# Numerical Flow Simulation and Rendering Using MPI

John Stone
Department of Computer Science
E-Mail: johns@cs.umr.edu

Mark Underwood
Department of Mechanical and Aerospace Engineering
E-Mail: munder@umr.edu

University of Missouri-Rolla

April 17, 1996

**Abstract**

In this paper, results obtained with a parallel computational fluid dynamics (CFD) simulation code with an embedded parallel rendering library are presented. By linking the rendering library into the CFD code, visualization of CFD data can be done at runtime. Several factors make in-place rendering of CFD data preferable to the use of external rendering packages or dedicated graphics workstations. One of the biggest advantages in doing rendering in-place is avoiding significant I/O overhead typically incurred when moving CFD data to disk files or remote systems. Another advantage to runtime rendering is the ability to monitor simulations as they progress, interactive rendering and simulation control. The use of MPI [1] in both codes helped facilitate their combination into a single application.

**Key Words**: Computational Fluid Dynamics, Visualization, Parallel Volume Rendering, Distributed Memory, MPI.

## 1 Introduction

In this paper, a parallel application for the simulation and visualization of high-speed fluid flow is presented. One of the most common problems in visualizing large CFD simulations is transferring large amounts of data between the simulation process and the rendering process. In many cases, CFD simulation data is stored in temporary disk files. These files are then read into a rendering program, or transmitted over a network to another machine for rendering. A good way of improving overall performance for this task is to join the simulation and rendering programs into a single unit. By joining the separate processes into one program, unnecessary I/O may be avoided. Rendering can be done in-place on the CFD data as it is stored in main memory. Along the same lines, an MPEG or JPEG compression engine may be tied to the rendering process to further streamline the compression of animated video. By merging the simulation, rendering, and compression processes, overall runtime is greatly reduced. MPI facilitates merging multiple parallel applications.

# 2 Parallel Computational Fluid Dynamics (CFD) Simulation

In the past twenty years, Computational Fluid Dynamics (CFD) has gained a very important place in the analysis of complex fluid flows. The CFD code described in this paper numerically simulates compressible fluid flow by solving the three-dimensional Navier-Stokes equations. The code has the capability to handle both multiple species and finite-rate kinetics for hydrogen-air combustion. The Navier-Stokes equations are coupled, non-linear partial differential equations which describe the conservation of mass, momentum, and energy. Due to these equations being very strongly coupled and non-linear, typical CFD solutions are both time and memory intensive. However, recent research has demonstrated the feasibility of using parallel computing to help reduce the computational cost of these simulations [2],[3].

The parallel CFD code PARAFLOW uses a finite difference formulation which relies on the explicit MacCormack method to time-march the solution. The versions of the MacCormack method available in the code are either second-order in time and space or second-order in time and, nominally, fourth-order in space.

For portability, PARAFLOW can be used in either sequential or parallel mode. The code has been validated for both modes and achieves near-linear speedup in the parallel mode for several test cases. The parallel implementation is directed toward a homogeneous domain decomposition on a distributed-memory architecture with point-to-point communication between neighboring processors and a limited amount of collective operations. All message-passing is included explicitly into the code; there is no reliance on parallelizing compilers. Additionally, processor boundaries are overlapped with neighboring processors in order to overlap communication with computation. Furthermore, the CFD code has the option of either using point-to-point communications to perform the collective operations (with either NX or MPI) or using the MPI collective operations. As a general rule, there are only minor performance differences between similar implementations of NX or MPI. However, MPI allows for portability across different platforms.

# 3 Parallel Volume Rendering

Volume rendering is a technique for rendering multidimensional grids of data using the data values to determine visual characteristics such as opacity and color. Volume rendering can also be used to enhance existing ray tracing [4] software with features such as *Hypertextures* [5]. In many cases, it is useful to combine volume data with spheres, planes, polygons or other objects. This can be a difficult problem depending on the construction of the rendering engine. The parallel ray tracing library used for this research includes standard ray tracing primitives as well as volume rendering capabilities [6]. In order to render the CFD data in-place, the ray tracing library was extended to handle distributed data, and volumes based on irregular grids. These extensions are based on user supplied mapping functions which deal with irregular geometry, value interpolation and any other preprocessing of volume data.

# 4 Results

The initial implementation in this investigation has been very encouraging. The rendering and simulation codes have been successfully merged. Several test simulations have been performed with various rendering characteristics. Both the CFD and rendering codes achieve linear speedup individually; similar speedup was observed in the runtime of the merged code. Since the rendering code depends on user supplied mapping functions for geometric transforms and value interpolation, a high percentage of the rendering runtime involves calling the mapping functions. Overall rendering time is reduced significantly by making mapping functions efficient.

The physical layout of the test case which was modeled is a constant cross-sectional area duct with viscous (no-slip) top and bottom walls. The front and back walls are treated as inviscid (velocity tangent to wall). The main flow is Mach 3 air moving from left to right with a flush wall injector on the bottom wall. The injection is Mach 1 air directed 20 degrees up from the horizontal. Figure 1 shows a conceptual schematic of the flow field described. This schematic is not to scale but shows the essential features of the flow field. Also note that the different shocks vary considerably in strength. For the test case, the injector bow shock should be the strongest followed by the recirculation shock. The bottom boundary layer shock and the top boundary layer shock should be of different strengths, however. The reason for the difference in the top and bottom boundary layer shocks is due to the recirculation region which causes the boundary layer to thicken ahead of the recirculation region on the bottom wall. This thickening results in the bottom boundary layer generating a stronger shock than the top boundary layer.

