Real-Time State Estimation of Laboratory Flows

by

Scott Stransky

S.B., Mathematics with Computer Science (2005)

Massachusetts Institute of Technology

Submitted to the Department of Earth, Atmospheric, and Planetary Sciences
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Atmospheric Science

at the

Massachusetts Institute of Technology

June 2007

Signature of Author .....
Department of Earth, Atmospheric, and Planetary Sciences
May 25, 2007

Certified by .....
John Marshall
Professor of Atmospheric and Oceanic Sciences
Thesis Advisor

Accepted by .....
Maria T. Zuber
E.A. Griswold Professor of Geophysics
Head, Department of Earth, Atmospheric, and Planetary Sciences

Real-Time State Estimation of Laboratory Flows

by

Scott Stransky

Submitted to the Department of Earth, Atmospheric, and Planetary Sciences
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Atmospheric Science

ABSTRACT

In this project, we use a real time computer model to simulate a differentially heated laboratory annulus. The laboratory annulus allows us to study chaotic flows typical of the atmosphere. Our objective is to bring the numerical model into close alignment with the laboratory system. Parameter estimation and data assimilation of real time model runs with the tank experiment can be used to improve numerical model fidelity, provided the model operates in real time and the model parameters are in acceptable regimes. We describe how to modify the default configuration of the MITgcm to tackle this new problem. We also run laboratory experiments. Using an iterative process, we update the model parameters (such as diffusion and viscosity), and observe that in at least some regimes, there is excellent agreement between observations and simulations. Much of this effort required the development of infrastructure, which is discussed in this document. Finally, we create and test a complete, real-time system, with data sent across the network from a parallel computer running the numerical model to the laboratory computer, where data assimilation takes place. We further modify the model to allow it to pause mid-run, and restart with the most recent state estimates of velocity and temperature.

Thesis Advisor: John Marshall
Title: Professor of Atmospheric and Oceanic Sciences

Table of Contents

Acknowledgements

I am extremely grateful for the support, advice, and encouragement given to me by my extraordinary advisor, Professor John Marshall. I would also like to give a huge thank you to Dr. Sai Ravela, who guided me throughout the process.

The following people also deserve mention, for their assistance throughout this project: Dr. Constantinos Evangelinos, Dr. Chris Hill, Andrew Wong, Ryan Abernathey, Dr. Ed Hill, and Bud Brown.

This research was supported, in part, by the National Science Foundation (Grant CNS-0540248).

1 Introduction

Numerical models, in conjunction with data acquired from a limited number of observation stations, are the main tools used in weather prediction. The models attempt to predict the future state for the entire planet, even though the initial observations are only in select areas. As it is running, the model receives updated observations about the current conditions. It incorporates those into its state, allowing more accurate predictions to be made.[1] This project attempts to replicate this model/observation interaction on a small scale, in real time – something that has never been attempted before.

Our goal is to model, in real time, rotating fluids in parallel with running actual laboratory experiments. In the particular laboratory experiments detailed here, we use a rotating water-filled cylinder with a metal can in the center, containing a chiller unit. The chiller creates a radial temperature gradient across the rotating fluid, an analogue to the pole/equator temperature gradient on Earth.

At fast rotation speeds, the flow in the tank becomes unstable and baroclinic eddies form (Figure 1). These eddies are analogous to the cyclones and anticyclones that form in the actual atmosphere. They are an efficient method of transporting heat inwards from the outer part of the tank offsetting the cooling in the center. In the real atmosphere (Figure 2), storms transport cold air toward the equator and warm air toward the pole, affecting meridional energy transport.

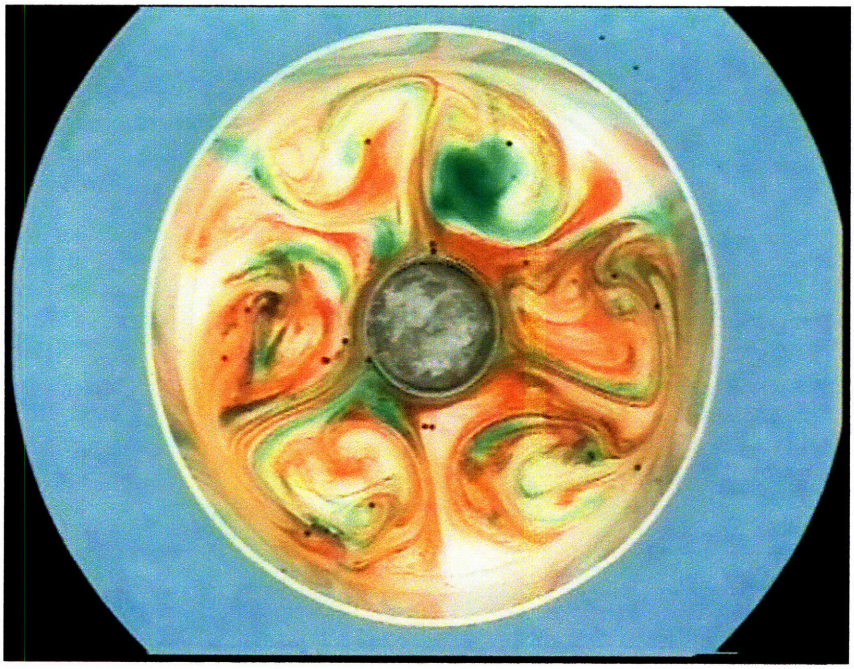

Figure 1, Rotating tank with eddies visualized using food coloring.

[1] If the model were to run without these additional observations, it would quickly diverge from the real world. See "Background and Motivation".

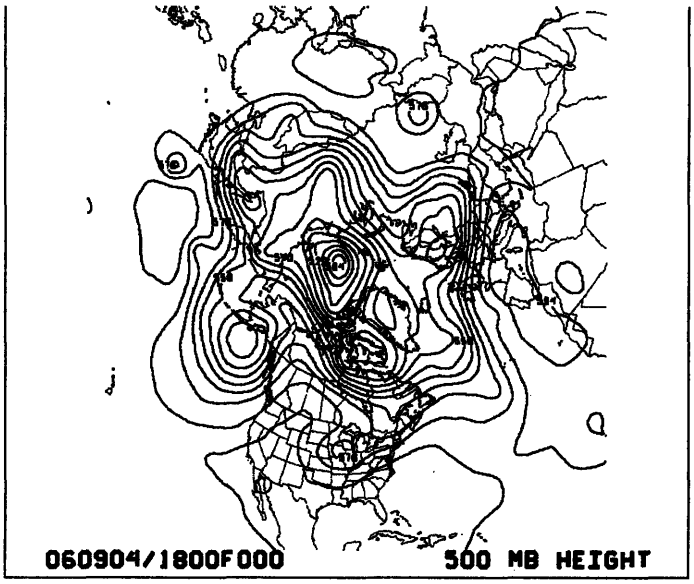

Figure 2, A synoptic map, showing eddies in the atmosphere, at the 500 millibar level (from September 4, 2006, 18z)

1.1 Project Goals

The goals of this project are:

- To develop a real time analysis tool for laboratory experimentation.
- To produce state estimates of laboratory flows that are superior to either the model or the tank alone.

We will perform data assimilation by running the model simulation in conjunction with the tank experiment. Every few seconds throughout the experiment, we will adjust the velocity and temperature fields of the model simulation to represent the current state of the tank, slowly bringing it into alignment with the observations. To accomplish this, we will need to allow the model to pause mid-run, and allow it to accept updated parameters based on the tank data.

Additionally, we can use model "data" to identify specific areas of the laboratory experiment on which to focus. For instance, if we do not have enough tank data near the edge of an eddy, we can (in real time) begin collecting more data from that area. In this way, we develop a closed-loop system: data from the tank will improve the model projection, and the model run will help us analyze the tank in more detail. Moreover, it is an analogue of procedures used in numerical weather prediction.

By running the two systems together, we will be able to improve the numerical model, which has positive implications for the many applications to which MITgcm is being deployed.

The result will be a state estimate that represents the tank experiment in every variable: velocity profiles, temperature profiles, wavenumber, and other quantitative information, not just those that can be easily observed in the tank.

In order to proceed, we must bring the model into near-alignment with the tank data before we can begin the data assimilation. The first part of this paper focuses on this process. Not only do we modify the model's default parameters and parameterizations, but we also add new ones to help improve the correlation between the model and the tank. Additionally, we study the science behind the flow patterns we observe, namely Hadley circulation and the thermal wind balance (see Appendix A).

It is important to note that once this model/laboratory system has been developed, we can apply its hardware and software infrastructures to many other experiments; we are not limited to a rotating cylinder with a chiller in the center. Thus, the groundwork described here will lay the foundation for much future activity, experimentation, and discovery.

1.2 Background and Motivation

In our annulus experiment, we use a rotating cylinder filled with water and a small chiller in the very center of the cylinder to introduce a lateral temperature gradient, simulating the pole/equator gradient. At slow rotation speeds, we would expect a stable flow. The flow is not turbulent, and the tank is in the Hadley regime. At higher rotation speeds, we would expect an unstable flow, with the formation of eddies. The eddies in the tank are the analogue of the baroclinic instabilities that form in the atmosphere.

Lorenz's chaos theory[2] predicts that in a stable regime (with the tank spinning at a slow rotation rate, the Hadley regime), a small perturbation to the initial state (or a small model error, in our case), causes the parameters (such as temperature and velocity at each point) to diverge from their unperturbed state, though the perturbation does not grow. In an unstable regime (the fast rotation case), a small perturbation of a field will grow exponentially in time. Figure 3 shows a field (this can be thought of as temperature at a given point, for example) with a small perturbation at time "A". In the unstable case, the perturbation causes the fields to diverge, while in the stable case, the fields remain close together.

[2] Lorenz (1963)

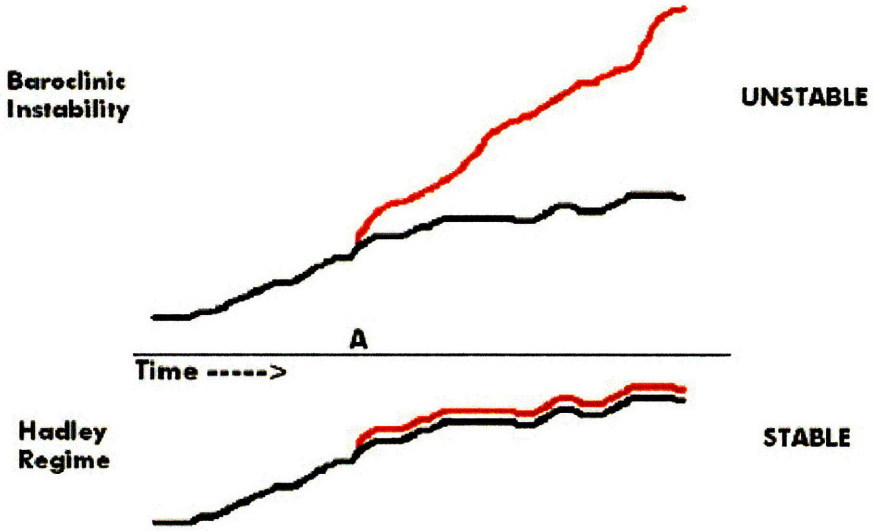

Figure 3, Evolution of a field with (red) and without (black) a small perturbation at time "A". Upper: high rotation speed (baroclinic instability)– the red line departs rapidly from the black, Lower: low rotation speed (Hadley regime) – the red line remains close to the black line. The real atmosphere at mid latitudes is unstable, and is therefore most similar to the upper case.

In the fast rotation experiment, we expect errors in the initial conditions and model errors to be amplified dramatically. Therefore, we will use real time model data assimilation to compensate for these errors. Figure 4 shows how assimilation can correct these model errors at each timestep, preventing them from growing exponentially.

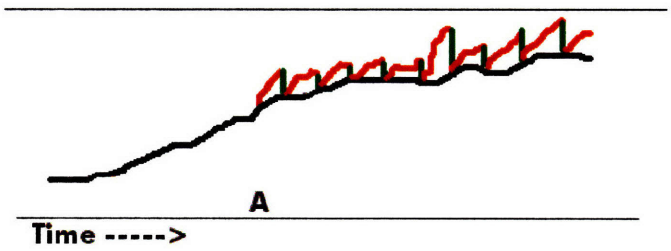

Figure 4, Constraining a parameter with assimilation. Red indicates the perturbed field, with the green being the assimilative corrections.

In this particular project, we create the technology needed to perform these assimilative steps. The system we create will need to be extensible, and able to accept various data assimilation modules (allowing the efficacy of the various schemes to be explored and tested). We test the system using one such assimilation method, developed by Dr. Sai Ravela. In the future, the data assimilation scheme can be switched without requiring any modification to the technology.

2 Setup of the Laboratory Experiment

The setup for the experiment has two main components, the experiment itself, and the data acquisition system.

2.1 Rotating Tank

The rotating apparatus is a cylindrical water-filled tank with a metal can in the center. The tank is filled with room temperature distilled water, laser sensitive particles (for the purpose of particle tracking), and enough salt to make the particles neutrally buoyant. The particles must be mixed with photograph processing solution prior to being placed in the tank, to avoid coagulation. The tank is placed on a rotating turntable, with the rotation rate determined by the experimenter. After about twenty minutes, the tank comes into solid body rotation. At this point, a chiller inserted in the central metal can is activated, cooling the center of the tank. See Appendix C for the complete setup information.

The physical system represents a complex set of interactions with its environment. To make the physical system tractable, we make certain assumptions[3] that we consider reasonable. There are no slip conditions on the bottom of the tank, the inner can, and the outer rim of the tank. This means that infinitesimally close to these surfaces, the velocity of the fluid is 0. Through the outer rim of the tank, and the bottom of the tank, we assume heat flux ($H = -k \nabla T = 0$). We also assume an insulating boundary condition at the air/water boundary. At the inner cylinder, we assume that the vertical temperature profile is known, and fixed, as a function of height. The water in the tank has an initial temperature profile. The temperature is approximately the room temperature (assuming the tank has been sitting in the room for a number of hours). Due to random processes, the initial temperature profile may vary slightly throughout the tank. The diffusion for the water in the tank is 10^{-7} m^2/s, and the viscosity is 10^{-6} m^2/s at laboratory temperature (23 degrees C).[4]

3 This describes the physical assumptions we made about the tank setup. The model assumptions are given later.

4 Marshall et. al. (2002)

Figure 5 shows the complete tank setup, with the boundary conditions labeled.

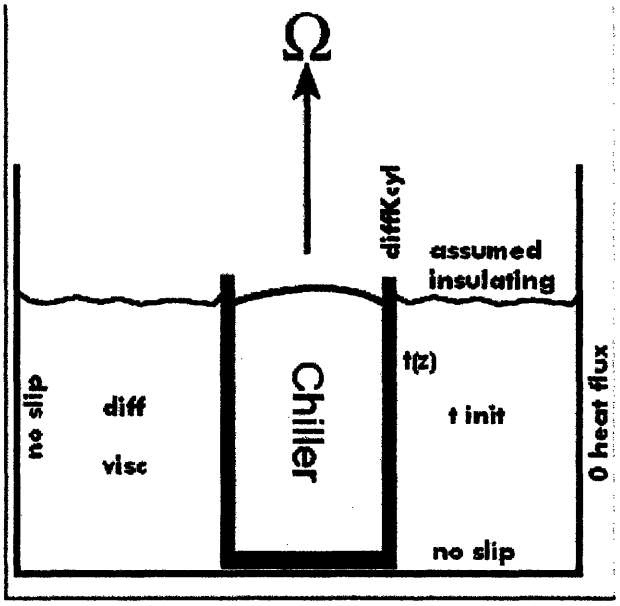

Figure 5, Schematic of the tank setup indicating initial boundary conditions.

In a real-world setting, the chiller in the can represents the cold pole, while the outside of the tank represents the equator. Therefore, we would expect to see a westerly "jetstream" develop in the tank as it does at mid latitudes on Earth.

To collect data from the experiment, we developed a high-tech system to collect velocity data from the tank in real time.

2.2 Electronic Data Collection

The data collection system (shown as a schematic in Figure 6 and photographs in Figure 7, Figure 8, Figure 9, Figure 10, and Figure 11) is composed of a green laser, a movable mirror, and a high-resolution digital camera. The particles that we add to the water are highly visible in green laser light. We use a mirror to direct the laser light to form a sheet at a single level of the tank, and photograph the illuminated layer from above with a digital camera. The digital camera is in the rotating frame (supported by a set of steel towers), so we use a fiber optic rotating joint (FORJ) to allow the data to cross into the stationary frame (after using a PHOX device to convert the electrical signal of the camera to a light signal that can be passed using fiber optics, and then a second PHOX device to convert the signal back into the laboratory frame), grabbing up to fifteen 2048 by 2048 pixel images per second. Using a Particle Imaging Velocimetry (PIV)[5] program, we can

[5] In PIV, the recorded image is divided into small interrogation windows typically 64x64 pixels in size. During the time interval dt between the laser shots the particles of each interrogation window have moved by a displacement ds. The velocity is then simply given by the ratio ds/dt. The calculation of the particle

take two snapshots of each layer in succession yielding a velocity profile at that level[6]. After moving the mirror so the laser shines on a different level of the tank, we repeat the process. In this manner, we can construct a three dimensional version of the current tank velocities, which can then be analyzed and compared to model runs.

