

## ON USING SPEEDES AS A PLATFORM FOR A PARALLEL SWARM SIMULATION

Matthew A. Russell  
Gary B. Lamont  
Kenneth Melendez

Department of Electrical and Computer Engineering  
Graduate School of Engineering & Management  
Air Force Institute of Technology  
WPAFB (Dayton), OH 45433-7765, U.S.A.

### ABSTRACT

Unmanned Aerial Vehicle (UAV) research is an increasingly important pillar of national security and military interest. A high fidelity discrete event simulation is prerequisite to any systems implementation. The Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) is a versatile and powerful tool that can be used for realization of this objective. A suite of five experiments measures the efficiency a parallel UAV swarming SPEEDES application. Results indicate that the conservative time management produces more than twice the speedup as optimistic time management.

### 1 INTRODUCTION

Militaries around the world are becoming increasingly interested in using Unmanned Aerial Vehicles (UAVs) to accomplish objectives that are more dangerous and expensive to do with conventional aircraft. As developments in UAVs unfold, the capability of building smaller and smaller UAVs is becoming a reality. Miniaturization of UAVs opens a wide variety of possible applications. The ability to deploy large numbers (swarms) of inexpensive UAVs for classical military missions of target detection and destruction will soon arrive. The desirability of UAV swarms achieving their mission in a hostile environment with the loss of potentially many of the individual UAVs requires an investigation of the nature of swarm behavior. A broad goal of this research is to develop models of UAV swarm behavior that ultimately promote mission accomplishment by a UAV swarm. An initial approach to meeting this goal is simulation. Simulating such applications with as much fidelity as possible is prerequisite to physical implementation.

### 2 BACKGROUND

Tools and techniques used in this investigation are discussed in this section.

#### 2.1 Parallel Discrete Event Simulation

Parallel algorithm design and data decomposition strategies can be applied to increase efficiency and provide effectiveness that could not otherwise be achieved. Dividing a computational task into smaller pieces that can be scheduled to run concurrently on multiple processors is the key when designing parallel algorithms and simulations.

Two of the central lynchpins in realistic simulation include a realistic simulation model and an efficient simulation implementation. A realistic simulation model accounts for the necessary constraints in the problem application, and the efficient simulation implementation provides a way to analyze and improve the interactions with the simulation. Also For many high fidelity simulations such as UAV routing, it is desirable for the simulation to be efficient enough for real-time visualization.

The Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) (Metron 2003) is a versatile and powerful tool that can be used to fill both the need for a realistic simulation and a efficient simulation. SPEEDES object-oriented modeling capability permits the encapsulation of UAVs as objects which models optimistic and conservative techniques.

#### 2.2 Time Management

Central to Parallel Discrete Event Simulation (PDES) is the local causality constraint (LCS), which requires that simulation events may only be processed in non-decreasing time stamp order. Enforcing the LCS can be accomplished using

either conservative time management (CTM) or optimistic time management (OTM) techniques.

CTM fulfills the LCS by processing events only when there is no chance of a synchronization violation. In other words, events containing a time stamp marked  $t$  are handled by a process  $p$  only when it is certain that  $p$  cannot receive any other event with time stamp less than  $t$ . The prevalent shortcoming of CTM is its inability to fully exploit the concurrency available in the simulation application. Common techniques for CTM include First-In-First-Out Queues, the Chandy/Misra/Bryand Algorithm, and the Demand Driven Approach.

OTM processes events in the order they are received, but ensures fulfillment of the LCS by “rolling back” any events that turn out to violate the LCS. For example, when a simulation object receives a message with a time stamp  $t_1$  marked as less than one already processed  $t_2$ , a rollback restores the simulation object’s state to the time stamp  $t_1$ . Simply rolling back the process to a state at a previous time does nothing to nullify cascaded messages, so techniques such as Jefferson’s Time Warp elegantly handles the situation by having each object that was “rolled back” send anti-messages. Anti-messages cascade if necessary. Other common OTM techniques include Breathing Time Buckets and Breathing Time Warp (Metron 2003).

### 2.3 Parallel Computing Metrics

At the core of parallel computing is *speedup*. Speedup can be described as the ratio of the serial time  $S$  of the best sequential algorithm for solving a problem to the time  $P$  taken by a parallel algorithm to solve the same problem on  $p > 1$  processors and is given in Equation 1. Amdahl, Gustafson-Barsis, and Sun-Ni give different conjectures but related insights into speedup.

$$Speedup = \frac{S}{P} \quad (1)$$

Amdahl makes the case for fixed size speedup (Grama et al. 2003). This conjecture states that speedup saturates and efficiency drops as a consequence of holding the problem size constant and increasing the number of processors. In other words, it implies that diminishing returns occur for speedup because of increasing communication cost. The equation for Amdahl’s conjecture is given by Equation 2, where  $P$  represents the parallel processing time, and  $S$  represents the best possible serial processing time.

