Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

PARALLEL SIMULATION OF UAV SWARM SCENARIOS

Joshua J. Corner

Rome Research Site
Air Force Research Lab
Rome, NY 13441, U.S.A.

Gary B. Lamont

Department of Computer Engineering
Air Force Institute of Technology
WPAFB, OH 45433, U.S.A.

ABSTRACT

The concept of operations for a micro-UAV system is adopted from nature from the appearance of flocking birds, movement of a school of fish, and swarming bees among others. This "emergent behavior" is the aggregate result of many simple interactions occurring within the flock, school, or swarm. Exploration of this emergent behavior in a swarm is accomplished through a high performance computing parallel discrete event simulation. After design of the system, several experiments are designed, tested, and analyzed for efficiency and effectiveness.

1 INTRODUCTION

The ability to use cooperative autonomous vehicles to perform a wartime mission is an important application in future military operations (Brown 1998, Davis 1995, McHale 2003, French 2003). Technology in the Unmanned Aerial Vehicles arena is moving toward smaller and more capable systems and is becoming available at a fraction of the cost. Unmanned Aerial Vehicles (UAVs) are mobile airborne machines that do not require an on-board human operator. Typically they are controlled by a remote operator or autonomous control logic.

2 PROBLEM DOMAIN

2.1 UAVs

The Department of Defense defines UAVs as "powered, aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" (Bone and Bolkcom 2003). The autonomous UAVs considered in this research are comparable in size to a bumblebee. Great strides are being made in miniaturization of electronic and electro-mechanical systems (Garrison 2000, Anonymous 1999, Michelson 2002, Dickinson 1999, McHale 2003). Ron Fearing, a UC Berkeley electrical engineer, is designing wings capable of mimicking the movements of a house fly for use in a micro-mechanical flying insect (Squatriglia 2002). Another study focuses on how flies navigate with reaction speeds that allow them to change course in just 30-thousandths of a second (Squatriglia 2002). Outside the United States is the Seiko Epson Corporation's "Micro Flying Robot" (uFR) (Newswire 2003). This uFR demonstrated its micromechatronics technology in November 2003 at the International Robot Exhibition and is known as the world's smallest flying prototype micro robot (Newswire 2003).

2.2 Swarming

Swarming is an emergent behavior of simple autonomous individuals according to (Clough 2003). Simply stated, using swarms is the same as "getting a bunch of small cheap dump things to do the same job as an expensive smart thing" (Clough 2003). Formally, it is a collection of autonomous individuals relying on local sensing and reactive behaviors interacting such that a global behavior emerges from the interactions (Clough 2003). All members have locally controlled behavior constrained by simple rules. All have predesigned reactive behaviors. Swarms in nature are best characterized by their behavior. When attempting to mimic the behaviors found in nature, a swarm model is not limited to simply the movement of the individual members of the swarm but all the faculties of the swarm. For example, a classification of swarm behavior is found in (Dudek et al. 1996). In it, Dudek indicates that such items as communications topology, range, and bandwidth, and collective reconfigurability, size, and composition, and processing ability all play a role in the classifying a swarm. Kadrovach introduces a swarm taxonomy that uses a three tier continuum: scale, coupling, behavior (Kadrovach 2003). The examples he gives include a single large school of fish as a {global, ordered, loose} swarm, and a colony of ants foraging in widely scattered groups as a {regional, chaotic, tight} swarm. Both of these classification methods are helpful in understanding the characteristics in nature and in man-made swarm models that represent "swarming."

3 ALGORITHM DOMAIN

3.1 Swarm Model

Several mathematical swarm models have been developed by researchers. A partial list includes particle swarm simulation (Trahan et al. 1998), physical robots (Mataric 1995), minefield clearing (Cassinis et al. 1999), cooperative control of autonomous air vehicles (McLain 1999) chemical cloud detection (Kovacina et al. 2002), and distributed sensor networks (Kadrovach 2003). This research intends to use an existing model of acceptable fidelity.

An accurate model represents the real-world object at the highest level of fidelity. The corresponding real-world object being modeled are those swarm entities in nature such as bees, fish, birds. From those natural occurrences one can derive a common basic behavior for any kind of swarm. That basic behavior is a small piece of the contribution made by Craig Reynolds in 1987 (Reynolds 1987). His flocking model is based on three simple steering behaviors as described in Figure 1–reproduced from (Reynolds 2001). The weighting and implementation of those steering behaviors distinguishes differing swarm models. The figure shows a circle which represents what he calls 'local' in the description next to each behavior; it is the same concept of a swarm particle's neighborhood.

