Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

TIME-PARALLEL SIMULATION OF AIR TRAFFIC NETWORKS

Young Jin Kim

Artificial Intelligence Products Group
Intel Corporation
Santa Clara, CA 95054, USA

Dimitri Mavris

School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

Richard Fujimoto

School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

ABSTRACT

Air traffic management is widely studied in several different fields because of its complexity and criticality to a variety of stakeholders. However, the exploding amount of air traffic in recent years has created new challenges to ensure effective management of the airspace. A fast time simulation capability is essential to effectively explore the consequences of decisions of air traffic management. A new algorithm for simulating air traffic networks using a time-parallel simulation approach is proposed that distributes time segments of the simulation scenarios across different processors. A simulation model for the National Airspace System (NAS) is described and validated. The components of the simulator are described as well as the parallel simulation algorithms. Experimental results utilizing real-world traffic data for the continental U.S. are presented demonstrating the speed ups achieved by a prototype simulator. These results illustrate that time-parallel simulation can be used to significantly accelerate certain air traffic simulations.

1 INTRODUCTION

According to the Federal Aviation Administration (FAA)'s latest forecast, domestic enplanement will be increased by 1.45 times over the next twenty years. Over the same time period, it is estimated that the number of passengers taking international flights into or leaving the U.S. will be double (Aerospace Forecast, FAA 2016). This increased traffic could result in significant delays in the National Airspace System (NAS). According to one study (Ball et al. 2010), air traveler delays accounted for approximately $33 billion in direct or indirect costs to passengers, airlines and other parts of the NAS in 2007.

Computer simulations of the NAS play a key role in helping to alleviate this concern by evaluating many different possible scenarios and situations. Rapid execution of simulation models is important in order to explore a wide variety of scenarios quickly. Parallel processing offers an approach to accelerate simulation executions, and several different parallel simulation algorithms have been explored. These approaches use spatial parallelism where one divides the NAS into distinct regions, and one distributes the state and associated computations to transform this state across different processors so that they can be performed concurrently. To ensure that the computation is correct, it is necessary to properly synchronize these computations. In an air traffic network simulation, the components of the system such as airports, air traffic control centers, flights etc. for a region are typically mapped to a single logical process (LP). Each LP computes its internal states during the simulation run and communicates necessary events with other

LPs. Challenges for achieving significant speed up of these simulations include the realization of efficient synchronization among the LPs and managing communication overheads.

Another approach to parallel simulation is referred to as the *time-parallel* approach. Time-parallel simulation involves dividing the simulation time axis into non-overlapping segments T_1, T_2, ... T_N where the end of time segment T_i coincides with the beginning of time segment T_{i+1}. We then assign the simulation of each time segment to a different logical process LP_1, LP_2, ... LP_N, with LP_i responsible for simulating time interval T_i. Each such LP may then execute concurrently on a different processor. This would be a very straightforward approach to parallelization if the simulations of the different time segments were independent of each other. This is rarely the case, however, because (1) the initial state of LP_i depends on (is identical to) the final state of LP_{i-1} — the so-called *state matching problem* — and (2) the events occurring in LP_i may depend on the simulation computations performed in earlier time segments. To address this problem, time-parallel simulations typically require multiple rounds of execution, and utilize a *fix up* computation to correct errors resulting from data dependencies between LPs that were not accounted for in the initial execution. Time-parallel simulations can become inefficient if an error propagates through multiple time segments because an additional round of fix up computations is needed to propagate the correction through each successive time segment.

Here, we exploit three properties of realistic air traffic simulations to mitigate the problem of fix up computations having to propagate through multiple time segments. First, the simulations are driven by statically defined *flight schedules* that indicate the scheduled departure and arrival times of each aircraft at each airport it visits. These schedules are known prior to the execution of the simulation and can be used by each LP to initially simulate the activity within its time segment. Second, so long as the computed arrival of a simulated aircraft leads to an on-time departure, an error in the computed arrival time will not impact the aircraft's departure, i.e., errors in computing the arrival time will not propagate further in the future trajectory of the aircraft. This greatly lessens the propagation of errors through the time segment. Error propagation resulting from fix up computations largely occur when the aircraft arrives sufficiently late to affect its subsequent departure time from the airport.

A third factor that facilitates the use of time-parallel simulation concerns the fact that airlines are often organized around a "hub" model where many flights travel through certain *hub* airports where passengers change flights to reach their eventual destination. A typical airline schedule will include many flights arriving within a small time window, a short period of time where passengers change flights, followed by the departure of many flights from the hub. This cycle repeats throughout the day. This results in bursts of high activity at the airport followed by periods of light activity before the next round of flights arrive. This is advantageous for time-parallel simulations because the effects of congestion-induced delays occurring in one busy period are mitigated by periods of light traffic, meaning computation errors due to delayed aircraft are less likely to propagate throughout the day in a typical airport in the absence of severe events that impact the entire airport for a prolonged period of time.