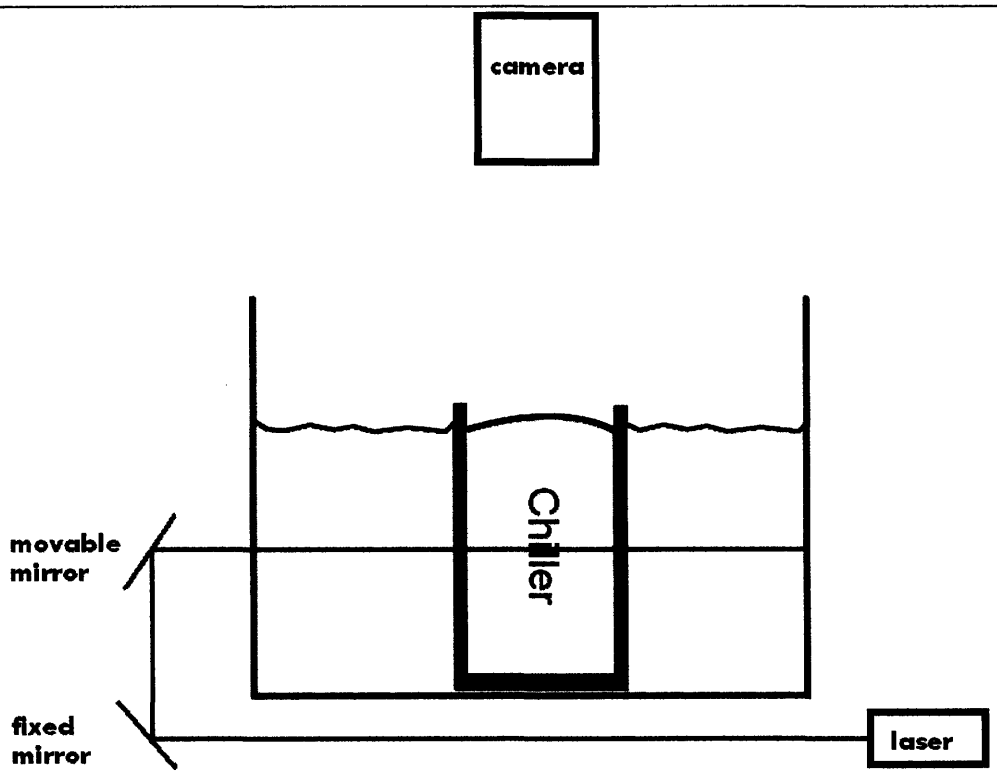

Figure 6, Diagram of the data collection system. The laser is directed at a pair of mirrors, one movable, and one immobile. The movable mirror, controlled by the graphical interface, directs the laser light to form an illuminated sheet in the tank. The camera sits above the tank, in the rotating frame.

displacement ds is done by cross-correlating the two corresponding interrogation windows. For high-speed calculation this is typically done via a Fast Fourier Transform using the Wiener-Kinchin (http://www.cs.huji.ac.il/~randoms/WienerKinchin.pdf) theorem. (from http://www.piv.de/piv.pdf)

[6] The image is split in half and transmitted over the network to two PIV servers. Each server computes velocity vectors in its half, and then the vectors are transmitted back to the data collection system.

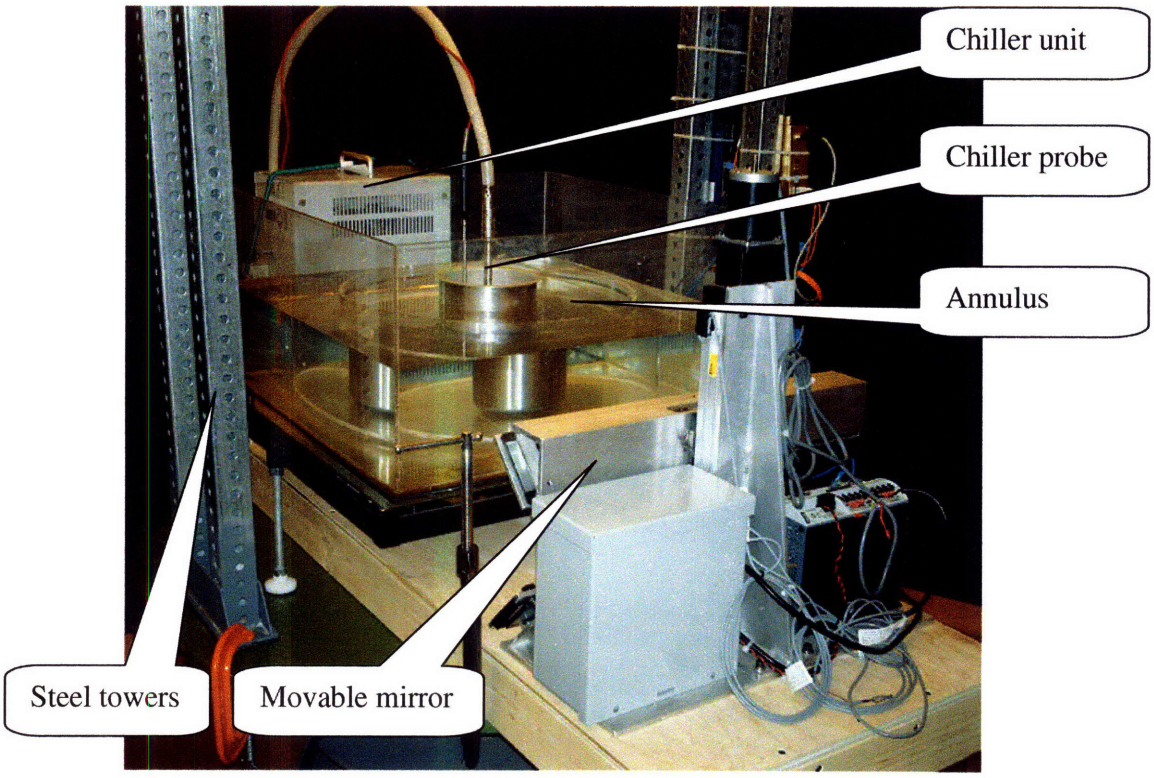

Figure 7, The lab apparatus, showing the steel towers (which support the camera and FORJ). In the rear is the chiller unit, with its probes entering the metal can in the center of the inner tank. In the foreground is the movable mirror.

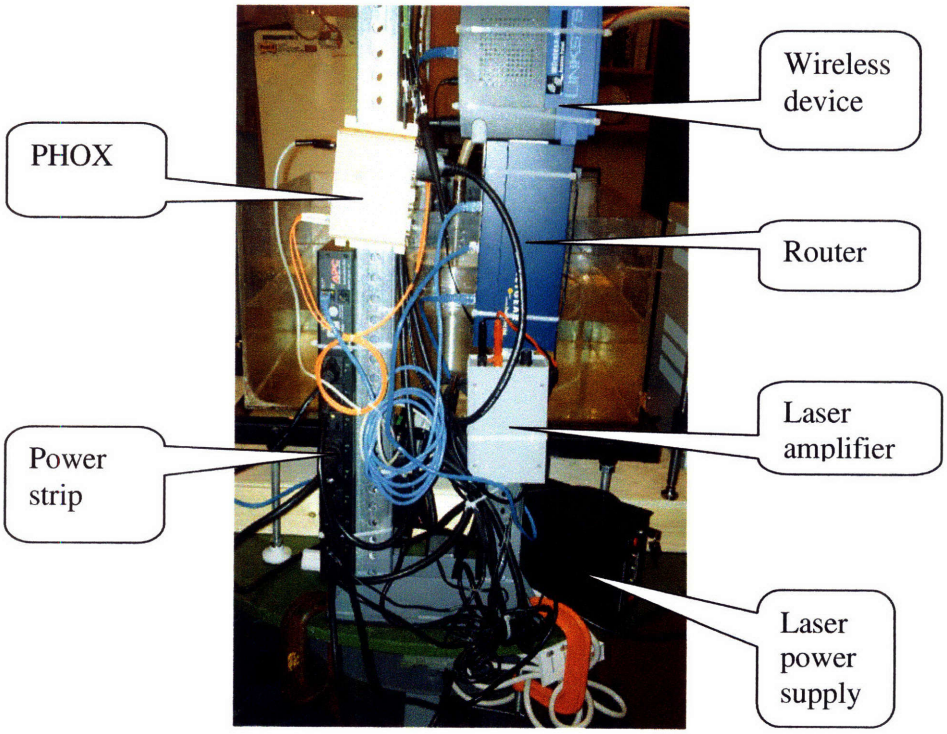

Figure 8, Devices mounted on the steel tower. Clockwise from the upper right: wireless device, router, laser amplifier, laser power source (on table), computer controlled power strip, PHOX.

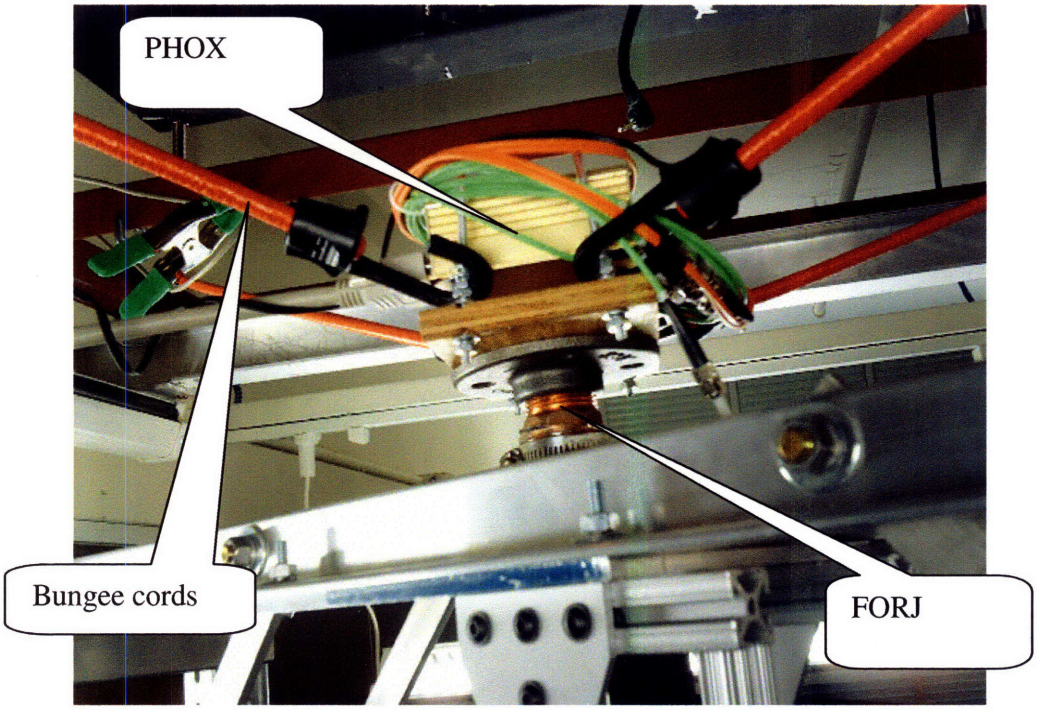

Figure 9, The FORJ, held in place by bungee cords.

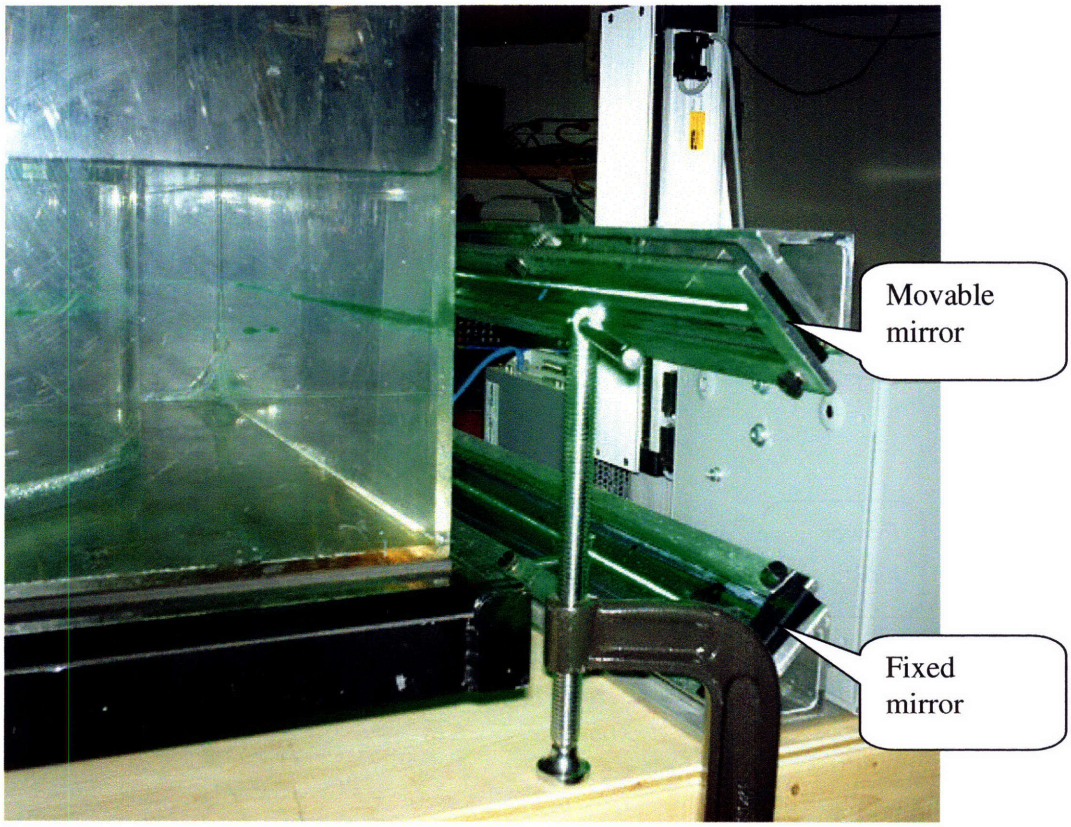

Figure 10, The movable mirror, with the laser in operation.

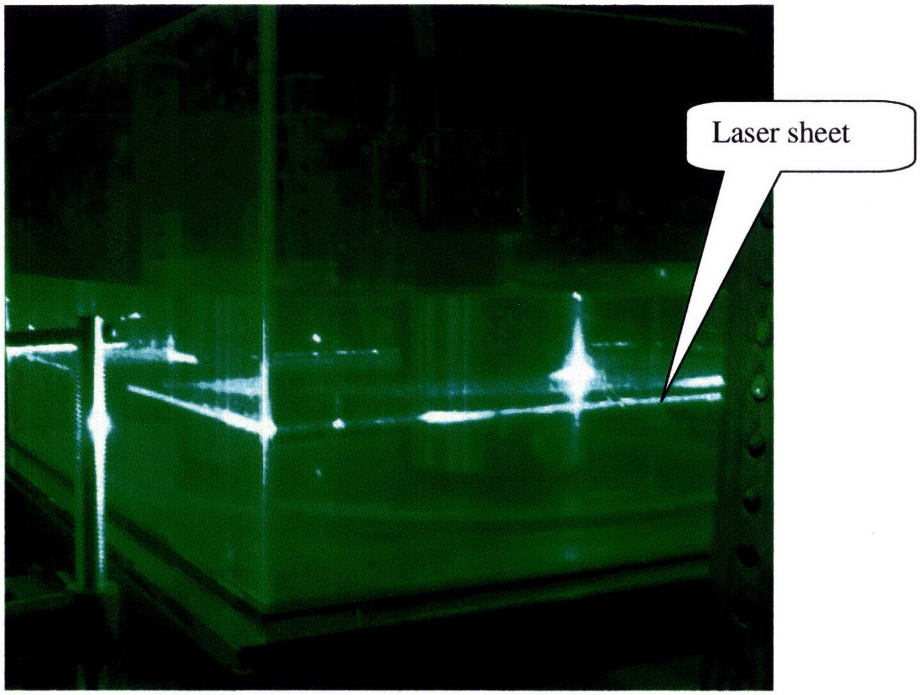

Figure 11. The tank, with the laser illuminating one layer of fluid.

We developed a graphical user interface, shown in Figure 12, to control these systems. The movable mirror, camera, and PIV can be operated from the interface. (In addition, we can switch on and off individual outlets on the power strip – see discussion on this later. This allows us to also control the chiller and laser from within the interface.) We tested the interface under various conditions.

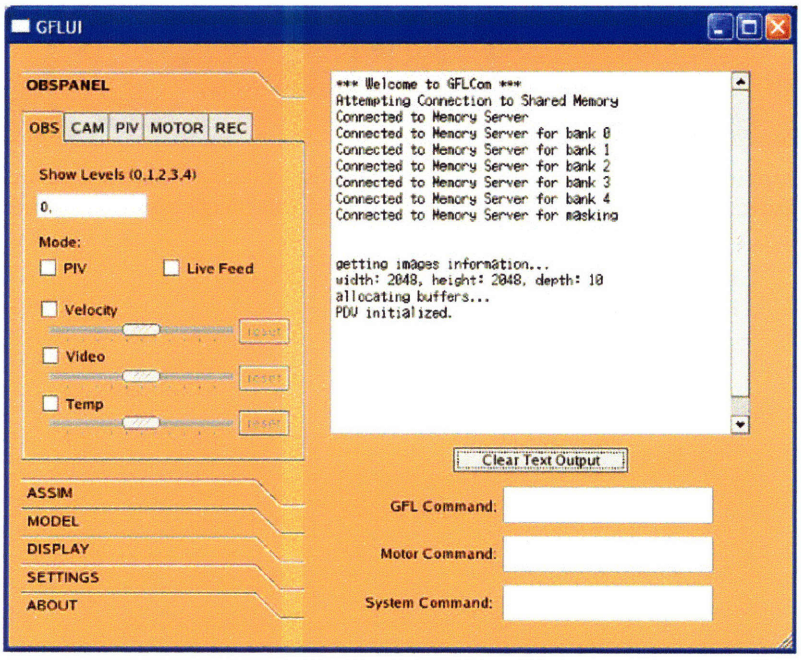

Figure 12, Graphical interface.

Figure 13 is an example of a picture that was taken by the camera. In this setup, there are laser fluorescent particles in the water of the tank, and the laser is on, illuminating them. The graphical interface displayed this image on the screen.

