January 2007

On The Development Of Self-adapting (rans/les) Turbulence Models For Fluid Simulation At Any Mesh Resolution

Jason A. Gadebusch
University of Massachusetts Amherst, jgadebus@ecs.umass.edu

ON THE DEVELOPMENT OF SELF-ADAPTING (RANS/LES) TURBULENCE MODELS FOR FLUID SIMULATION AT ANY MESH RESOLUTION

A Thesis Presented

by

JASON A. GADEBUSCH

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

September 2007

Mechanical and Industrial Engineering

ON THE DEVELOPMENT OF SELF-ADAPTING (RANS/LES) TURBULENCE MODELS FOR FLUID SIMULATION AT ANY MESH RESOLUTION

A Thesis Presented

by

JASON A. GADEBUSCH

Approved as to style and content by:

J. Blair Perot, Chair

Stephen de Bruyn Kops, Member

Robert Hyers, Member

Mario Rotea, Department Head
Mechanical and Industrial Engineering

To my family for all of their love and support.

ACKNOWLEDGMENTS

I would like to acknowledge all of my colleagues in the UMass fluids group. Particularly I would like to thank Shivasubramanian Gopalakrishnan for all of his help in the Linux realm. I thank David Hebert for constantly answering endless questions about life, the universe, and everything. I would also like to thank Venkataramanan Subramanian for many long hours discussing Fortran and fluid dynamics. Of course, a big thank you to others for creating coffee thats more volatile than most jet fuels to discussing the latest supercomputer compilers. Thanks to Mike M., Mike N., Sandeep M., Saba A., and Kaustubh R.

Lastly (certainly not least) I am extremely grateful for the continuous support and guidance from my advisor Blair Perot.

ABSTRACT

ON THE DEVELOPMENT OF SELF-ADAPTING (RANS/LES) TURBULENCE MODELS FOR FLUID SIMULATION AT ANY MESH RESOLUTION

SEPTEMBER 2007

JASON A. GADEBUSCH

B.S., WORCESTER POLYTECHNIC INSTITUTE

M.S.M.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor J. Blair Perot

Solving the Navier-Stokes equations using direct numerical simulation (DNS) is computationally impractical, especially at high Reynolds numbers. Recent technological advances in supercomputing have paved the way for Large Eddy Simulations (LES) to circumvent this problem by resolving large scale turbulence motions and modeling only the small (subgrid) scales. However, LES modeling still requires advanced knowledge of the turbulence (integral length scale must lie in the inertial range) and LES models are currently very simplistic. Because of this, there has been considerable interest in hybrid turbulence models, which can perform either Reynolds Averaged Navier-Stokes (RANS) modeling or Large Eddy Simulation (LES). With a hybrid model, one can obtain initial predictions with coarse meshes (RANS) or more accuracy can be obtained with finer meshes (LES). The self-adapting model proposed in this work is fundamentally different from prior LES models and current hybrid models in that it achieves a completely natural evolution from RANS to LES

to (with enough mesh resolution) DNS. In this work, transport equation (RANS) models are used to predict the subgrid-scale stress tensor. The model predictions are compared to DNS data of isotropic decaying turbulence, and the results indicate that this modeling approach is practical and efficient. In addition, this approach is extensible and not restricted to a particular (RANS) transport equation. Theoretically any transport equation model can be utilized.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Turbulence models are frequently classified by the ratio of how much turbulent energy is represented by the model compared to how much turbulent energy is computed via first principles. RANS (Reynolds averaged Navier-Stokes) models represent the most turbulent energy in the model. LES (large eddy simulation) computes considerably more of the turbulent energy via first principles and DNS (direct numerical simulation) simulates all the turbulent energy correctly and models none. If a more detailed terminology is desired, in between RANS and LES lies URANS (unsteady RANS) and VLES (very large eddy simulation). It is sometimes useful to use the term QR-LES (quasi-resolved LES) referring to an LES simulation that is very nearly DNS. The range of these turbulence models are shown in Figure 1.1 in relation to a 3D turbulent energy spectrum. Each model, tries to represent the energy in the spectrum to the right of the model's name.

Classic subgrid-scale models are algebraic in nature and assume that equilibrium is present between the unresolved and resolved scales. The motivation for using a transport equation model is both theoretical and practical. Theoretically, a transport equation model can capture the inherently non-equilibrium subgrid-scale turbulence physics more accurately. Practically, a transport equation based model can easily transition to a RANS or URANS model at coarse mesh resolutions. In low Mach number flows, the solution for the resolved pressure dominates the calculation time, and the solution of additional transport equations are not expected to incur a large performance penalty.

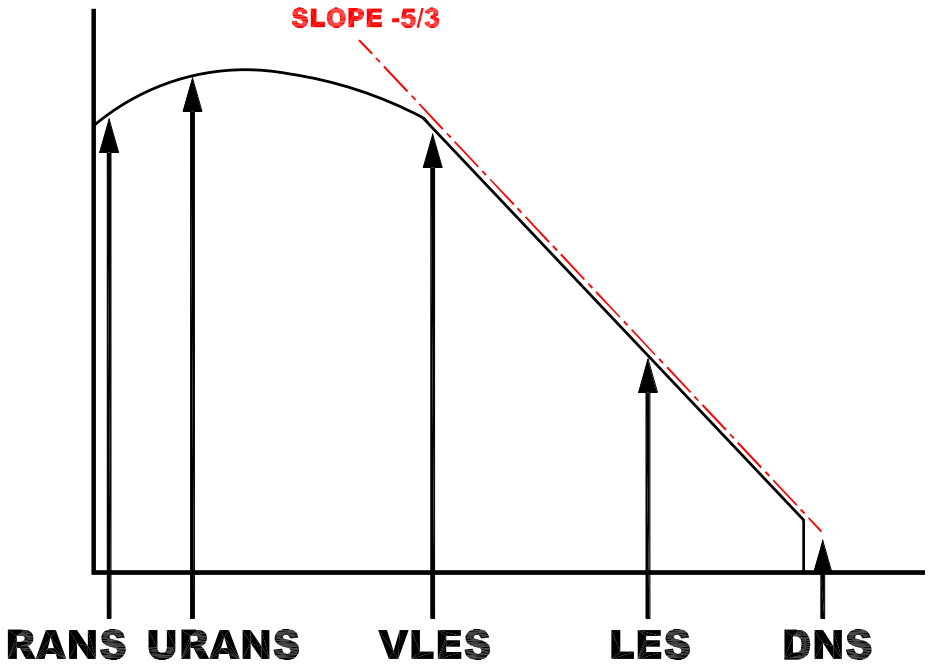

Figure 1.1. Illustration of turbulence modeling (RANS-DNS) on a 3D energy spectra.

Very recently there have been a number of hybrid turbulence models developed that are designed to be able to perform over a broader range of the spectrum (i.e. URANS down through LES) depending on the situation. For example, the very popular DES (detached eddy simulation) model [24] behaves like a RANS or URANS model near walls, but away from walls the lengthscale is changed to the mesh size and the model has an LES character. Other hybrid models do not change their character based on location relative to a wall, but on whether the mesh is much smaller than the energy containing turbulence scales (leading to LES) or not (leading to RANS or URANS). The earliest implementation of such an approach Speziale [25] used classic LES (Smagorinsky) and RANS (k/ϵ) models to solve for both an LES and a RANS eddy viscosity and then blended these two viscosities together based on a function of the mesh size. Girimaji [10] has developed a hybrid model that can change its character based on input from the user (the user sets the desired ratio of modeled

turbulent kinetic energy). The SAS model of Menter [14] is an attempt to fix the lengthscale deficiency of DES. The SAS idea has also been applied to k/ϵ models (and involves adding another term to the dissipation equation similar to the RNG correction).