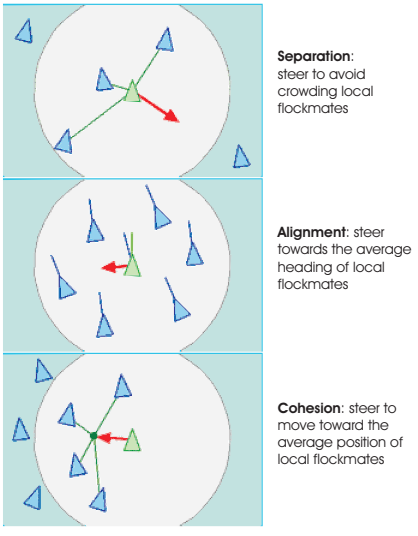


Figure 1: Reynolds' Distributed Behavior Model

Kadrovach designed a swarm algorithm based directly on Reynolds three steering behaviors, with scalability improvements (Kadrovach 2003). The elements of cohesion, avoidance, and attraction are enhanced with the concept of a visibility model. He observes that flocking birds only use a certain visual periphery of influence to adjust their position in the swarm (also noted by Reynolds in (Reynolds 2001)). Building on that observation the behavior of one member of the swarm is influenced by only those members within a fixed angle of perception. Scalability is achieved because the algorithm complexity does not increase as the size of the swarm increases. This swarm model is used by the author to build the parallel swarm simulation.

3.2 Parallel Simulation

Parallel algorithm design and data decomposition strategies can be applied to increase efficiency and provide effectiveness that could not otherwise be achieved. One distributed processing technique is dynamic load balancing. This section discusses these parallel concepts for a distributed sensor application: swarming reconnaissance–using a swarm of sensors to perform a wartime mission (Corner 2004).

Dividing a computational task into smaller pieces that can be scheduled to run concurrently on multiple processors is the key when designing parallel algorithms. This division can be done by decomposing the data structures on which the algorithm operates and then scheduling multiple tasks of a computation simultaneously.

How does one determine the appropriate technique for swarming reconnaissance? If the search space can be represented as a matrix the techniques mentioned above are appropriate. However, if the purpose of the decomposition is applied to the swarm communications problem to optimize the throughput and minimize the latency, then one can consider this an optimization problem in which exploratory decomposition is appropriate because the underlying computations correspond to finding a solution in the search space. If on the other hand one considers the finer level detail of the simulation of the swarm network then the problem turns into a discrete event simulation of the packets traveling across a network topology–at which point speculative decomposition is appropriate because the program may take one of many possible computationally significant branches depending on the output of other computations that are predecessors. The answer depends on the selected models to be implemented and associated model fidelity. Ideally all of the aforementioned techniques are applied to the swarm reconnaissance simulation.

Once the domain has been decomposed the parallel machines can be scheduled to start solving their portion of the problem. The idea is to balance the load evenly across all processing elements. Scheduling these decomposed tasks can itself be a whole new problem because of the possible inter-dependencies within the problem domain. This results then in a task-dependency graph also called a directed acyclic graph (DAG). In Chapter 23 of (Buyya 1999) a description of the scheduling problem as being an NP-complete problem is presented as well as algorithms for the static scheduling problem.

An example task schedule for a reconnaissance swarm application is decomposing the tasks into one of three types: position updates, search strategy and bookkeeping, and sensor data collection and processing.

4 IMPLEMENTATION

Commonly taught among software engineering institutions are techniques that increase the software quality by improving code maintainability, reliability, re-usability, testability, usability, traceability, learn-ability, and portability (Bruegge and Dutoit 1999). Code reuse is one way to gain the most benefit across the range of software quality measures. Simulating a swarming reconnaissance mission as shown in the previous chapter has many complex associated models. To write all of those models from the ground up is a task for the individual without time constraints. Therefore, motivated by software quality and reasonable expectations a selection process occurred which analyzed existing software models and simulation frameworks pertinent to this research effort, see (Corner and Lamont 2003).