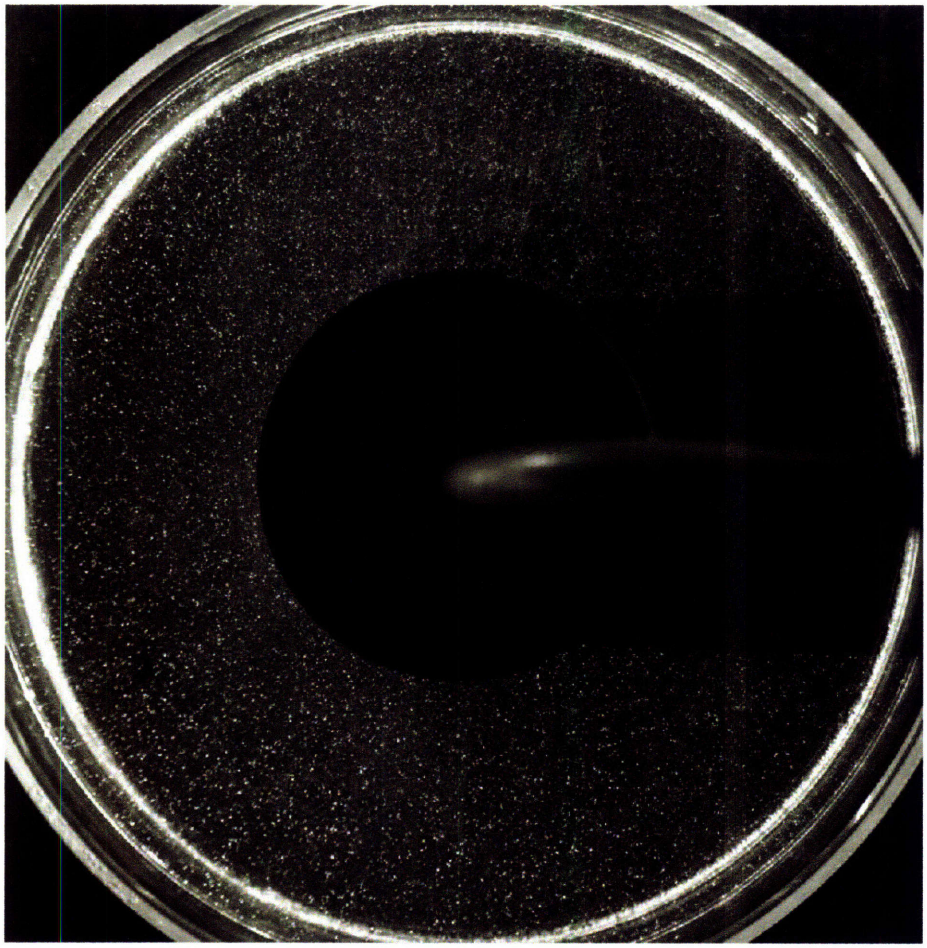

Figure 13, Graphical user interface's display of the laboratory tank. There are particles in the tank which scatter the laser light – the small white dots in this image captured by the camera.

In addition, we can use the interface to display the PIV vectors overlaid on top of the camera's image. Displaying these vectors allows us to easily see where the flow is strongest, and to see where the eddies have formed in the tank. Figure 14 shows the display of the interface with the vectors activated.

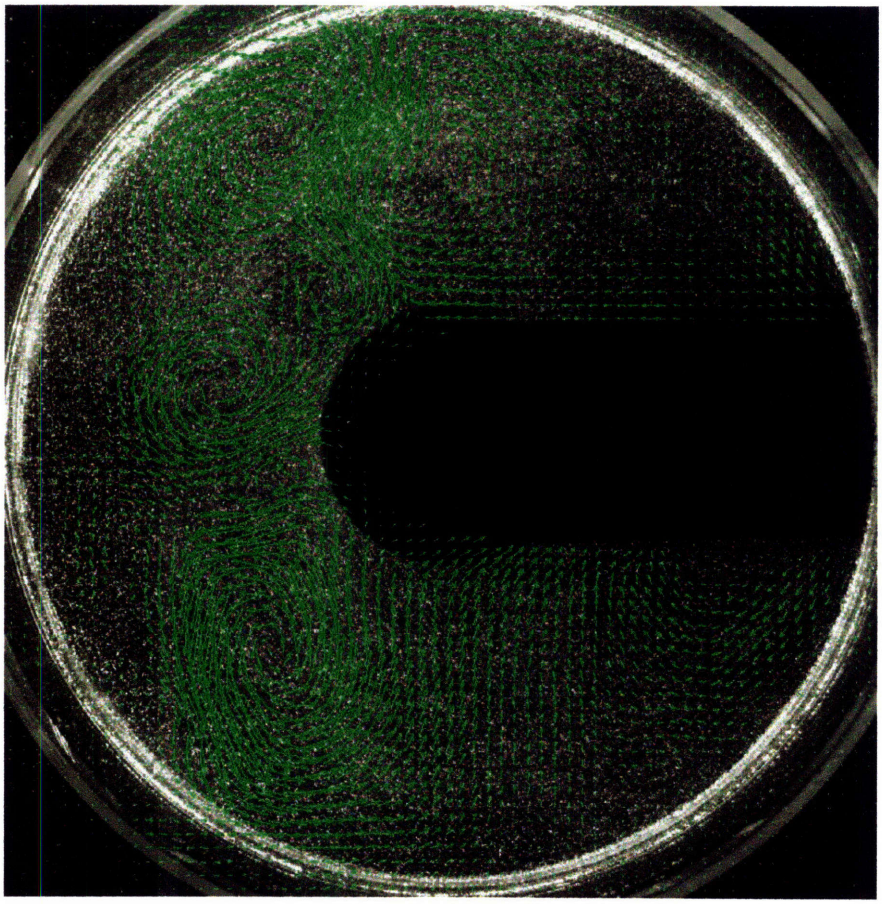

Figure 14, PIV vectors overlaid on the tank image. The green arrows represent the flow computed by PIV for this model run. A number of eddies of varying size can be observed.

2.3 Thermometer Calibration and Experiments

The hobo-thermometers used to determine the temperature of the water and chiller during the experiment record data every half second. Collecting this data allowed us to set the computer model initial conditions to more accurate values for temperature both in the chiller and in the water. In addition, results of the experiments with these thermometers made us realize that the vertical temperature structure near the chiller was a function of height, and not constant, as we had originally assumed.

2.3.1 Experiments

For the actual tank experiments, we used six thermometers.[7] To determine the temperature profile in the tank (to provide initial conditions for the model), we ran several experiments with the thermometers placed at various points in the tank and chiller. We measured the temperature at the top and bottom of the tank on the outside, the top and bottom of the chiller, and the ambient temperature of the water in the tank.

Figure 15 shows a complete temperature profile for one experiment. The blue line is measuring the temperature at the bottom of the inside of the can and the red line is measuring the temperature at the top of the inside of the can. The ambient air temperature of the room was 23 degrees C. Once the experiment is underway, the temperatures remain relatively constant.

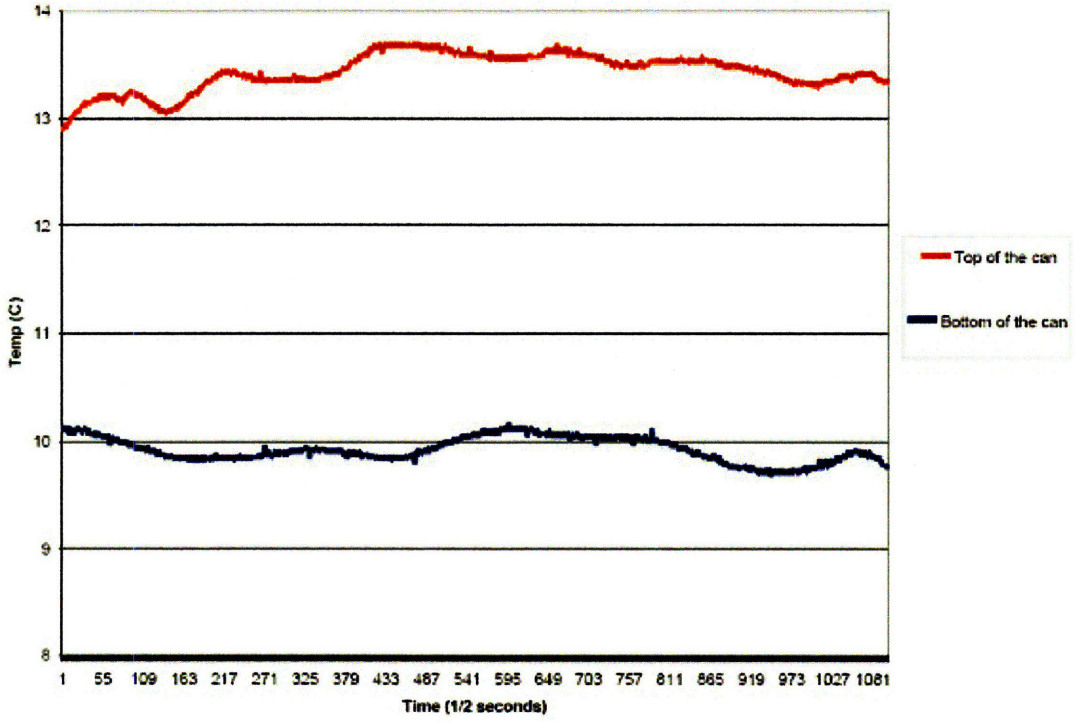

Figure 15, Tank experiment temperature data. The red line is the temperature near the top of the center can containing the cooling device, while the blue line is the temperature near its base. Once the experiment is underway, the temperatures remain relatively constant. The gentle undulations are due to passing eddies.

We used these results to set our parameter file for many of the model runs. These experiments made us realize that not only was the temperature outside the chiller considerably higher than 0° C, but that there was also a substantial vertical temperature gradient.

---

7 See Appendix B.

3 Numerical Model

We can represent the fluid flow in the tank using a computer model. In this project, we are using the MITgcm[8], a computational fluid dynamics code that can simulate flow in an incompressible, baroclinic fluid, such as our tank. MITgcm is a discrete numerical solver of the equations of fluid motion. The model is comprised of an executable file, created from code written in the FORTRAN language, and a set of runtime parameter files. These parameter files can be modified without having to recompile the entire model code. In addition, we have developed a graphical user interface to easily create the runtime parameter files. In this project, we also modified the FORTRAN source code to add a number of scientifically beneficial features to the model that pertain to our setup.

3.1 Solving the Equations

The model uses a timestep loop to solve the equations for motion. The following is the basic scheme of the model step:

For each timestep:

1. Maintain a state vector with fields such as u-velocity, v-velocity, temperature, salinity, and pressure, together with the time derivatives of these.
2. Compute the new state vector at the timestep using a series of function evaluations.
3. Repeat steps 1 and 2 for every model grid cell.

3.1.1 Diffusion and Heat Flux

To give an example of how the model calculates the heat flux out of the metal can at the center of the tank (one of the many function evaluations performed at each timestep of the model run), we will examine a small piece of the model source code from the file external_forcing.F.

```
            faceArea = drF(kLev)*dyG(i+1,j,bi,bj)
            dFlux =
     &       -faceArea*kDiffCyl*(tCyl -
theta(i,j,kLev,bi,bj))
     &       *recip_dxC(i,j,bi,bj)
```

8 Marshall, et. al. (1997)

These equations are evaluated at every model grid point on the outside of the can (as denoted by the subscripts in the FORTRAN code). They perform the following actions:

- Determine the area of the grid cell on the can by multiplying the grid spacings
- Compute the flux as follows:
  - Multiply the area by the diffusion coefficient of the cylinder, kDiffCyl
  - Multiply by the difference in temperature between the cylinder at the current grid point and the water just outside of the cylinder
  - Divide by the grid spacing in the theta direction (the model considers the theta direction to be x)

The following discrete formula represents the continuous equation for the heat flux, $H = -k\nabla T$, where ∇T is the temperature gradient:

$$H = \frac{-k\Delta A(T_{can} - T_{fluid})}{\Delta x}.$$

3.2 Model Grid

Although the fluid flow is itself continuous, the computer model solves discrete equations on a grid. For the rotating tank of this project we used a grid with 15 vertical levels, 23 radial cells (we do not solve the equations of the model within the chiller), and 120 slices at 3 degree intervals going around the tank. The length scales of each of these spacings is set using the parameter file, as described in a later section. Figure 16 graphically illustrates the model grid in its three dimensions. However, as shown in the figure, the vertical levels are not evenly spaced. We wanted to get higher resolution near the bottom of the tank, where frictional effects are greatest, and the boundary layer needed to be better represented.

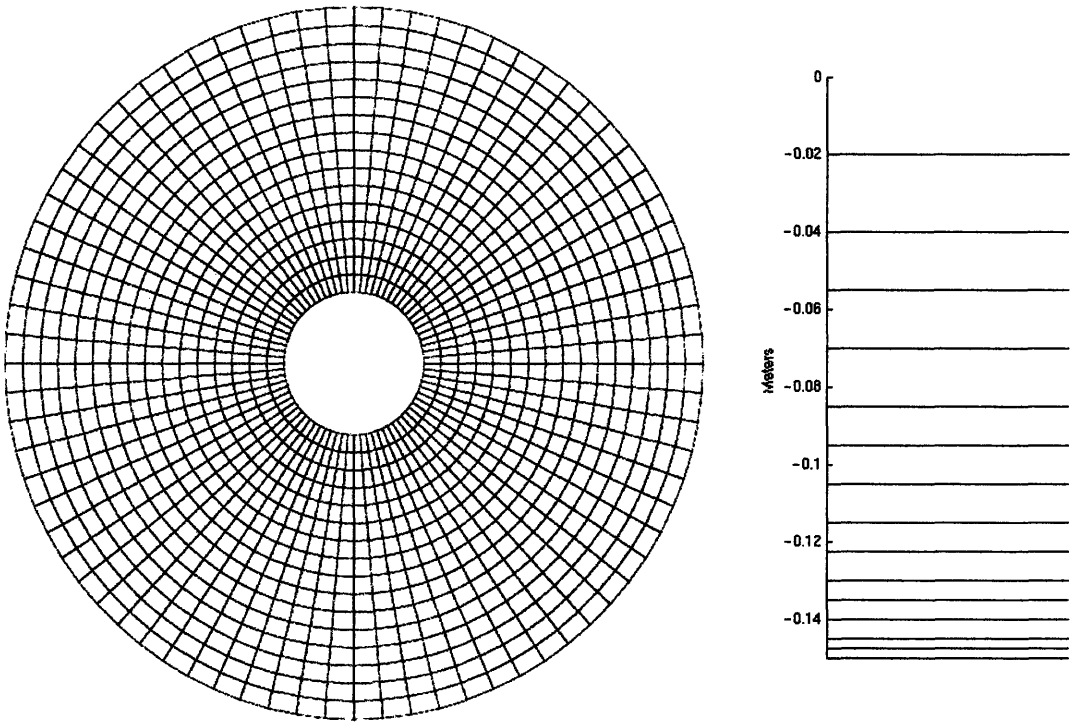

Figure 16, Two figures showing the model grid. Left: 23 radial cells and 120 slices in theta; Right: 15 vertical cells, with the diagram depicting their spacing within the model (the levels are closer together at the bottom of the tank, where frictional boundary layers need to be represented).

3.3 Model Parameters

A parameter file is required to run the MITgcm. This file contains information about such parameters as viscosity, diffusion, and rotation rate of the tank, as shown below:

Variable	*Data File Variable Name*	*Default Rotating Tank Model Value*
Water temperature in the tank	tRef	All vertical levels, 20 degrees C
Viscosity of the tank water	viscAh and viscAz	$5.0*10^{-6}$ m^2/s
Diffusion of heat in the tank	diffKzT and diffKhT	$2.5*10^{-6}$ m^2/s
Temperature of the inner can	tCylIn	0 degrees C
Temperature of the outer rim	tCylOut	20 degrees C

3.4 The Interface

We have created a graphical user interface to create and modify the parameter files for the model runs. The interface (Figure 17) is based on past work of Dr. Constantinos Evangelinos.[9] In addition, the interface can execute the model with the click of a button. See Appendix B for more information on this interface.

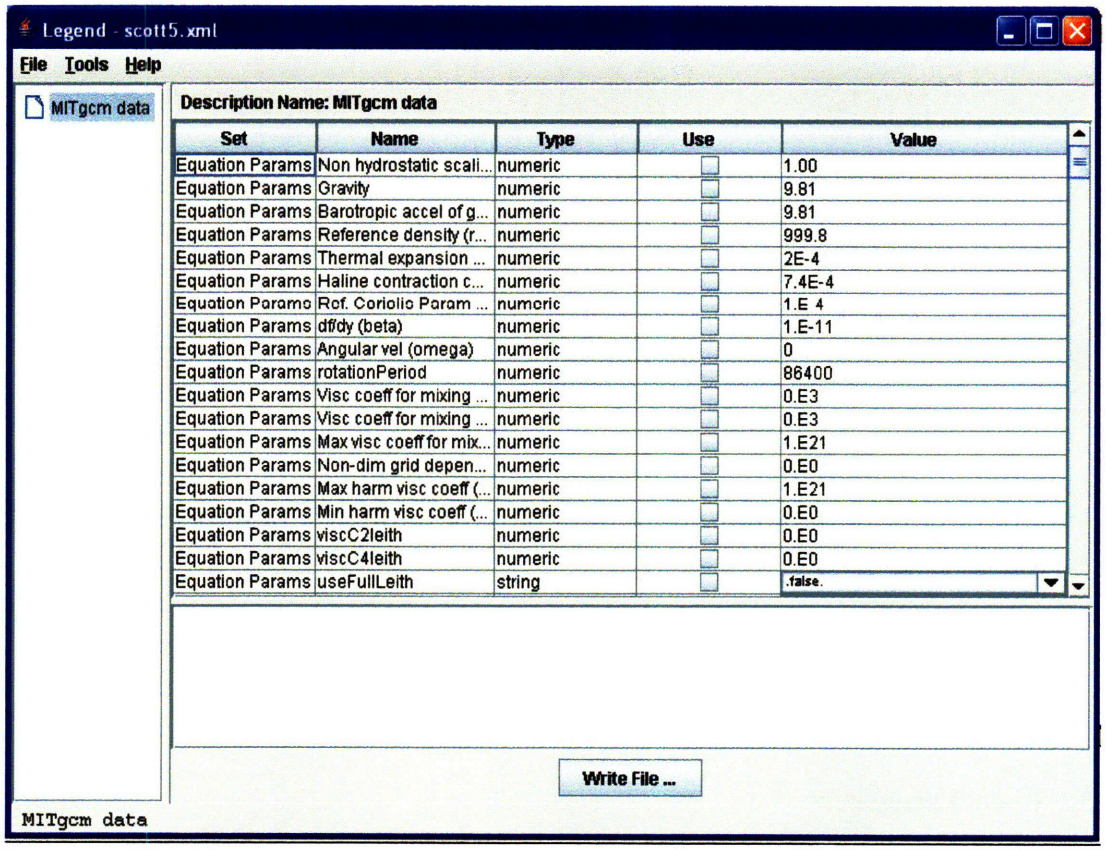

Figure 17, Interface for controlling the model parameters.

3.5 Model Output

The model makes use of various output methods both to screen and files. At each timestep in the model run, a line of output is displayed on the screen. In the parameter file, the user can set how often more detailed statistics are displayed to the screen. These detailed statistics show values such as the mean temperature in the tank at the current timestep.

Files are also saved to disk at user-defined intervals (also set in the parameter file). Files for the following parameters are saved: temperature, u-velocity, v-velocity, vertical motion, and salinity.

