Using Unified Parallel C to Enable New Types of CFD Applications on the Cray X1/E

Andrew A. Johnson
Army High Performance Computing Research Center,
Network Computing Services, Inc.

ABSTRACT: *We are using parallel global address-space (PGAS) programming models, supported at the hardware-level on the Cray X1/E, that allow for the development of more complex parallel algorithms and methods and are enabling us to create new types of computational fluid dynamics codes that have advanced capabilities. These capabilities are the ability to solve complex fluid-structure interaction applications and those involving moving mechanical components or changing domain shapes, and this is a result of coupling automatic mesh generation tools and techniques with parallel flow solver technology. Several complex CFD applications are currently being simulated and studied using this method on the Cray X1E and will be presented in this paper.*

KEYWORDS: Cray X1E, Unified Parallel C, CFD, Applications, Mesh Generation

1. Introduction

We are currently involved in a project to develop new types of computational fluid dynamics (CFD) codes and methods in order to solve a class of scientific and engineering applications that have traditionally been difficult to solve using numerical methods. These are fluid-structure interaction applications that have moving mechanical components and/or changing domain shapes. Such applications could include flapping-wing vehicles, rotorcraft, engines, turbines, pumps, airdrop systems, free-surface flow, fluid-particle flow [1-3], energy/nuclear systems, and many bio-medical applications. In some cases, the motion of the mechanical components or surfaces is know (prescribed), and in others, the motion of the components is part of the solution and coupled to the fluid dynamics results. In either case, the changes in the domain shape cause complications to the underlying mesh being used in the discretization of the fluid.

We are concerned with conforming-mesh methods where the geometry of the mesh matches the geometry of the problem. This represents a Lagrangian or Arbitrary Lagrange-Eulerian (ALE) framework as opposed to a pure Eulerian framework which assumes the grid remains fixed throughout the entire time-integration of the simulation. In an ALE simulation, like those being discussed in this paper, the 3D mesh must move and/or deform in order to take-up the motion of any mechanical components (surfaces) involved in the application. Using special mesh moving methods involving formulations based on linear-elasticity theory [1,4], the mesh can be moved to a small extent, but soon, the mesh quality can degrade significantly and eventually tangles and/or becomes "invalid" when any mesh-element's volume becomes zero or negative. Traditionally, one would then re-mesh and project the solution onto the new mesh [1,5], but that procedure is very complicated, time consuming, and introduces significant projection errors, especially for incompressible flow [5].

Our new parallel CFD method for solving these types of applications in an ALE framework is called "Dynamic-Mesh CFD" and is based on automatic mesh re-generation ideas discussed in [6]. At that time, work on dynamic-mesh methods was limited to serial (single-processor) codes due to the complexity of the methods involved. Any practical 3D CFD simulation must run in parallel due to the large mesh sizes, memory requirements, and significant run-times of practical 3D applications. Limitations of the MPI parallel programming model prevented us to develop a parallel implementation of automatic mesh re-generation methods at that time due to its complexity. Currently, we are using the more flexible and intuitive Unified Parallel C (UPC) parallel programming model [7] on the Cray X1/E architecture which is allowing us to develop a Dynamic-Mesh CFD code, called XFlow, and use it to solve several complex 3D fluid-structure interaction applications.

In Section 2, we will present some high-level details about the Dynamic-Mesh CFD method, and then in Section 3, we will give some information on three of the initial applications that have been solved using this methods and code. All current work on Dynamic-Mesh CFD methods and codes, as well as all applications, have

been performed on the Cray X1E located at the Army High Performance Computing Research Center (AHPCRC). This is a liquid-cooled system consisting of 256 Multi-Streaming Processors (MSPs) and is shown in Figure 1.

Figure 1. The AHPCRC's 256-processor Cray X1E at Network Computing Services, Inc. (Minneapolis, Minnesota).

On Cray X1/E systems, the parallel global address-space (PGAS) concepts of UPC are supported at the hardware level. This leads to significant performance gains for applications using UPC over MPI, especially when message-latency time is important [8].

2. Technology Involved

Dynamic-Mesh CFD, as embodied in the new XFlow simulation code, involves the coupling of several distinct methods and techniques. These include "traditional" parallel finite-element based flow solvers [9], automatic mesh generation (AMG) techniques based on Delaunay methods [6,9], and the UPC parallel programming model. Although our parallel finite-element based flow solvers such as Aeolus and BenchC have been running in parallel since the early '90s [10,11], AMG methods have traditionally been difficult to parallelize, and impossible to do so in many cases. This difficulty is due mainly to the complexity of the algorithms, sparse and complex nature of the calculations and searches involved, and the very dynamic and changing data structures.