4.1 Simulator

A layered approach is determined in (Corner and Lamont 2003) where the SPEEDES parallel environment is chosen as the lowest layer and the initial component model to build on top is Kadrovach's swarm behavior model (Kadrovach 2003).

4.2 Fundamental Differences

Swarm visualization can dramatically effect the validation of a swarm model. As understanding was gained about the implementation of the serial swarm model some questions came to mind about the validity of the model that was used. Can one separate the swarm behavior algorithm from the visualization of that algorithm? No! One assumption Kadrovach did not make is that of physically implementing his model in a real world swarm of micro UAVs. A global data structure would never work, but more importantly the concept of time as implemented appears inaccurate. Common to both a visualization system and a physical implementation is the hard requirement that the laws of time be maintained. When visualizing time-stepped data it is crucial to maintain synchronization with the second hand on the clock or else the results are skewed. Similarly, when implementing a physical representation of a synchronized system those interacting elements must maintain a common clock. In both cases, the serial swarm implementation lacks consistency. For the visualization system, assuming each time unit is equal steps (seconds for example), then any time-dependent movement must occur in sync with one second. That is not the case for the serial implementation.

A simple change allows for a consistent time representation, which is how the SPEEDES "process & proxy" implementation models the swarm. All that is needed for correct reporting of position time update pairs is the visibility of each position update as well as the time that it took to perform the movement. Thus t_2 is always $> t_1$ when a second particles receives an update based on a historical activity. After making this change, there were slightly noticeable differences in the behavior model. The flocking behavior showed little difference, but the swarming behavior is considerably smoother (almost flocking). The quick bursting random direction movements are now smoothed into smaller changes in direction and their is much less perceived acceleration. The above discussion is formed on a basis that the swarm members make updates in a serial fashion with relatively equal intervals. That does not necessarily need to be the case.

How are each members swarm position updated in a parallel implementation where each swarm member might reside on a different CPU? Can the existing algorithm be improved? The answers to these questions dramatically affect the performance of the SPEEDES swarm application. Below are some alternative ways to implement the position update portion of the algorithm.

Position updates in the context of the following discussion implies not only a coordinate pair, but also a heading (direction.) Kadrovach's original implementation of the swarm model calculates the swarm member position updates in a strictly equal time-stepped fashion, which implies that he used a synchronous (time-based) rather than an asynchronous (event-based) simulation.

A difficult part of understanding how to implement this shift in position update time frame is understanding how things are started. At first, one might be confused as to how the position update occurs as it sounds like circular reasoning; however, what must not be forgotten is that at some discrete point in time the swarm is going to be initialized with one member in the lead and the others following. So the steady-state condition involves updates continuously propagating throughout the swarm, while the initial-state has far fewer updates that are propagated–at least until all swarm members are deployed out of their cargo transport or ground launch. In this way, the initial update occurs from the swarm members who are leading. Because the leaders of the swarm have very few (or possibly no) visible neighbors to influence them their update cycle is more rapid than the others. In short, from the time the UAVs are launched until the swarm reaches a steady-state, the number of swarm members who are communicating updates starts with 1 and exponentially (depends on the swarm topology) increases until all members are either publishing position update information or calculating it.

A final alternative for the position update dilemma uses angular information to filter what neighbors are influenc-

ing the current particles direction. This approach can be constructed in the SPEEDES environment by receiving position updates through a proxy for every particle in the current particle's neighborhood of interest. Deciding on which neighbor to process first or to process at all can be implemented through a filter that detects angular movement. Thus neighbors that have larger angular movement have greater influence and are considered a higher priority when calculating a position update. A particle then has variable influence with varying priorities resulting in non-linear positional influence–which can be far more efficient than a forced equal interval update pattern as currently implemented.

For this study, a position update is synchronized at every time step, thus the swarm's movement is formed by discrete updates at each time step. This is not the most efficient choice, however, due to time constraints is the most economical.

5 DESIGN OF EXPERIMENTS

5.1 Parallel Discrete Event Simulation Experiments

To find an efficient configuration for the SPEEDES framework one can characterize the communications impact that results from the internal communication libraries and algorithms of the parallel discrete event simulation framework. Why only characterize the communications impact? Because when changing from a serial to parallel computing platform, the improved performance is primarily based on the speed of the communications between processes (Grama et al. 2003). Given a fixed network topology (cross-bar), fixed data handling and routing (cut-through), and a fixed protocol (TCP) the variable becomes the process communication algorithms. Using gridmanagers, proxies, and events in the SPEEDES framework all require communications between simulation objects–some of which are located on a physically separate nodes, therefore the efficiency of these built in communication algorithms is characterized.