---

9 Evangelinos, C. (2005)

3.6 Model Output Analysis Scripts

To visualize the model output in various manners, we wrote Matlab scripts that take the model output files and create pictures and movies. There are three types of output visualizations: color contour maps of horizontal sections at any level, velocity vector plots of horizontal sections at any level, and color contour maps of vertical sections at any of the slices around the tank. For any of these types of plots, we can analyze one timestep and level as a single jpeg image, or create an avi movie that advances either through time or space. Movies that advance through time can show any number of timesteps with any space between frames. Movies that advance through space must be at a given timestep and advance vertically for horizontal sections and around the tank for vertical sections. Again, the spacing between frames of the movie is user defined.

3.7 Model Modification

To better represent the Hadley regime (slow rotation) in the model, we realized that the actual GCM would have to be modified to accommodate additional parameters. We added parameters to the model, which would allow for independent boundary values for both diffusion and viscosity in the tank. We needed to be able to adjust the diffusion coefficient to represent a flux of heat through the side of the metal cylinder containing the chiller. We also had to allow for a vertical temperature gradient on the cylinder. Finally, for the fast rotation experiments, we wanted to introduce random perturbations in the initial temperature field to instigate baroclinic instabilities.

3.7.1 Boundary Diffusion

After running several experiments with the standard MITgcm rotating tank configuration, it became clear that too much heat was being diffused within the main body of the tank, while not enough cold temperature was being diffused out of the chiller's can. In fact, we would prefer that the diffusion coefficient within the body of the tank was equal to that of water, 10^{-7} m^2/s.

Scientifically, we reasoned that adding a boundary diffusion layer would be a way to represent the extremely small-scale processes that take place very close to the center can. These small-scale effects transfer heat to and from the metal of the cylinder. The rapid transport of heat near the can is due to both advection and diffusion. Because of the granularity of the model grid, we did not believe that we could properly model the advection close to the can. To compensate for this, we raised the diffusion in the grid cells directly next to the can. To avoid a sharp gradient of diffusion, we discretized the diffusion in the tank into three bands, beginning with a high value near the can. We used a medium value for the second band, and a low value (the value of the diffusion of water) further away from the center (this band contains the majority of the fluid in the tank). Each band's diffusion coefficient can be independently set. This counterintuitive method

allowed us to compensate for the lack of model grid resolution at the center of the tank, as shown in Figure 18.

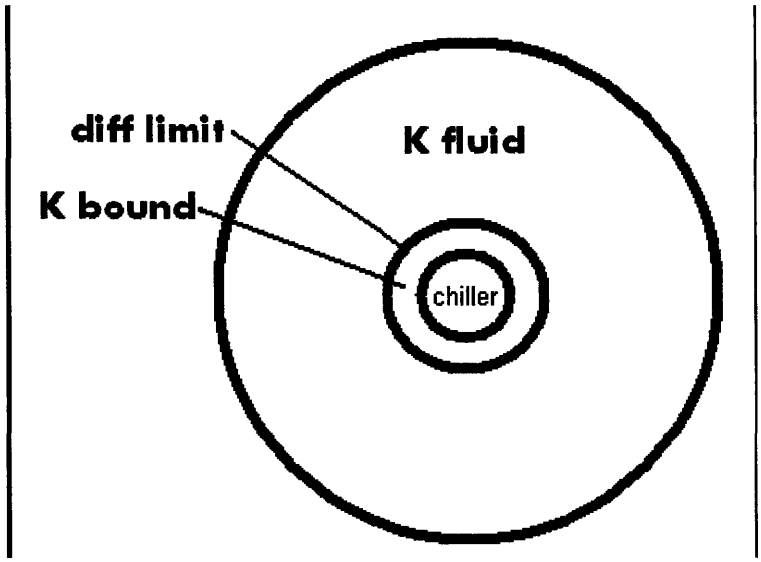

Figure 18, Schematic showing the diffusion values in the tank after our modification. We use the diffusion coefficient of water for the value of K_{fluid}. The value of K_{bound} is slightly higher to simulate the small-scale processes that occur right near the metal can in the center of the tank.

3.7.2 Boundary Viscosity

Similarly to the boundary diffusional layer, we felt that we could improve the model's ability to simulate the tank by adding a boundary viscosity layer. In this case the additional viscosity coefficient, viscAbound, acts only at the lowest grid cell within the model tank. To change the viscosity at the very bottom of the tank, we used the parameter bottomDragLinear, a linear coefficient of drag that acts between the bottom of the tank and the lowest water filled grid cell by the following equation:

$$\tau_b = L\mu + 2\mu \frac{visc_{interior}}{(\Delta z)}$$

3.7.3 Vertical Temperature Gradient

Based on our hobo-thermometer experiments, it was apparent that one single temperature for the entire center can would not be realistic. We decided that it would be best to change the model to be able to set a different temperature at each of the 15 vertical levels in the model grid.

3.7.4 Random Perturbations

In some model runs, even at high rotation speeds, the model did not go unstable and form eddies. We assumed that this was due to the temperature profile in the tank being perfectly symmetrical. In the actual laboratory configuration, this would be impossible to achieve. To compensate for this, and to create the eddies that should exist in a physical simulation, we added small random perturbations to the initial conditions of the model. Originally, the model uses a vector called tRef to determine the temperature at each of the vertical levels within the body of the tank, with the temperature being constant at each level. In our current modification, we randomly add or subtract a very small number (a small fraction of a degree) from each initial temperature in the model tank. This creates a tank with the average temperature at each vertical level equal to that of the tRef values, but with a minute variation from one point to the next on each level. These perturbations create instabilities which then grow into the synoptic scale eddies that we observe in the tank. Analogues of these random perturbations in the real life tank experiment could be a slight temperature gradient in the tank to begin with or air currents in the laboratory room.

Figure 19 shows the temperature field at the bottom level of the tank with the random perturbations after 100 timesteps (or 10 seconds) of model time.

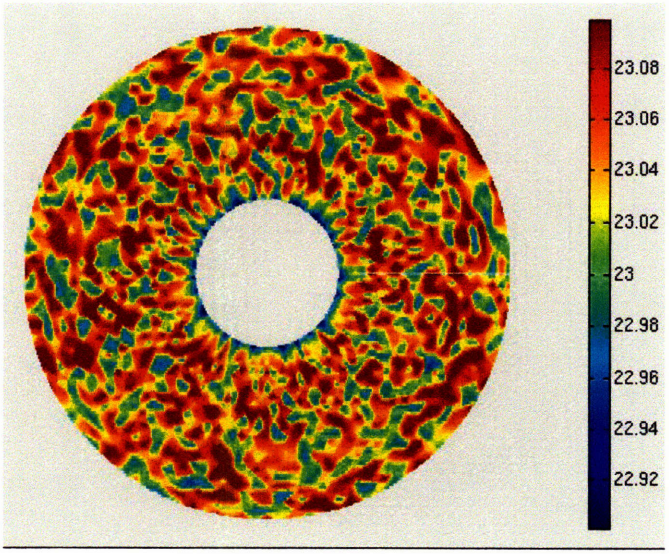

Figure 19, Random temperature perturbations after 100 timesteps (10 seconds). The scale is degrees Celsius.

4 Experiments

In this section, we describe the various experiments that we performed to determine the optimal model parameters for simulating the tank in the two regimes: slow rotation and fast rotation.

4.1 Slow Rotation

We began by running the model at slow rotation speeds and fine-tuning the parameters to make the model match the tank experiment runs that we had performed. It was our hope that if we properly tuned the model at slow speeds, it would scale to fast rotation without changing any of the parameters.

For our slow rotation experiments, we used an f-value of 0.1, equivalent to a rotation period of about two minutes. As expected, the Hadley regime is stable and has no eddies. The flow has a general outward flow at the bottom of the tank and a general inward flow at the top. Because of angular momentum conservation, the flow does not move in straight lines; instead, it spirals outwards.

The following (Figure 20 and Figure 21) are two vector fields of the flow in the tank experiment. The first one is at the bottom of the tank, while the second one is at the top of the tank.

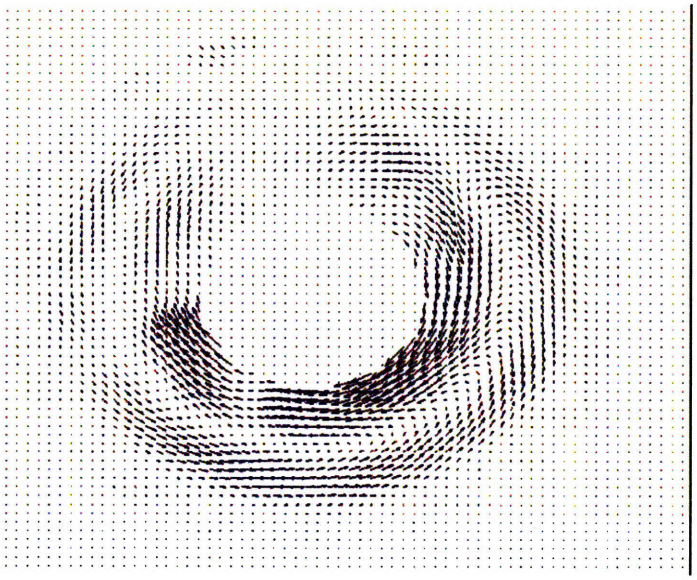

Figure 20, Hadley regime flow at the bottom of the tank. This data field is from PIV. The flow speed is on the order of a few millimeters per second.

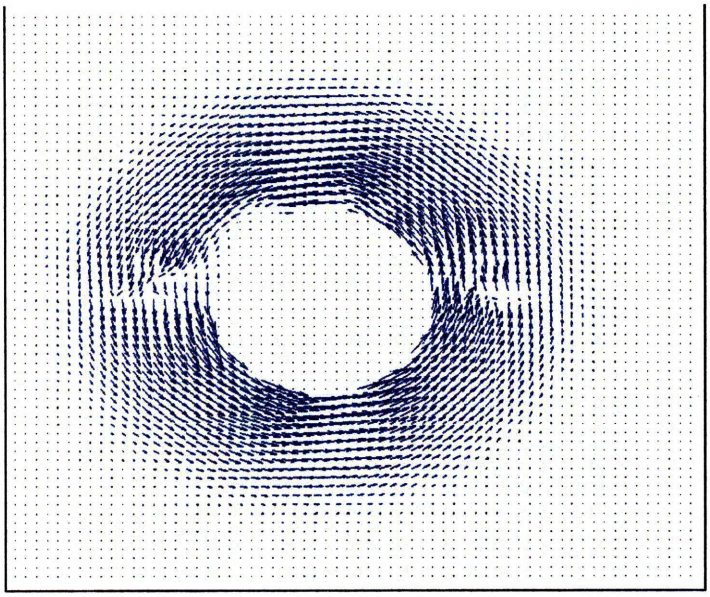

Figure 21, Hadley regime flow at the top of the tank. This data field is also from PIV.

Figure 22 shows a vertical section of velocity (in the theta direction) in the tank, averaged over 6 movable mirror passes (where the mirror allows data to be taken at 50 levels per pass). Red colors indicate counterclockwise movement, while blue colors indicate clockwise motion. The data is plotted in mm/s.

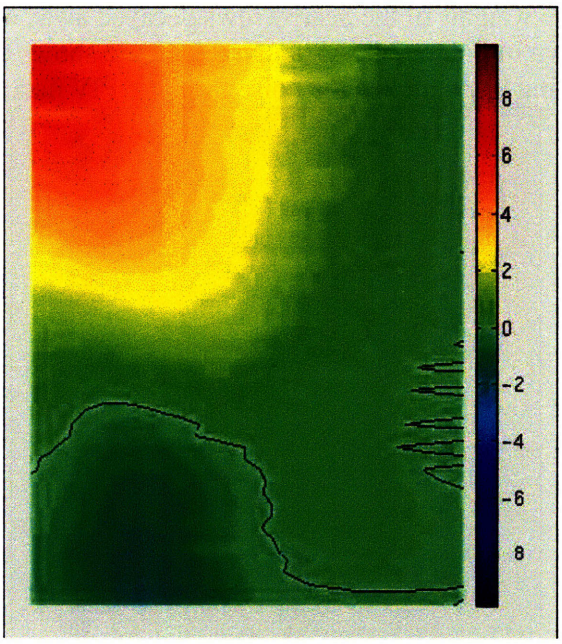

Figure 22, Vertical cross section of the azimuthal tank velocity (from PIV). Units are millimeters per second. There is strong counterclockwise flow near the interior top, weaker clockwise flow near the interior bottom, and little flow on the outside of the tank

Since we wanted to compare the vertical shear of the model to the vertical shear in the tank, we computed du/dz. The data from the previous chart is messy, leading to an unphysical shear profile, as shown in Figure 23:

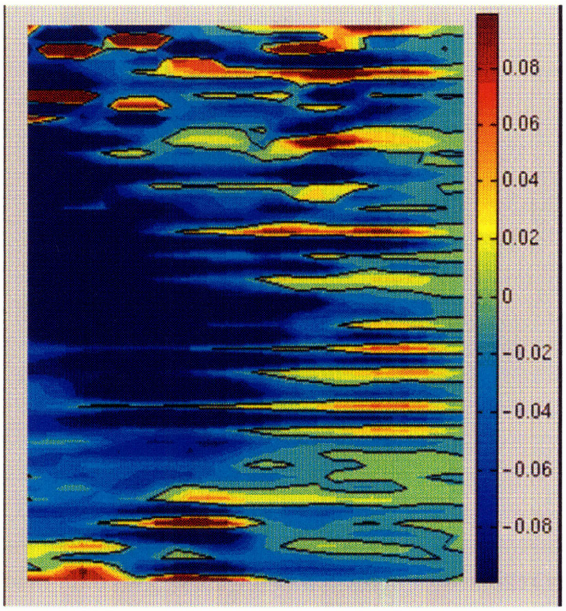

Figure 23, Vertical shear in the tank, without smoothing (s^{-1})

Since the data in Figure 23 is relatively messy, we applied a smoothing algorithm. The smoothing procedure was row-wise. For each row in the figure, we computed a weighted average of that row and the 3 rows above and below it. The row we were computing was weighted most, and the weights of the nearby rows decayed away as a Gaussian, yielding the velocity field shown in Figure 24. Although this figure shows the same data as Figure 22, the smoothing has removed the noise from the PIV data.

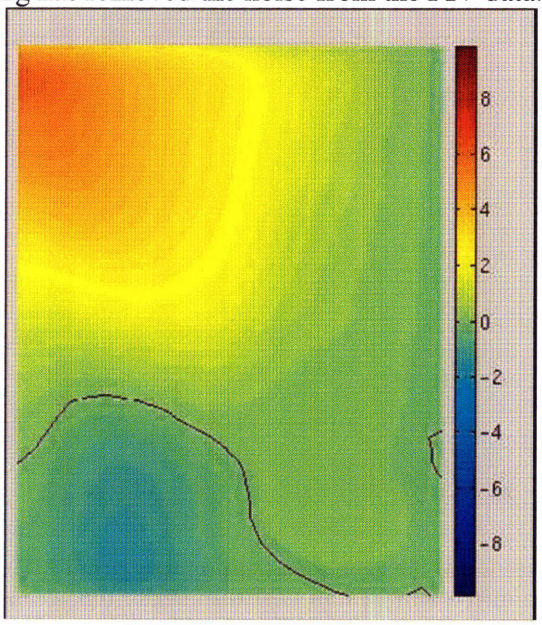

Figure 24, Vertical cross section of the azimuthal velocity in the tank after processing with the smoothing algorithm (mm/s)

Finally, we took du/dz of the smoothed data, and plotted the shear in the tank (Figure 25), which, by the thermal wind equation, we expect to be equal to the horizontal temperature gradient (adjusted by constants):

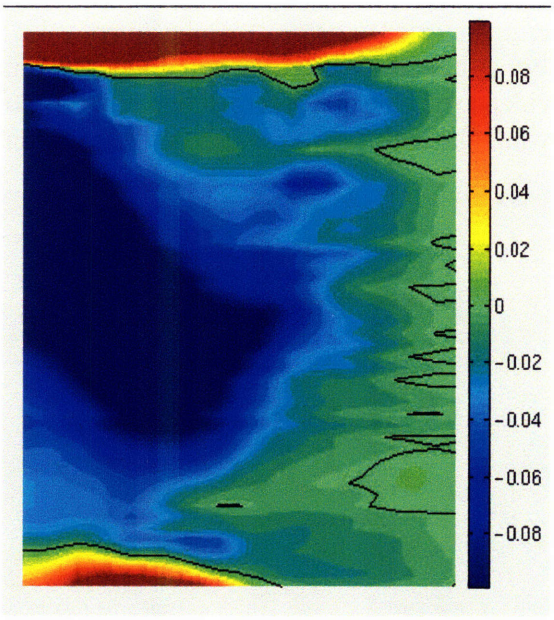

Figure 25, Vertical shear in the tank, with smoothing (s^{-1})

4.1.1 Slow Rotation Experiment 1

Figure 26 shows a comparison of the velocity profile in the model run and the velocity profile in the tank experiment, prior to parameter adjustment. We noticed that these profiles are very different in both the magnitude and direction of the velocity field. The model run shows no evidence of a zero-crossing (a horizontal layer where there is no velocity whatsoever), while the tank experiment shows this crossing.