$$Speedup = \frac{1}{1 - P + \frac{P}{S}} \quad (2)$$

Gustafson-Barsis’ conjecture (Grama et al. 2003) circumvents Amdahl’s by increasing the problem size as the

number of processors increase. It exploits the ratio between the portion of serial code in the problem and the problem size. For Gustafson’s conjecture to hold, it requires the serial portion of the workload stay constant and only the parallel portion of the workload to increase as we scale up the problem and add machines. In this formulation, the number of processors is defined by  $N$ , and  $R_s$  represents the portion of code that is serial.

$$Speedup = N - (N - 1)R_s \quad (3)$$

Sun-Ni’s conjecture is a generalization of the two previous ones and makes the case for memory bounded speedup. It states that making all data accessible by all processors, using all available memory space, and increasing memory space as needed results in nearly ideal speedup (Sun and Ni 1993). Without a shared memory system this metric is not possible to exploit.

## 3 MODELING

This section outlines the architecture of the parallel swarm model of UAVs and discusses its application to routing swarms of UAVs.

### 3.1 Simulation Architecture

To design something is to conceive or systematically fashion it in the mind. Design is a creative and non-algorithmic process, and a good one produces a precise model. A model is an encapsulation of some slice of the real world within the confines of the relationships constituting a formal mathematical system (Casti 1997). Because any given element of reality cannot ever be precisely modeled, various constraints are an essential part of the formal system because they define the boundaries for which the model holds. A "good" model characterizes a system with a desired degree of accuracy, and thus, enables accurate predictions of the natural system in consideration to be made. The hardware and software platforms as well as the mathematical swarm model selected have a strong influence on the design of the simulation. The selection of SPEEDES as the software platform enables an object oriented approach and provides the ability to build a parallel simulation. The target hardware platform, a homogeneous Linux based cluster of processors, influences the approach to distributing the processing tasks.

The existing AFIT Swarm Simulator is the logical fruition of Kadrovach’s (Kadrovach 2003) swarm model with parallel extensions by Corner (Corner 2004)(Corner and Lamont 2004). Kadrovach’s work produced an implementation of an unscalable swarm model that uses a global data structure and is designed for a serial processor. A benefit of this model is that it does accurately simulate the

flocking behavior of a swarm and can be used in applications that require analysis of sensor coverage within the swarm.

### 3.2 Parallel Swarm of UAVs Simulation

The simulation model is based on an object-oriented design. In the model a swarm consists of a collection of UAVs where each UAV is represented as a SPEEDES object. During a simulation run, the UAV objects are distributed uniformly across the processing nodes in the assigned cluster. Conceptually, each UAV should have minimal communication and visibility to the state of the other UAVs in the cluster, but yet it must have sufficient interaction with fellow swarm UAVs in order to maintain proper separation, alignment, and cohesion. In a single processor simulation global data structures can be used to provide total visibility to state variables among the UAV, however, in a cluster environment exchanging state information among all UAVs yield a high network communication cost. Thus, minimizing interprocessor communication has been a significant challenge in the design of the simulation software. Once a UAV object is assigned to a processor in the cluster it continues to be resident on that processor throughout the simulation. Dynamic load balancing (Jiang et al. 1994) (Gan et al. 2000) can reduce interprocessor communication, however, a redesign of the simulation model is required.

### 3.3 Routing Swarms of UAVs

Originally the swarm model as developed by Kadrovach (Kadrovach 2003) and parallelized by Corner (Corner 2004) provided a concept of targets that repelled or attracted the swarm, however, the swarm basically wandered in a pseudo-random fashion and was not routable. In the current AFIT Swarm Simulator, a route is modeled as an object and each UAV object is issued a copy of the route. A route consists of a sequence of waypoints (coordinate locations), which the swarm must visit. Based on the next waypoint the swarm must visit, a vector component is added to each UAV's movement vector, which moves the swarm in the direction of the waypoint. This vector is added after swarm movement behavior has been applied to each UAV.

The route of waypoints is input to the swarm model. In his research Russell addresses the problem of routing and simulating swarms of UAVs (Russell 2005). Sorties are modeled as instantiations of the NP-Complete Vehicle Routing problem. His work uses genetic algorithms to provide a fast and robust algorithm for a priori and dynamic routing applications of the UAV swarm.

## 4 EXPERIMENTATION

Five experiments form a suite designed for analyzing the performance of a parallel swarm simulation employing

Kadrovach's swarm model (Kadrovach 2003), parallelized by Corner (Corner and Lamont 2004). The primary objective of these experiments is to measure and improve the parallel simulation's current performance, to better understand the anomaly from Corner's results showing no speedup (Corner 2004), and to improve the simulation's performance for this particular application.