In practical applied simulations it is not always possible to ensure that the mesh spacing lies in some inertial or self-similar regime. This occurs very near walls, where there is no inertial range, or because the flow is too complex to predict these ranges *a priori*. In addition, in many situations it is wasteful to perform LES in many regions of the computation (flat plate boundary layers and simple mixing layers) where RANS models are known to perform quite well. LES becomes more useful if it can function in the RANS limit as well. Classic LES models can not function in the RANS (coarse mesh) limit because the mesh size in RANS is not a useful indicator of the turbulent length scales. The mesh size is also not appropriate in the DNS (fine mesh) limit. In this thesis, a self-adapting turbulence modeling approach is shown that works for any mesh resolution and over the entire energy spectrum. It can therefore do, RANS, URANS, VLES, LES and even DNS. More importantly, the character of the model is not set by the user or geometric location, but instead will adapt to whatever level the mesh can support. The proposed approach therefore models only as much turbulent kinetic energy is necessary (for a given mesh) and resolves as much of the energy using first principles as possible. It is not technically correct to consider the proposed approach to be a hybrid model in the classic sense (though it has many similarities to those models) because it does not blend an LES and a RANS model together. The proposed model is closer to DES in philosophy in that it is a single set of transport equations that changes its character (RANS, LES or DNS) depending on the flow situation. However, in contrast to DES this change does not depend on the distance to the wall, but rather on the available mesh resolution.

1.1 Isotropic Decaying Turbulence

In order to validate a new approach to turbulence modeling, DNS simulations for isotropic decaying turbulence (similar to [6]) were obtained for comparison against model predictions. Admittedly isotropic decaying turbulence is not a very complex flow, however its simplicity allows for rapid testing and implementation at a low cost. This is a necessary step toward the development of any new modeling approach. Figure 1.2 shows a mid-plane slice of an initial condition 256x256x512 isotropic field for reader visualization.

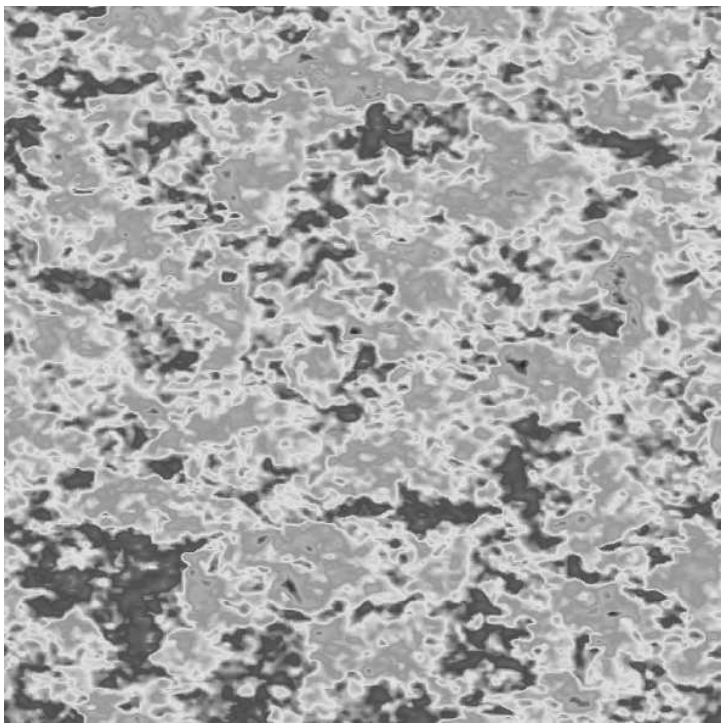

Figure 1.2. Mid-plane slice (x-y plane) through initial 256x256x512 isotropic field, Re = 640.

It is apparent from Figure 1.2 that the 256x256x512 initial condition appears turbulent and clearly isotropic (no preferential direction for fluid movement). It will be shown that the proposed approach does indeed give good predictions for

the turbulent quantities for this test case. However, it must be stressed that this methodology can easily be applied to any flow field, not just isotropic turbulence.

CHAPTER 2

MATHEMATICS

2.1 Mathematical Preliminaries

The classic mathematical theory behind RANS and LES makes these two modeling approaches look fundamentally different. RANS is based on ensemble averages and LES on filtering. At first glance, the possibility of a single model that does both (without some sort of switch or blending function) seems remote. However, a closer examination by Germano [8] revealed some very important insights. Most importantly, the exact but unclosed governing equations for RANS and LES (and URANS, VLES and DNS) are mathematically identical. While the RANS equations can be derived from the assumption of ensemble averaging and the LES equations from filtering operations, these assumptions are overly restrictive and neither system must be derived with those assumptions. The only required assumption is that the velocity field can be split into two parts and that this splitting operation commutes with differentiation. With this assumption the equations for turbulence evolution are (from Appendix A)

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{u_i}\,\overline{u_j}) = -\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u_i}}{\partial x_j^2} - \frac{\partial R_{ij}}{\partial x_j} \tag{2.1}$$

where $\overline{u_i}$ and $\overline{p}$ are the computed velocity and pressure and $R_{ij} = \overline{u_i u_j} - \overline{u_i}\,\overline{u_j}$ is the unknown turbulent stress tensor. The exact (but unclosed) evolution equation for this stress tensor is

$$\frac{\partial R_{ij}}{\partial t} + \overline{u_k}\frac{\partial R_{ij}}{\partial x_k} = \nu\frac{\partial^2 R_{ij}}{\partial x_k^2} - (R_{jk}\frac{\partial \overline{u_i}}{\partial x_k} + R_{ik}\frac{\partial \overline{u_j}}{\partial x_k}) - \frac{\partial}{\partial x_k}T_{ijk} \\ -(< \frac{\partial p}{\partial x_i}, u_j > + < \frac{\partial p}{\partial x_j}, u_i >) - 2\nu < \frac{\partial u_i}{\partial x_k}, \frac{\partial u_j}{\partial x_k} > \quad (2.2)$$

where the bracket operation is given by $< a_i, b_j > \equiv \overline{a_i b_j} - \overline{a_i}\,\overline{b_j}$ and the turbulent transport is defined by $T_{ijk} \equiv \overline{u_i u_j u_k} - \bar{u}_i R_{jk} - \bar{u}_j R_{ik} - \bar{u}_k R_{ij} - \bar{u}_i \bar{u}_j \bar{u}_k$. The well known closure problem arises when the evolution equations (for a set of statistics) contain additional statistics to the set in consideration. Without the additional statistics, these equations cannot be solved for. These equations therefore have more unknowns than equations and are said to be unclosed. In order to solve these equations, models are used to represent the unknowns and hence a solution can be obtained. The turbulent transport and bracketed terms represent these unknown terms and will require a model if the system is to be solved.

In RANS the overbar might denote an ensemble average. In LES the overbar might be an explicit filtering operation. However, it can also be an implicit operation, because in practice (when these equations are modeled and then solved on a computer) the overbar operation is never actually performed. In this case it is assumed that an overbar represents whatever the calculation computes. It is not possible to prove that an implicit filter commutes with differentiation but it is a fairly reasonable assumption to make (at least to first order).

To accurately model some of the terms in 2.1 and 2.2, a third 'scale' equation is often postulated that captures the turbulent energetic length or timescale. The epsilon equation [12] is perhaps the most commonly used scale equation, but omega (inverse timescale) [26] and lengthscale equations [22, 16] also are possible and can be advantageous. These scale equations can, in theory, be derived from first principles but the number of modeling assumptions, in practice, makes them largely empirical.

Starting from these exact equations numerous modeling approaches are possible. In order of increasing simplicity a brief description is given for some of the more common approaches. Reynolds stress transport (RST) models use a modeled form of equation 2.2. The more popular two equation RANS models (such as k/ϵ or k/ω) are a simplification of equation 2.2 from a tensor equation to a scalar equation (by taking its trace). The primary unknown, R_{ij} must then be reconstructed from this scalar kinetic energy, k, using a hypothesized algebraic relation such as the eddy viscosity hypothesis. One equation models, such as the Spalart-Allmaras model [23] used in DES [24], solve a single transport equation directly for the eddy viscosity and use an algebraic expression for the lengthscale (such as the distance to the wall). Classic LES models (such as Smagorinsky and its variants) are the simplest of all, as they solve only 2.1 and algebraically obtain the necessary length and timescales from the mesh size and the resolved velocity gradients. Traditionally, LES models have used the simplest modeling approach because of the issue of cost. Early attempts at using more complex transport equations were deemed not worth the extra effort. Deardorff [7] used a RST model and Schumann [21] used a k/ϵ model. Models and computing power have changed considerably in the 30 years since those first simulations. Some more recent LES models now carry a single transport equation. This is certainly the case in DES, and a kinetic energy transport equation was used by Ghosal e.t. al. [9]. As the mathematical analysis of Germano [8] makes clear, there is no fundamental reason why these more complex modeling approaches (used currently only by RANS models) can not also be applied to LES. The apparent natural evolution of turbulence models to include more physics and therefore complexity, suggests that two-equation transport models for LES are, in fact, the next logical step.