The UPC language greatly aids in allowing us to implement our AMG methods on a distributed-memory parallel architecture and integrating those methods with the flow-solver's routines and methods. UPC shared-data constructs allows us to make the entire mesh structure "visible" to all processors (threads in UPC-speak) so that computations and searches can easily span processor (i.e. mesh partition) boundaries which normally would be barriers to MPI-based algorithms and codes. The low-latency features of the Cray X1/E make sure that when these UPC-based algorithms and codes do span these processor boundaries, a prohibitively expensive cost is not incurred.

By integrating AMG procedures directly with the flow solver, we can change the nature of performing a numerical simulation. Typically, when performing a CFD simulation, a user would first generate a 3D mesh, and when that is done, use that mesh to carry-out the simulation. The mesh is then, generally, static and doesn't change its structure throughout the run, and thus, un-able to change due to either moving mechanical components or changing domain shapes. By coupling the mesh generator directly with the simulation code, automatic mesh generation continues throughout the simulation and never stops, and therefore, isn't a static process anymore. In Dynamic-Mesh CFD, we still move/deform the mesh using our formulation based on linear-elasticity models, but after that happens, the automatic mesh generation routines are there to "clean-up" the mesh structure where required. The AMG procedures re-arrange the mesh-elements to improve their quality, add new mesh-nodes to locations where there may not be enough, and delete existing mesh-nodes from locations where there may be too many.

Because we are using general-purpose mesh moving methods and "automatic" mesh generation/update techniques, this entire procedure to move and update the 3D mesh throughout the simulation is automatic to a user of XFlow. This feature makes the set-up and simulation of complex applications fairly easy, at least when compared to other methods and codes. Because of this ease-of-use and generality of the method, many different types of complex 3D fluid-structure interaction applications can be simulated using the same XFlow code. We highlight three of the initial test/demonstration applications in the next section.

3. Applications

We have been using the Dynamic-Mesh CFD method, as implemented in the XFlow code, on the AHPCRC's Cray X1E to test and demonstrate its capability through the simulation of several different types of fluid-structure interaction application. All of these 3D applications involve moving mechanical components with large, time varying displacements. We present details on three of these early test applications.

3.1 Micro-Unmanned Aerial Vehicle

One of the first applications simulated using the new XFlow code was airflow past a cruising micro-unmanned aerial vehicle (MUAV), which could have use as a small reconnaissance vehicle. These are autonomous vehicles with wingspans on the order of a few centimetres, on the scale of a large insect or small bird. Although such mechanical systems can't be built today, one can perform numerical simulations of such hypothetical vehicles to obtain information on the aerodynamic factors involved, test and develop control algorithms, and come up with power and weight estimates for possible future systems. Our goal for these simulations is to demonstrate the applicability of XFlow technology for studying these types of systems and vehicles.

Our hypothetical MUAV being simulated here involves a long slender body with two flapping wings.

We envision thin wings made up of an Electro-Active Polymer material that would bend up-and-down based on the amount of electrical current fed to it. We programmed a fast downward bending motion and slower upward motion, with a forward sweep to the downward motion in order to create thrust. Based on the average wing chord length and cruising speed, a Reynolds number of around 400 was set for the simulation which is similar to the flight conditions of some insects.

The 3D mesh used in the simulation contained, on average, around 5.2 million tetrahedral elements. Due to the symmetry of the geometry, only half of the vehicle was modelled with symmetry conditions imposed at the center. A total of 6,000 time steps were computed (15 wing-beat cycles) using 32 multi-streaming processors (MSPs) of the Cray X1E.

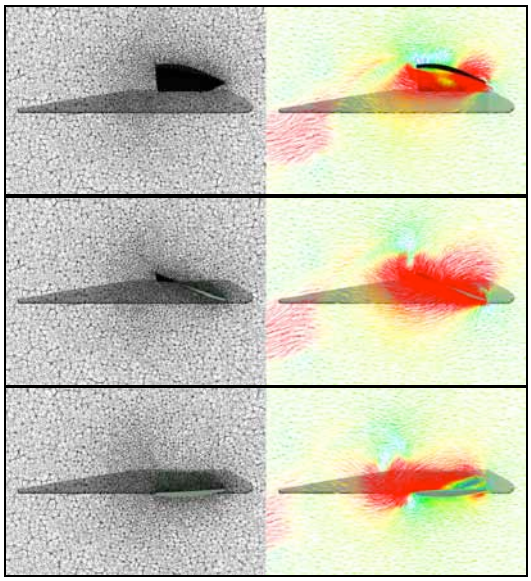

Figure 2. Shown is the mesh (left) and velocity vectors at a cross-section at a location towards the tip of the micro-UAV wing at three time instances. The colors in the vector field correspond to the speed of the airflow. In this sequence of images, the wing is moving down during its thrusting-stroke.