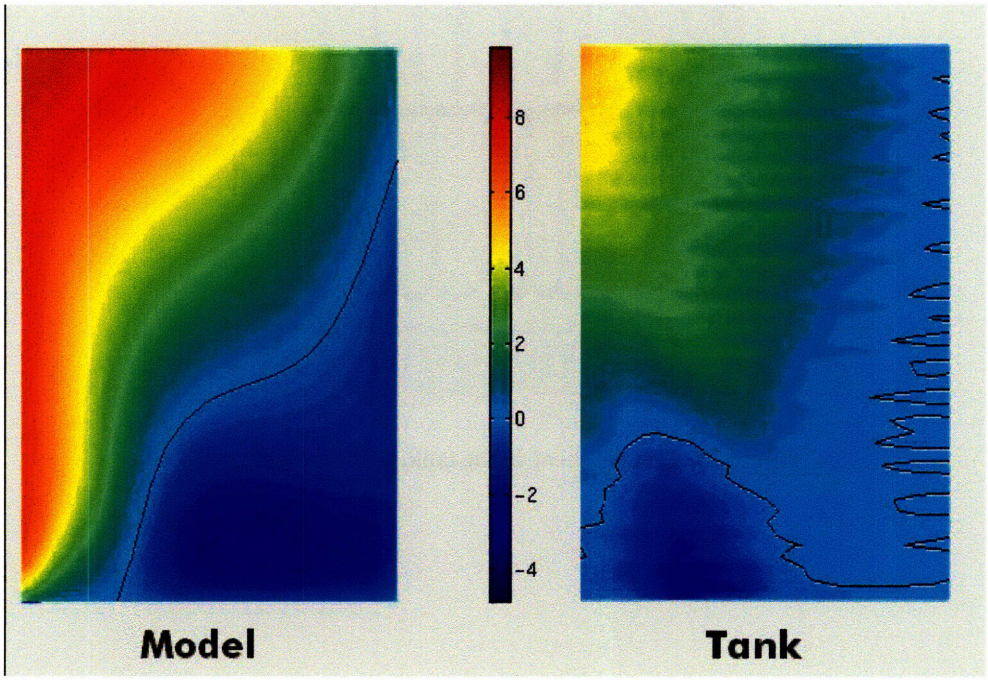

Figure 26, Direct azimuthal velocity comparison of the model and the tank, prior to parameter adjustment (mm/s)

Figure 27 analyzes the thermal wind balance in the model. The chart in the upper left shows the model's horizontal temperature gradient, adjusted by the appropriate constants. The upper right shows the vertical shear in the model. The lower chart shows the difference of the two. If the model were in perfect thermal wind balance, the lower chart would be zero everywhere except the frictional boundary layer, where thermal wind does not apply.

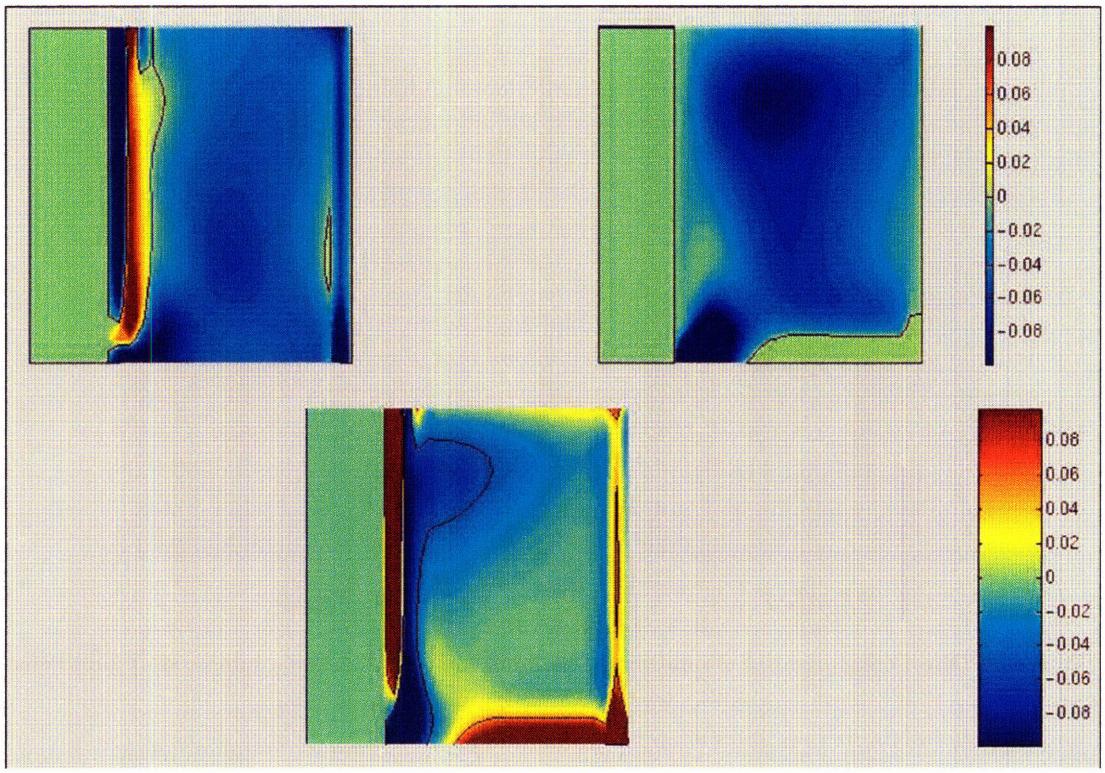

Figure 27, Upper left: horizontal temperature gradient in the model (multiplied by appropriate constants from the thermal wind equation), upper right: vertical shear in the model, lower: difference between the two upper charts.

4.1.2 Slow Rotation Experiment 2

After applying the various changes to the model (boundary diffusion, boundary viscosity, center can temperature profile) and adjusting existing parameters (such as the value of diffusion of the water in the can, viscosity parameters, slip conditions, usage of hydrostatic balance, and others), we arrived at a model configuration that was able to simulate the velocity profile of the tank experiment.

As above, we created a set of plots to analyze the thermal wind balance in this run. The velocity profile in the model is also close to that of the tank experiment. We see a similar zero-crossing in both cases. The magnitudes and directions of the velocity also match. The thermal wind balance of this simulation is nearly perfect, as shown in Figure 28 and Figure 29, suggesting that the Rossby number in the experiment was very small.

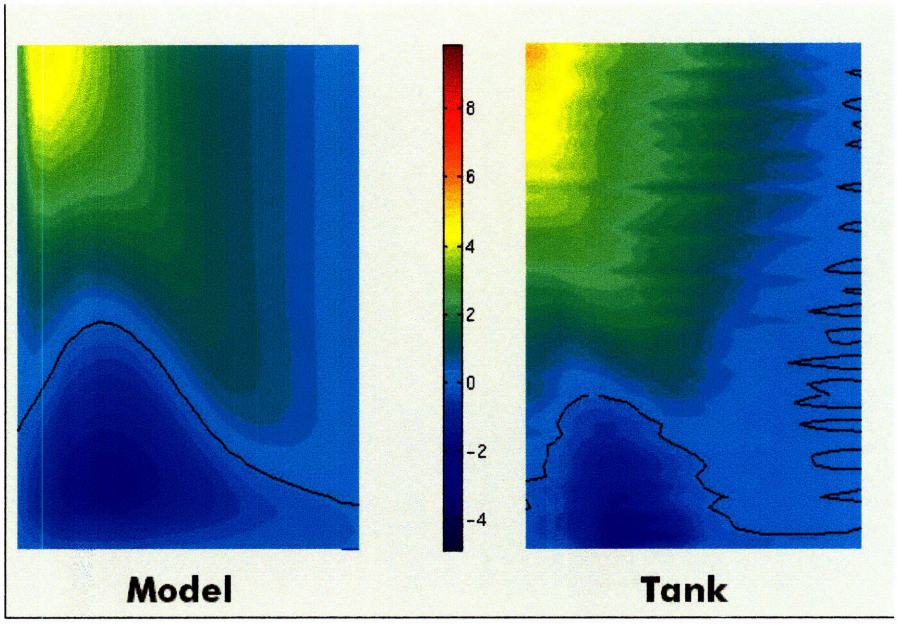

Figure 28, Direct comparison of the model and the tank (azimuthal velocity, mm/s), after parameter adjustment.

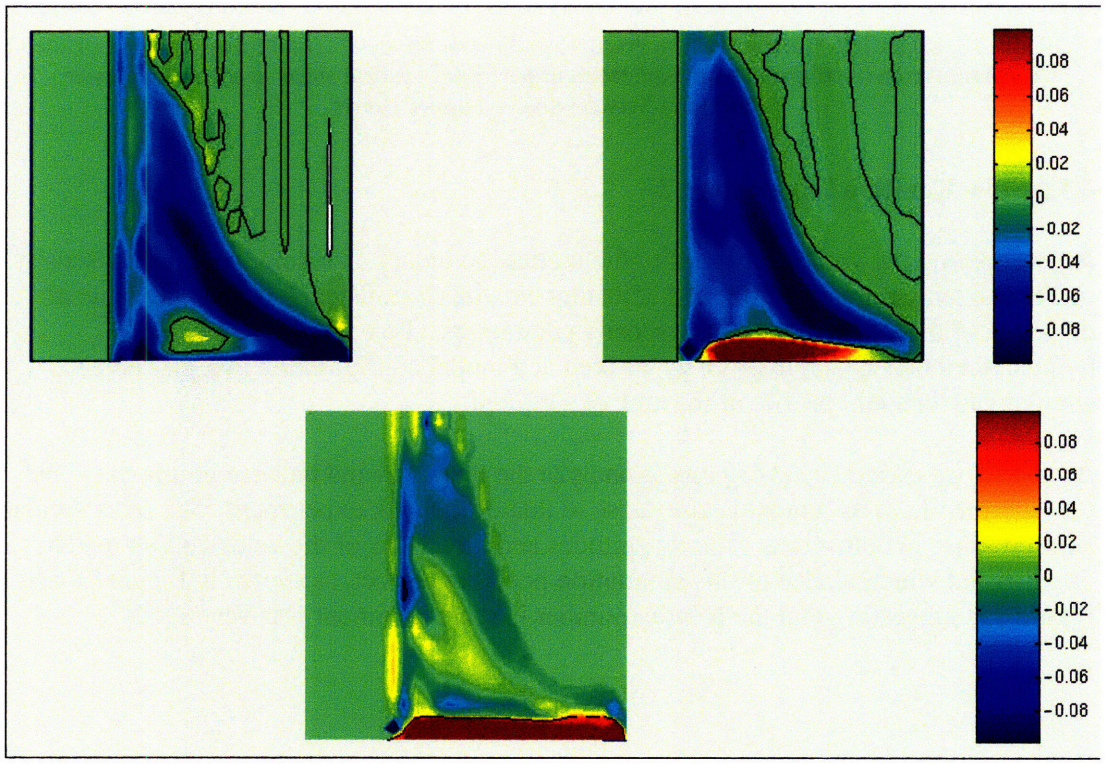

Figure 29, Upper left: horizontal temperature gradient in the model (multiplied by appropriate constants from the thermal wind equation), upper right: vertical shear in the model, lower: difference between the two upper charts.

4.2 Fast Rotation

In the fast rotation experiments, we used f-values of 1.0 and 2.0, equivalent to rotation periods of twelve and six seconds respectively.

After turning the chiller on at fast rotation speeds, we observe several eddies forming very quickly (on the order of 1 or 2 minutes). These eddies persist for the duration of the experiment, though they progress around the tank slowly and do change shape over time.

The vector field in Figure 30 shows eddies observed in a high rotation speed tank experiment. The upper center is in the can's shadow.

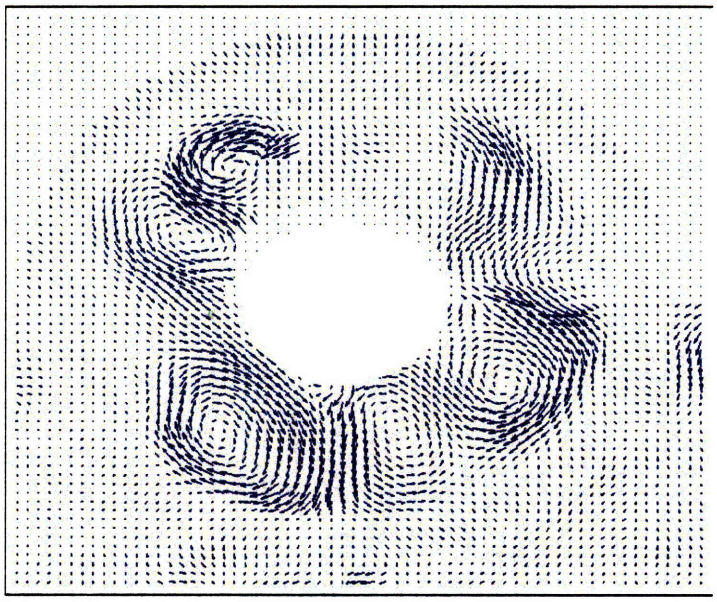

Figure 30, Velocity profile of the middle level of the tank at a high rotation speed, from PIV data. The region in the upper center is the shadow of the chiller can. The velocities are on the order of millimeters per second.

4.2.1 Fast Rotation Experiment 1

Figure 31 shows the temperature field that the model produces in its default configuration at high rotation speeds (in this case a 12 second rotation period, or f=1). It has five main eddies at first. Unlike in the tank experiment, these eddies disappear as time progresses, and are replaced by less distinct structures, as can be seen in Figure 32.

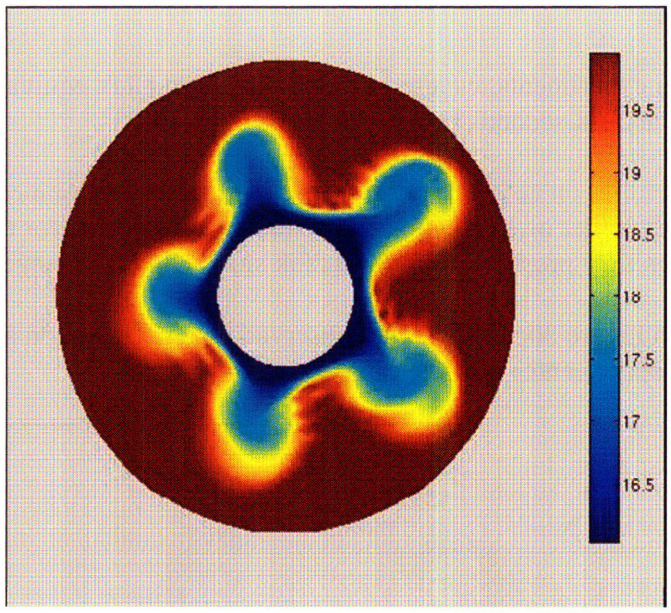

Figure 31, Model temperature field in its default configuration, after about 30 rotation periods, showing five eddies.

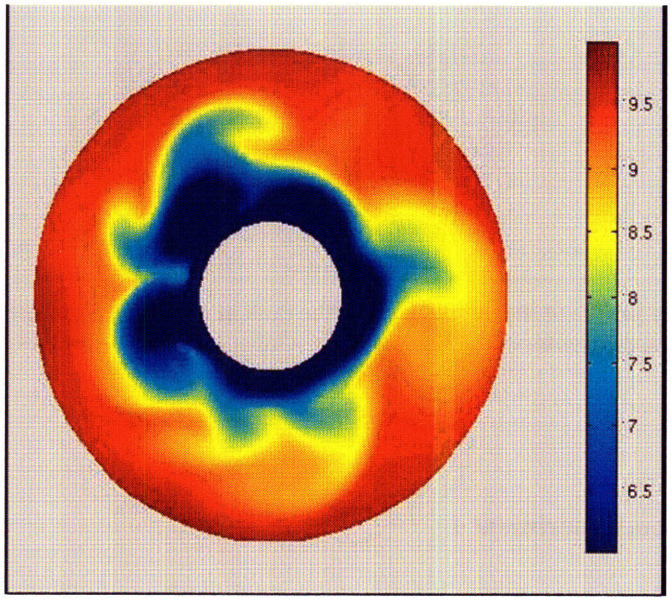

Figure 32, Lack of eddies later in the model run (after about 80 rotation periods).

Figure 33 shows the velocity field at the bottom of the tank that the model produces after about 80 rotation periods:

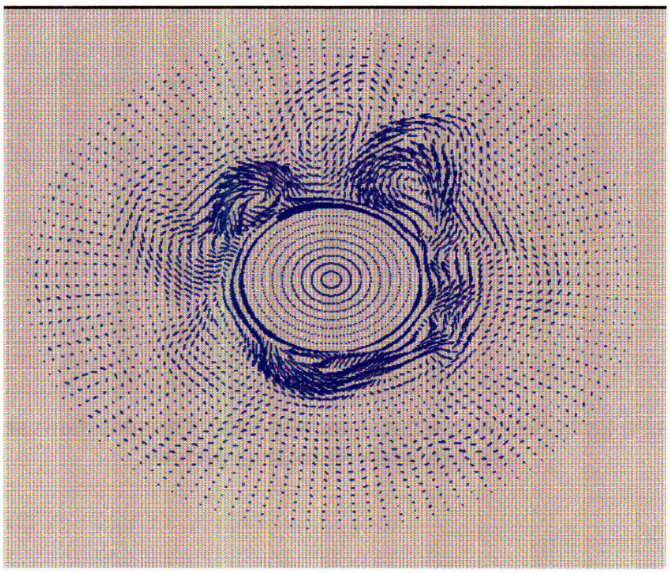

Figure 33, Model velocity field at the bottom of the tank in its default configuration after 80 rotation periods.

4.2.2 Biharmonic Viscosity Experiment

We experimented with biharmonic viscosity instead of Laplacian viscosity. Biharmonic viscosity acts on smaller scales than the standard default Laplacian viscosity. The Laplacian form (viscAh) acts as follows:

$$\frac{\partial u}{\partial t} = viscAh\nabla^2 u$$

while biharmonic viscosity (viscA4) takes the following form:

$$\frac{\partial u}{\partial t} = -viscA4\nabla^4 u$$

Using dimensional analysis, we can determine the approximate relative scale of viscA4 as opposed to viscAh. Values of viscA4 greater than about 10^{-9} m^2/s cause the model to crash, while values smaller than 10^{-11} m^2/s cause the model to be extremely unstable with turbulent flows, instead of stable eddies.

Judicious choices for the biharmonic frictional parameters enabled the model to capture the large-scale baroclinic instability observed in the tank, whilst maintaining smooth fields on the grid-scale (Figure 34).

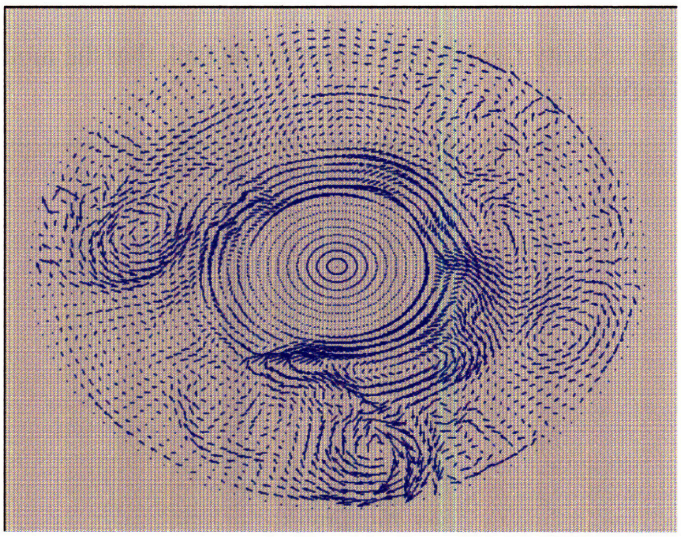

Figure 34, Model velocity field with biharmonic viscosity. This plot shows stable eddies after approximately 80 rotation periods.