2.1.1 Two-Equation Self-Adapting LES Model

The k/ϵ system is used in this work in order to reach the largest audience possible. There are very good reasons to prefer other two-equation model systems. However, since all two-equation transport models are closely related, the proposed modeling ideas can be easily generalized to these other transport equation models. The unclosed equations 2.1 and 2.2 will be modeled using the following transport equations,

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i\, u_j) = -\frac{\partial (p + \frac{2}{3}k)}{\partial x_i} + \frac{\partial}{\partial x_j}[(\nu + \nu_T \alpha)(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})] \tag{2.3}$$

$$\frac{\partial k}{\partial t} + \frac{\partial}{\partial x_j}(k u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_k})\frac{\partial k}{\partial x_j}] + \alpha P - \epsilon \tag{2.4}$$

$$\frac{\partial \epsilon}{\partial t} + \frac{\partial}{\partial x_j}(\epsilon u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_\epsilon})\frac{\partial \epsilon}{\partial x_j}] + \frac{\epsilon}{k}[C_{\epsilon 1} P - C_{\epsilon 2}\epsilon] \tag{2.5}$$

where the overbar on the velocity and pressure have been dropped for convenience. The production is given by $P = \nu_T(u_{i,j} + u_{j,i})u_{i,j}$ and eddy viscosity is given by $\nu_T = C_\mu \frac{k^2}{\epsilon}(\frac{k}{k+k_r})$. The addition of the ratio of model to total turbulent kinetic energy $(\frac{k}{k+k_r})$ was included in the eddy viscosity formula to help adjust the turbulent viscosity at any mesh resolution. The constants are fairly standard k/ϵ constants, $C_{\epsilon 1} = 1.55$, $\sigma_\epsilon = 1.2$, $\sigma_k = 1.0$, $C_\mu = 0.18$. The parameter $C_{\epsilon 2} = \frac{11}{6} f + \frac{25}{Re_T} f^2$ is sensitive to the local turbulent Reynolds number $Re_T = \frac{k^2}{\nu\epsilon}$ of the modeled turbulence via the function $f = \frac{Re_T}{30}(\sqrt{1 + \frac{60}{Re_T}} - 1)$ as per the analysis of Perot and de Bruyn Kops [16]. This varies $C_{\epsilon 2}$ from its theoretical limits of 11/6 at high Reynolds numbers to 3/2 at low Reynolds numbers. Any Reynolds number dependent $C_{\epsilon 2}$ would probably be sufficient. Reynolds number dependence is important in LES because the effective turbulent Reynolds number $Re_T = k^2/\nu\epsilon$ gets smaller as the mesh is refined (and goes toward zero in DNS). For incompressible flow, the pressure in 2.3 is determined from the constraint $u_{j,j} = 0$.

2.2 Backscatter of Energy

One key component of a self-adaptive turbulence model is that it must be able to backscatter energy from the unresolved (modeled) turbulence to the calculated (resolved) velocity field. For example, a fine grid (128^3) simulation of isotropic turbulence would resolve most of the energy and very little would need to be modeled. However, if this simulation were set up incorrectly, and most of the energy were defined to be modeled and very little resolved (i.e. initialized from a RANS simulation), the adaptive turbulence model should displace energy from the modeled kinetic energy and energize the resolved velocity field to correct this error. Figure 2.1 illustrates this desired effect of backscatter and forward scatter for a 1d energy spectra. Here the resolved turbulence is on the left and the modeled turbulence is to the right (shaded), the arrows indicate the direction of energy flow.

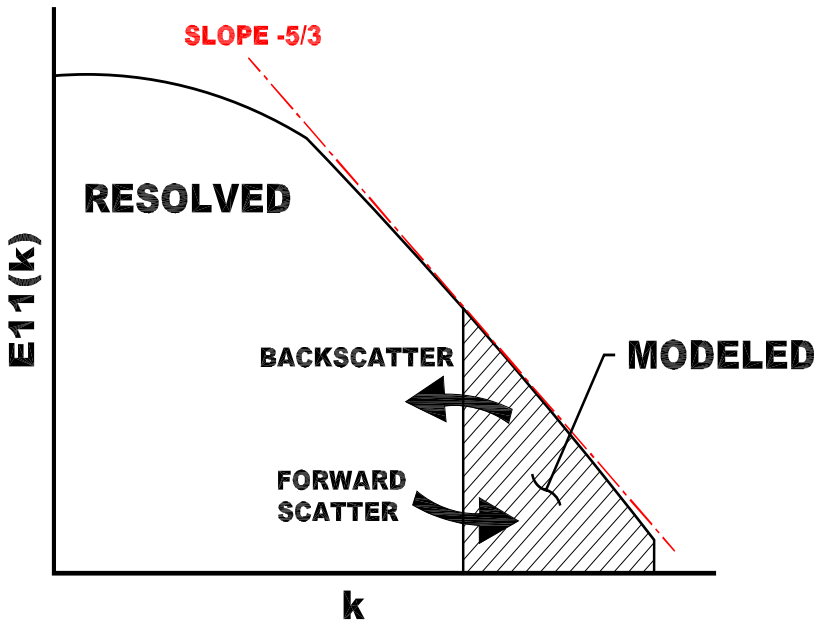

Figure 2.1. Illustration of backscatter and forward scatter on a 1D energy spectra.

With a feature to control the flow of energy, a turbulence model could indeed give accurate flow predictions, correct errors in the initial conditions, and most importantly, perform these functions without any user intervention. The idea of allowing backscatter in a turbulence model is not a new one. It has been shown by Chasnov [3] and Carati et al. [1], that the $-5/3$ power law in isotropic decay is better predicted by LES (dynamic) models that account for backscatter. Along similar lines, classical DES can not backscatter energy but it has been recently shown by Piomelli et al. [18] that adding noise, a crude form of energy backscatter, to the DES model improved results for channel flow.

If $\alpha = 1$, this system (Equations 2.3, 2.4, 2.5) is very close to a classic k/ϵ model. The proposed formulation assumes that the turbulent stress tensor is reconstructed using the eddy viscosity hypothesis, $R_{ij} = \frac{2}{3}k\delta_{ij} - \nu_T\alpha(u_{i,j} + u_{j,i})$. This of course is the simplest reconstruction hypothesis to use. Nonlinear eddy viscosity and algebraic Reynolds stress models use more complex (but still algebraic) reconstructions. Again, the proposed ideas can easily be generalized to that context as well. The simplest model possible is used in this work in order to focus as directly as possible on the key idea - it is possible to develop turbulence models that automatically work at any mesh resolution.

RST models based on equation 2.2 can, and do, backscatter energy. However, any model based on a positive definite eddy viscosity (such as the classic k/ϵ model or any two-equation model) is too simplified and can not backscatter energy. The eddy viscosity is always positive and therefore always removes energy from the mean flow and puts it into the modeled kinetic energy, k, where it is eventually dissipated by ϵ to heat. This is the correct average behavior for a turbulence model, but not necessarily locally correct (in space or time).

The additional parameter α has been added to the classic k/ϵ model to correct this important flaw and control the energy flow. Usually α is positive (and order

1), but it can become small or negative. When $\alpha < 0$ the model is backscattering energy. This parameter is not a model constant, instead it is a field that varies in space and time, so that backscatter can happen in different regions at different times. The transfer variable, α, also appears in Equation 2.4, the k-equation, so that the total kinetic energy (modeled k plus resolved k_r) is a conserved quantity and can only disappear via dissipation to heat. Its presence is not necessary in the scale equation, Equation 2.4.