Moreover, these factors suggest that a time-parallel simulation approach may be well suited for air traffic simulation. It is conceivable that one might exploit *both* space and time-parallel methods to accelerate air traffic simulations. However, the work described here focuses only on the time-parallel approach. Extension to also exploit spatial parallelism is an area of future research.

The remainder of this paper is organized as follows. The next section describes related work. The discrete event simulation model utilized here is then described. The time-parallel simulation algorithm is described, followed by presentation of experimental results based on actual historical data for the NAS and concluding remarks.

2 RELATED WORK

There is a significant literature concerning different modeling approaches for simulating air traffic network systems including queueing network based models, agent-based models and system dynamics models, e.g., see (Long et al. 1999, Pinon 2012, Kim et al. 2015). Several studies have focused on the use of parallel

simulation algorithms to speed up the fast time simulation models. Wieland reports results concerning the use of parallel simulation algorithms for aviation applications (Wieland 1998). Lee, Pritchett et al. presented several different parallel simulation algorithms for the analysis of the NAS (Lee et al. 2001). Hybinette and Fujimoto proposed a new method to maximize the parallel efficiency of parallel aviation simulation by using a simulation cloning technique (Hybinette and Fujimoto 2001).

There have been several studies using the time-parallel simulation approach for a variety of applications. It was used for trace-driven cache simulation (Heidelberger and Stone 1990), stochastic automata networks (Thi et al. 2014), multigrid PDE simulation (Grasedyck et al. 2016), and other applications. Several approaches attack the state matching problem. Some use repeated fix up computations to correct errors in guessed initial states (Fujimoto 2000). Time-parallel simulation of queues using parallel prefix computation algorithms were introduced by Greenberg, Lubachevsky et al. (Greenberg et al. 1991). In order to improve parallel efficiency, there have been several approximate state matching algorithms developed. Wang and Abrams proposed an approximate time-parallel simulation algorithm of queueing systems with losses (Wang and Abrams 1992). Kiesling, Tobias et al. proposed a time-parallel simulation algorithm with approximate state matching and analyzed its efficiency and accuracy (Kiesling and Pohl 2004).

Although there is a growing literature in time-parallel simulation, to our knowledge, there has not been an attempt to apply it to the problem of air traffic network simulations. The effort described here addresses this question.

3 SIMULATION MODEL

Here, a discrete event simulation model is used to model the NAS for a single day in the continental U.S. Specifically, we model the domestic airports in the U.S. with the commercial flights using a queuing network-like model. Each flight departs from an airport based on the flight schedule for the day and the availability of resources for the flights. These resources typically include runways, gates, taxi ways and the air traffic controllers in the airport. Here, we focus on modeling airport runways and delays associated with waiting to utilize the runways. Each airport models arriving and departing flights based on the schedule for the day. The sequence of arriving and departing flights is modeled as events within the simulation. The main goal of the simulation is to track delays encountered by each flight in order to improve management of the NAS by minimizing delays and the number of diverted flights. Further, each aircraft is used for a sequence of flights throughout the day resulting in correlations and interactions among those flights. Details of the event definitions, resources and the sequence of events in the system are described next.

3.1 Simulation Model Resources

Runways are the main resource modeled within this simulation system. A principal task of the simulation is to compute queuing delays resulting from congestion as many flights compete for the available runways. The hub model described earlier that is used by major airlines can exasperate runway delays because arrivals and departures are scheduled to cluster into certain, busy time periods. All incoming and outgoing flights must wait until a runway is available to land or take off.

3.2 Event Definitions

The simulation utilizes five different types of events:

1. Arrived - The arrived event represents the arrival of an aircraft at either its original destination or the alternate airport in case a flight is diverted. This event is typically scheduled when a departed event is processed at the airport originating the flight. Alternatively, it can be also scheduled when a diverted event is processed. When this event is processed a runway is assigned as a resource that is used by the flight. In the time-parallel simulation, the time stamp of the arrived event could be changed by the fix up computation. This is discussed in greater detail later.

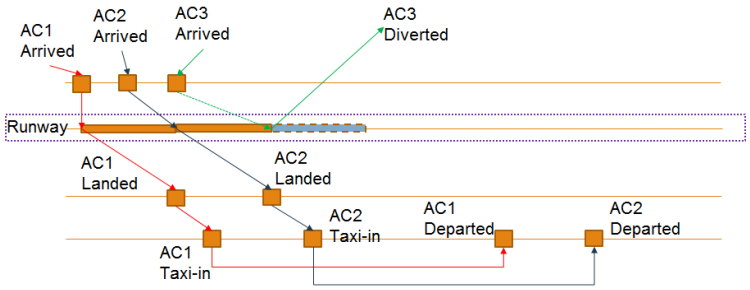


Figure 1: Single airport event flow.