The simulation using XFlow ran very well, and a positive lift and thrust force, generated by the wings, was measured. Based on the wing area, average lift and thrust coefficients of 0.16 and 0.10, respectively, were produced. In Figure 2 is shown the mesh and velocity vector field at a cross-section towards the tip of the vehicle. In Figure 3 is shown a volume-rendered image of vorticity magnitude, viewed from the top of the vehicle. In this figure, the effects of the individual wing beats on the flow can easily be seen by the observed centers of strong vorticity.

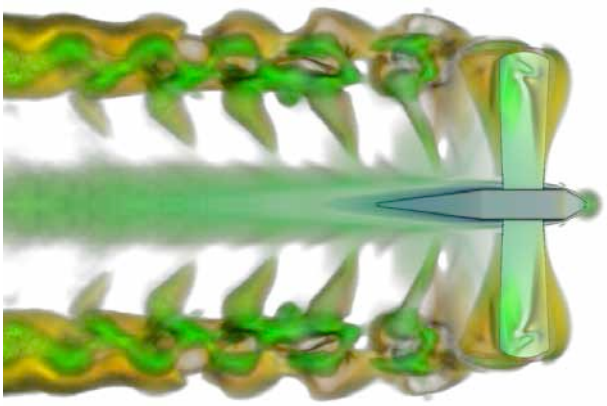

Figure 3. Shown is a volume-rendering of vorticity magnitude at one instance of the flapping-wing micro-UAV simulation. The generation of centers-of-vorticity from each wing beat cycle can clearly be seen.

3.2 Aerodynamics of a Hovering Hummingbird

To expand on the work and simulations involved in the micro-UAV demonstration of Section 3.1, we developed a model of the geometry and wing motion of a hovering hummingbird. The idea here is to demonstrate the capabilities of XFlow for simulating the flapping-wing flight of both birds and insects. In this case, we are simulating a hovering hummingbird.

The study of flapping-wing flight of animals has received a great amount of attention lately in the literature [12,13]. The goal in these studies is to learn from nature how those biological systems fly, perform manoeuvres, control, and hover, all with low weight and power. By studying these animals, it is possible that future micro-UAV designs and functionality could be modelled after actual biological systems.

Our computational hummingbird has a very realistic geometry and wing motion. The wing motion was designed to match real hummingbird wings and motions [13], and involves a forward and back sweeping motion with wing rotations and twists. The simulation performed models a "hovering" condition where the bird is held fixed and the entire airflow around the bird is driven by the motion of the wings alone. Based on the average chord length and velocity of the wing, a Reynolds Number of 2,000 was set and a total of 3,000 time steps were computed. This corresponds to about 15 flapping cycles of the wing which is about ¼ second real-time. The 3D mesh started with around 4.5 million tetrahedral elements and ended with around 7.3 million elements (i.e. the mesh was growing throughout the simulation). As in the micro-UAV simulation, only half of the geometry was modelled due to the symmetries involved. The simulation was performed using 20 MSPs.