When $\alpha < 0$, the eddy viscosity in 2.4 is essentially negative. Negative viscosity is anti-diffusive, it amplifies (rather than damps) existing resolved velocity fluctuations. It amplifies small wavelength modes (those closest to the mesh resolution) the most rapidly. This is a very reasonable model for backscatter. It is not injecting energy via some random forcing of the resolved flow, rather it works to enhance the existing instabilities and modes. Moreover, the energy transfer is local in spectral space. It tends to take energy from the model (which has most of its energy at scales just below the mesh resolution) and preferentially delivers it to the resolved flow at almost the same length scale (but just above the mesh resolution).

The model for α the energy flow parameter, uses ideas from error estimation for mesh adaptation. If the error in the computed solution is small, then the mesh is deemed to be sufficient for first principles simulation and the model should go away (the modeled kinetic energy should become very small). This is akin to the situation described earlier, where a 128^3 grid is initialized with a RANS solution and some small resolved flow fluctuations. The model will detect this situation as very highly resolved (lots of mesh resolution). Due to energy conservation inherent in Equations 2.3 and 2.4, the modeled kinetic energy can not just disappear. In order to become small the modeled kinetic energy must be transferred somewhere - the only possibility is to the resolved flow. Small estimated errors (in the resolved quantities) implies there should be energy backscatter (the mesh can handle more fluctuations via direct simulation).

At some point later in time, the resolved velocity on the 128^3 mesh will be fully energized. If the Reynolds number is high, then a 128^3 mesh is still not sufficient to perform DNS (a model is still necessary). In this case the error estimate will become larger and larger until α becomes positive and a normal forward energy scatter down the energy cascade will be imposed. The larger the error, the more energy is fed to the modeled kinetic energy and the more the model influences the resolved flow evolution. If the Reynolds number is low enough, than a 128^3 mesh may actually be sufficient resolution for DNS and the error estimate stays small. In this case the model continues to backscatter until the model has almost no energy. At this point the model has no affect on the resolved modes and DNS is achieved.

2.2.1 Energy Transfer Variable

The proposed equation for the energy transfer is,

$$\alpha = 1.5(1.0 - C^*(\frac{k}{k+k_r})^2[(\frac{\Delta x_i}{\sqrt{k_r}}\frac{\partial\sqrt{k_r}}{\partial x_i})^2 + 0.11]^{-1}) \tag{2.6}$$

where k_r is the resolved turbulent kinetic energy (at a certain location and time), k is the modeled kinetic energy, and $C^* = 0.28$. The quantity $(\Delta x_i\frac{\partial\sqrt{k_r}}{\partial x_i})^2/k_r = ((\Delta x\frac{\partial\sqrt{k_r}}{\partial x})^2 + (\Delta y\frac{\partial\sqrt{k_r}}{\partial y})^2 + (\Delta z\frac{\partial\sqrt{k_r}}{\partial z})^2)/k_r$ is a dimensionless measure of the error (similar to what is sometimes used in mesh adaptation). In this formulation the resolved kinetic energy $k_r = \frac{1}{2}(u_1^2 + u_2^2 + u_3^2)$ is the indicator function that is being used to estimate the mesh resolution. If the flow is DNS or over-resolved (like a RANS initial condition on an LES mesh) then this quantity is small, its inverse is large (but limited away from infinity by the fairly arbitrary 0.11 term), and the model tends to backscatter energy. In contrast, normal energy transfer (from resolved scales to the modeled scales) occurs in the regions of the flow where the gradient length scales are comparable to the mesh size. On very coarse meshes, RANS like behavior should be recovered. In this limit, k_r is expected to be very small, but note that $(\Delta x_i\frac{\partial\sqrt{k_r}}{\partial x_i})^2/k_r$

will remain finite and independent of the magnitude of the resolved fluctuations. For the simulations of isotropic turbulence performed, this term obtains an average value of 0.8 in the RANS limit. This means that $\alpha \to 1.5 - 0.42/(0.8 + 0.11) \approx 1.04$ and the standard RANS model is very closely recovered in the RANS limit.

The particular form of the energy transfer function was developed and tuned solely to obtain the correct limits. Many other functional expressions and/or indicator quantities are certainly possible. The goal of this work is not to advocate for this particular function but to demonstrate that self-adaptive turbulence models are possible, and this particular function serves this purpose adequately.

CHAPTER 3

SIMULATION RESULTS

The governing equations were solved using a Cartesian staggered mesh (Figure 3.1) method [11] with exact projection [2] for the pressure solution. Isotropic decaying turbulence was calculated using periodic boundary conditions on a box of size of $18\pi\, x\, 18\pi\, x\, 36\pi$. The numerical method is second order accurate in space and time and locally conserves mass and momentum. In addition, it locally conserves circulation and kinetic energy in the absence of viscosity. The code is fully parallel (using MPI) and optimized for execution on PC clusters.

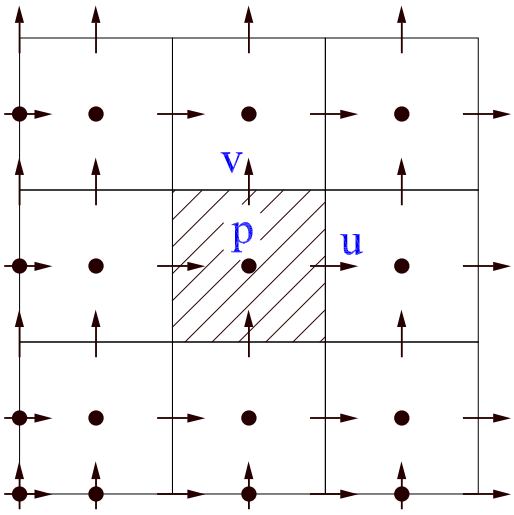

Figure 3.1. Fully staggered arrangement of velocity (u,v) and pressure (p).

3.1 High Re Isotropic Decay

In isotropic turbulence, the turbulent kinetic energy decays with time. This process can be simulated with a simple RANS model using only one spatial grid cell, with LES using many cells, or with DNS using enough cells to resolve the smallest length scales of motion. Results from such a test are shown in Figure 3.2. The ordinate shows total kinetic energy $(k + k_r)$ normalized by the initial total kinetic energy at time t=0.0, the abscissa τ, is given by simulation time (seconds) non-dimensionalized by the inverse time scale $(\frac{\epsilon_t}{k_t})$ at time t=0.0 (where $_t$ indicates total quantities). Normalization of the abscissa can be interpreted as 'eddy' turn over times.

The initial turbulent Reynolds number, $Re_T = \frac{k_t^2}{\nu\epsilon_t}$, for this test case is 640, comparable to the Comte-Bellot and Corrsin 1971 experiment [4]. The DNS result (similar to [6]) was performed independently by de Bruyn Kops on a 768x768x1536 mesh using a Fourier spectral method and is given by the large circles. Many aspects of the DNS (including spectra) have been closely compared with Comte-Bellot and Corrsin and shown [5] to agree well. The mesh resolution is identical in each direction, with the box size in the z-direction twice as large.

A number of different simulations were performed using mesh resolutions from 1x1x2 to 256x256x512. In each case the model is identical and only the mesh and initial conditions are changed. The initial conditions for the full DNS were obtained by running, for a long time with as little forcing of the large scales as possible, to maintain the kinetic energy. The initial condition is therefore not random Fourier modes - but Navier-Stokes turbulence. Each model initial condition was formed by averaging the same initial 768x768x1536 velocity field to the appropriate mesh size using a simple box average. The initial modeled kinetic energy at each mesh location was then determined by comparing the exact 768x768x1536 velocity field to the smoother coarse mesh field and calculating the sum of its difference. The dissipation was calculated similarly, by using a box average of the exact DNS dissipation field.