2. Diverted - The diverted event represents the diversion of an aircraft to land at an alternate airport because of limited fuel and/or inability of the destination airport to handle the incoming flight due to limited capacity. The processing of this event will schedule another arrived event at the alternate airport.
3. Landed - The landed event represents the completion of the landing sequence for an aircraft. It represents the exit of the aircraft from the runway it used in the landing sequence. This event is scheduled when an arrived event is processed with some amount of time to model the landing of the aircraft on its assigned runway.
4. Taxi-in - The taxi-in event represents the arrival of an aircraft to a gate in order to unload and load passengers. In this simulation the traffic on the ground at the airport is modeled by a fixed time delay. Therefore, the event is scheduled when a landed event is processed with some amount of time delay determined by an analysis of historical data for that airport.
5. Departed - The departed event represents the departure of an aircraft from an airport based on the schedule and availability of resources at the destination airport. The event is scheduled when a taxi-in event is processed using certain delay computations. This computation utilizes a flight delay model from a delay analysis study in (Wong and Tsai 2012). The model uses the delay of the previous flight to compute the delay of the next flight. The model is described in greater detail later.

When the initial arrival and departure schedules are given for an airport, the events described above are scheduled according to the schedule. The arrived events are scheduled based on the anticipated arrival times. Then, the arrived events are processed with queuing delays in the runway resource. As a result of processing arrived events, the landed and taxi-in events are then scheduled. If the airport could not process the arrived event because of the limited capacity in the airport, then a new diverted event will be scheduled. Then, the delay model and the ground delay program model are utilized for the computation of the next time for departure while processing the taxi-in event. As a result of the combined model computation, a new departed event is scheduled. The sequence of event processing is illustrated in Figure 1.

3.3 Delay Model

In the simulation of one day of the NAS, an aircraft can have multiple flights. When it arrives at a destination airport, it will prepare for the next flight to be flown. Therefore, the delay from the previous flight might affect the delay of subsequent flights by that aircraft. To accommodate possible delays, schedules are defined that include buffer times are used to absorb delays in both ground operations and flight delays en route. Turn-around buffer time is used to mitigate possible departure delays by adding extra time for ground operations. On the other hand, block buffer time is used for mitigating possible arrival delays by adding extra time in the sky. When these buffer times are able to absorb delays on the ground and in the sky, there will be no delays relative to the published flight schedule. However, if the actual delays exceed the buffer times, there will be delays that can propagate through to other flights.

The formulas to compute the departure and arrival delays are shown below. D^d and A^d represent the actual departure delay and the actual arrival delays, respectively. G^d represents the ground handling time and b^{sg} represents the scheduled turn around buffer time on the ground. Similarly, R^d represents the block time in the air and b^{sr} represents the scheduled block buffer time.

$$D^d = \mathbf{max}\{0, A^d_{-1} + G^d - b^{sg}\}. \tag{1}$$

$$A^d = \mathbf{max}\{0, D^d + R^d - b^{sr}\}. \tag{2}$$

As stated earlier, new departed events and arrived events can be scheduled based on the computation in equations (1) and (2). The delay model for the departure is illustrated in Figure 2. It shows the case where a delayed taxi-in event doesn't affect the next departure schedule because there is sufficient buffer time on the ground.

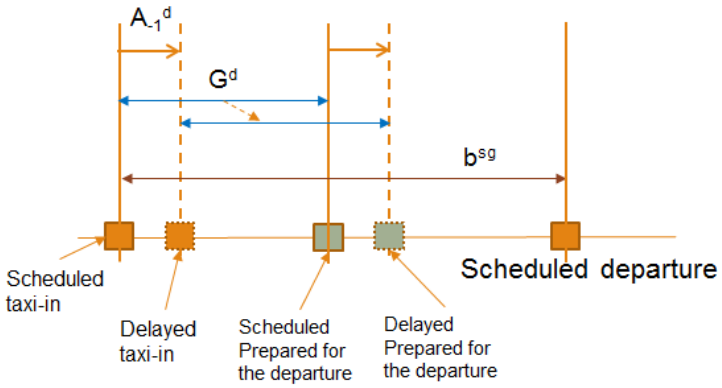


Figure 2: Delay model of the departure.

4 ALGORITHM

This section explains the algorithms used to execute the air traffic model described in the previous section. In particular, the time-parallel simulation algorithm is described as well as the fix up computation.

4.1 Time-Parallel Simulation

The goal of the simulation is to compute the state of the entire system, in this case the NAS, over a single day. While the space-parallel simulation assigns a set of nodes (airports) in the air traffic network to a single LP, the time-parallel simulation assigns a segment of simulation time to each LP. Then, the results from each LP are merged to produce the final result. In principle, one could use both time and space parallelism to simulate the NAS, however, here we focus exclusively on the time-parallel simulation approach.

Each LP simulates the NAS over a time segment, say $[T_y, T_z]$. In other words, the simulator must compute the trajectory of each aircraft over $[T_y, T_z]$, as it visits different airports during that time segment. The simulator has the scheduled arrival (and departure) times of aircraft. But initially, the location of each aircraft A_i at the start of the time interval T_y is unknown. However, the simulator does have the scheduled departure times. So, in the initial simulation, the simulator assume A_i leaves the gate on time from the first airport it visits during $[T_y, T_z]$, and it then simulates A_i's trajectory based on this assumption. This information, the *departure time of the aircraft from the gate of the first airport it visits during* $[T_y, T_z]$, is the key piece of information needed for the time parallel algorithm to work correctly.