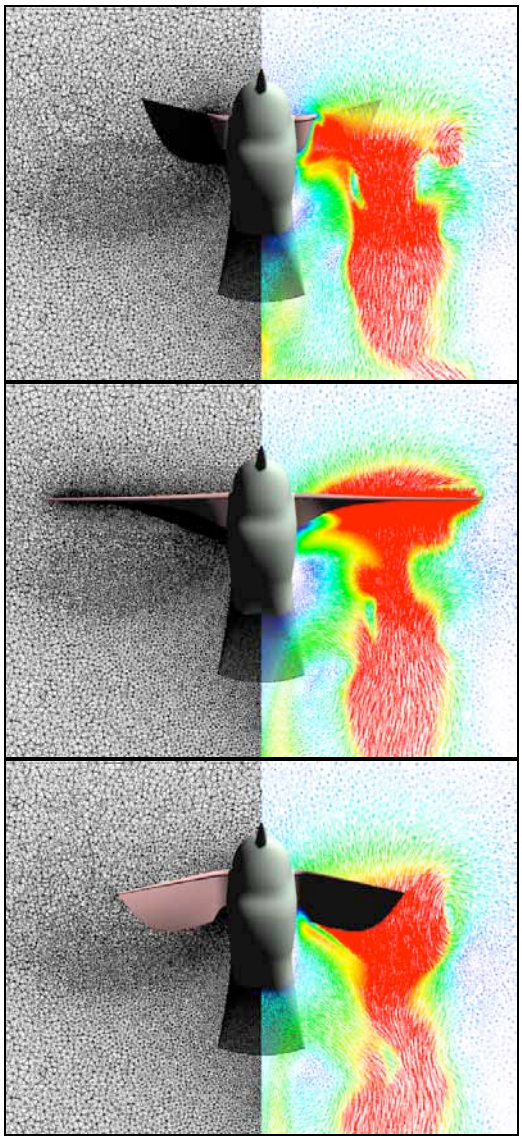

Figure 4. Shown is the mesh (left) and velocity vectors at a cross-section through the center of the bird at three time instances. In this sequence of images, the wing is moving forward, and the strong downward stream of air generated by this motion can clearly be seen.

The simulation ran very well and produced a very realistic flow field around the bird and its flapping wings. The average lift force generated was around 4.5 grams which is larger than the average weight for such birds which is between 3 and 4 grams. Discussions with biologists performing experimental measurements of airflow around a hovering hummingbird in a wind tunnel [13,14] have shown that our computational results compare very well with the actual flow features observed. Shown in Figure 4 is the mesh and velocity vectors at a cross-section at three instances during the bird's wing-beat cycle. Shown in Figure 5 is a volume rendering of vorticity magnitude at an instant during the bird's mid-forward wing stroke.

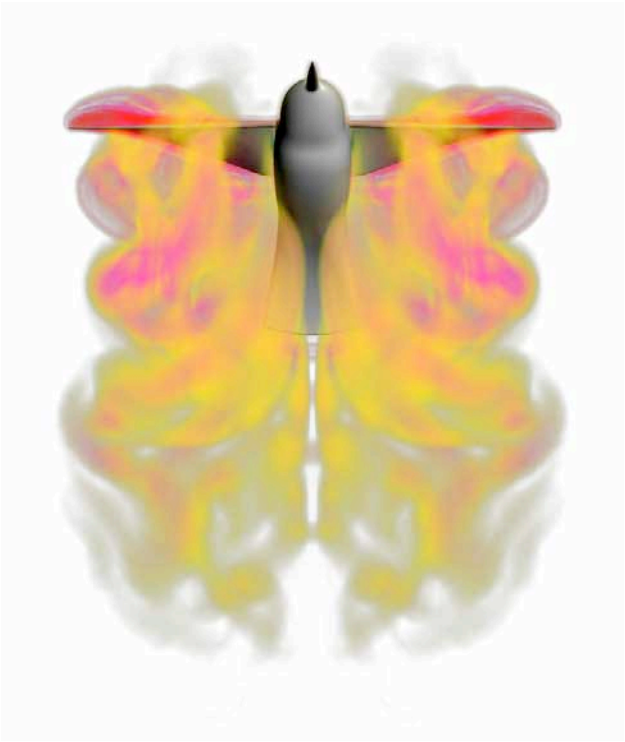

Figure 5. Shown is a volume-rendering of vorticity magnitude at one instance of the hovering hummingbird simulation. The strong downward streams of air, generated by the wing motion, can clearly be seen. Also noted here is the complex nature of the airflow patterns generated by the bird during hovering.

3.3 Pumping Artificial Heart

Another application area that XFlow has great potential for is complex bio-medical applications. These applications can be very challenging to simulate due to the changing domain shapes and moving geometries involved in many of these applications. Such applications could include, for example, the blood-flow through arteries, airflow through moving and expanding lungs, flow through opening and closing heart valves, and simulations of the entire human heart.

To demonstrate XFlow's applicability in this area, we chose to model flow through a small pumping device, which could represent a total artificial heart. In our hypothetical pump design, there is a cylindrical chamber about 8 cm in diameter, and on one side of the chamber there is a pulsating membrane which drives the flow. Attached to this chamber are two cylinders that have valves in them to allow flow to come into the pump from one side and leave through the other. We don't actually model the valves in these cylinders mechanically, but we have developed a special boundary conditions that only allow flow in one direction through these "virtual" valves. It is this combination of the pulsating membrane, along with the inflow and outflow cylinders and valves, which causes the blood to flow through the chamber in an efficient manner. The parameters of the simulation are set

to very realistic conditions for actual blood-flow and pumping rates of the human circulatory system.