The simulation is implemented using a Beowulf cluster of 32 nodes running Red Hat Linux (Shrike) with 2.2GHz AMD Opteron processors with 4GB of memory and a fast ethernet backplane. SPEEDES is the discrete-event simulation platform.

### 4.1 Experiment 0

*Objective:* To visualize the 3-dimensional response surface for runtimes and speedup by varying UAV sizes and CPU nodes. The number of data points required to obtain a relatively smooth surface and the corresponding runtimes constrained the number of replications of the experiment, hence, three replications of 100 simulation time units were used per data point.

### 4.2 Experiment 1

*Objective:* To measure runtimes and calculate speedup and efficiency metrics for small swarms,  $|SWARM| \in \{32, 64, 128, 256, 512\}$ , on varying numbers of processors,  $P \in \{1, 2, 4, 8, 16, 32\}$  using *optimistic* event processing for 12 simulation time units.

This experiment is designed to reproduce data from Corner's research (Corner 2004) showing a lack of speedup for even small swarm sizes on various CPU configurations using OTM with SPEEDES, and provides a basis and direction for all other experiments. By analyzing results for small swarm sizes using the standard parallel computing metrics, some insight may be gleaned for how larger swarms perform using OTM.

### 4.3 Experiment 2

*Objective:* To compare runtimes between conservative and optimistic techniques using SPEEDES for various swarm sizes,  $|SWARM| \in \{20, 40, 60, \dots, 160, 180, 200\}$ , on a single processor.

Previous research (Corner 2004) implicitly hypothesized that the Breathing Time Warp Algorithm results in a more efficient simulation. Research results, however, did not confirm this hypothesis. In fact, results show that there are more than 1.6 million rollbacks and the ratio of simulation time units to wall clock time varied between ratios of 1:1000 and 1:2000 for a simulation involving only 500 swarm members for 12 simulation time units. This experiment is designed to validate that there are no significant efficiency

differences between OTM and CTM for runs on a *single processor*, and thus, measure the overhead of OTM in SPEEDES for this application. Theoretically, there should not be a significant difference because no rollbacks should occur for the OTM.

#### 4.4 Experiment 3

*Objective:* To compare runtimes for CTM using varying lookahead window values  $W \in \{0, 3, 4, 5, 6, 7\}$  with SPEEDES for various swarm sizes,  $|SWARM| \in \{50, 55, 60, \dots, 95, 100\}$  on 5 processors.

This experiment follows a procedure similar to Experiment 2, except that the runs are accomplished for a parallel configuration and for various lookahead windows. Results from this experiment are expected to provide insight into the sensitivity of the lookahead value for this particular application and show any gain in speedup realized by using a lookahead window.

#### 4.5 Experiment 4

*Objective:* To compare the difference in CTM and OTM with SPEEDES using variable CPUs and swarm sizes.

This experiment is the capstone of the suite. It provides data necessary to calculate the standard metrics for CTM with SPEEDES and compares it with the OTM results from Experiment 2. A specific comparison is made between results from CTM using a particular lookahead window value and results from OTM using BTB.

### 5 EXPERIMENTAL RESULTS ANALYSIS

Data from the experimental suite is analyzed using statistics and visualized using tables and graphs.

#### 5.1 Experiment 0

All simulations in this experiment were performed using OTM. This experiment used UAV swarm sizes ranging from 32 to 200 and CPU sizes ranging in number from 2 to 30. The objective of this experiment was to understand the overall parallel behavior and scalability, so the experiment begins with two CPUs as the baseline. Figure 1 shows the response surface for runtimes. Note that for a fixed number of UAVs the runtime first increases as more CPUs are added. When a sufficient number of CPU have been added, the runtimes begin to decrease and continue until the minimum is reached at which point there is a gradual increase in runtime. Observe the valley that is formed by the minimums. With the exception of the curvature associated with CPU size of 2, curvature evident at the other edges of the graphs are false artifacts due to smoothing. Figure 2 depicts speedup. The graphs show that in the range of 32

to 150 speedup increases sharply only for CPU sizes in the range of 5 to 20. Outside of that range speedup is minimal.