EXPERIMENT SPEEDES-1 (S1). This experiment expects to learn the most efficient configuration for a given set of system parameters for a generic SPEEDES application.

EFFICIENCY. How can SPEEDES run most efficiently on the available Beowulf system configurations? Beowulf systems can be designed with a wide variety of parameters. The parallel computing system options can include the type of hard disk array, memory capacity (fast/slow), cache sizes on almost every hardware component, operating system, management software/hardware, hard disk interface, communications backplane selection, additional specialized hardware requirements, 32/64 bit processor, processor manufacturer, along with many additional managerial and support related items. A detailed look at these configuration details for a large scale computing application is presented in (Corner 2003). The systems used in this experiment were not custom designed for a parallel discrete event simulation application, however they have a few options that can be used to configure the parallel environment at runtime: processors per node, type of backplane, and number of nodes.

PARALLEL CONFIGURATIONS. The configuration variability when setting up the Beowulf system for running the simulation combines permutations of the number of nodes, backplane, and number of CPUs per node. For example, the first configuration uses only 1 node with 1 CPU per node–the Ethernet backplane is not used of course. A second configuration uses 2 nodes for the application with 1 processor per node with the Ethernet backplane. A variation of this second configuration is using the Myrinet backplane.

APPLICATION. Each configuration is tested with each SPEEDES application. The SPEEDES application used for this experiment is a simple straight-line movement pattern for the UAVs. The number of UAVs varies according to this set: {10 , 20, 50, 100, 500, 1000}–which is representative of pedagogical, medium, and larger problem sizes. No rollbacks should occur in this application, thus the major reason for variation is communication delays between simulation objects.

The application is a simple simulation in which n UAVs are moving synchronously at each simulation time step. In addition, each UAV simulation object has a DDM subscription to every UAV within a three unit range. This means that if UAV 1 the x,y coordinate pair of $(5, 8)$ that it will detect through the DDM proxy all other UAVs within this square $(5 \pm 3, 8 \pm 3)$. Using DDM in the application requires SPEEDES to create the gridmanagers to implement the DDM function thus utilizing the communications to the fullest extent. The simulation end time is set to 12 time units. That number is chosen because the amount of communication that is occurring between the SPEEDES applications and the number of events in the queue in addition to those that are being processed by the SPEEDES framework are enough to produce an acceptable "steady state" for this research. An external module is connected to record the movement of each UAV–this is the program that produces the '.swh' history file (a binary file recording swarm positions for every time step.)

5.2 Parallel Swarm Algorithm Experiments

PARALLEL EXPERIMENT #1 (P1). What experiments are necessary to understand if fidelity has been maintained or increased for the swarm simulation model? The behavior of a swarm across the various implementations (Microsoft, Linux, SPEEDES) is comparable only by observation of identical behavior. It is known that the exact reproduction of the sequence of moves is not possible even with the same

set of parameters due to the random number generation variants in each of the implementations. Also, SPEEDES is not expected to produce the same sequence of moves as the Linux version even though these implementations are using the same random number generators because in a distributed simulation UAVs are their own process and thus what would have been a random number in the sequential Linux algorithm for the 40th swarm member is now the 2nd random number for a local process on the 8th node. The result is that even with exact parameters the movements are anticipated to be different on all three platforms. However, forcing the SPEEDES implementation to behave in a serial fashion like the Linux variant is possible. It is also possible to temporarily remove the random number generation forcing a fixed number instead, thereby producing identical results.

CONFIGURATION. A necessary requirement for this test is that the randomness be removed from the swarming algorithm. The programs that are tested include Kadrovach's original swarm movement algorithm (command line version), the author's Linux port, and the SPEEDES version. The random number function that is called throughout the algorithm is defined in the drand() function, so all three implementations are recompiled with this function simply returning a value of 34.03 (arbitrary choice.)