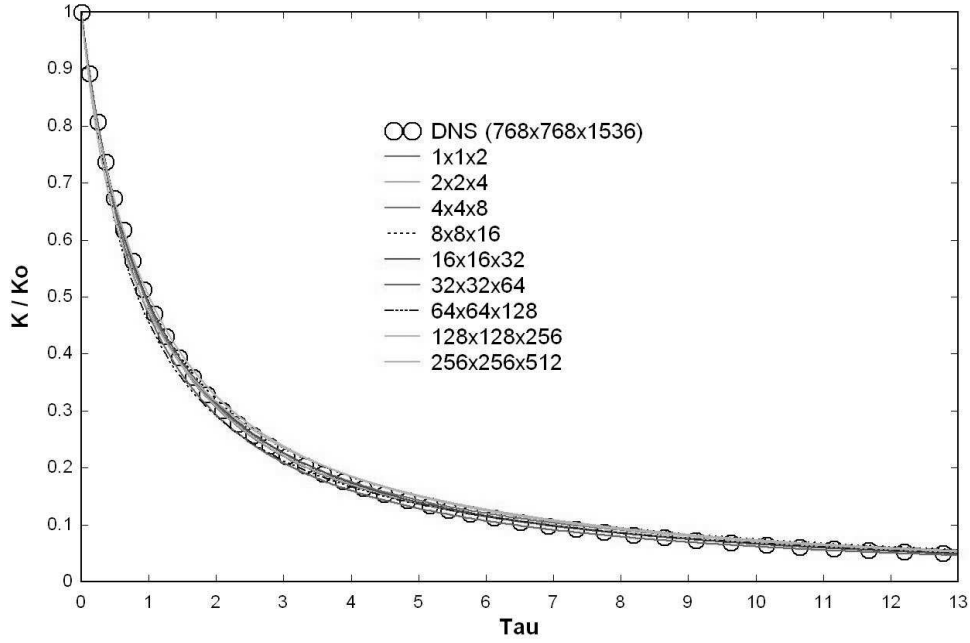

Figure 3.2. Total turbulent kinetic energy predictions of isotropic decay at initial Re=640.

The simulations in Figure 3.2 show that at any mesh resolution, the model predicts the decay of the turbulence well. The very lowest mesh resolution is clearly a RANS (or URANS) simulation and the largest mesh, 256x256x512, is a QR-LES (nearly DNS) simulation (with the mesh size roughly in the inertial range). The intermediate resolutions might be considered URANS, VLES, or LES. The spectra for the initial conditions are shown in Figure 3.3. These spectra represent VLES (32x32x64), LES (64x64x128, 128x128x256), and QR-LES (256x256x512) simulations. The box average of the initial data takes some energy from the lowest wavenumbers but most of the energy from the highest wavenumbers. All energy that is not resolved is modeled by the variable k.

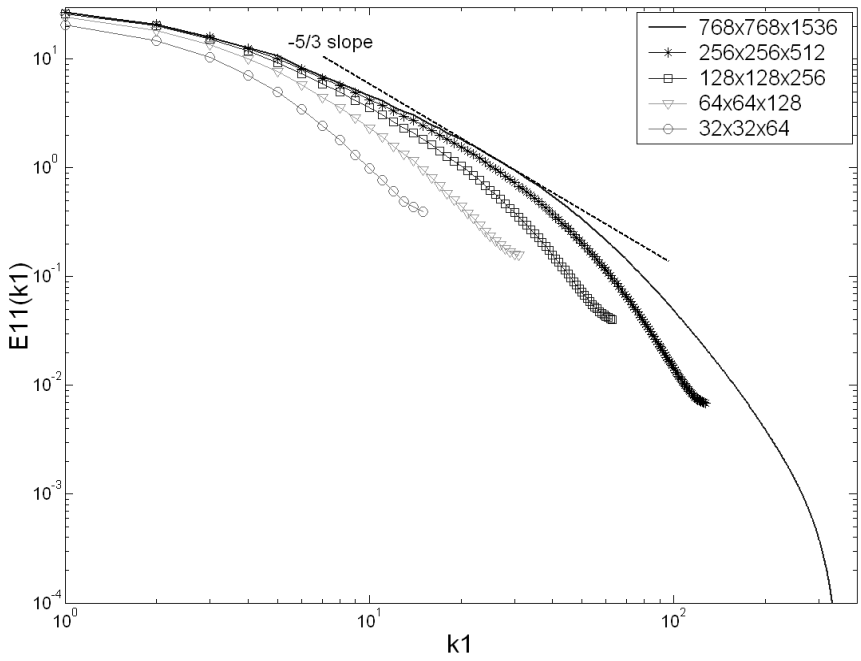

Figure 3.3. 1D energy spectrum of initial conditions for (Re=640) isotropic decaying turbulence.

3.1.1 Turbulent Kinetic Energy Ratio

It might be expected that RANS equations should give RANS (essentially the 1x1x2) results irrespective of the mesh resolution used to solve those equations. This is not the case, for two reasons. First the error will be different for different meshes which will affect the backscatter term. More importantly, the equations are nonlinear, and like the Navier-Stokes equations, have multiple possible solutions.

The ratio of the modeled kinetic energy to the total kinetic energy is shown in Figure 3.4 with one curve for each of the mesh resolutions. The 1x1x2 solution is the top curve, with all its energy contained in the model (giving a ratio of 1.0) and the 256x256x512 simulation is the bottom line, with the smallest ratio of modeled kinetic energy ($<$ 10 percent). Note that these curves are relatively constant but decrease slightly with time (as the simulation proceeds). Even though, the equation system looks like a classic RANS model, it is not. The analysis of Germano [8] shows

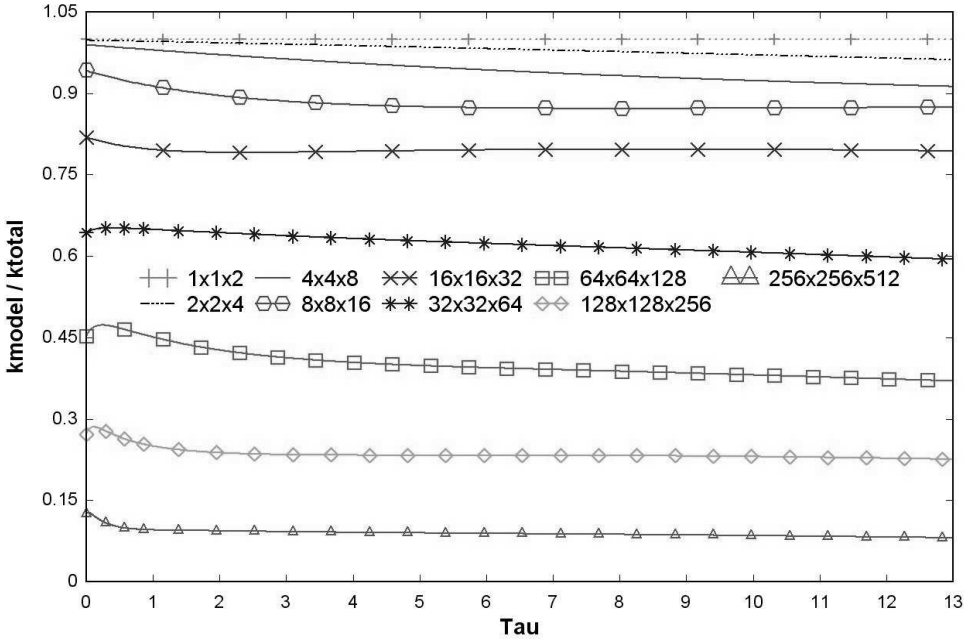

Figure 3.4. Ratio of modeled turbulent kinetic energy to total kinetic energy, Re=640.

that this two-equation self-adaptive model is actually a 'universal' turbulence model applicable at any mesh resolution. There is no tendency for the solutions to move toward the RANS solution (a ratio of 1.0).

The LES solutions stay entirely unsteady and three-dimensional. The slight decrease in this ratio over time is the correct behavior. It is due to the fact that over time the Reynolds number of the flow is slowly decreasing and the mesh can, and therefore does, resolve a larger percentage of the turbulent fluctuations. If the simulation was run long enough, then the Reynolds number would drop sufficiently for the 256x256x512 mesh to perform DNS, and at this point the ratio should be essentially zero. The small variation at the beginning of each simulation shows the initial condition is close to the correct ratio supportable by that mesh, but not perfect. The long time behavior of this ratio is shown in Figure 3.5.