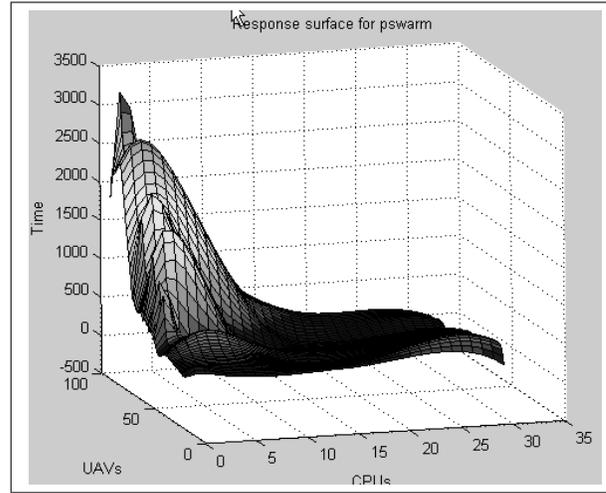


Figure 1: Response Surface for Simulation Run Times Using OTM

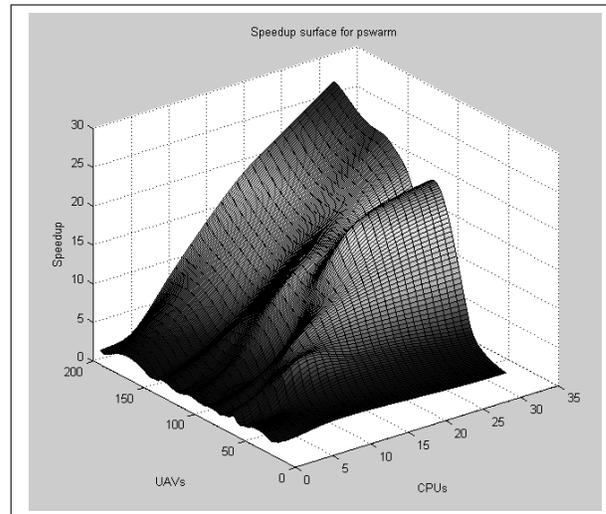


Figure 2: Response Surface for Simulation Speedup Using OTM

Minimum simulation execution time can be achieved by selecting the number of CPUs corresponding to the UAV swarm size by selecting points in the valley shown in the graph in Figure 1. Given a fixed UAV size for the swarm simulation, good parallel speedup can be achieved by selecting the CPU numbers using Figure 2. Scalability is an issue with the AFIT parallel swarm simulator. The two response surface graphs do not show with a UAV size greater than 200. Scaling the UAV sizes greater than 200 especially with a small number of CPU causes the simulation to terminate. No evidence is apparent indicating a scalability issue with respect to CPU sizes. However, the speedup graphs does indicate that there is a knee in

the surface at which increasing the number of CPU in the simulation yields diminishing returns.

## 5.2 Experiment 1

This experiment used UAV swarm sizes  $|SWARM| \in \{32, 64, 128, 256\}$  and CPU sizes that are powers of 2 (so that the UAVs per CPU could divide evenly) to measure performance of the BTB algorithm. Given previous results from Corner (Corner 2004), OTM for UAV sizes much larger than these is not possible because of the sheer number of rollbacks and long run times that occur via OTM for this application with SPEEDES.

Experimentation in this suite confirmed that the simulation chokes for swarm sizes of 512 or larger using OTM. This is likely because of the excessive memory required by SPEEDES for OTM due to the large number of rollbacks. Manually inspecting memory usage using standard system commands for CPUs running ratios of more than 32 swarm members per CPU reveals that eventually the entire node’s memory is used up by the parallel swarm and its proxy processes, resulting in an eventual node crash. Results can probably be obtained by running much smaller simulation durations, but it is unlikely that reliable steady state results can be obtained for much less than 12 simulation time units (Corner 2004). Even if results could be obtained for less than this steady state threshold, it would provide almost no benefit for simulation purposes; any realistic UAV simulation contains thousands of time increments.

Speedup and efficiency calculations for this experiment define the best serial runtimes as the ones using SPEEDES with a single CPU. It is crucial to note that Corner’s speedup calculations used serial run times from the original Linux port of Kadrovach’s swarm model. Corner was not able to produce any speedup using the best serial runtimes as the ones shown in Table 5.2. This conundrum can be explained by recalling that the Kadrovach’s swarm model is designed using a global data structure in which all swarm members communicated with one another, yielding  $O(|SWARM|^2)$  communications at each step. Although there may be hope for gaining speedup using a shared memory model with Kadrovach’s algorithm, empirical results show no indication that a parallelization of the serial model as SPEEDES application running on a Beowulf cluster can produce speedup.

When making speedup comparisons for a SPEEDES application, it is most appropriate to use results from the SPEEDES application running on a single CPU as the basis of calculation in order to achieve an ‘apples to apples’ comparison. Any SPEEDES application is subject to more overhead than a streamlined serial algorithm. As Corner’s results show for this particular application, the speedup calculations are paradoxical without this understanding. Comparing streamlined serial runtimes to parallel SPEEDES

Table 1: Results From Page 94 of Corner’s Research (Corner 2004), Citing the Best Serial Runtimes for Swarm Simulation. Parallel Runtimes Using the SPEEDES Library Do Not Even Begin to Approach These Runtimes Even in the Best Cases.

UAV Count	Execution Time (sec)
100	4
500	198
1000	1478

runs is not a fair comparison for measuring speedup and cannot be considered an ‘apples to apples’ comparison.