TEST SETUP. In this experiment the well formed formula is a deterministic program which is a formal mathematical expression represented by byte code at its lowest level. The parameter that varies is t, the current simulation time. The condition that needs to hold true is the value of the program at time t on multiple platforms. Thus there are six variants needed for this experiment: 2 for each of the 3 configurations ($t = 1$ and $t = 89$). The first is the base case, the second is the induction step where $n + 1 = 89$. The remainder of the experiment, however, can be assumed true by the second principle of mathematical induction (Shaffer 2001).

PARALLEL EXPERIMENT #2 (P2). Accomplishing efficiency testing with a parallelized version of a serial program is straightforward. The simplest test is to compare run times for identical parameters for a given variety of parameters.

APPLICATION. The application to test the efficiency of the parallelized version of the swarm program uses a similar application as above only running until a steady-state condition is reached (12 time steps). The number of UAVs used in this application vary also according to the findings in the first experiment indicating three transitions in the SPEEDES framework's performance {100, 500, 1000}.

Statistical runs are still needed because of the inconsistencies when running a SPEEDES framework application (probably due to the TCP connections and higher level proxy connection oriented protocol). The entries in the test matrix that have 30 statistical runs refer to running the parallel SPEEDES swarming application. Some entries have only 1 run and are those running the original single-CPU swarming application.

6 TESTING & ANALYSIS

6.1 Parallel Discrete Event Simulation Experiments

S1 ANALYSIS. The metric of interest concerning efficiency is elapsed time. SPEEDES has a default output field that provides the total wall time used to indicate the length of the entire simulation. That field is specified by "wall=" in the standard output of the end of a SPEEDES application.

Since 30 runs were performed with each of the entries in the test matrix a box plot analysis is conducted to show the sample median, the interquartile range (middle 50% range of the data), and any outliers of the execution time for the given parameter sets. Figure 2 shows this data for 1000 UAVs. What additional insight does this box plot

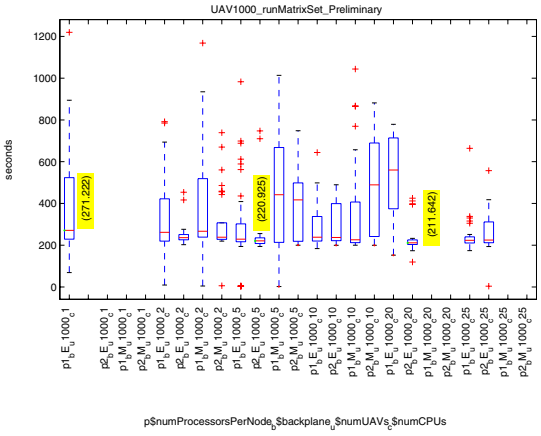


Figure 2: Box Plot of Elapsed Time Data for Experiment 1 for 1000 UAVs

give? The 3 boxed numbers written parallel to the y-axis indicate actual values of the median in the nearby box revealing similar execution time at the lowest levels. No one configuration stands out above the rest when comparing median values. It does not make sense to perform speedup calculations on the SPEEDES program itself with varying numbers of UAVs, although that information is available. One might ask about the outliers down near the '0' value. After probing the data, these runs were not true runs.

It contains the same format of the configurations across the bottom as in the figure above, but this time in addition to the base UAV count (1000) for the figure, all of the remaining UAV counts are laid over this plot to contrast the various execution medians. The data points are the median values of the 30 statistical runs, identical to the solid line in the middle of the box in the box plot figure. This figure shows a major demarcation for the SPEEDES framework when

transitioning from hundreds to thousands of UAV objects. Thus it can be expected that simulations involving over 1000 simulation objects that require intercommunication will take at least 200 seconds to execute 12 simulation time units.

Probing further into the plot reveals another solid demarcation at 500 UAVs, however at 100 UAVs and below the efficiency of the SPEEDES framework is relatively close ($< 5 seconds$).

The final analysis of interest is finding a subset of the configurations which are consistently more efficient. Figure 3 reveals there are certain configurations which always take longer no matter how many UAVs are used. The configurations with 2 processors per node are eliminated. The remaining values are easily rank ordered by the average normalized median values. The top three configurations across all remaining configurations and UAV counts are shown in Figure 4.