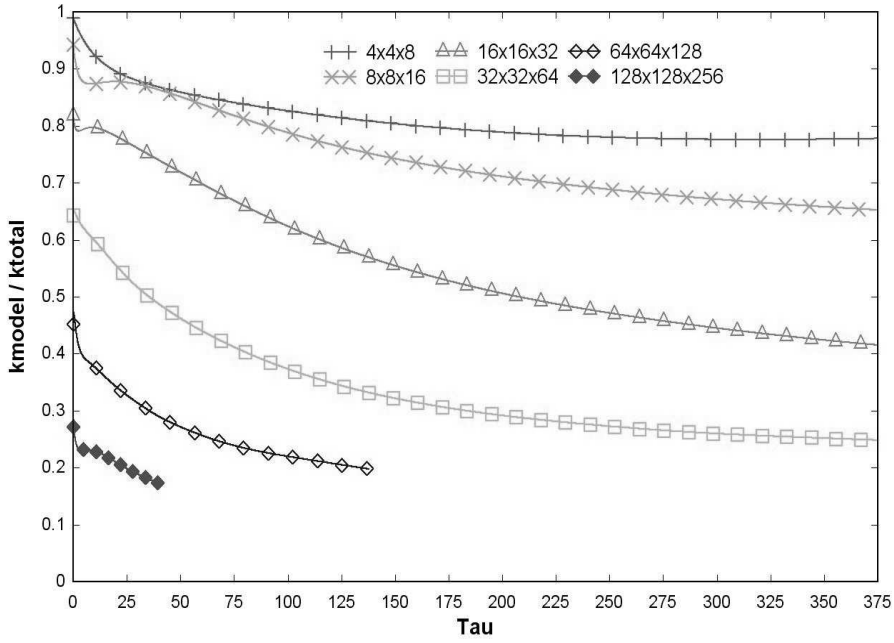

Figure 3.5. Ratio of modeled turbulent kinetic energy to total kinetic energy for extended time, Re=640.

At these very long times, the turbulence scales grow large enough to feel the influence of the box size and the decay rate no longer follows the classical theory for homogeneous decay in an infinite domain. For this reason the total kinetic energy at these long times is not of interest. However, the long time behavior of the model ratio is still of interest. Figure 3.5 shows that the coarse mesh (4x4x8) appears to be approaching the 'RANS' solution (a ratio of 1.0) at very long times. The 8x8x16 simulation initially tries to approach the RANS solution, however this ratio eventually starts to decrease. The finer meshes (16x16x32 - 128x128x256) have enough resolution to compute a solution that does not return to the RANS limit. It is hypothesized that steady (RANS-like) solutions occur when the resolution is not sufficient to maintain an energy cascade. For this test case, the initial large eddy lengthscale of the turbulence ($L = k^{3/2}/\epsilon$) is 6.0, and the 8x8x16 simulation has a mesh size close to 7.0. Nikitin et al. [15] has noted that a similar affect occurs in very under-resolved DES simulations.

3.1.2 Perturbation of Initial Conditions

A truly adaptive model should be able to obtain the correct behavior from incorrect initial conditions. For example, it is of considerable interest to see if a 64x64x128 mesh initialized with a RANS solution can, over time, develop into a full LES simulation. In order to test the model in this way, the initial conditions were either smoothed or sharpened using a filtering operation. The filter used to alter the initial conditions was a nearest neighbor averaging procedure, $u_{ijk}^{filtered} = \beta u_{ijk} + (1-\beta)(u_{i+1jk} + u_{i-1jk} + u_{ij+1k} + u_{ij-1k} + u_{ijk+1} + u_{ijk-1})\frac{1}{6}$. For smoothing, $\beta = 0$ was used. This replaces the value at a mesh point by the average of its nearest neighbors. This type of filter removes energy primarily from the highly oscillatory modes with wavelengths close to the mesh size. In spectral terms it damps the spectra in the region just above the cutoff wave number. The effect is shown in Figure 3.6, which shows the original initial spectra for the 64x64x128 simulation (normal, smooth, sharpened) as the top three curves. For now the reader is referred to only the top three curves as the bottom curves (almost on top of each other) will be discussed in more detail shortly.

Sharpening the velocity field is performed by using $\beta = 1.5$. This adds energy to the existing high frequency modes. Figure 3.7 shows the affect of smoothing and sharpening the initial conditions on the kinetic energy ratio. When smoothing is used, energy is removed from the resolved modes. In order to keep the total kinetic energy the same, the model now must start with more energy. The ratio therefore starts higher than before. As time proceeds the model achieves the same ratio irrespective of the initial conditions. At early times, the smoothed solution has less error and therefore backscatters somewhat more than the unperturbed initial condition. This removes energy from the model and makes the ratio decrease faster, so that it approaches its original state. A similar (but opposite) process happens when the spectra is sharpened. In this case, the model senses that the mesh can

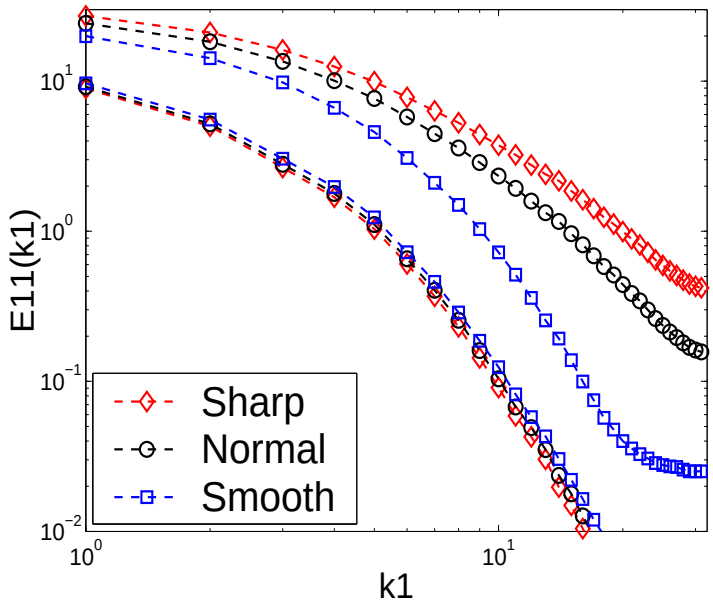

Figure 3.6. 1D energy spectra for 64x64x128 case, Re=640 (Sharp, Normal, Smooth).

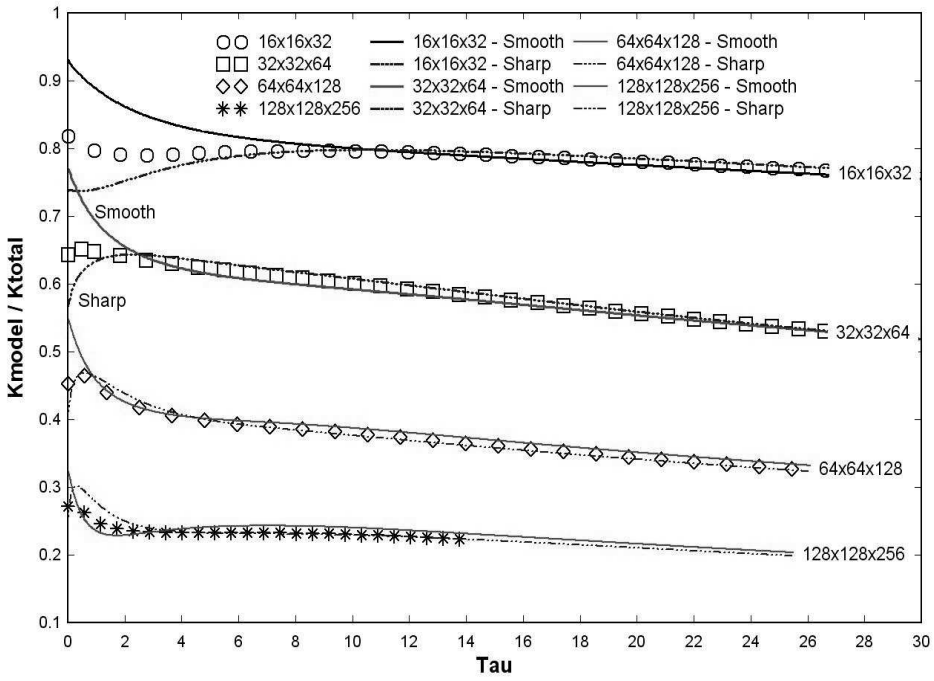

Figure 3.7. Ratio of modeled turbulent kinetic energy to total kinetic energy, Re=640, for perturbed initial conditions.

not support the input resolved fluctuations, and quickly sends that energy to the model. Note that the rate at which the model adjusts to perturbed initial conditions depends on the mesh resolution. The higher mesh resolutions adjust more rapidly. It is hypothesized that the time it takes to transfer the energy scales on the timescale of the turbulence at the cutoff (transfer) lengthscale (k/ϵ).