5 Building a Real Time Model Control System

5.1 Introduction to the Model Control System

The overall goal of the project is to have real time data assimilation of a multi-ensemble model with real time tank observations. The MITgcm needed to be modified to allow it to pause at user defined frequencies, and restart with new, analyzed assimilated model fields.

The main purpose of this part of the project was building a Model Control System (MCS), which would allow us to perform real time data assimilation of the model and the tank experiment. Data assimilation and data display would be performed on our laboratory computer, while the model ensembles would be running on an Altix supercomputer. We needed to control the number of ensembles running at a given time, to give complete flexibility to the experimenter. In addition, the MITgcm had no facility to stop running and restart under new, user-provided, conditions. Therefore, numerous modifications and additions to the MITgcm were required:

- Display the results visually in a graphical interface
- Interpolate tank observation fields to a polar coordinate system (which is required by the data assimilation module) and store them in shared memory
- Pause model runs, to allow time for the data assimilation to take place
- Write model fields to shared memory at given intervals
- Read model fields from shared memory, and restart the model with these fields as new "initial" conditions
- Create an ensemble generator, which will be used to generate random initial conditions for numerous model runs

- Utilize a network connection to transmit the model fields from our Altix supercomputer to our lab computer, and vice versa

5.2 Modifications to the Graphical Interface

5.2.1 Power System Control

After installing a network ready APC power strip on the rotating table, we were able to configure it to have a network IP address. We were then able to communicate with it directly from our user interface. The code for this communication was written in the Expect language[10], a language that allows a user to telnet across a network in an automated fashion. We implemented functions to switch individual power outlets on and off, and to reset the entire strip. These functions allow us to reset an individual item on the table if there is a failure during the experiment. For example, if our program realizes that the camera is not being responsive, we have coded the power strip to automatically reboot its power. In addition, we are able to remotely control the power to the laser and chiller, creating a safer environment for the experimenter. This also allows experiments to be controlled from any computer in the world with an Internet connection, assuming the table apparatus is set up.[11] To make these features as user friendly as possible, all one needs to do to turn on an outlet of the strip, is type "start [outletnum]" in the GUI text field. To turn an outlet off, type "kill [outletnum]".

5.2.2 Display of Model Fields

We modified the graphical user interface to display both the PIV vectors and model vectors simultaneously (or independently, as it had operated previously). The PIV vectors are displayed in green, and the model vectors are displayed in yellow. Their scales can be adjusted independently. Both fields update as often as new data is available. In addition, the interface places the observation fields in a shared memory, enabling other modules (such as the data assimilation module) to access it.

5.3 Timing Data

After the experiment with the model on a single processor "slow" computer, we needed to get accurate timing data for the model on various computers. Since the goal of this project was real time data assimilation, the model would have to substantially outperform real time, allowing the data assimilation module time to do its computations. Figure 35 shows the results of these tests. We ran a 200 timestep (20 second) model run on AO, a single processor "slow" computer, and using one and four processors of a 16-processor Altix supercomputer. We also experimented with two different vertical structures of the grid: an unevenly spaced 15 level grid, and an evenly spaced 29 level grid.

---

[10] http://expect.nist.gov/

[11] We have a VNC server running on the computer. This allows the visual output of the experiment to be viewed remotely, as well.

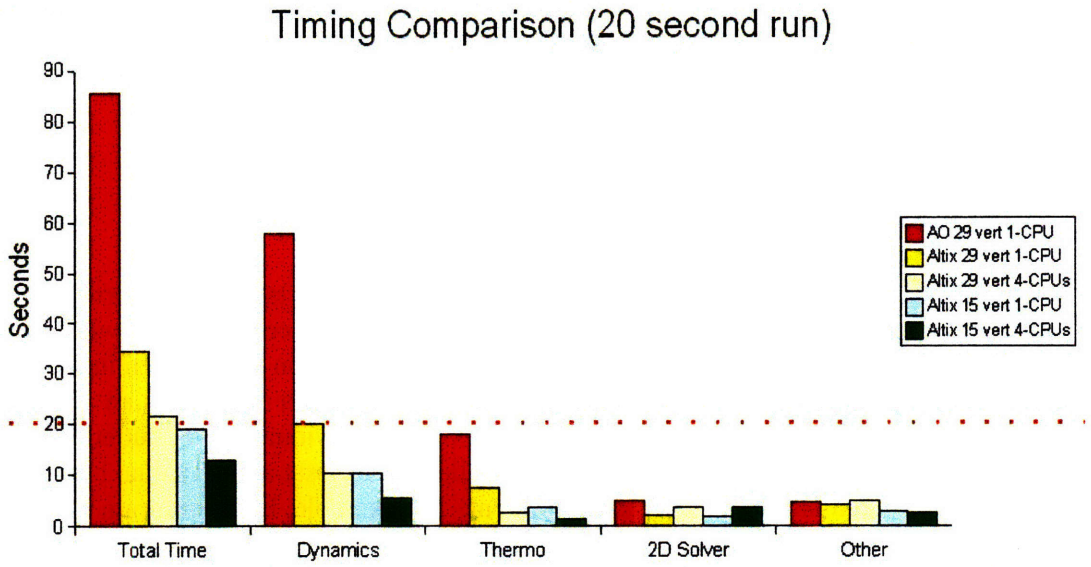

Figure 35, Model timing comparison. The dotted line shows real time. The total run time is the left set of data. The various components of the model run are indicated, as well.

The only model runs that were faster than real time were the 15 level runs on the Altix supercomputer. Therefore, that is the configuration we chose for the experiments.

5.4 Interpolation

The MITgcm outputs its data in polar coordinates, while the PIV software outputs its data in a Cartesian grid. We needed to develop two inverse interpolation schemes: one to convert model data to Cartesian data for display in our graphical user interface, and one to convert PIV data to polar coordinates for data assimilation processing. In addition, these interpolation routines needed to be extremely rapid, enabling the user to visualize the model data on the screen in real time, and to perform the data assimilation in real time.

5.4.1 Radial Basis Functions

We needed an interpolation scheme that would not only be fast, but would also preserve the integrity of the small-scale flow features of our tank experiment. For example, near the center of an eddy, the flow may reverse from one grid cell to the next (due to the resolution of our PIV grid). Gaussian radial basis functions provided this speed and accuracy.

In general, the equation for interpolating with radial basis functions is:

$$I(x,y) = \sum_{i=1}^{I}\sum_{j=1}^{J} w_{(i,j)\to(x,y)} U(i,j)$$

where I(x,y) is the interpolated value, w is the weight function, and U(i,j) is the uninterpolated value. In our Gaussian setup, the weight function was calculated as follows:

$w_{(i,j)\to(x,y)} = e^{\frac{(y-j)^2+(x-i)^2}{c}}$, where c is a constant.

The weight function decays exponentially as the uninterpolated grid point gets further from the interpolated grid point. As discussed below, we use that fact to improve the speed of the interpolation, by eliminating negligible weights.

5.4.2 Pregeneration of Weights

Since our model and Cartesian grids remain constant, it saves time to generate the weights in advance, storing them in compact files. We developed Matlab scripts to pregenerate the weights for both directions of interpolation (the weights are specific to our tank and model configurations, and can be regenerated if those configurations change). The scripts compute the weight of each point in the initial grid for each grid point in the interpolated grid. All weights above a certain, user defined, threshold are stored in a file named aNbM, where (N,M) is the point in the interpolated grid. Keeping only the weights above the threshold allows the files to be small, yet still accurate. The Labsim user interface application loads these files into an object in memory each time it is started. These weight objects (one for each direction of interpolation) contain all of the information needed to interpolate any point in the grid. Using a get(int i, int j) method, the specific weights for a given point can be accessed.

5.4.3 Model Interpolation

To allow the model to be smoothly interpolated at the boundaries of the tank, we pad the model data with zeros, both on the inside (where the center can is located), and the outside (where the outer rim is located). Prior to applying the weights, a transformation matrix is applied to each velocity vector in the polar grid, converting the vectors from polar space to Cartesian space. Then, a loop runs through each point in the Cartesian grid, uses the weights and the formula above, and determines the velocity field. A similar loop is used to interpolate the model's temperature field; however, a transformation matrix is not needed.

5.4.4 PIV Interpolation

The interpolation process of the PIV vectors from Cartesian to polar spaces works in a similar manner. It uses a separate set of weights, and the inverse transformation matrix as in the opposite interpolation.

5.5 Model Modifications

By default, the MITgcm writes its output to standard format data files, with one file for each output variable (such as u velocity, v velocity, and temperature). However, reading and writing files, especially on the Altix supercomputer, is very slow.

To get the most accurate results possible, we needed to run 100's of ensemble members. Due to being limited by the 16 processors in the Altix supercomputer, we needed a method to generate multiple ensembles per processor (and hence, per model run). To accomplish this, we use the method of "snapshots". In this method, we take output from one model run at various timesteps near the point in the integration we are considering. We regard each of these realizations to be an individual ensemble member. In our current configuration, we are taking nine time snapshots of the model prior to the output of an "official" model state, giving us 10 ensembles per processor, or a total of up to 160 ensemble members (if we were to use all 16 of the Altix's processors).

We modified the model itself to allow writing of data fields to shared memory, which allows rapid reading and writing of data. First, we added a new module to the model for these shared memory reading and writing routines. We then developed an infrastructure to control how the model writes to memory, how often it does so, when it pauses, how it knows when to restart the time integration, and how it reads new data from the memory when it restarts.

We added the following parameters to the model's parameter file:

- useShmem. A boolean flag that controls whether the model uses our new infrastructure at all. When it is set to false, the model runs as it did prior to these modifications.
- shmemKey. This is the key that identifies which shared memory this specific model should use for writing and reading fields. If the user intends to run ensembles of models (in addition to the snapshots), it is important to pick different shared memory keys for each run.
- shmemWaitInit. This initializes the number of timesteps of the model that are run before each snapshot is written to shared memory.

In addition, we hard coded the number of snapshots per model to be 10.

When the model is initialized, it computes an internal variable, called the "waitflag". This variable is initially equal to the product of the number of snapshots and the value in shmemWaitInit. The waitflag indicates how many timesteps remain before the model pauses iterating to allow the data assimilation module to do its computations. In addition, the model initializes the shared memory indicated by the shmemKey parameter. It places the waitflag in the memory, and allocates storage for 10 snapshots of model parameters.

Once the model begins stepping forward, it performs a check after each timestep. If the waitflag is greater than zero and the waitflag modulo the number of snapshots to be

written between pauses (the waitflag modulo 10 in our current configuration) is non-zero, the waitflag is decremented, and the model continues iterating. If the waitflag is greater than zero, but the waitflag modulo the number of snapshots to be written is equal to zero, then it performs a shared memory write. It writes the u velocity, v velocity, and temperature fields of the model into the n-th[12] slot of the shared memory in row major form. After decrementing the waitflag, it continues iterating. If the waitflag is equal to zero, the model suspends its iterations, and sets shmemWaitInit to zero. After writing its fields to the final slot in the shared memory, it enters a passive mode. While it is paused, other processes can read any of the snapshots of the model (for data assimilation), and modify the fields located in the final slot of the shared memory[13]. Every 50 milliseconds, the model checks the shmemWaitInit flag in the shared memory. If that flag has been changed to a positive value, it recomputes the waitflag (by multiplying by the number of snapshots), reads the (possibly) updated fields from shared memory, and starts iterating the model again with those new fields as initial conditions.

[12] Where n = the waitflag divided by the number of snapshots

[13] As discussed previously, the data assimilation tries to "nudge" the model towards the observations. The data assimilation module has access to the observations, as well, allowing it to perform these computations.

This set of processes is shown as a flowchart in Figure 36.

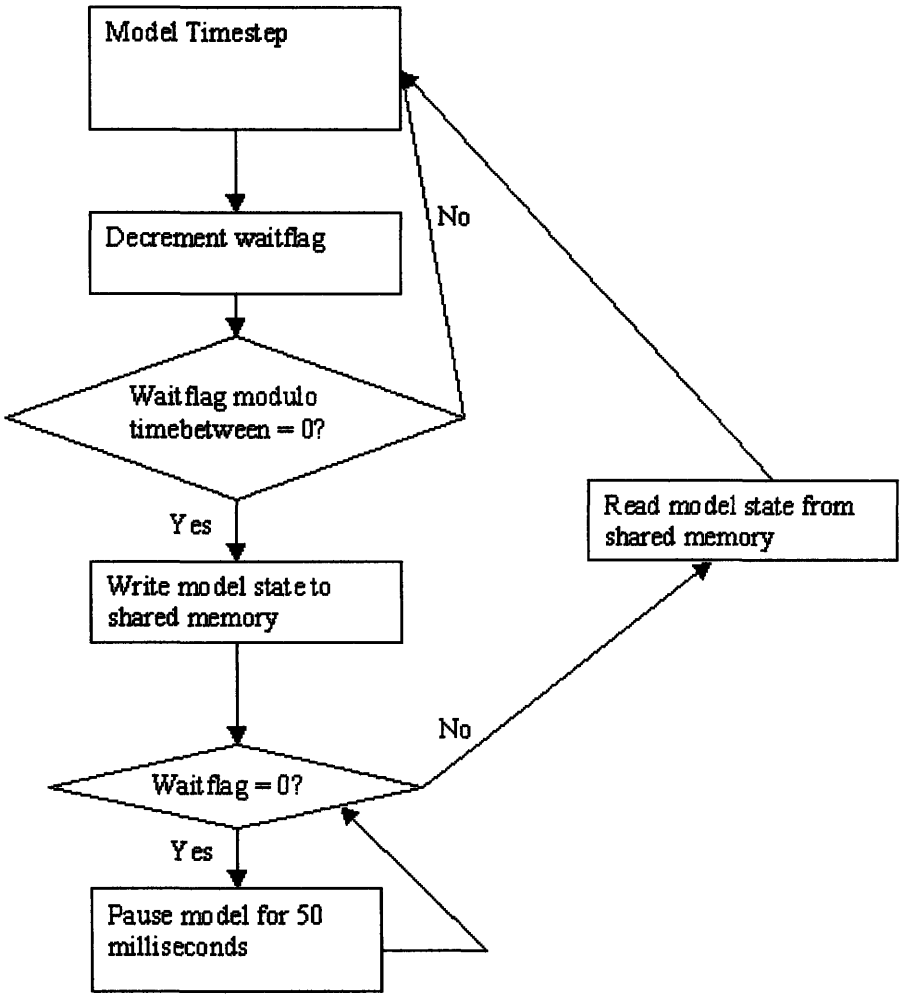

Figure 36, Flowchart of model, when run in shared memory mode. The chart indicates the model enhancements discussed in the previous section, including the model pauses and shared memory writes and reads.

5.6 Ensemble Generation and Runner

For the data assimilation routines to function properly and with high accuracy, we needed to run many perturbed ensembles of the MITgcm. These ensembles needed to be quickly and automatically generated and initialized. To accomplish this task, we wrote a pair of scripts in Perl and Matlab. The first script generates the perturbed models, and the second script automatically runs as many ensembles as the user chooses.

5.6.1 Ensemble Generator

The ensemble generator script can generate an arbitrary number of perturbed initial states for the MITgcm. The generator reads a file called "config.ens", which contains a list of the shared memory keys for each ensemble member (the number of keys in the list determines the number of ensembles generated). The ensemble generator takes a template of the model parameter file and appends the shared memory key specified in the configuration file. It then uses Matlab to take a standard temperature file (see the section on the initial temperature perturbations in Part 1), and perturbs each grid point by a value selected from a Gaussian probability distribution function, with a mean perturbation of 0.05 degrees Celsius.

5.6.2 Ensemble Runner

The ensemble runner can run one or more of the ensemble members created by the ensemble generator. The script reads a configuration file name "config.ens.run", which lists the ensemble numbers that it is supposed to run.

5.7 Network Connection

Given that we needed to run the model on the Altix supercomputer, and that the Ravela data assimilation module and the graphical user interface were on Coetzee, a protocol needed to be established for two-way communication of model data between these two machines. We determined that the best implementation would be to have the Altix function as a TCP/IP server, with Coetzee as the client. However, each machine would need to be capable of both receiving and transmitting data. The client issues the commands to the server, as described below.

5.7.1 Commands

All of the commands passed back and forth are character based. For example, when a model field is to be sent over the network, it is converted into a stream of characters before transmission. Every command is issued by Coetzee. The Altix interprets the initial part of the command, and takes the appropriate action. This is discussed in more detail in the following sections.

There are four main commands that are passed in this system, and various other commands, as well. The four commands are:

- Check. This command returns the current wait flag for a given ensemble member running on the Altix. Coetzee sends a command to the Altix in the form "check#####", where the five digits represent the ensemble number that the Altix is to check. Upon receiving this message, the Altix transmits the data back to Coetzee.

- Get. This command instructs the Altix to transmit the complete model state (all ten snapshots) of a given ensemble member. Coetzee transmits the command in the form "get#####", where the digits represent the ensemble number that is to be returned. When the Altix receives a get message, it converts the model data, reads from the shared memory associated with that ensemble to a character array (9936016 bytes long), and transmits it across the network.
- Restart. The restart command tells the Altix to restart a given ensemble member for a set number of timesteps between each shared memory write. This command is transmitted in the form "restart#####%%%", where the pound signs indicate the ensemble number that is to be restarted, and the percent signs indicate the number of timesteps between each write to shared memory (given that there are 10 writes before the model pauses again, this number is $1/10^{th}$ of the number of timesteps that will be completed before the next pause). Allowing the timing information to change mid-run is crucial to developing a real time system. The data assimilation module can determine if it is running ahead or behind real time when it establishes the new value for this parameter. When the Altix receives a restart message, it places the new "time between" flag into the proper ensemble. As discussed previously, updating this flag causes the model to automatically begin iterating again.
- Send. This command tells the Altix to place a set of analyzed model fields (u velocity, v velocity, and temperature) into the "read slot" of the shared memory (the snapshot that is read when a model is restarted after pausing) of a given ensemble member. The command sent by Coetzee is in two parts. The first command is "send#####%%%%%", which instructs the Altix that it is about to receive a set of model data of size indicated by the % signs, to be placed into the ensemble member indicated by the # signs. The next transmission is the actual model data fields. The Altix uses memcpy to copy the fields into the proper shared memory.