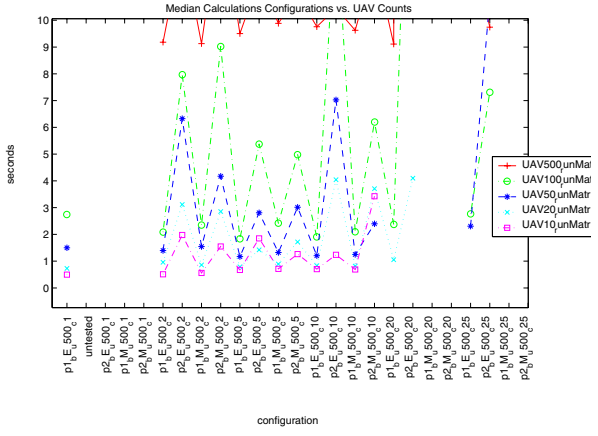


Figure 3: Median Values of Configurations vs. UAV counts (up to 100)

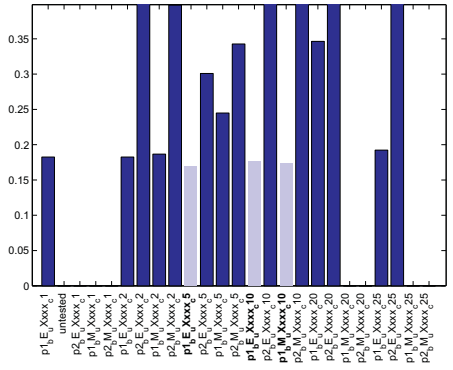
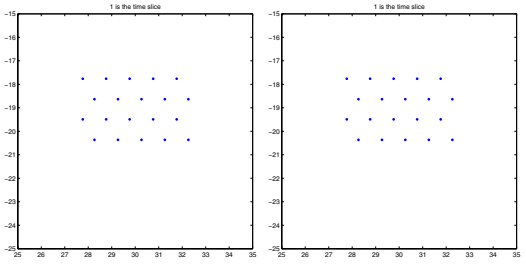


Figure 4: Average Normalized Median Values for Each Configurations

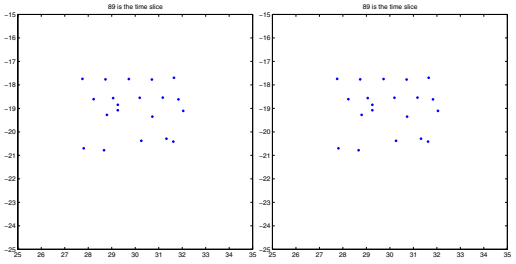
6.2 Parallel Swarm Experiments

P1 ANALYSIS. Running the experiments with varying values of t results in a binary output file that must be viewed through a visualization program. Thus the results are evaluated empirically. Execution of the test is simply using the correct parameter files and command line arguments. The output is captured for all six runs and shown in Figures 5 and 6. By observation, the output is identical for both cases of the induction procedure, therefore it is proven that these algorithms behave accurately for all values of t.



(a) Kadrovach's cline (b) Linux port / SPEEDES

Figure 5: Experiment P1 at $t = 1$



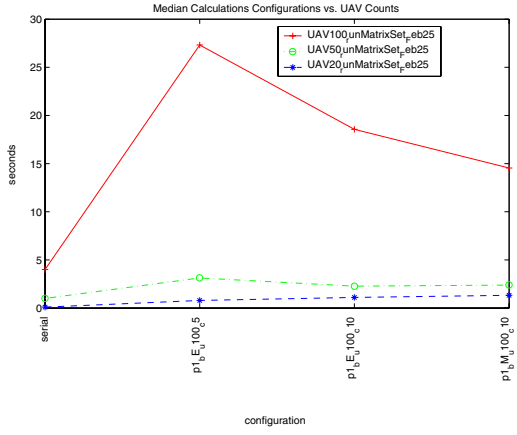
(a) Kadrovach's cline (b) Linux port / SPEEDES

Figure 6: Experiment P1 at $t = 89$

P2 ANALYSIS. When running this experiment, it did not take long before it was evident that there were some efficiency problems. The first row in the test matrix was not the issue, however when the runs from the second row started executing, hours had passed and the SPEEDES run with 500 UAVs was still not complete even though the original serial implementation had finished within the first 5 minutes of execution. After probing the output files it was clear that a number of inefficiencies were discovered. Those items include the number of proxy connections, number of rollback events, and the number of events needing processed before simulation time was allowed to advance. At that point it was clear that the working parallel swarm SPEEDES

implementation was not efficient enough for running larger UAV counts (> 100). Further investigation is conducted in the *Analysis* paragraph below.