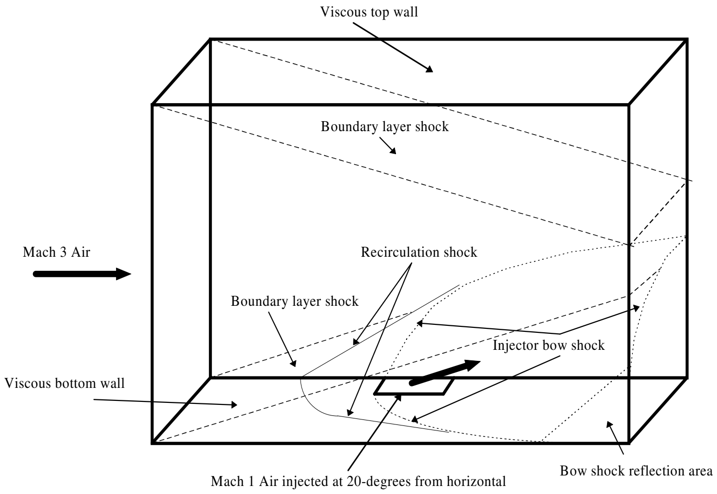


Figure 1: Conceptual Schematic of the Injector Flow Field

Figure 2 shows the pressure contours as predicted by the parallel CFD code and as rendered using the parallel ray tracing library at one particular instant of time. The injector is the white rectangular region at the middle of the bottom wall. The bow shock due to the injection is the most prominent feature (white and pink). Note that the schematic is in an isometric perspective view while the rendering is in one-point perspective view. This difference accounts for the different direction for the curvature of the bow shock on the right hand side. Slightly ahead of the bow shock and near the bottom wall, the triangular recirculation shock can be seen. Due to the normalization of the pressure contours, the weak top boundary layer shock is very faint, while the bottom boundary layer shock is slightly more visible and is seen upstream of the recirculation shock. Therefore, the rendering displays the same major flow features as shown in the conceptual schematic.

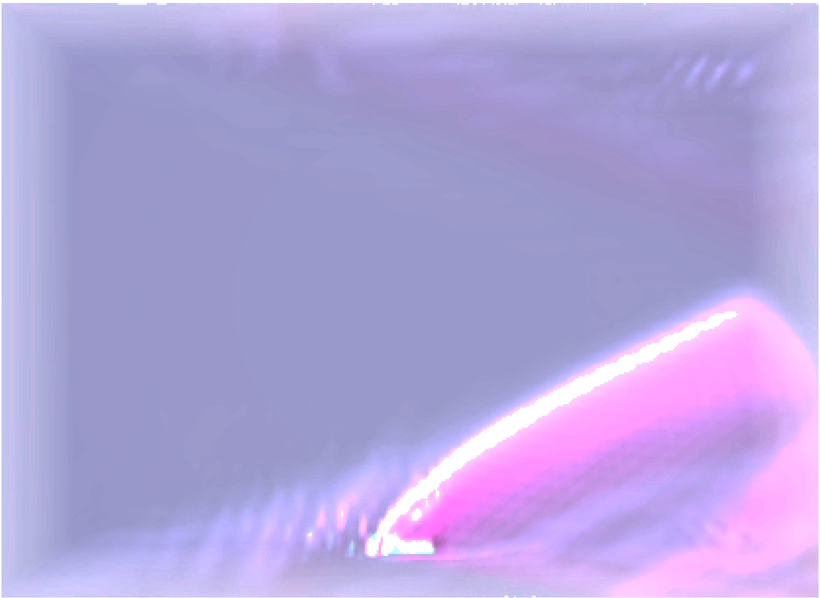

Figure 2: Example Rendering of CFD Data

## 5 Conclusions and Future Work

MPI is an essential part of this investigation due to its ability to easily merge separate parallel applications. It is clear from this and previous work that writing portable parallel code, utilizing MPI or its equivalent, will become increasingly important in order to achieve realistic simulation times.

The results from this work are very promising. In the near future, it is hoped to explore several different approaches in mapping the volume data, hence potentially improving image quality and rendering performance. The next stage of work will involve adding MPI safety features such as contexts and communicators in order to further insure reliable message passing and portability. Once the current implementation has been refined and safety features have been employed, an interactive simulation application will be built and tested.

## 6 Acknowledgements

The authors would like to acknowledge support and computational resources provided by Oak Ridge National Laboratory Center for Computational Sciences, the Hypersonic Vehicles Office at NASA Langley Research Center, NASA NAS, Washington University Computer and Communications Research Center, and the University of Missouri-Rolla. One of the authors (MU) would also like to acknowledge support provided through the NASA Graduate Student Researchers Program (GSRP) by the Hypersonic Vehicles Office at NASA Langley Research Center.

## References

[1] M. P. I. Forum, "Mpi: A message-passing interface standard," Tech. Rep. Computer Science Department Technical Report CS-94-230, University of Tennessee, Knoxville, TN, May 5 1994. To appear in the International Journal of Supercomputing Applications, Volume 8, Number 3/4, 1994.

[2] M. L. Underwood, "The simulation of supersonic flows using parallel computing," Master's thesis, University of Missouri-Rolla, 1993.

[3] D. Riggins, M. Underwood, B. McMillin, L. Reeves, and E. J.-L. Lu, "Modeling of supersonic combustor flows using parallel computing," *Computing Systems in Engineering*, vol. 3, no. 1-4, pp. 217–219, 1992.

[4] A. Glassner, *An Introduction to Ray Tracing.* Academic Press, 1989.

[5] Ebert, Musgrave, Musgrave, Peachey, Perlin, and Worley, *Texturing and Modeling, A Procedural Approach.* Academic Press, 1994.

[6] Intel Supercomputer Users Group, *An Efficient Library for Parallel Ray Tracing and Animation*, 1995.