two dimensions:

$$time\_min = max\{min\_x, min\_y\} \quad (13)$$

$$time\_max = min\{max\_x, max\_y\} \quad (14)$$

Compare $time\_min$ with $time\_max$, if $time\_min$ is less than $time\_max$, indicate that there is a potential conflict between the $collision.ID$ and $track.ID$. Check whether $time\_min$ is less than $collision.time\_till$, if so, $collision.time\_till$ is updated to $time\_min$. Then all *collision* records in each PE are passed to next PEs by *swazzle* function and are compared to *tracks* in that PE using the same algorithm. After 96 iterations, all *collisions* have been compared with all *tracks*. The $time\_till$ and $collide\_ID$ of each *collision* is its soonest collision time with another track and that track's ID .

4.4 Conflict Resolution

We first find the minimum $time\_till$ of *collisions* in each PE:

$$for(i = 0; i < numberperpe; i++)\{$$
$$if(collision[i].time\_till < mintimetillonpe)$$
$$mintimetillonpe = collision[i].time\_till;\}$$

Find the minimum $time\_till$ across PEs:

$$mintimetillonmono = cs\_reduce\_min(mintimetillonpe);$$

Transfer *collision* records from PEs to mono:

$$c\_dst\_addr = tmp\_c + numberperpe * get\_penum();$$

Table 6. Data Structures of *Projected*

Attribute	Type	Comments
ID	int	Best flight's ID
X	float	Its X position
Y	float	Its Y position
H	int	Its altitude
V_{xhc}	float	X velocity by heading change
V_{yhc}	float	Y velocity by heading change
$time\_till$	float	Soonest collision time

$$memcpyp2m(c\_dst\_addr, \&collision, numberperpe*sizeof$$

(struct collision);

Search for the ID of the flight that has maximum $time\_till$:

int minid $= 0$;

$$while(tmp\_c[minid].time\_till! = mintimetillonmono)\{$$

$$minid++;\}$$

This is the best flight that will make the heading change. Calculate that flight's *velocity* and *angle*:

$$velocity = \sqrt{(tmp\_c[minid].V_{xc})^2 + (tmp\_c[minid].V_{yc})^2}$$

$$angle = \arctan(\frac{tmp\_c[minid].V_{yc}}{tmp\_c[minid].V_{xc}})$$

We design a *projected* struct, its data structures are shown in Table 6. We have a *projected* struct $tmp\_f2[96]$ in mono, copy the best flight's ID, X_c, Y_c and H_c to ID, X, Y and H of $tmp\_f2[0], tmp\_f2[1], \cdots, tmp\_f2[95]$, initialize *time_till* to 1200.00. Set a float variable *frac_rad* to be 0.0109083 that is the radian of $30 \div 48 = 0.625$ degree. Each of *tmp_f2[i]* represents a path where the best flight makes a different heading change from left to right 30 degrees. The program segments are shown below:

$$for(i = 0; i < 48; i++)\{$$

$tmp\_f2[i].V_{xhc} = velocity \times \cos(angle + (i+1) \times frac\_rad);$

$tmp\_f2[i].V_{yhc} = velocity \times \sin(angle + (i+1) \times frac\_rad);$

$tmp\_f2[i+48].V_{xhc} = velocity \times \cos(angle - (i+1) \times frac\_rad);$

$tmp\_f2[i+48].V_{yhc} = velocity \times \sin(angle - (i+1) \times frac\_rad);\}$

Transfer the $tmp\_f2[]$ from mono to a poly *projected* struct *projectedpath* in PEs, the programs are shown below:

$$f2\_src = tmp\_f2 + get\_penum();$$

$$memcpym2p(\&projectedpath, f2\_src, sizeof(struct$$

$$projected));$$

Each *projectedpath* is compared to all *collision* records in each PE. If their flight *IDs* are different and altitudes are within 1000, 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 9, 10, 11 and 12, just replace $track.X_t(t_{k+1})$ with $projectedpath.X$, $track.Y_t(t_{k+1})$ with $projectedpath.Y$, $track.V_x(t_{k+1})$ with $projectedpath.V_{xhc}$, and $track.V_y(t_{k+1})$ with $projectedpath.V_{yhc}$. Then use equations 13 and 14 to get $time\_min$ and $time\_max$. If $time\_min$ is less than $time\_max$, there is a potential conflict between the *collision ID* and this path. Check whether $time\_min$ is less than $projectedpath.time\_till$, if so, $projectedpath.time\_till$ is updated to $time\_min$. The *projectedpath* records in each PE are then passed to the next PE to compare with *collision* records in that PE. After 96 iterations, all *projectedpath* envelopes have been compared to all *collision* envelopes and their $time\_till$ values are their soonest collision time with another flight. We find the maximum one:

$$maxtime = cs\_reduce\_max(projectedpath.time\_till);$$

Transfer *projectedpath* records from PEs to mono again:

$$dst\_addr = dest + get\_penum();$$

$$memcpyp2m(dst\_addr, \&projectedpath, sizeof(struct$$

$$projected));$$

Find which new path has the maximum $time\_till$, that path is the best scenario:

$$m = 0;$$

$$while(dest[m].time\_till! = maxtime)\{$$

$$m++;\}$$

Transfer *track* records from PEs to mono:

$$track\_dst = tmp\_track + numberperpe * get\_penum();$$

$$memcpyp2m(track\_dst, \&track, numberperpe * sizeof$$

(struct track));