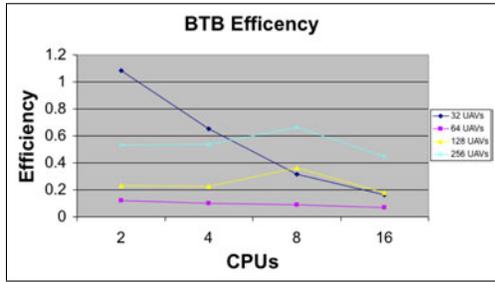
Results from this experiment indicate that for the given swarm configurations and for 12 simulation time units, the wall clock time can be decreased and speedup can be gained using BTB. Figure 3(c) shows the runtimes from a matrix of runs on various processors. Figures 3(b) and 3(a) show the speedup and efficiency trends.

Somewhat unexpected is the increase in the run times when transitioning from 1 to 2 CPUs. This is likely explained by the communication inefficiencies introduced because of the proxies (external modules) used by a SPEEDES application and is consistent with the trends found in Corner’s research (Corner 2004). Transitioning from 2 to 4 CPUs appears to relieve some of the administrative overhead, thereby decreasing some of the communication inefficiencies introduced and facilitating speedup.

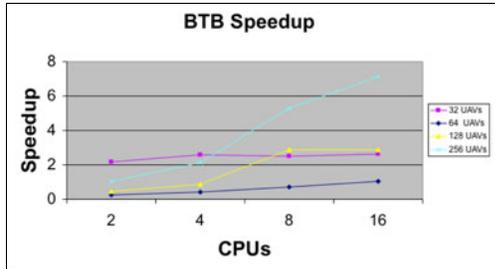
## 5.3 Experiment 2

This experiment measures the difference between using CTM and OTM differences for identical runs of a SPEEDES application on a single CPU. As could be expected, results show that there is some additional overhead for using OTM, but this overhead is small enough to be considered negligible. For the smallest swarm sizes  $|SWARM| \in \{20, 40\}$ , the percent difference is quite large, but only because of a constant overhead imposed by starting up and configuring the SPEEDES server in comparison to the very short runtime for such a small swarm size. Results from this experiment validate that there is not a significant overhead imposed by SPEEDES for OTM. Given OTM’s small overhead, it is likely that performance issues alluded to by Corner are *primarily* the result of excessive rollbacks. From “Experiment P2” in Corner’s research (Corner 2004):

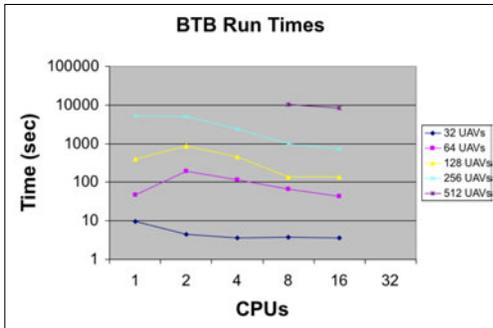
“Obvious eyesores are the millions of rollbacks that are occurring on a regular basis. This is probably due to the optimistic processing of future events that need rolled back because each UAV depends on the data that has already been



(a) Efficiency



(b) Speedup



(c) Run Times

Figure 3: Parallel Results: For a Small Swarm of UAVs, There is Eventually Only Negligible Gain for Parallelizing the Swarm Algorithm Using SPEEDES. For a Larger Swarm Size, However, There is Significant Gain, but Efficiency Quickly Becomes a Concern.

modified at a future simulation time, so that  $n$  UAVs are causing rollbacks on all other UAVs."

The final experiments in this suite explicitly measure and compare performance between OTM and CTM with SPEEDES.

### 5.4 Experiment 3

This experiment compares the results for various lookahead window values for CTM with SPEEDES to determine the general sensitivity to the lookahead value. Overall results are given in Figure 4 and indicate that using even small values for a lookahead window significantly decreases the

Table 2: Comparison of Wall Clock Times for Simulating Various Swarm Sizes on a Single CPU Using CTM and OTM with SPEEDES

SWARM	conservative	optimistic	% diff
20	3.77	5.23	27.91
40	13.35	15.67	14.81
60	39.17	40.58	3.47
80	86.73	88.84	2.37
100	170.79	176.37	3.16
120	312.03	326.49	4.42
140	534.98	551.74	3.03
160	860.05	886.45	2.98
180	1326.09	1382.55	4.08
200	1963.76	2053.95	4.39

Table 3: Explicit Runtimes from Various Swarm Sizes on 5 CPUs with Various Lookahead Values

SWARM	0.0	3.0	4.0	5.0	6.0	7.0
55	28.26	6.57	6.05	4.92	5.14	4.78
60	33.21	7.97	7.54	5.99	7.28	5.66
65	43.66	9.65	9.08	9.11	8.6	8.03
70	46.59	10.5	12.01	11.59	9.65	10.22
75	54.13	12.99	12.98	12.57	12.54	12.07
80	67.97	16.68	16.48	15.14	15.83	14.15
85	67.98	20.29	19.16	18.89	17.39	17.73
90	94.59	24.83	23	22.17	22.8	21.29
95	103.69	27.21	27.06	27.05	25.55	25.38
100	117.25	33.57	33.19	31.92	30.52	30.66

runtime for CTM compared to not using a lookahead value. Differences in lookahead values can be explicitly compared using the values from Table 3.