The test was modified then to only include smaller UAV counts (< 100) and rerun. The output is shown in Figure 7. As expected, the difference in execution times is negligible for values of 50 and under. Unexpected, however, is the lack of speedup for the largest data set. Speedup measures how much quicker the parallel implementation executes the program over the serial, but as seen in this chart any speedup calculation is less than 1, indicating no speedup at all. The larger data runs did finish on the serial version and the execution times are presented in Table 1.



(a) SPEEDES

Figure 7: Experiment P2 Median SPEEDES Execution for UAVs$\in \{20, 50, 100\}$ vs. Serial Execution

Table 1: Experiment P2 Serial Execution times for UAVs $\in \{100, 500, 1000\}$

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

An intrusive look into the SPEEDES framework is required for understanding the reason for this inefficiency. A preliminary investigation modified configurable SPEEDES implementation options, re-executed the run, and found the following:

- original unsuccessful run
  - last output line after 12512 seconds of execution averaged 1042 wall clock seconds/1 simulation second
- 2nd attempt
- CHANGE: basic SPEEDES algorithm from Breathing Time Warp to Breathing Time Buckets
- THEORY: Breathing Time Buckets will always process at least 1 event so no chance for deadlock to occur
  - last output line after 11590 seconds of execution averaged 966/1
- 3rd attempt
- CHANGE: number of processors: 5 CPUs
- THEORY: Communication is causing the simulation delay, therefore with half the CPUs there is much less needed communication
  - last output line after 22237 seconds of execution; averaged 1853/1.

From these limited attempts there is no obvious solution to improving the efficiency of processing the of the discrete events for this swarming application. 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 modified at a future simulation time, so that n UAVs are causing rollbacks on all other UAVs. This potentially could lead to thrashing and even deadlock–but that is obviously not the case here. Other attempts were made by changing the update interval so that each UAV has a designated time slot offset by $1/numUAVs$ but this took longer even though it had 0 rollbacks! One other possibility is that the performance of SPEEDES gets worse before it gets better, so that at even higher data loads, swarming SPEEDES outperforms the serial version. This is certainly true for data sets that run out of available memory on 1 CPU.

7 CONCLUSIONS

Current research has not seen a parallel implementation of a swarming system. This unique contribution is even more pivotal because the framework upon which it is built includes support for discrete event simulations. This allows for quickly integrating higher fidelity or specialized support models as needed. While speedup is not achieved with the current implementation, it is achievable in future implementations that incorporate optimized parallel algorithms.

ACKNOWLEDGMENTS

The authors appreciate the initial design support received from the diligent researchers at AFRL/IFTC: Robert Hillman, James Hanna, William McKeever, and Doug Holzhauer. Also, real-time visualization support was provided by the great folks at AFRL/SNZW: Mike Foster, Mark Speed, and Ken Sewell.

REFERENCES

Anonymous. 1999. Current and future uav military users and applications. *Air & Space Europe* 1 (5 of 6): 55.

Bone, E., and C. Bolkcom. 2003. Unmanned aerial vehicles: Background and issues for congress. Report for Congress.

Brown, D. A. 1998. Medusa's Mirror: Stepping Forward to Look Back"Future UAV Design Implications From the 21st Century Battlefield". School of Advanced Military Studies; United States Army Command and General Staff College.

Bruegge, B., and A. A. Dutoit. 1999. *Object-oriented software engineering; conquering complex and changing systems*. Upper Saddle River, NJ: Prentice Hall PTR.

Buyya, R. 1999. *High performance cluster computing*, Volume 1. Prentice Hall.

Cassinis, R., G. Bianco, A. Cavagnini, and P. Ransenigo. 1999. Strategies for navigation of robot swarms to be used in landmines detection. In *Proceeding of the Third European Workshop on Advanced Mobile Robots*.

Clough, B. 2003. Uav swarming? so what are those swarms, what are the implications, and how do we handle them? In *Proceedings of 3rd Annual Conference on Future Unmanned Vehicles*. Air Force Research Laboratory, Control Automation.

Corner, J. 2003. Beowulf configuration for large-scale generic application. unpublished; Air Force Institute of Technology.