It could be theorized that Figure 3.7 does not necessarily imply that the model has corrected itself. Although the ratio of ($\frac{k}{k+k_r}$) indeed shows the correct behavior, a more definitive test would be to see if the energy spectra for all three cases (sharp, normal, smooth) are the same. These spectra are shown in Figure 3.6. As mentioned, the top three curves shows the initial energy spectra at (t=0.0s) and the bottom three curves correspond to the spectra at some later time (t=0.4s) where equilibrium is obtained. These spectra (almost right on top of each other) along with the turbulent energy ratio clearly show that the model is self-adapting and does indeed correct itself for perturbed initial conditions.

3.1.3 Alpha Histogram

It had been stated earlier that the energy transfer function (α) was usually positive and order one. In order to visualize exactly what happens during a simulation with correct initial conditions and with perturbed initial conditions (sharp and smooth) Figure 3.8 shows a histogram of α (positive values) and Figure 3.9 shows a histogram of α (negative values).

Figure 3.8 shows that with the correct initial conditions (solid line) α obtains an average value of around 0.87 for this mesh resolution of 64x64x128. Figure 3.9 is an enlarged view of the negative α values from negative one to zero. The histograms also include values of α for the smooth (squares) and sharpened (diamonds) initial conditions whose spectra were shown in Figure 3.6. As mentioned previously, smoothing the velocity field removes energy from the resolved flow and places too

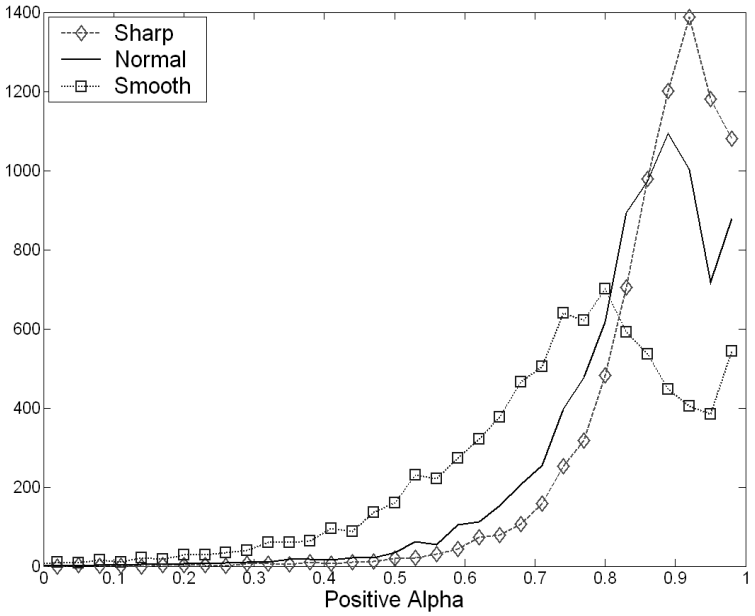

Figure 3.8. Histogram of energy transfer function (α) positive values.

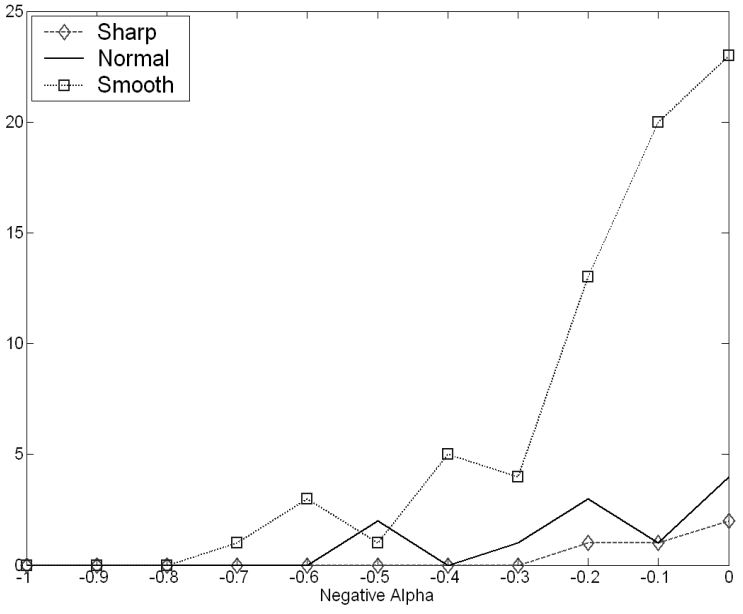

Figure 3.9. Histogram of energy transfer function (α) negative values.

much energy into the model. To compensate for this error Figure 3.8 shows that α takes on a smaller average value and Figure 3.9 shows a drastic increase in negative values (more backscatter). By having more negative values α is correctly displacing energy from the model to the resolved field to correct for the wrong initial condition.

The opposite effect is shown for the sharpened case. The simulation has been started incorrectly with too much energy in the resolved field. Here it is seen that α obtains a slightly larger value than 0.87 in Figure 3.8 to ensure more normal forward transfer of energy. Because of this shift to the right, there is considerably less (almost non-existent) negative values observed in Figure 3.9. This guarantees almost no backscatter and has an increase in forward scatter to correct the initial conditions.

3.2 Low Re Isotropic Decay

In order to verify that the model works in the DNS regime, a lower Reynolds number simulation was performed which is well resolved (DNS) at the largest simulation of 256x256x512. The initial condition for this lower Reynolds number case was generated by running the high Re simulation with a larger viscosity ($\nu = 0.3743\, cm^2/s$) and with a very small amount of modeled kinetic energy ($k = 10^{-3}\, cm^2/s^2$). Using the higher viscosity reduces the Reynolds number and also quickly damps the smallest scales of the turbulence.

The turbulent time scale (k_t/ϵ_t), is plotted versus simulation time (solid line) in Figure 3.10 for the low Re case. As was expected, at early times, the small scales are adjusting to the low Re (solid line). This is due to the simulation reacting to the increased viscosity and reduced modeled kinetic energy. Eventually, a power law decay for the kinetic energy is obtained, and the simulation is considered fully adjusted. A power law decay should appear as a straight line in Figure 3.10. A linear curve fit is given by the plus symbols. The inverse of the slope given by the curve fit is the kinetic energy decay exponent, $n \approx 1.38$. The simulation was considered