After the first round, i.e., all time segment simulations complete their execution, the simulator for the previous time segment $[T_x, T_y]$ will have computed the actual time A_i left the gate at the first airport it

visited during $[T_y, T_z]$. Now, this time was computed based on the assumption of an on-time departure at the first airport A_i visited during $[T_x, T_y]$ which may not be correct. Nevertheless, the simulator for $[T_y, T_z]$ compares the time computed by the simulator for $[T_x, T_y]$ that A_i departed from the first airport it visited in $[T_y, T_z]$ with the on-time departure it had assumed. If A_i did in fact depart on time, A_i was simulated correctly in the first round so nothing more needs to be done. If A_i departed late, then the trajectory of T_i (and possibly other aircraft) during $[T_y, T_z]$ needs to be fixed up or re-simulated. If there is no differences in event ordering, the fix up computation can correct the simulation, however, if the correction involves reordering events, the initial simulation must be discarded and repeated using the corrected event ordering.

At the end of round 2, the simulator for the previous time segment $[T_x, T_y]$ will have re-computed a new time A_i left the gate for the first airport it visited during $[T_y, T_z]$. Here, another round of fix up computation may be needed. If this time matches what it had reported to the simulator of $[T_y, T_z]$ after the first round, nothing needs to be done. However, if it does not match, the trajectory of A_i during $[T_y, T_z]$ needs to be re-computed again. The above process repeats until no mismatches occur. Once this becomes correct for each aircraft, for each time segment, the time parallel simulation is done.

4.2 Fix Up Computation

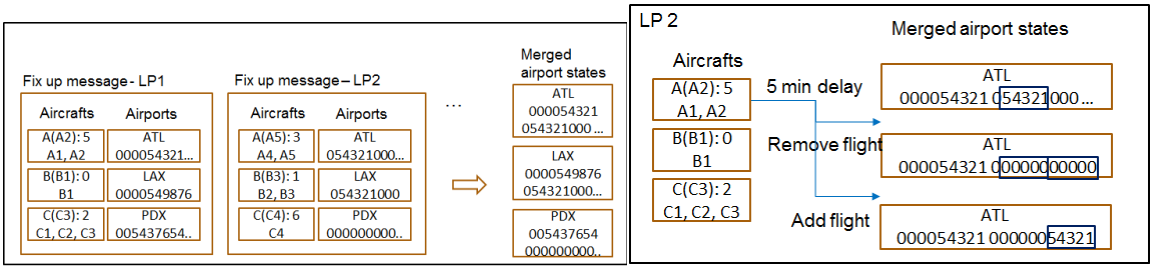
A more detailed description of the fix up computation, that occurs at the end of each round of simulation is described next.

4.2.1 Update and Communication Strategies

At the end of the simulation run, each LP constructs a fix up message. This fix up message consists of two components. The first component is the changed schedules for the last flights flown in the LP for each aircraft. All of the LPs maintain a list of the aircraft flown throughout the day, so this is simply an array of delay times (T_{delay}). More specifically, each delay time means the time difference($T_{delay} = T_{act} - T_{org}$) between the original scheduled arrival times(T_{org}) for the flights and the actual arrival times(T_{act}) for the flights after the simulation run. The other component is the queueing status of each airport at every time stamp. This means the wait time when a new flight arrives to the airport at that specific time point. In other words, based on the queueing status, we know how much time each aircraft needs to wait. These two fix up messages are shown in Figure 3(a). In this specific notional example, there are three different aircraft in the system and each LP is simulating 9 minutes each. They are used for several different flights and the fix up message holds the last flights' delay for each LP. In the figure, each box indicates *'Aircraft(Last flight number in the LP) : delay'*. For instance, aircraft A's last flight in LP 1 was delayed 5 minutes and aircraft B's last flight in LP 1 has no additional delay. In case of the airport queueing status, LP 1 has a sequence of numbers represents the delay time in the runway queue at a specific time for each airport. In this example, the aircraft A is supposed to arrive five minutes later from the beginning time of LP 1. If we assume that the landing takes 5 minutes, the expected waiting time will be five, four, three, two and one for the next five minutes. Here, the next step is iterating through the updated aircraft in the LP which have different arrival times from the previous LP. When it goes through the aircraft's flights, it removes flights from the queue at the original schedule spot and add it to the new spot in the queue. As a result of the evaluation, one can determine whether a new simulation run is necessary in the specific LP or not. If all the changes are simply updating the arrival times and do not propagate to the next flight, the LP does not need to re-compute the simulation. On the other hand, if the evaluation results in a change to the upcoming flights, another round of simulation is run based on the updated flight times. Depending on the algorithms, the evaluation of updates can be done sequentially or collectively. This means that the fix up messages are either sent to the next LP or collected into the first LP.