Other commands that the system understands include:

- Quit. When Coetzee sends the text "quit", the Altix server stops.
- GetNumEns. When Coetzee sends the "getnumens" command, the Altix responds by transmitting the number of ensembles listed in the config.ens.run file.

5.7.2 Server

The server resides on the Altix computer. It is automatically started by the ensemble runner (discussed in the previous section). However, it also can be started manually. The only parameter required to initialize the server is a port number that the Altix is to listen on. The server has two modes: listen and command. In listen mode, it waits for an incoming connection. When the server begins, it starts in listen mode. If a connection is offered, the server goes into command mode. Command mode allows a connected computer to pass one of the valid commands listed in the previous section, with the Altix returning data, if applicable, to that command. If the connected computer should disconnect from the server, it returns to listening mode, allowing the server to remain active indefinitely, regardless of the connecting computer's state.

5.7.3 Client / GUI Modifications

The client is built as a library on Coetzee, allowing its functions to be called from both the graphical interface and the data assimilation module. When the client is initialized, using the InitClient method, it attempts to connect to the server on the Altix. If the connection is successful, the client awaits commands to send, such as those issued in the interface or assimilation procedure. When the client is finished, it can disconnect from the Altix. The graphical user interface has been modified to operate under the files method or the new network connection method. If the network connection method is chosen, the user can decide whether to be in assimilation mode (in which it does not issue any network commands – as those would be controlled by the assimilation package), or standard network mode (in which the interface controls the model).

When the client receives data from the Altix, it puts it into a shared memory dedicated to assimilation. The assimilation routine can perform its operations on it. The graphical interface can be configured to display model data from this shared memory.

Figure 37 shows the complete workings of this network connection, in schematic form:

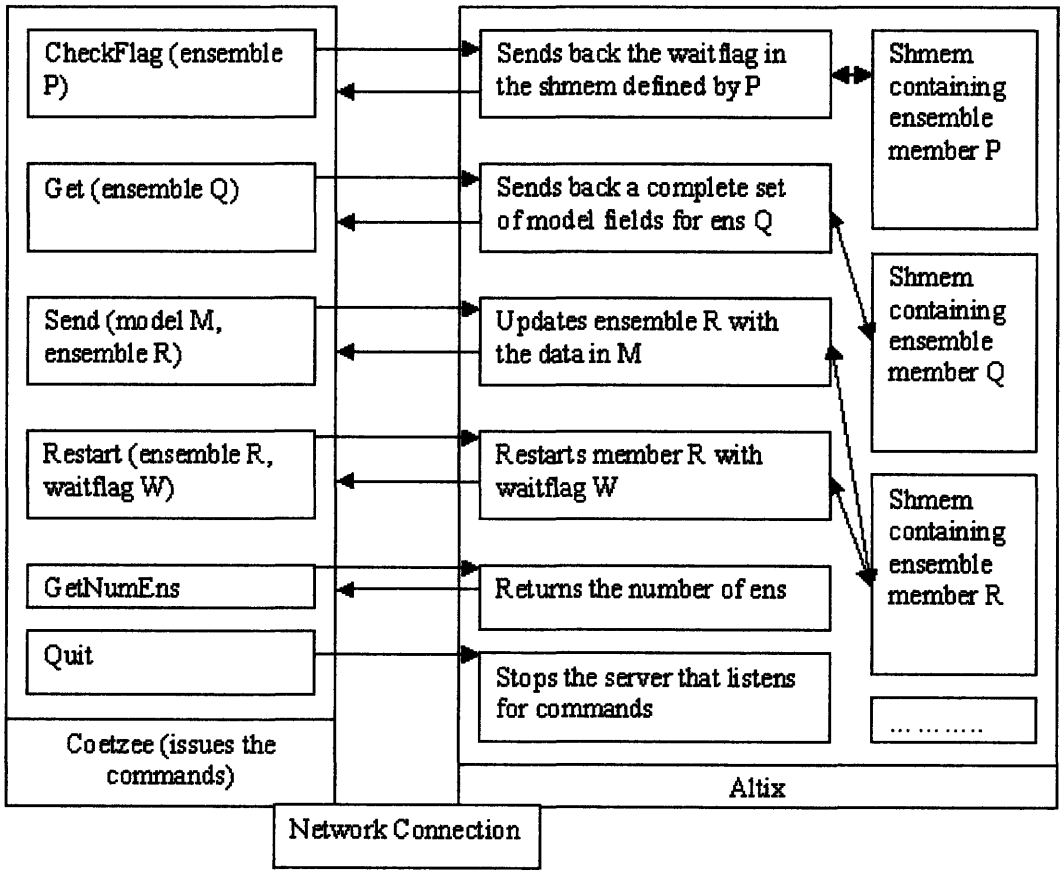

Figure 37, Schematic of the network connections and commands

5.8 Testing Procedures

We implemented various testing procedures to ensure that both the shared memory model writing and the network connection were bug-free. In addition, we tested the interpolation routines, both to and from polar coordinates.

5.8.1 Shared Memory Testing

To test that the proper model field was in the shared memory, we implemented the following routine:

- Output the model's u velocity, v velocity, and temperature fields to files after every 50 timesteps.
- Output the shared memory fields at the same timestep.
- Using a Matlab script, we compare the data in the files to the data in the shared memory.

The data in the shared memory was identical to the data in the files.

5.8.2 Network Connection Testing

To test the functionality of the network connection we employed a number of tests. The following explains how the four main commands were tested.

- Check. We tested the check method by displaying the model's wait flag on the screen during the model run. We compared that value to the value received across the network.
- Get. To analyze the efficacy of the get method, we ran the model with its standard file output – as a control, comparing each timestep to a separate model run (with identical parameters) using shared memory and the network connection.
- Restart. To test our restart method, we passed various wait flags back to the model, and watched the output of the model, confirming that it was using the flags properly.
- Send. Testing the send routine was accomplished by sending back the model vectors (unchanged) to the Altix supercomputer prior to restarting the model. The model produced the same results as when we did not send the vectors back, confirming that the send procedure worked.

5.8.3 Interpolation Testing

In order to test our interpolation routines, we could not make use of the paradigm of the above tests. In any interpolation process, there is smoothing of the data, so we would not expect the interpolated fields to match the uninterpolated ones. In the qualitative tests of our polar to Cartesian interpolator, we compared graphical model output from the polar coordinates to the GUI's Cartesian display. For the Cartesian to polar tests, we compared

the GUI's Cartesian display to the same data interpolated to polar and then back to Cartesian.

6 Data Assimilation System

As a final test of the system, we connected Dr. Ravela's data assimilation module to the infrastructure developed in this project. His module uses the commands provided by the client that we built in this project to communicate with the Altix supercomputer running ensembles of the MITgcm model. It also makes use of the shared memory and interpolation schemes that we developed and tested, both for display purposes in the interface, and to ensure that the data from the tank and the fields from the model were in the same coordinate system.

Prior to initiating the model, we generated a set of fifteen perturbed ensemble members using the ensemble generator. We ensured that the PIV servers were operational. After switching the chiller unit on, we waited for stable eddies to form. We then ran the model for a number of timesteps, until it formed eddies, as well.

After completing those setup steps, we ran the experiment in the laboratory tank, the server on the Altix computer, and continued the numerical model (with pauses for the assimilation). At each pause of the model, which we set to be every 100 timesteps (10 seconds of real time), Ravela's module assimilated the tank observation data (5 levels of equally spaced tank PIV data, with the levels automatically controlled by the movable mirror) and the model data it received over the network connection. Each assimilation stage was equivalent to the green lines in Figure 4, and brings the model into a closer alignment with the observations. After about fifty iterations, the state estimate of the fields appeared as they do in Figure 38 (the yellow vector field). The location and magnitude of the eddies in the state estimation match up very well to the observed eddies (the green vector field).

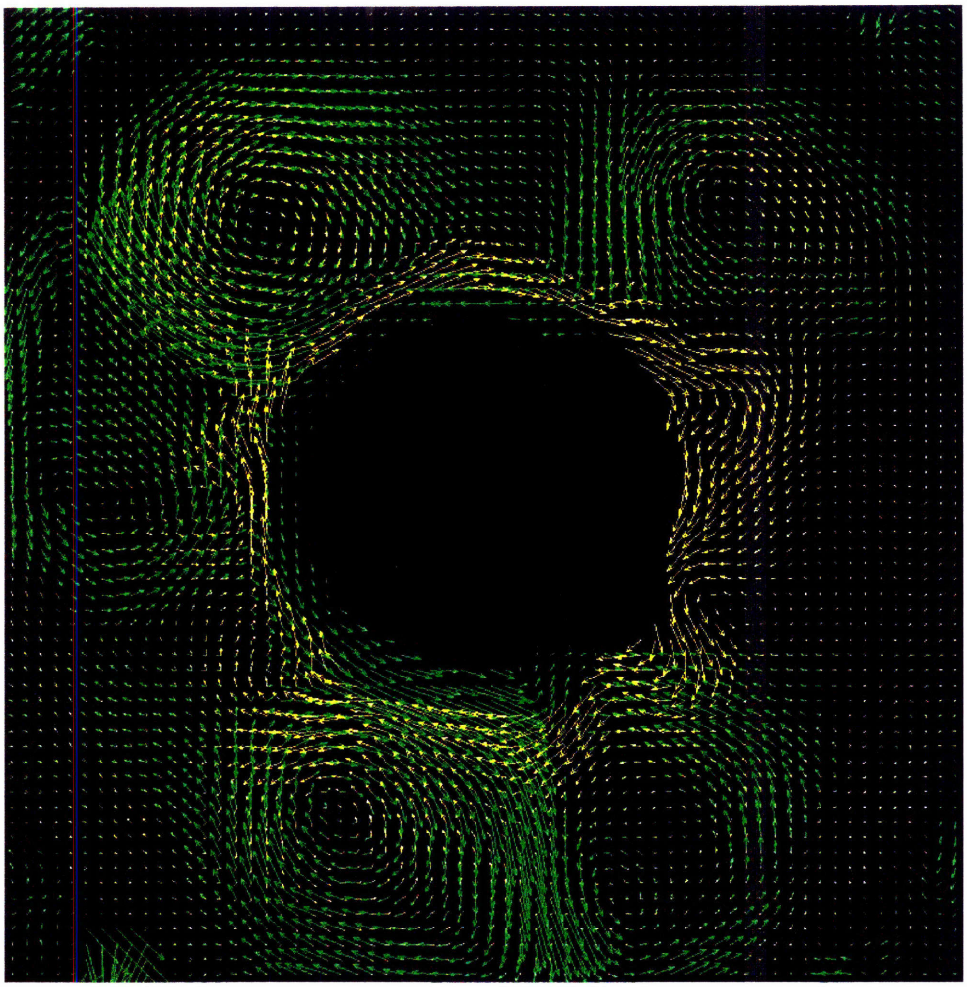

Figure 38, An example of a partially assimilated model field / state estimate (yellow), along with the observation field (green). Note that the eddy locations in each field are similar (whereas in an unassimilated run, they would not be in similar locations).

7 Conclusion

We have developed a robust system for performing real time data assimilation.

For the first time, a system has been created that allows a laboratory experiment to be run in conjunction with a real time numerical model. It was designed in three stages. First, we configured the parameters of the model to produce eddies of the approximate size, shape, magnitude, and longevity that we observed in the tank experiment. Next, we adjusted the code of the MIT General Circulation model, allowing it to pause in the middle of its run. In addition, the code was modified to let the model, when it restarted, use a new set of velocity and temperature fields (as output by the data assimilation

module) as initial conditions. Finally, we created a TCP/IP client/server to quickly transfer data over a network from the supercomputer where the model is processed to the laboratory computer where we controlled the experiments. After building these systems, we tested each component individually, and as a whole.

It is also important to note that we are not limited to the rotating annulus experiment, now that this system has been created. Any tank configuration that can be placed on the rotating table can be analyzed using the system.

The tools created in this project can also be used for teaching. The model is now adjustable without knowledge of its inner workings, using the interface we developed. Its parameters can easily be tweaked, allowing students to witness their impact on the resultant flow. In addition, the Model Control System can be used to teach students about various methods of data assimilation. Since the Model Control System is independent of the assimilation module itself, the assimilation module can be swapped to use a different scheme, without having to change anything else in the code.

Finally, as an example of a different experiment that benefited from the tools developed during the course of this project, we show results from a plume evolution experiment. In this experiment, there is no center can. A density stratification is established in the main tank, with a density of 1.06 g/cm^3 at the bottom, and 1.00 g/cm^3 at the top (giving an average of 1.03 g/cm^3, which is needed for the particles to operate properly). After the tank reached solid body rotation, salt water containing the particles was injected into the middle of the tank. We were able to use our system to capture the velocity vectors (using PIV) of the induced anticyclone, as shown in Figure 39. We were also able to use the movable mirror to determine the vertical extent of the anticyclone.

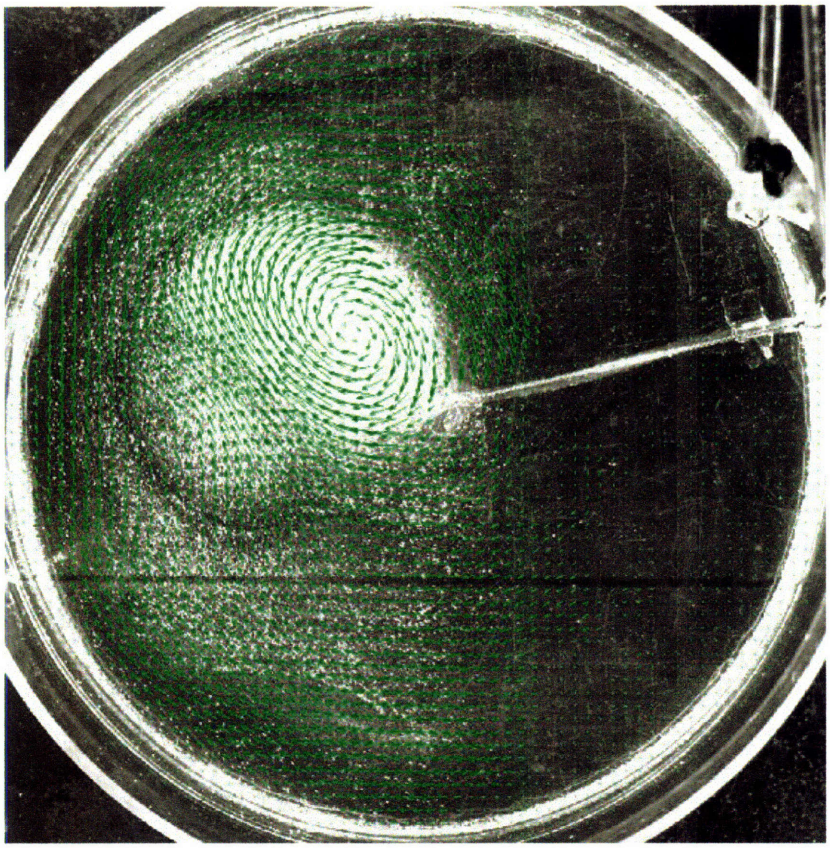

Figure 39, Density stratification experiment showing a large anticyclone, as indicated by the green vectors acquired using PIV.

References

- Deterministic Chaos. http://www.exploratorium.edu/complexity/CompLexicon/chaos.html
- Evangalinos, C. Web-enabled configuration and control of legacy codes: an application to ocean modeling. Journal of Ocean Modeling. Volume 13/3-4, pages 197-220. 2005.
- Expect. http://expect.nist.gov/
- LaVision. PIV. http://www.piv.de/index.htm.
- Lorenz, E.N. Deterministic Nonperiodic Flow. Journal of the Atmospheric Sciences, Volume 20, pages 130-141. 1963.
- Marshall, J, et al. "A finite-volume, incompressible Navier Stokes model for studies of the ocean of parallel computers." Journal of Geophysical Research. 102(C3), 5753-5766.
- Marshall, J., Plumb, A., and Illari, L. "Thermal wind." (12.307 notes). http://paoc.mit.edu/12307/front/thermal%20wind/thermal_wind.pdf. March 4, 2003.
- Marshall, J., et al. Journal of Physical Oceanography. "Can eddies set ocean stratification?" Vol 32, No. 1, Jan 2002.
- Marshall, J., et al. Grant application for the Planet-in-a-Bottle DDDAS.
- MITgcm documentation. http://mitgcm.org.
- Read P. L., et al. "An Evaluation of Eulerian and Semi-Lagrangian Advection Schemes in Simulations of Rotating Stratified Flows in the Laboratory." Monthly Weather Review, page 2835, August 2000.

Appendices

Appendix A: The Thermal Wind Equation

The thermal wind relation states that the vertical wind shear, du/dz, is proportional to the horizontal temperature gradient, dT/dr. This is due to a relationship between geostrophic balance and hydrostatic balance. In our flow, the Rossby number is small, $R_0 = \frac{U}{fL}$, therefore we can assume geostrophic balance because the Coriolis force is balanced by the pressure gradient force. Geostrophic balance is given by:

$$u_g = \frac{1}{\rho f} \hat{z} \times \nabla p \quad .$$

Next we assume that the density of the water varies as:

$$\rho = \rho_0 + \delta\rho \quad \text{where} \quad \frac{\delta\rho}{\rho_0} \ll 1 \quad .$$

We then use the hydrostatic balance

$$\frac{dp}{dz} + \rho g = 0$$

and the z partial derivative of the geostrophic flow equation giving us:

$$\frac{du_g}{dz} = \frac{-g}{\rho_0 f} \hat{\mathbf{z}} \times \nabla\rho$$

Finally, to convert this formula to one that expresses the relationship we desire, we convert pressure to temperature using

$$\rho = \rho_0 (1 - \alpha (T - T_0)) \quad .$$

where α is the thermal expansion coefficient of water at temperature T_0. Using this equation and the previous one, we use the simplified equation of state:

$$\frac{du}{dz} = \left(\alpha \frac{g}{f}\right)\left(\frac{dT}{dr}\right) \quad .$$

The vertical wind shear is hence equal to the thermal expansion coefficient of water multiplied by the horizontal temperature gradient times gravity, all divided by f, which is proportional to the rotation rate.

We take this as a given in the tank experiment, and hope to achieve this relationship in the model. Not only do we wish to have thermal wind balance in the model, we would like the velocity and temperature profiles of the tank to match the profiles of the model.

These equations apply to rotating fluids with a small Rossby number and minimal frictional effects.