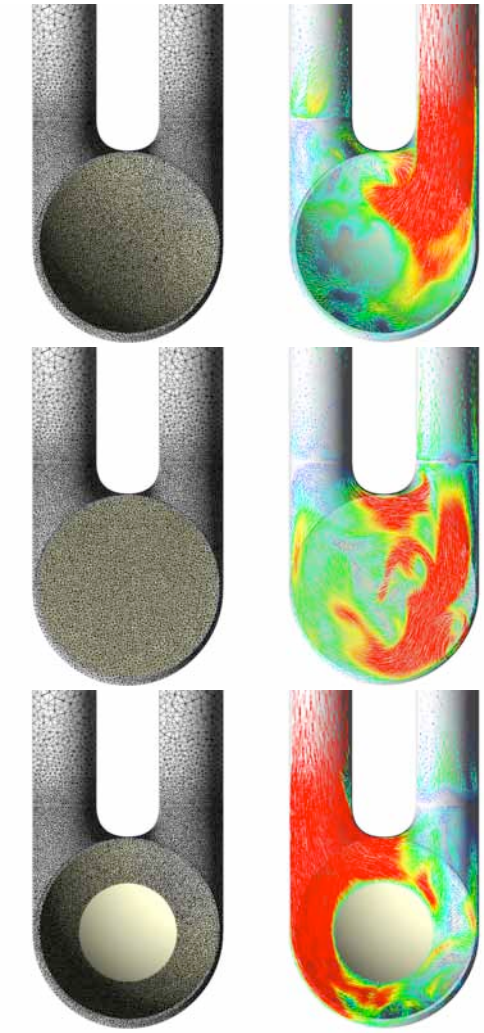

Figure 6. Shown is the mesh (left) and velocity vectors at a cross-section down the center of the pump geometry at three time instances. The fluid is pulled into the chamber at the first time instance, the pumping membrane is "at-rest" in the second time and both valves are closed, and the fluid is pushed out of the chamber at the third time instance.

Since this is an "internal" flow system, the tetrahedral element mesh used was smaller than the others, on the order of about 1 million elements. Many beat cycles of the pump were simulated, and the computed results are very realistic and provide a large amount of details of the blood-flow through the system. Shown in Figure 6 are velocity vectors and the mesh at a vertical cross-section through the geometry at three time instances. Shown in Figure 7 are the velocity vectors and the mesh at a horizontal cross section. The movement and deformation of the mesh caused by the pulsating membrane can be clearly seen in this figure.

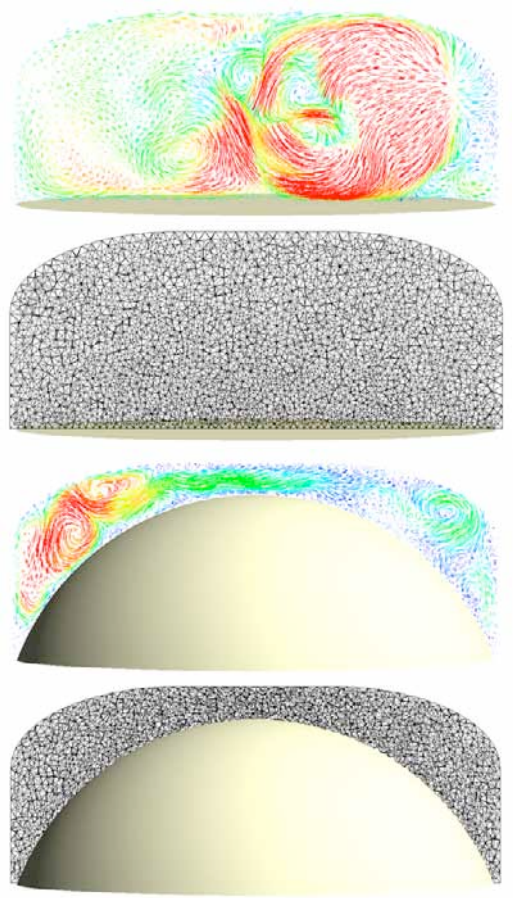

Figure 7. Shown is the mesh and velocity vectors at a horizontal cross-section at two time instances. The deformation of the mesh due to the motion of the pump's membrane can be seen, as well as its effect on the flow.