This modification of queue status is illustrated in Figure 3(b). The aircraft A's update message from LP 1 to LP 2 was that 5 minutes delay was added to the original scheduled arrival. Therefore, LP 2 does the evaluation step by firstly removing the flights at the original place which is the second time spot in

LP 2's status. This removes all expected delays for five minutes. Then, it adds the flight into the delayed time slot which is five minutes later. The expected delay was zero for that time slot but it has 5 minutes expected delay at that time point after evaluation.

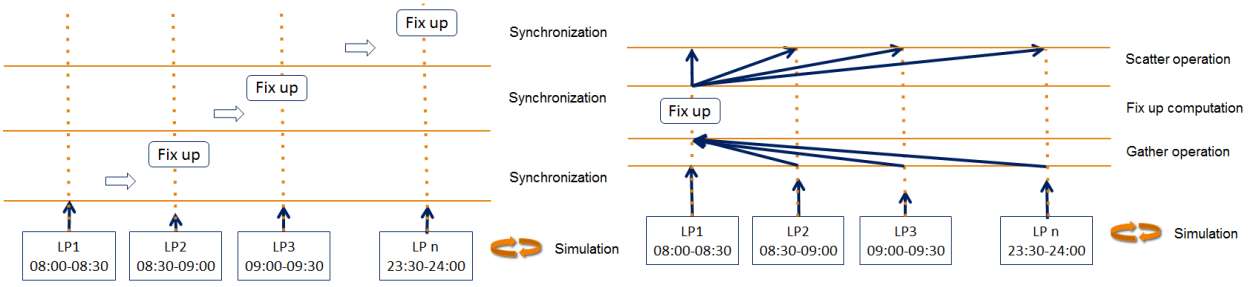


(a) Fix up message (b) Evaluation of fix up message

Figure 3: Evaluation of update for the fix up computation.

A high level view of the two different communication strategies is illustrated in Figure 4. Figure 4(a) shows a fix up computation that is accomplished separately by a sequential communication that propagates corrections in earlier time intervals(LP_i) to later ones(LP_{i+1}). This approach assures that all communications among the LPs happen only once. Based on the fix up message received from the previous LP, it might or might not need to do simulation run again. However, the simulation results should be final at every LP, so it avoids unnecessary rounds of fix up computations. On the other hand, because this fix up computation is sequential, there could be performance degradation especially if there are a large number of LPs.

Figure 4(b) shows the collective computation of the output variable by gathering all the results into the first logical process. This utilizes the collective API from MPI such as *MPI_Gather* and *MPI_Scatter*. By utilizing this collective approach, we can reduce the number of communications to a constant number instead of the number proportional to the number of LPs. Here, two rounds of communications are required for one fix up computation. Because general implementations of the collective APIs of MPI roughly has a complexity of $O(\log n)$, it could reduce the burden of communications significantly. After collecting all the updated information into one LP, it can quickly evaluate all the states need to be fixed. In the next part of the section, the criteria for rerunning the simulation after the fix up computations are described.



(a) Sequential fix up computation (b) Collective fix up computation

Figure 4: High level view of the fix up computations.

4.2.2 Rerunning the Simulation

When the updated events from the LP processing the previous time segment, i.e., the *preceding LP*, have no impact on the subsequent event scheduled by the updated event, we can simply update state variables affected by the changed event in order to ensure computed statistics are correct. This is when the updated next departure time based on the queue evaluation is still the same as the original scheduled departure

time. This is computed by Equation 1. In this case the fix up computation will not propagate beyond the LP and the update will remain within this LP.

If the updated events affect the scheduling of the subsequent departed events within an LP, another round of the simulation is needed to correct the differences. Based on the updated arrival time, all the affected time pieces are running the simulation again. At the end of the simulation, another round of fix up computation is performed for the evaluation. The explained procedure is presented in Algorithm 1.

Algorithm 1 Fix up computation

```
1:  procedure COMMUNICATEFIXUPMSG
2:      Msg.flights ← flight_result
3:      Msg.queue_state ← airport_state
4:      SendAndReceive(Msg)

5:  procedure EVALUATEUPDATE
6:      queue_states ← Msg.queue_state
7:      for flight_update ∈ Msg.flights do
8:          UpdateQueue(flight_update)
9:          if IsDepChanged(flight_update) then
10:             isRerunNeeded ← TRUE

11: procedure ISDEPCHANGED(offset)
12:     if offset + G^d − b^sg > 0 then
13:         return TRUE
14: procedure POSTFIXUP
15:     isRerunNeeded ← FALSE
16:     CommunicateFixupMsg()
17:     EvaluateUpdate()
18:     return isRerunNeeded
```

4.3 Workload Distribution

It is important to balance the computational workload across the different LPs because the slowest LP will dominate the execution time. In order to achieve this, we may partition the time domain so that all the LPs have a similar amount of traffic to model. In order to measure how this balancing affects the efficiency of the time-parallel simulation, two workload distribution algorithms were tested. The initialization portion of the algorithm is presented in Algorithm 2. Based on the options which are "SAME_TIME" and "SAME_TRAFFIC", the workload is distributed across the LPs.