Appendix B: The Model Interface

Introduction and Definitions

In this part of the project, I created a piece of computer code that will allow users with a strong background in oceanic and atmospheric sciences, yet little knowledge of computer programming, to set the runtime parameters for running the MIT circulation model, MITgcm[14], which is written in FORTRAN. This code will create a graphical interface that will let users easily manipulate all of the variables in the model without knowing anything about the source code of the model itself. My code is written in Java, as well as a markup language called LCML[15], a type of XML[16] that can be understood by a Java application called Legend[17]. When the LCML code is loaded in Legend, an interface appears that lets users adjust the values of all the variables in the problem (they are given default values in the LCML code). Upon completion of any adjustment of parameters, Legend can check their validity and output a FORTRAN readable file for input into the model.

XML is a "markup" language for computer programming, similar in structure to HTML. XML schemas provide a set of rules for defining the structure, content, and semantics of an XML code.[18]
MITgcm employs several namelists for runtime parameterization. FORTRAN namelists are parameter files for use in FORTRAN codes. They list all of the variables in a certain code, and can be given default values.
LCML is a form of XML with certain rules and constraints described in a schema. LCML files can form the description of FORTRAN namelists, and can be interpreted by Legend.
Legend is a Java application that can interpret LCML descriptions and produce a data file that contains values for the variables in a FORTRAN namelist.

Background Material

To learn about the Legend software and the LCML, I studied the thesis *Legacy Computing Markup Language (LCML) and LEGEND - LEGacy Encapsulation for Network Distribution* by Stephen Kurt Geiger. In addition, I learned about the XML language, XML schemas, the Legend software, and the LCML markup language. I studied sample LCML description files that were done for other projects. My final piece

---

14 Marshall, J, et al. A finite-volume, incompressible Navier Stokes model for studies of the ocean of parallel computers. Journal of Geophysical Research. 102(C3), 5753-5766.
15 Geiger, S. Legacy Computing Markup Language (LCML) and LEGEND - LEGacy Encapsulation for Network Distribution. Thesis.
16 Extensible Markup Language, www.w3.org/XML
17 Evangalinos, C. Web-enabled configuration and control of legacy codes: an application to ocean modeling. Journal of Ocean Modeling. Volume 13/3-4, pages 197-220. 2005.
18 W3C XML Schema, www.w3.org/XML/Schema

of background material was learning about the actual FORTRAN namelists that I would be using in this project.

Legend, LCML, and XML

The Legend program allows users to create parameter files for FORTRAN files without any knowledge of FORTRAN.

The following is a basic description of the LCML language.

As in HTML (the standard language for building web pages), all information is encapsulated with tags. The format of a first tag is "<tag>". This is followed by the information that the tag refers to. The end tag is formatted as "</tag>".

The entire file is contained between <description> and </description> tags. The content of the file is contained between <descriptionContent> tags.

The variable descriptions themselves, using the <var> tag, are divided into sets (using the <set> tag), based on the source code itself; in this case, MITgcm.

This is an example of an extremely basic LCML skeleton:

```
<set>
    <var>var1</var>
    <var>var2</var>
</set>
```

In the above case, "var1" and "var2" would be replaced with items discussed below in the variables section.

MITgcm

MITgcm has over 250 parameters that had to be described in XML. Creating this description involved carefully gathering the list of variables from various FORTRAN files, figuring out what each variable signified to the model (from other FORTRAN files), and determining appropriate default values and types (numeric, string, boolean, etc.) for each (from yet more FORTRAN files).

As discussed above, the variables to be described have to be divided into sets. For the MITgcm description of the main runtime file (invariably called "data"), there are 5 such sets: continuous equation parameters, elliptic solver parameters, time stepping parameters, gridding parameters, and input files. Within each set, there can be one or more variable descriptions, contained within <var> tags.

Variables

Here is an example of the XML description of one of the diffusion variables in MITgcm:

```
        <var>
         <name>Laplacian diff coeff for heat mixing (diffKhT)</name>
         <header>diffKhT = </header>
         <trailer>, &#x000A;</trailer>
         <info>Laplacian diffusion coeff for mixing of heat laterally
(m^2/s )</info>
         <type>numeric</type>
         <precision>double</precision>
         <range>[0,INFINITY)</range>
         <value>0.E3</value>
         <use>false</use>
         <hidden>false</hidden>
       </var>
```

The entire description for each variable is encapsulated within `<var>` and `</var>` tags. The tags within the variable's description are as follows:

- Name: a descriptive name of the variable that will be displayed to the user in the interface (in this case, a brief summary of what the variable is used for).
- Header: this field contains the text that will be output to the data file before the value of the variable. Typically, this contains the actual name of the variable followed by an equals sign.
- Trailer: this field contains the text that will be output to the data file following the header and the value of the variable. Typically, this will be a comma followed by the XML symbol for a new line: `&#x000A`.
- Info: a full description of the variable's use in MITgcm. These were taken directly from the comments of the FORTRAN files of MITgcm.
- Units: an optional field, listing the units of the variable.
- Type: in this case a "numeric", though this field accepts "string" as well.
- Precision: for numeric fields, it could be integer, float, or double.
- Range: for numeric fields. The format of range uses standard algebraic notation with "(" representing greater than, and "[" representing greater than or equal to.
- Value: the default value for the variable.
- Use: either true or false, determines whether Legend will output this variable to the output data file.
- Hidden: either true or false, determines whether Legend will display this variable in its interface.

Other tags not used in this example include the "enumeration", which is used for variables that have a set of possible values (for instance, booleans). In the interface, enumerations appear as drop down menus. An example of a non-boolean enumeration from MITgcm is the buoyancy relation. The following snippet of code shows the relevant section. In this case, there are three possible choices for the enumeration: oceanic, atmospheric, and oceanicp.

```
        <type>string</type>
        <enumeration>'OCEANIC';'ATMOSPHERIC';'OCEANICP'</enumeration>
        <value>'OCEANIC'</value>
```

It is also possible to use LCML to describe vectors, using something called a "data structure". We create a "dummy" variable that the user can change to determine the number of elements in the vector. The vector itself then refers to the value of the dummy variable to set its length. By convention, the dummy variable will precede the structure in the interface.

When creating the descriptions of the variables, it was important to not include any "<", ">" or "&" symbols in the text. XML interprets these characters as parts of tags and escape codes (an example of an escape code is the symbol for newline, discussed above).

Progress and Screenshots

I have completed a working XML description for MITgcm in Legend.

Figure 40 is a screenshot of the interface produced by my code.

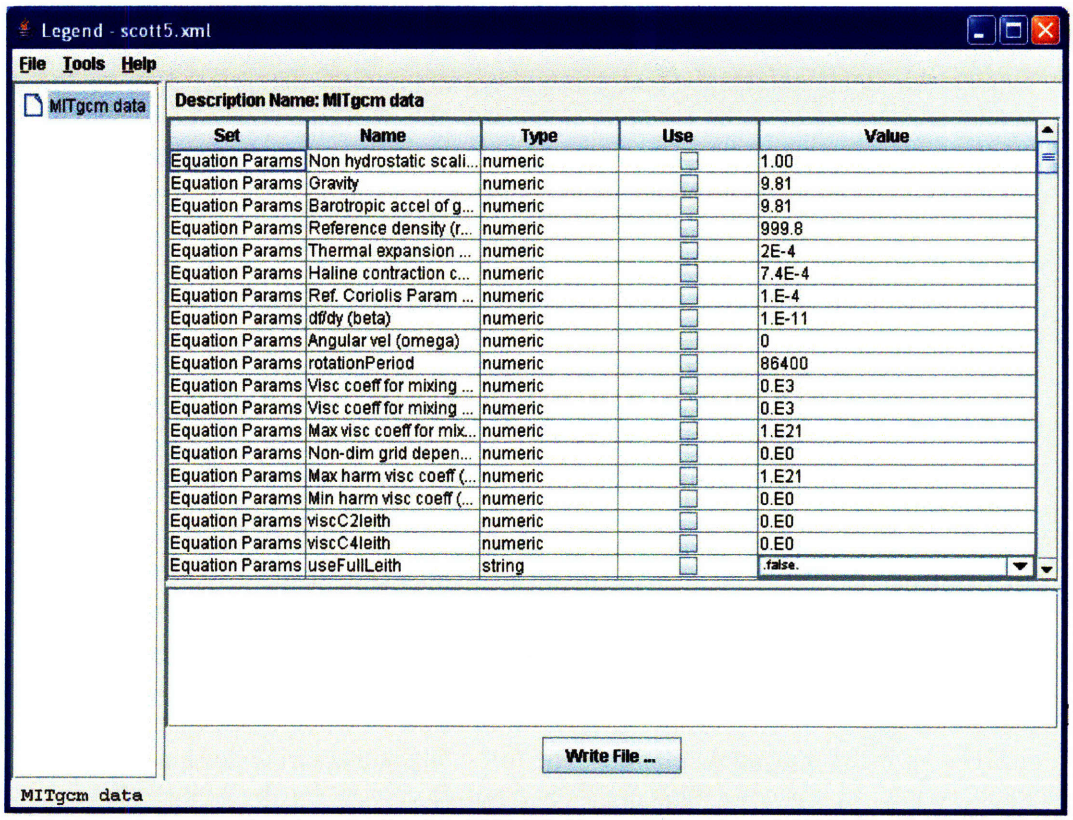

Figure 40, Legend interface.

This table can be sorted by any of its columns (name, type, value, etc.). The interface allows the user to manipulate the data that will be used as input to the MITgcm run. For instance, to change the value of gravity, the user just clicks the number in that field and types a new number. When the user is finished changing values, he or she clicks the "Write File…" button, which outputs the data file in a FORTRAN readable format. Not all variables need to be output to the file. To select which ones to output, check the "Use" box for each.

Figure 41 is a screenshot of the same code producing a different style of interface, the panel interface (this can easily be switched in the toolbar menu):

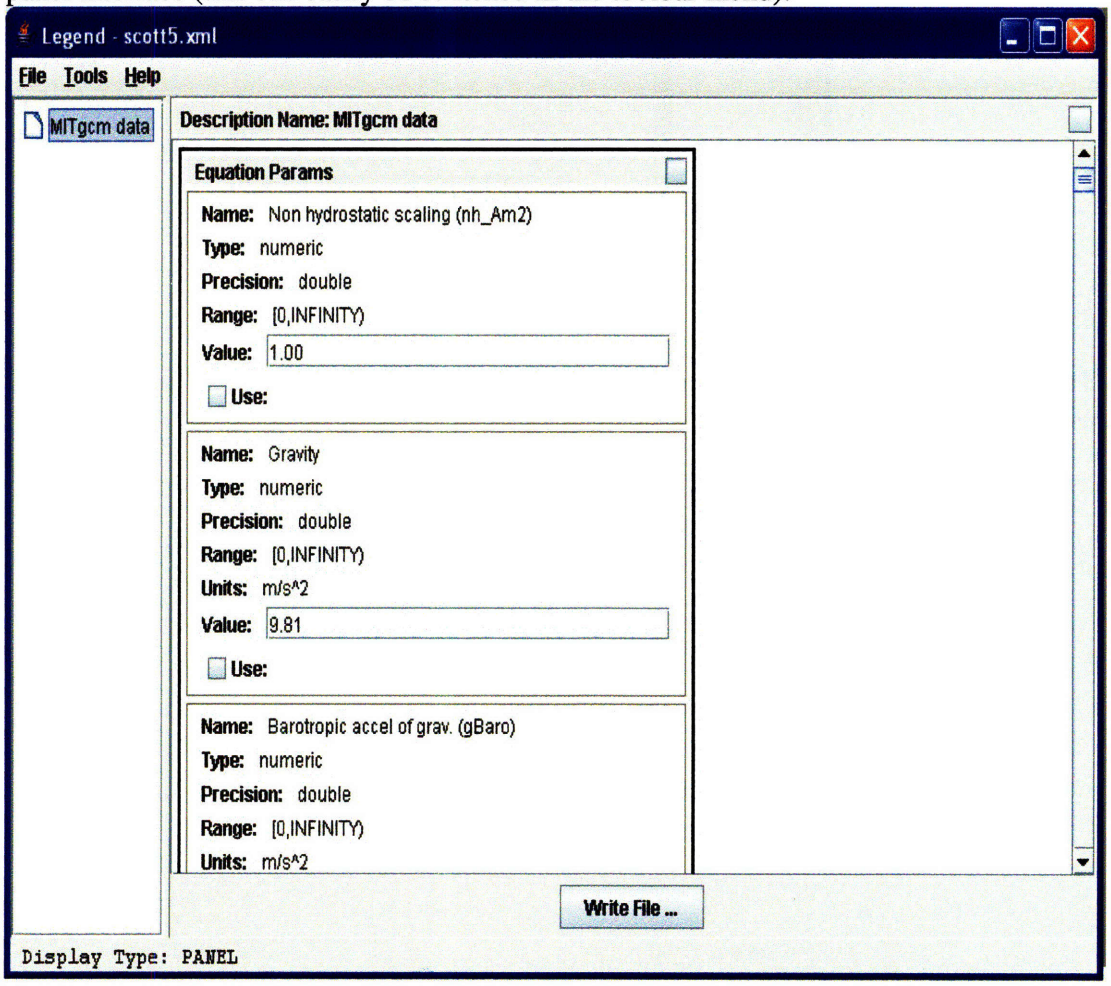

Figure 41, Changing parameters

Figure 42 is an example of an enumeration drop down menu as discussed previously:

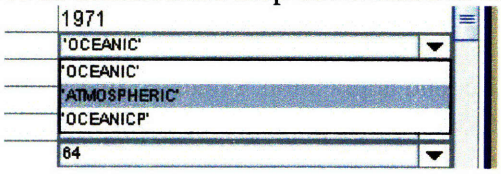

Figure 42, Enumerations

Figure 43 is an example of editing a vector (you access this popup screen via the "Edit Structure" button):

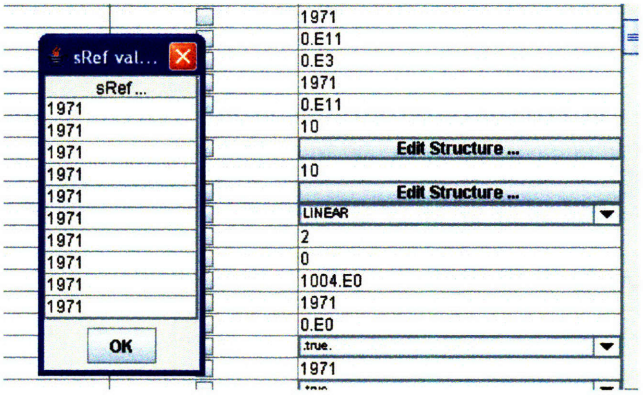

Figure 43, A large array

Then we change the dummy variable (from 10 to 4), and the vector is a different length, as shown in Figure 44.

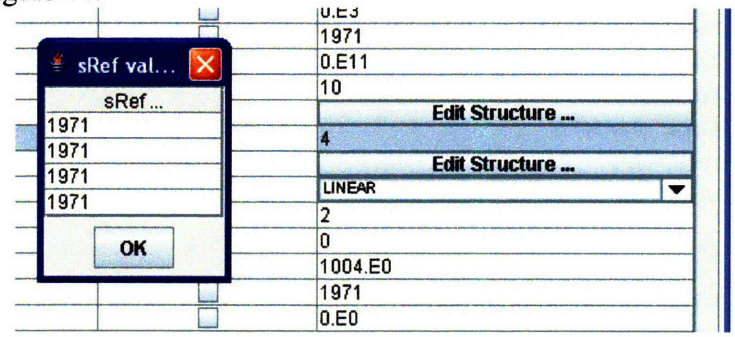

Figure 44, A smaller array

Modifications to the Legend Application

I have studied the Java source code of the Legend application itself. Dr. Evangelinos and I figured what specifically needed to be updated and modified to best suit the project's needs.

When I began using the Legend application, I realized that vectors were not being output to the data file, nor was a "use" check box showing up for them. I have fixed this bug.

Other enhancements that I have made to the application include:

- Storing the working directory in memory, making it much easier to open additional LCML files from one directory.
- Adding keyboard shortcuts to speed up usage.
- Changing the default display of the interface to table mode, as opposed to panel mode.

Legend Usage Guide

To use the interface:

- Run Legend.jar
- Choose File/Open, and then pick the XML file for MITgcm
- Adjust values as needed
- Press the "Write File..." button
- The output file will be called "data" by default

Appendix C: The Complete Tank Setup

The following is the complete rotating tank setup procedure:

- Mount the steel tower (holding the camera, FORJ, wireless network device, PHOX device – the unit that converts the signal from electronic to optic, and power strip), and secure it to the rotating table with clamps.
- Connect bungee cords to the top of the steel tower, to secure it in place, and prevent the FORJ from experiencing unnecessary torques.
- Connect the power cable and the data cable to PHOX on top.
- Plug in the power strip on the steel tower to the outlets on the table.
- Place a glass table with a black rubber mat onto the table. This will hold the actual tank. It is important to have the black rubber to prevent the tank from slipping, and to have a black background for higher quality digital images.
- Place a large wooden "table extender" onto the rotating table. This will allow the table to have more room to place the chiller and moving mirror apparatus.
- Place the chiller and moving mirror on opposite sides of the wooden tray. Secure them both (with bungee cords and clamps).
- Setup and align the laser.
- Place a large rectangular tank on the glass table. Place a cylindrical tank inside of it. (Having a rectangular tank on the outside helps eliminate some of the optical issues of having a light sheet pass through a round plastic surface.)
- Place the probe of the chiller in its aluminum casing along with the chiller's temperature sensor, as shown in Figure 45. Place the aluminum casing in a metal can, and place the can (centered) within the inner tank.

Figure 45, The chiller can

- Measure the appropriate amount of salt, such that the inner cylinder's water will have density 1.03 g/cm^3 when it is filled with water. This is the density of the laser fluorescent neutrally buoyant particles. For our standard setup, this amount is approximately 590 grams of salt.
- Fill both the outer and inner tanks with water, ensuring that the salt is thoroughly mixed and that there are no bubbles.
- Mix a small amount of the neutrally buoyant particles with photo fluid (to prevent them from clumping). Slowly add salt water from the tank to the mixed solution. Then place the entire mixture into the tank, and stir the tank.
- Plug in the laboratory frame PHOX, and then plug in the table's power.
- Spin the rotating table until the water is in solid body rotation.
- The experiment is now ready to be performed.

THE END