4. Conclusion

We have developed a novel and unique numerical method for simulating some of the most challenging fluid-structure interaction applications. These applications involve moving mechanical components or changing domain shapes, which have always been difficult to solve numerically due to the requirement that the underlying 3D mesh used in the computation must move and change to conform to the changing geometry. These new methods are enabled by the capabilities of the Unified Parallel C programming model, as implemented on the Cray X1/E.

Our approach involves a dynamic-mesh technique that changes and modifies the mesh throughout the simulation to account for these, sometimes drastic, geometric changes. We have implemented these methods within the new XFlow CFD code and have demonstrated its functionality and potential through a series of complex engineering and scientific applications, some of which have not been solved before at this level of detail. Initial results of these applications look very promising and points to great potential to use XFlow and Dynamic-Mesh

CFD methods for a wide variety of complex 3D fluid-structure interaction applications.

References

1. A. Johnson and T. Tezduyar, "Simulation of multiple spheres falling in a liquid-filled tube", *Computer Methods in Applied Mechanics and Engineering*, No. 134 (1996), 351-373.

2. A. Johnson and T. Tezduyar, "3D Simulation of fluid-particle interactions with the number of particles reaching 100", *Computer Methods in Applied Mechanics and Engineering*, No. 145 (1997), 301-321.

3. A. Johnson and T. Tezduyar, "Methods for 3D computation of fluid-object interactions in spatially-periodic flows", *Computer Methods in Applied Mechanics and Engineering*, No. 190 (2001), 3201-3221.

4. A. Johnson and T. Tezduyar, "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces", *Computer Methods in Applied Mechanics and Engineering*, No. 119 (1994), 73-94.

5. A. Johnson, "Mesh generation and update strategies for parallel computation of flow problems with moving boundaries and interfaces", Ph.D. thesis, University of Minnesota, 1995.

6. A. Johnson and T. Tezduyar, "Advanced mesh generation and update methods for 3D flow simulations", *Computational Mechanics*, No. 23 (1999), 130-143.

7. S. Chauvin, P. Saha, F. Cantonnet, S. Annareddy and T. El-Ghazawi, "UPC Manual", *The George Washington University High Performance Computing Laboratory*, Version 1.2, available at http://upc.gwu.edu/

8. A. Johnson, "Unified Parallel C within computational fluid dynamics applications on the Cray X1(E)", *Proceedings of the Cray User's Group Conference 2005*, Albuquerque New Mexico, May 2005.

9. A. Johnson and T. Tezduyar, "Parallel computation of incompressible flows with complex geometries", *International Journal for Numerical Methods in Fluids*, No. 24 (1997), 1321-1340.

10. M. Behr, A. Johnson, J. Kennedy, S. Mittal and T. Tezduyar, "Computation of incompressible flows with implicit finite element implementations on the Connection Machine", *Computer Methods in Applied Mechanics and Engineering*, No. 108 (1993), 99-118.

11. T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson and S. Mittal, "Parallel finite element computation of 3D flows", *IEEE Computer*, October (1993), 27-36.

12. M. Dickinson, F. Lehmann and S. Sane, "Wing rotation and the aerodynamic basis of insect flight", *Science*, Vol. 284 (1999), 1954-1960.

13. D. Warrick, B. Tobalske and D. Powers, "Aerodynamics of the hovering hummingbird", *Nature*, Vol. 435/23 (2005), 1094-1096.

14. D. Warrick and B. Tobalske, personal communications, April 2006.

Acknowledgments

This document was developed in connection with contract DAAD19-03-D-0001 with the U.S. Army Research Laboratory. The views and conclusions contained in this presentation are those of the authors and should not be interpreted as presenting the official policies or positions, either expressed or implied, of the U.S. Army Research Laboratory or the U.S. Government unless so designated by other authorized documents. Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

About the Author

Dr. Andrew Johnson is a Senior Scientist at Network Computing Services, Inc. working on the Army High Performance Computing Research Center (AHPCRC) program. He has been with that program since its inception in 1990. Dr. Johnson performs research and development in the areas of Computational Fluid Dynamics, High Performance and Parallel Computing, Automatic Mesh Generation, Geometric Modelling, and Large-Scale Scientific Visualization on parallel architectures. Dr. Johnson holds a Ph.D. in Aerospace Engineering from the University of Minnesota. He can be reached at 1200 Washington Avenue South, Minneapolis, MN 55415 USA. E-mail ajohn@ahpcrc.org.