Algorithm 2 Initialization

```
1:  procedure DISTRIBUTE(option)
2:    if option = SAME_TIME then
3:      duration ← total_time/num_lps
4:      start_time ← lpID × duration
5:      end_time ← start_time + duration
6:    else if option = SAME_TRAFFIC then
7:      lp_traffic ← total_traffic/num_lps
8:      time ← 0
9:      while traffic < lpID × lp_traffic do
10:       traffic ← traffic + traffic_at_time
11:       time ← time + 1
12:     start_time ← time
13:     while traffic < (lpID + 1) × lp_traffic do
14:       traffic ← traffic + traffic_at_time
15:       time ← time + 1
16:     end_time ← time

17: procedure DOINITIALIZE()
18:   schedule ← LoadData(start_time, end_time)

19: procedure STARTSIMULATION()
20:   Distribute(option)
21:   DoInitialize()
22:   isRerunSimulation ← TRUE
23:   while isRerunSimulation do
24:     Barrier()
25:     executiveHandle ← StartExecutive()
26:     while executiveHandle.IsRunning() do
27:       Wait()
28:     Barrier()
29:     isRerunSimulation ← PostFixup()
30:   return
```

5 EXPERIMENTAL RESULTS

In order to investigate performance and various algorithm alternatives, the time-parallel simulator algorithm was implemented. The main focus of these experiments was to evaluate the initial workload distribution approaches as well as the variations on the fix up computation algorithms in addition to evaluating the overall speed up obtained by the time-parallel simulation of the NAS.

To partially verify the simulation, computed results were compared with a perfect schedule of historical data. When the departure schedules of all the flights from one specific date are given, the simulation model runs all the traffic during the day in the absence of airport capacity limitations. This scenario should produce simulation results that exactly match scheduled arrival times. In the validation test, the developed simulation model yielded the correct results except in some instances where inconsistencies in the historical data led to differences in model predictions.

5.1 Air Traffic Scenario and Data

To verify the speed up of the time-parallel simulation algorithm under realistic test conditions, historical, real world air traffic data was utilized. *TranStats* from the *Bureau of Transportation Statistics* is a large transportation database maintained by the U.S. Department of Transportation (Bureau of transports statistics, DOT 2017). For this experiment, air traffic data for one day was selected which includes 16,614 flights across 297 airports. As a first step, it is assumed that there is no major delay in the NAS. It will be expanded with the more complex delay patterns which has inclement weather conditions and high traffic volume.

5.2 Experimentation Environment

5.2.1 System Configuration

For the experimentation, a parallel machine using Intel's Xeon® CPU (E5-2699 v4, 2.20 GHz, Broadwell microarchitecture) was utilized. It has two sockets and each socket has 22 processing cores. Therefore, up to 44 physical cores were utilized in the experiments. Also, each socket has 64 GB DDR4 memory installed so in total 128 GB of memory. Because all the experiments are performed in a single node, there is no network connectivity for the experiments. Each thread corresponds to an LP and executes on one physical core. To maximize cache hit rates, each physical core executes only one LP during the simulation run.

C++11 standards and libraries are utilized for the implementation on top of CentOS v7.2. Even though this experiment is performed in a single machine configuration, it has been implemented using *MPICH* to enable later extension to a multi-machine clustering environment. The version of the *MPICH* used in this implementation is v3.2 and the version of *gcc* used is v4.8.5.

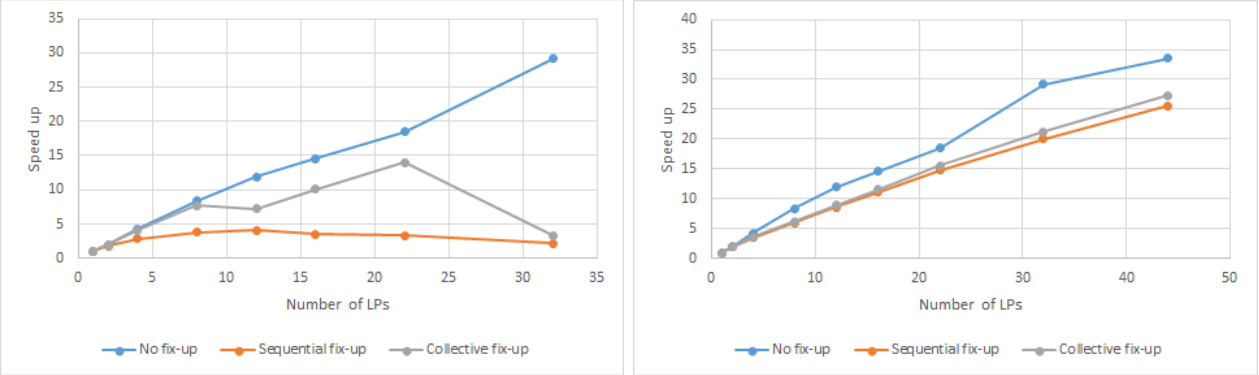
5.3 Fix Up Computation Comparison

Two different fix up computation algorithms described in Section 4.2 were implemented and tested. In these experiments, the wall clock time required for each LP was measured and the slowest time is used in computing speed ups. Figure 5 (a) shows the speed up as the number of LPs was varied for the two different fix up computation algorithms. As seen from the graph, the collective fix up computation algorithm executes faster than the sequential fix up computation. This can be explained by the communication overheads in the fix up computation. There should be *n-1* communications among all LPs which results in *O(n)* communication complexity. On the other hand, all the collective operations used in this implementation have *O(log n)* communication complexity.