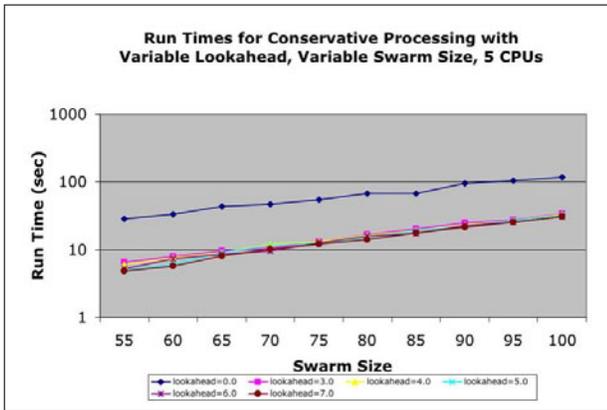
### 5.5 Experiment 4

Given that OTM does not impose a significant overhead when compared to CTM, and that results from OTM for a UAV swarming application with SPEEDES are less than impressive, an objective comparison is made between CTM and OTM for this particular SPEEDES application.

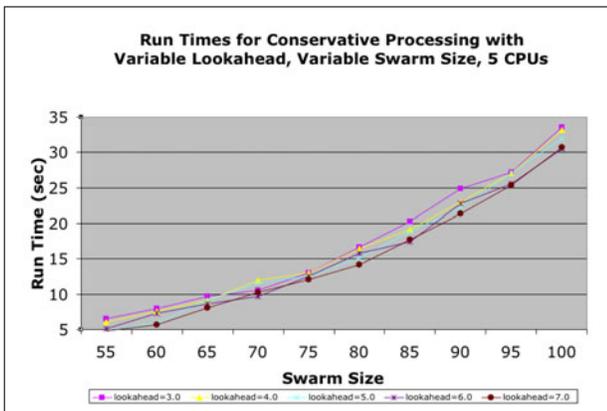
The runtimes, speedup, and efficiency graphs from a series of conservative runs comparable to the optimistic runs using BTB from Experiment 2 are given in Figure 7.

The metrics from the conservative runs as shown in this experiment exemplify the 'regular' behavior expected by adding additional processors. In particular, the runtimes decrease in all cases by the addition of more CPUs as show in Figure 7(c); there is observable speedup from the additional CPUs, and for these particular runs, the number of additional CPUs did not become great enough to begin leveling out the speedup curve shown in Figure 7(b); and the efficiency decreased within an acceptable tolerance that could have been expected.

In fact, the only abnormalities with the data collected using CTM are 'good' abnormalities—a slight superlinear speedup that can be observed for the largest swarm size of 512 UAVs. The superlinear speedup in this case is most



(a) All Conservative Runs on a Logarithmic Scale



(b) The Tightly Grouped Series of Runs on a Standard Linear Scale

Figure 4: Results From a Series of Runs Using 5 CPUs with Various Lookahead Window Values

likely explained by hardware features that put the serial implementation at a disadvantage (Grama et al. 2003). In this case, the limiting hardware feature is probably the memory limitations from processing 512 UAVs on a single CPU. Not given enough memory, the CPU would have had to write and read data from disk periodically, an operation that can take hundreds of times the duration from accessing memory in the worst case. Dividing the swarm size between 16 CPUs, however, eliminates this ‘problem’, thus resulting in a perceived superlinear speedup.

Regardless of philosophical arguments for OTM in PDES, the results from Experiments 1 and 4 demonstrate that that CTM produces superior results for this particular application. Consider the comparison of runtimes given in Figure 5.

## 6 CONCLUSION

Results from experiments suggest that a transition from OTM to CTM for this particular SPEEDES application

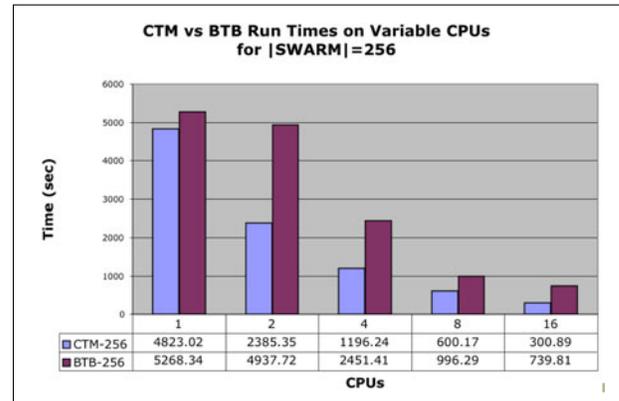


Figure 5: CTM Outperforms the OTM by Approximately a Factor of 2 for this Parallel Swarming Application

improves simulation runtimes in the general case by more than factor of 2 in addition to providing much more ‘regular’ behavior with regard to parallel computing analysis. Such improvement is a necessary step toward simulating swarms of UAVs in realtime.