The track whose index is $minid$ is the best flight that will make a maneuver. Change its x and y velocity to the best path's that have just been found:

$$tmp\_track[minid].V_x(t_{k+1}) = dest[m].V_{xhc};$$

$$tmp\_track[minid].V_y(t_{k+1}) = dest[m].V_{yhc};$$

Finally transfer *track* records back from mono to PEs:

$$track\_src = tmp\_track + numberperpe * get\_penum();$$

$$memcpym2p(\&track, track\_src, numberperpe * sizeof$$

(struct track));

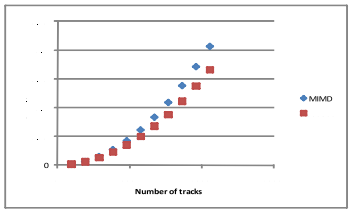

Figure 9. Tracking time comparison MIMD vs. CSX

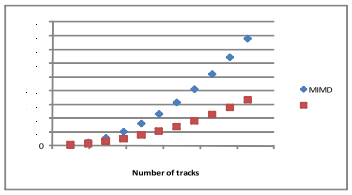

Figure 10. CD&R time comparison MIMD vs. CSX

5 Experiment Results

Radar reports in host and tracks in PEs are generated by giving each flight a random starting position and velocity. The altitude for each flight is an integer between $3,000$ and $6,000$ feet, the range for both X and Y is between -128 and 128 nautical miles (Nm) with the origin $(0,0)$ at its center. Target velocities are between 30 and 600 knots (nautical miles per hour) and an average of 250. We use a dual processor, each of which has four cores to simulate shared memory MIMD. We divide all aircraft into the eight threads, and execute the same tracking and CD&R algorithms.

We run 100 trial Monte Carlo simulations and compare the timing and scalability of tracking and CD&R between shared memory MIMD and CSX. The results are shown in Figure 9 and Figure 10. When there are more than 800 aircraft, MIMD will take double and even more time than CSX in tracking algorithm. It is difficult for the MIMD approach to finish CD&R tasks within deadline when the system has more than 500 flights, while CSX approach can finish CD&R tasks within deadline even when the system goes up to 1200 flights.

6 Conclusion

We propose a simple and predictable solution for our nation's ATC system based on ClearSpeed CSX620. The contributions of this paper are to use the SIMD and special features of CSX620 to guarantee the safety, efficiency and predictability of important ATC tasks such as report correlation and tracking, and CD&R. We will add more ATC tasks such as flight plan conformance and update, and terrain avoidance etc to complete our current prototype, and we will improve the scalability and efficiency of the tracking, and CD&R algorithms etc.

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, 33*(8), August 1988, 780-783.

[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, Artech House, 1990, pages 31-63.

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

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

[7] 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.

[8] 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.

[9] 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, 2001.

[10] S. Kahne and I. Frolow, Air traffic management: Evolution with technology, *IEEE Control Systems Magazine, 16*(4):1996, 12-21.

[11] 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.

[12] J.K. Kuchar and L.C. Yang, A review of conflict detection and resolution modeling methods, *IEEE Transactions on Intelligent Transportation Systems*, *1*(4): 2000, 179-189.

[13] 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, 1*(3): September 1993, 186-194.

[14] K.M. Liu, Composition of Kalman and heuristic tracking algorithms for air traffic control, master thesis, Kent State University, August 1999.

[15] 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, 34*(1): 1998, 103-123.

[16] 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*, Cambridge, MA, 2002, 483-488.

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

[18] 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.

[19] M.S. Nolan, *Fundamentals of Air Traffic Control* (Brooks/Cole: Wadsworth, 3rd edition, 1998).

[20] R.A. Paielli and H. Erzberger, Conflict probability estimation for free flight, *Journal of Guidance, Control, andDynamics, 20*(3):1997, 588-596.

[21] 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, April 2005.

[22] P. Stage, and J.L. Melsa, *Estimation Theory with Application to Communication and Control* (New York: McGraw-Hill, 1971).

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

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

[25] L.C. Yang and J.K. Kuchar, Prototype conflict alerting system for free flight, *Journal of Guidance, Control, and Dynamics*, *20*(4): July-August 1997, 768-773.