One further observation is that the parallel efficiency drops significantly after a certain number of LPs are reached. These results are compared with the speed up when no fix up computation is performed, as shown in Figure 5 (a). With no fix up computation, the simulation, of course, produces incorrect results, however, this simulation provides an upper bound on performance and illustrates the performance degradation that

results from the fix up computation. As expected, the speed up without fix up computations yields almost ideal, linear speed up, and sometimes even shows super linear speed up. We believe this is because more cache memory is available as the number of processors (LPs) increases.

Returning to the time-parallel simulation, We hypothesize that the performance degradation as the number of processors is increased is because the simulation computation models a single day of traffic, so the amount of computation in each LP decreases as more LPs are added. For a large number of LPs the fix up computation requires additional communication, incurs a more significant overhead. This explains the performance degradation in the time-parallel simulation as the number of LPs becomes large. In this experiment, the policy which distributes the same amount of traffic is used.



(a) Small workload (b) Large workload

Figure 5: Comparison of fix up computation algorithm.

In order to verify that the degradation of parallel efficiency is because of the small size of the original computations, another experiment was performed that included artificially enlarged event computations. This is done by adding a spin-loop in each event computation. Figure 5 (b) shows the result of this experiment. In this experiment, both the sequential fix up and collective fix up computations show good speed ups. The collective fix up computation shows slightly better performance. These speed ups can also be compared with the ideal speed up without fix up computation. All of these show significant speed ups demonstrating the potential performance improvement of the time-parallel algorithm.

5.4 Workload Distribution Comparison

For another experiment, the different initial workload distribution policies described in 4.3 were used in the simulation. In both cases, they scale well as shown in Figure 6. And, the case with the same amount of traffic shows better performance than the case with the same time distribution. This result shows that it is important to evenly distribute workloads across the LPs. In the case of 44 LPs, 1.65 times more speed up can be acquired by distributing workload uniformly. For this experiment the collective fix up computation is used.

6 CONCLUSION

Time-parallel simulation offers a new approach to accelerating air traffic simulations of the NAS. The schedule-driven nature and other aspects of the air transportation schedules such as the inclusion of buffer times makes it an application that appears to be well suited for the time-parallel simulation approach. Therefore, a time-parallel simulation algorithm for simulating the NAS was proposed. Experimental results illustrate that this algorithm can achieve high level of parallelism and speed up for this application. Preliminary measurements indicate that the collective fix up computation yields better performance than the sequential fix up computation. It is also seen that even distribution of workloads across the LPs is

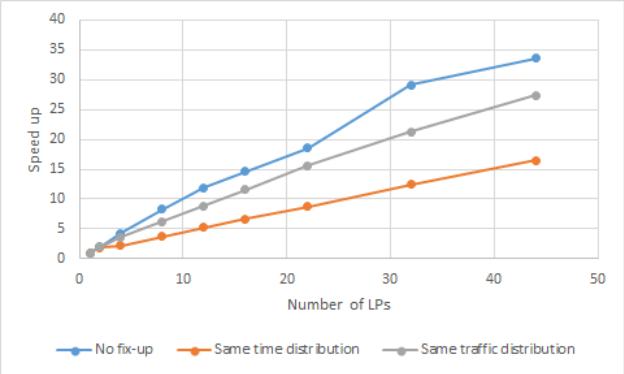


Figure 6: Same time intervals vs. Same amount of traffics.

necessary for efficient parallel simulation. These results suggest that time-parallel simulation offers a viable approach to accelerating certain air traffic simulations.

There are several open avenues for future research. Certain algorithmic improvements of the simulation method may yield additional performance enhancements. Approaches using both space and time-parallel simulation offer another area of research, as is realization exploiting SIMD architectures. Further, a shared memory model such as OpenMP can be integrated with the MPI based simulator and may achieve better speed up in many core systems; there are many physical cores in a single machine so it will be more efficient to exploit shared memory among LPs where possible, and rely on MPI for communication between LPs mapped to different machines. Finally, the simulations could be adopted for use in real-time symbiotic simulation applications.

REFERENCES

Aerospace Forecast, FAA 2016. "Forecast 2016–2036". https://www.faa.gov/data_research/aviation.

Ball, M., C. Barnhart, M. Dresner, M. Hansen, K. Neels, A. Odoni, E. Peterson, L. Sherry, A. A. Trani, and B. Zou. 2010. "Total Delay Impact Study: a Comprehensive Assessment of the Costs and Impacts of Flight Delay in the United States". Technical Report 01219967, UC Berkeley Transportation Library, Berkeley, California.