Once the simulation is extended to provide a swarm model that can be routed, the routing algorithm can provide an initial way point sequence for simulating sorties, and the simulation can then be extended to include threats, dynamic changes to the way point schedule, etc. One very high level architecture for a more unified simulation is shown in Figure 6.

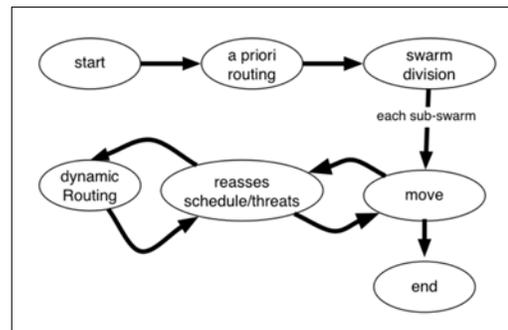
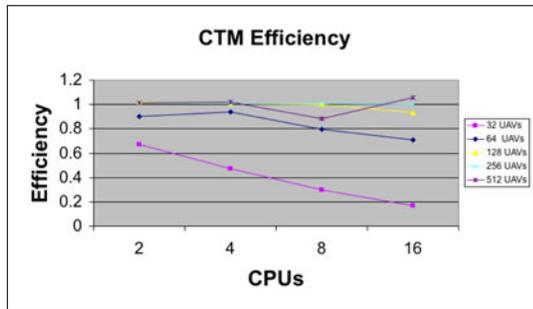
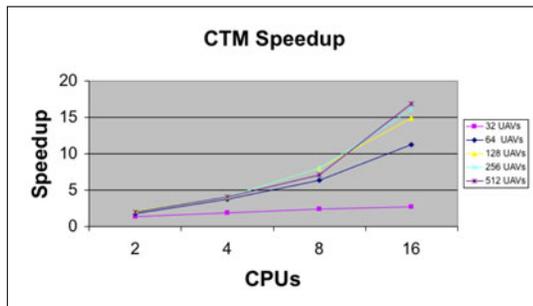


Figure 6: High Level Swarm Simulation Architecture: Capable *a Priori* and Dynamic Routing in Addition to Threat Detection and Other Constraints

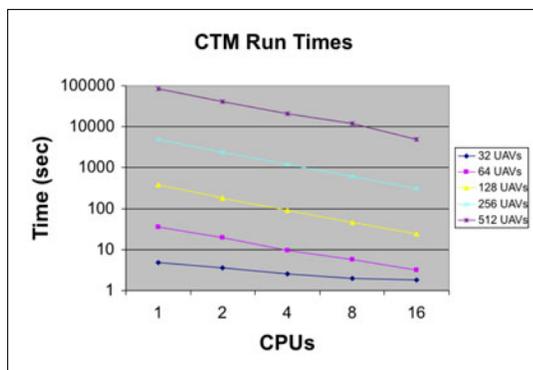
Analysis from various CPU and swarm configurations simulated as a SPEEDES application shows that SPEEDES applications exhibit ‘regular’ parallel behavior with regard to standard parallel metrics when run with CTM. Using OTM, on the other hand, produces some abnormalities with the introduction of additional CPUs. The results for a UAV swarming application suggest that CTM outperforms OTM by approximately a factor of 2.



(a) Efficiency



(b) Speedup



(c) Run Times

Figure 7: Parallel Results: For a Small Swarm of UAVs, There is Only Negligible Gain for Parallelizing the Swarm Algorithm. For a Larger Swarm Size, However, There is Significant Gain, but Efficiency Quickly Becomes of Predominant Concern.

## 7 FUTURE WORK

The scalability issues associated with swarm sizes larger than 200 need to be addressed. Data storage requirements for dynamic simulation state data and the communications between UAV objects need to be reduced. Building a simulation model that more closely follows the actual independence and low communication traffic between UAVs

is expected to reduce global dynamic data structures and communication traffic. UAV objects that are independent can run on separate CPUs and require a minimum amount of coordination, allowing the simulation model to execute using a CTM methodology as though it were using OTM. The swarm behavior implemented in the swarm model requires periodic synchronization between UAVs. Currently synchronization occurs at one simulation second intervals which yields a scalability and run time choke point. By adding dynamic load balancing (Jiang et al. 1994) (Gan et al. 2000) by hosting a neighborhood of UAVs on the same processing node the internode communications can be greatly reduced. Only UAVs on the neighborhood boundaries need communicate outside of their host node.