Corner, J. 2004. Swarming Reconnaissance Using Unmanned Aerial Vehilces in a Parallel Discrete Event Simulation. Master's thesis, Air Force Institute of Technology, Wright Patterson Air Force Base, Ohio.

Corner, J., and G. Lamont. 2003. Selecting a Simulator for Modeling UAV Swarms. unpublished; Air Force Institute of Technology.

Davis, C. R. 1995. Airborne reconnaissance: The leveraging tool for our future strategy. Executive Research Project, National Defense University. The Industrial College of the Armed Forces.

Dickinson, M. H. 1999. Biological insight to flight control. Available online via <www.darpa.mil/dso/thrust/biosci/cbs/ucb-v_ab.html> [accessed August 19, 2003].

Dudek, G., M. R. M. Jenkin, E. Milios, and D. Wilkes. 1996. A taxonomy for multi-agent robotics.

French, M. 2003. Uavs advance since desert storm. Available online via <www.fcw.com/fcw/articles/2003/0217/web-uav-02-19-03.asp>[accessed August 19, 2003].

Garrison, P. 2000. Microspies: A web anthology-a supplement to air & space magazine. Air & Space Magazine; Available online via <www.airspacemag.com/asm/mag/supp/am00/uSPY.html> [accessed August 19, 2003].

Grama, A., A. Gupta, G. Karypis, and V. Kumar. 2003. *Introduction to parallel computing*. Harlow England: Addison-Wesley.

Kadrovach, B. A. 2003. A communications modeling system for swarm-based sensors. Ph. D. thesis, Air Force Institute of Technology.

Kovacina, M. A., D. Palmer, G. Yang, and R. Vaidyanathan. 2002. Multi-agent algorithms for chemical cloud detection and mapping using unmanned air vehicles. Technical Report AFRL-VA-WP-TP-2002-322, Air Force Research Laboratory, Air Vehicles Directorate. In *Proceedings for 2002 Conference on Intelligent Robots and Systems Presentation*, Lausanne, Switzerland.

Mataric, M. J. 1995. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems* 16 (2-4): 321–331.

McHale, J. 2003. Unmanned aircraft armed and dangerous. *Military and Aerospace Electronics* 14 (11): 18–25.

McLain, T. W. 1999. Coordinated control of unmanned air vehicles. Air Vehicles Directorate, Wright-Patterson Air Force Base, Ohio.

Michelson, R. 2002. Entomopter project. Available online via <avdil.gtri.gatech.edu/RCM/RCM/Entomopter/EntomopterProject.html> [accessed August 19, 2003].

Newswire, J. 2003. Epson develops world's smallest flying microrobot. Available online via <www.japancorp.net/Article.asp?Art_ID=5967> [accessed August 19, 2003].

Reynolds, C. 2001. Boids(flocks, herds, and schools: a distributed behavior model). Available online via <www.red3d.com/cwr/boids> [accessed August 19, 2003].

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21 (4): 25–34.

Shaffer, C. 2001. *A practical introduction to data structures and algorithm analysis*. Upper Saddle River, New Jersey: Prentice Hall.

Squatriglia, C. 2002. Soaring achievement in spying uc berkeley team creating 'microfly' to infiltrate enemies. Ohio State University Transgenics in the News Publication. Available online via <ipm.osu.edu/trans/062_241.htm> [accessed August 19, 2003].

Trahan, M. W., J. S. Wagner, K. M. Stantz, P. C. Gray, and R. D. Robinett. 1998. Swarms of uavs and fighter aircraft. In *Proceedings of the 2nd Int'l Conference on Nonlinear Problems in Aviation and Aerospace*, Volume 2, 745–752.

AUTHOR BIOGRAPHIES

JOSHUA J. CORNER is a senior engineer in the Air Force Research Laboratory at the Rome Research Site, Advanced Computing Architectures branch of the information directorate. He completed his Bachelors Degree in Electrical Engineering from Cedarville University, Cedarville, OH. In 2004 he received a Master's degree in Computer Engineering from the Air Force Institute of Technology, WPAFB, OH. His e-mail address is <joshua.corner@rl.af.mil>.

GARY B. LAMONT, Professor of Electrical and Computer Engineering , Dept of Electrical and Computer Engineering, Graduate School of 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 algorithms, evolutionary strategies, …), artificial immune systems, information security, parallel & 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 e-mail address is <Gary.Lamont@afit.edu>.