Bureau of transports statistics, DOT 2017. "Research and Innovative Technology Administration (RITA)/Transtats". http://www.transtats.bts.gov.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems*, Volume 300. Hoboken, New Jersey: Wiley New York.

Grasedyck, L., C. Löbbert, G. Wittum, A. Nägel, V. Schulz, M. Siebenborn, R. Krause, P. Benedusi, U. Küster, and B. Dick. 2016. "Space and Time Parallel Multigrid for Optimization and Uncertainty Quantification in PDE Simulations". In *Software for Exascale Computing-SPPEXA 2013-2015*, edited by H. Bungartz, P. Neumann, and W. Nagel, 507–523. New York, New York: Springer.

Greenberg, A. G., B. D. Lubachevsky, and I. Mitrani. 1991. "Algorithms for Unboundedly Parallel Simulations". *ACM Transactions on Computer Systems (TOCS)* 9 (3): 201–221.

Heidelberger, P., and H. S. Stone. 1990. "Parallel Trace-Driven Cache Simulation by Time Partitioning". In *Proceedings of the 1990 Winter Simulation Conference*, edited by O. Balci, R. P. Sadowski, and R. E. Nance, 734–737. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hybinette, M., and R. M. Fujimoto. 2001. "Cloning Parallel Simulations". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11 (4): 378–407.

Kiesling, T., and S. Pohl. 2004. "Time-Parallel Simulation with Approximative State Matching". In *Proceedings of the eighteenth workshop on Parallel and distributed simulation*, 195–202. New York, New York: Association for Computing Machinery.

Kim, Y. J., O. J. Pinon-Fischer, and D. N. Mavris. 2015. "Parallel Simulation of Agent-Based Model for Air Traffic Network". In *AIAA Modeling and Simulation Technologies Conference*, 1–14. Reston, Virginia.

Lee, S., A. Pritchett, and D. Goldsman. 2001. "Hybrid Agent-Based Simulation for Analyzing the National Airspace System". In *Proceedings of the 2001 Winter Simulation Conference*, edited by B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 1029–1036. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Long, D., D. Lee, J. Johnson, E. Gaier, and P. Kostiuk. 1999. "Modeling Air Traffic Management Technologies with a Queuing Network Model of the National Airspace System". Technical Report NAS2-14361, NASA, Hampton, Virginia.

Pinon, O. J. 2012. *A Methodology for the Valuation and Selection of Adaptable Technology Portfolios and Its Application to Small and Medium Airports*. Ph.D. thesis, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, Georgia. Available via http://hdl.handle.net/1853/43632.

Thi, T. H. D., J.-M. Fourneau, and F. Quessette. 2014. "Time-Parallel Simulation for Stochastic Automata Networks and Stochastic Process Algebra". In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, edited by B. Sericola, M. Telek, and G. Horvth, 140–154. New York, New York: Springer.

Wang, J. J., and M. Abrams. 1992. "Approximate Time-Parallel Simulation of Queueing Systems with Losses". In *Proceedings of the 1992 Winter Simulation Conference*, edited by J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 700–708. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wieland, F. 1998. "Parallel Simulation for Aviation Applications". In *Proceedings of the 1998 Winter Simulation Conference*, edited by D. Medeiros, E. Watson, J. Carson, and M. Manivannan, 1191–1198. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wong, J.-T., and S.-C. Tsai. 2012. "A Survival Model for Flight Delay Propagation". *Journal of Air Transport Management* 23:5–11.

AUTHOR BIOGRAPHIES

YOUNG JIN KIM is a research scientist in artificial intelligence at Intel Corporation, USA. He is also a Ph.D. candidate in Computational Science and Engineering at the Georgia Institute of Technology, Atlanta, GA. His e-mail address is young.jin.kim@intel.com.

DIMITRI N. MAVRIS is a Regents' Professor of Aerospace Engineering at the Georgia Institute of Technology, Atlanta, GA, and director of the Aerospace Systems Design Laboratory. Dimitri Mavris received his B.S., M.S. and Ph.D. in Aerospace Engineering from the Georgia Institute of Technology. His primary areas of research interest include: advanced design methods, multi-disciplinary analysis, design and optimization, system of systems, and non-deterministic design theory. His email address is dimitri.mavris@aerospace.gatech.edu.

RICHARD M. FUJIMOTO is a Regents' Professor in the School of Computational Science and Engineering at the Georgia Institute of Technology, Atlanta, GA. He received the Ph.D. and M.S. degrees from the University of California-Berkeley in 1983 and 1980 in Computer Science and Electrical Engineering and B.S. degrees from the University of Illinois-Urbana in 1977 and 1978 in Computer Science and Computer Engineering. Fujimoto's research is concerned with the execution of discrete-event simulation programs on parallel and distributed computing platforms. His email address is fujimoto@cc.gatech.edu.