The initial focus has been on swarm behavior and swarm routing. These two areas present a significant challenge, and work will continue in these areas. The initial model is a 2-dimensional swarm behavior model with a 3-dimensional model planned.

Swarm routing is static and fixed for the entire mission. A dynamic and possibly interactive routing needs to be developed.

Other areas of future work include: deployment, communication, search, performance, and mission accomplishment. Aerial platform, missile, ground based, etc. are some of the deployment methods that need investigation. What forms of communications and communication capabilities are needed by the individual UAV swarm members? What are good search strategies for UAV swarms? The performance characteristic of climb, speed, and payload capability affects the swarm behaviors. All of the above considerations are relevant only with respect to mission accomplishment. The swarm need to accomplish its mission of sensing or neutralizing the target.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Mike R. Foster, Air Force Wright Laboratory (AFRL/SNZW), Virtual Combat Laboratory for his support to this project. We also acknowledge Captain Joshua J. Corner, USAF, for his contributions to the development of the parallel UAV swarm simulation model.

## REFERENCES

Casti, J. L. 1997. *Reality Rules I: Picturing the World in Mathematics*. John Wiley and Sons.  
 Corner, J. J., and G. B. Lamont. 2004. Parallel simulation of uav swarm scenarios. In *Proceedings of the 2004 Winter Simulation Conference*, ed. M. Rossetti, J. Smith, and B. Peters, Volume 1, 347–355.

- Corner, J. 2004. Swarming reconnaissance using unmanned aerial vehicles in a parallel discrete event simulation. Master's thesis, Air Force Institute of Technology.
- Gan, B. P., Y. H. Low, S. Jain, S. Turner, W. Cai, W. J. Hsu, and S. Y. Huang. 2000. Load balancing for conservative simulation on shared memory multiprocessor systems. In *Proceedings of Workshop on Parallel and Distributed Simulation, PADS*, ed. L. Donatiello, S. Turner, and D. Bruce, 139–146.
- Grama, A., A. Gupta, G. Karypis, and V. Kumar. 2003. *Introduction to Parallel Computing*. Second Edition ed. Addison Wesley.
- Jiang, M.-R., S.-P. Shieh, and C.-L. Liu. 1994. Dynamic load balancing in parallel simulation using time warp mechanism. In *Proceedings of International Conference on Parallel and Distributed Systems*, 222–227.
- Kadrovach, T. 2003. *Communications Modeling System For Swarm-Based Sensors*. Ph. D. thesis, Air Force Institute of Technology.
- Metron 2003. SPEEDES User's Guide. <<http://www.speedes.com>>.
- Russell, M. A. 2005. A genetic algorithm for uav routing integrated with a parallel swarm simulation. Master's thesis, Air Force Institute of Technology.
- Sun, X.-H. H., and L. M. Ni. 1993. Scalable Problems and Memory-Bounded Speedup. *Journal of Parallel and Distributed Computing* 19 (1): 27–37.
- rithms, evolutionary strategies), artificial immune systems, information security, parallel and distributed computation and simulation, combinatorial optimization problems (single objective, multi-objective), formal methods, software engineering, digital signal processing, intelligent and distributed control systems, computational and numerical methods, and computer-aided design. Dr. Lamont has authored various textbooks (Multi-objective EAs, Computer Control) and book chapters as well as over 150 papers on the above topics. Professor Lamont has advised over 250 MS students and 30 PhD students. Dr. Lamont was also an engineering systems analyst for the Honeywell Corp, for 6 years. His contact information is 937-255-3636, x4718, <GaryLamont@afit.edu>.

## AUTHOR BIOGRAPHIES

**MATTHEW A. RUSSELL** is a distinguished graduate of both the United States Air Force Academy the Air Force Institute of Technology. BSCS 2003 United States Air Force Academy; MS 2005 Air Force Institute of Technology. His M.S. thesis research is entitled "A Genetic Algorithm for UAV Routing Integrated with a Parallel Swarm Simulation." His interests primarily include software engineering and artificial intelligence.

**KENNETH MELENDEZ** Post Doctorial Researcher, Department of Electrical and Computer Engineering, Graduate School of Engineering and Management, Air Force Institute of Technology, WPAFB, Dayton, OH 45433. BS 1963 and MS 1965 New Mexico State University; PhD 1972 Oklahoma State University. His interests include artificial intelligence, parallel computing, and simulation. His e-mail address is <[kmelende@afit.edu](mailto:kmelende@afit.edu)>.

**GARY B. LAMONT** Professor of Electrical and Computer Engineering, Department of Electrical Engineering and Management, Air Force Institute of Technology, WPAFB, Dayton, OH, 45433, USA; B. of Physics, 1961; MSEE, 1967, PhD, 1970; University of Minnesota. His research interests include: evolutionary computation (genetic algo-