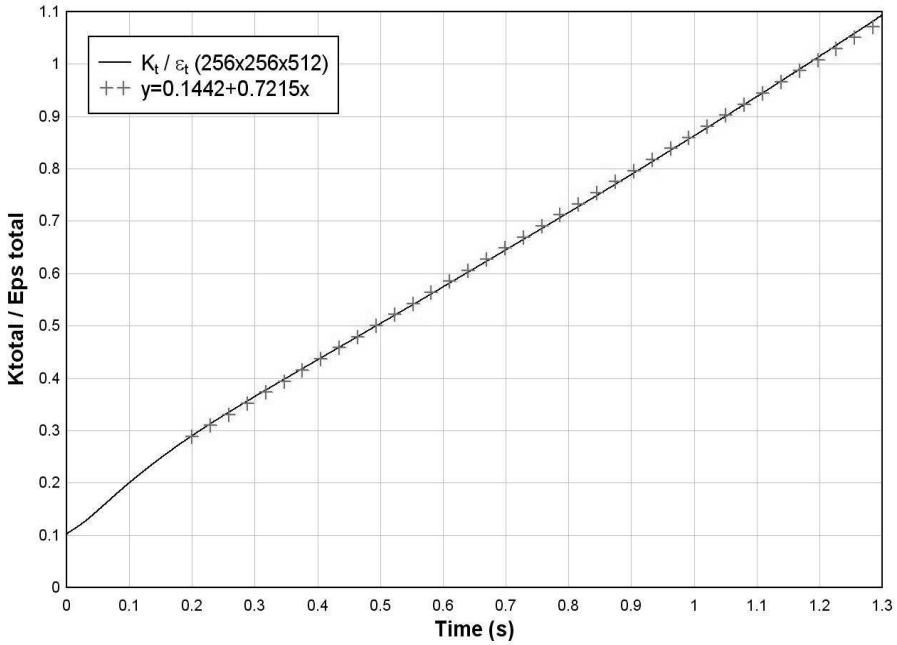

Figure 3.10. Turbulent time scale (k_t/ϵ_t) vs. simulation time for a 256x256x512 simulation with increased viscosity and small amount of modeled kinetic energy (DNS).

to be well developed at a time of 0.14s. This field (at t=0.14s) was used in all the subsequent simulations, as the well developed low Reynolds number (Re=211) initial condition. The same box averaging procedure for the high Re number was also used to create initial conditions for the smaller meshes in the low Re number case.

Figure 3.11 shows the total kinetic energy predictions for the low Re test case. The circles represent the essentially DNS simulation (though the model is on), the lines represent the various simulations from 1x1x2 through 128x128x256. These are in fairly good agreement with the DNS data, as was observed for the high Reynolds number case. The slight discrepancy at long times is most likely due to errors in the RANS model at lower Re.

The ratio of modeled kinetic energy is plotted in Figure 3.12 The full RANS simulation (1x1x2) is at the top of the figure (ratio of 1.0) and the DNS is shown at the bottom of the figure with a ratio of roughly 10^{-3}. Once again it is observed that

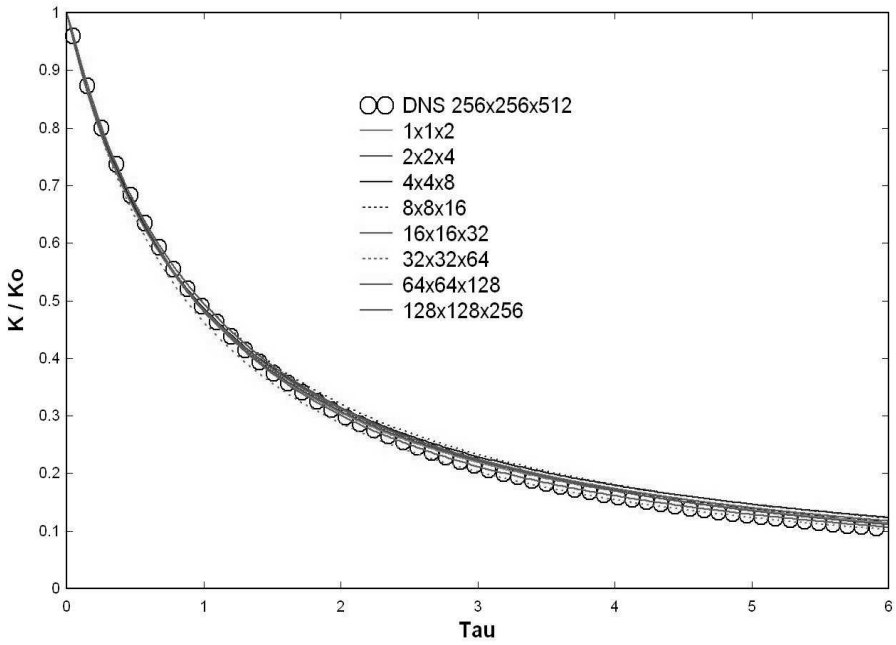

Figure 3.11. Total kinetic energy predictions of isotropic decay at initial Re=211.

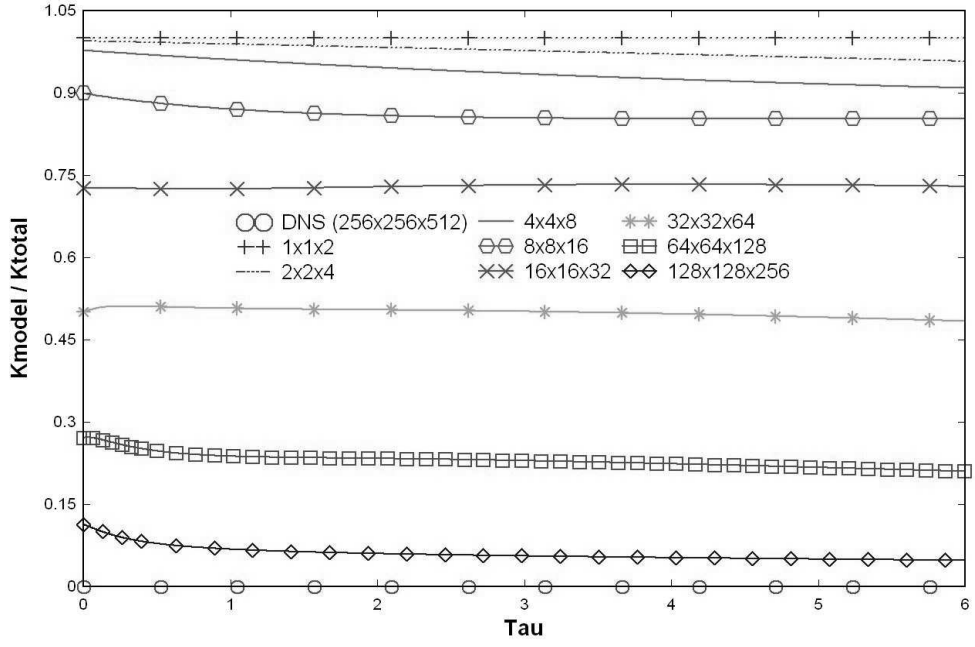

Figure 3.12. Ratio of modeled kinetic energy to total kinetic energy, Re=211.

there is no tendency for these solutions to move toward the RANS solution (ratio of 1.0). The LES solutions stay entirely unsteady and three-dimensional, and the correct decrease in this ratio over time is shown.

3.3 Scaling

In classic LES models the lengthscale is assumed to be proportional to the mesh size, Δx, and the gradients scale like $u_{i,j} \approx \epsilon^{\frac{1}{3}}\Delta^{-\frac{2}{3}}$. The eddy viscosity is then constructed from this scaling and has the form, $\nu_T \approx u_{i,j}\Delta^2 \approx \epsilon^{\frac{1}{3}}\Delta^{\frac{4}{3}}$. The use of the mesh size to infer the lengthscale is why classic LES fails outside of the inertial range (close to the DNS or RANS regimes), and one reason why the self-adaptive model (which predicts the lengthscale and does not infer it) can operate at any mesh size. However, the current model should still obtain the classical LES behavior in its range of applicability. The lengthscale predicted by the model ($L_m = k^{3/2}/\epsilon$) should be proportional to the mesh size, when the model is operating in the LES regime.

Figure 3.13 looks at predicted lengthscale (in log scale) at a fixed time t=0.5 for the various meshes. At small values of Δx (large number of mesh points) it is obvious that the lengthscale is indeed a function of grid spacing as would be expected with classic LES. A reference line has been added (dotted line) with a slope of 1.0 for comparison. If a well defined -5/3 slope were to be observed in the energy spectra, one would assume that classic LES would closely match the reference line in that -5/3 region. Because the spectrum shown (Figures 3.3 and 3.6) show only a weakly observable -5/3 slope, it is not surprising that the lengthscale shown in Figure 3.13 does not match the reference line exactly. One would expect viscous effects (the influence of a viscous lengthscale) to reduce the scaling below linear and this is indeed observed. The figure clearly shows how the turbulent lengthscale is not related to the mesh spacing outside the LES regime.

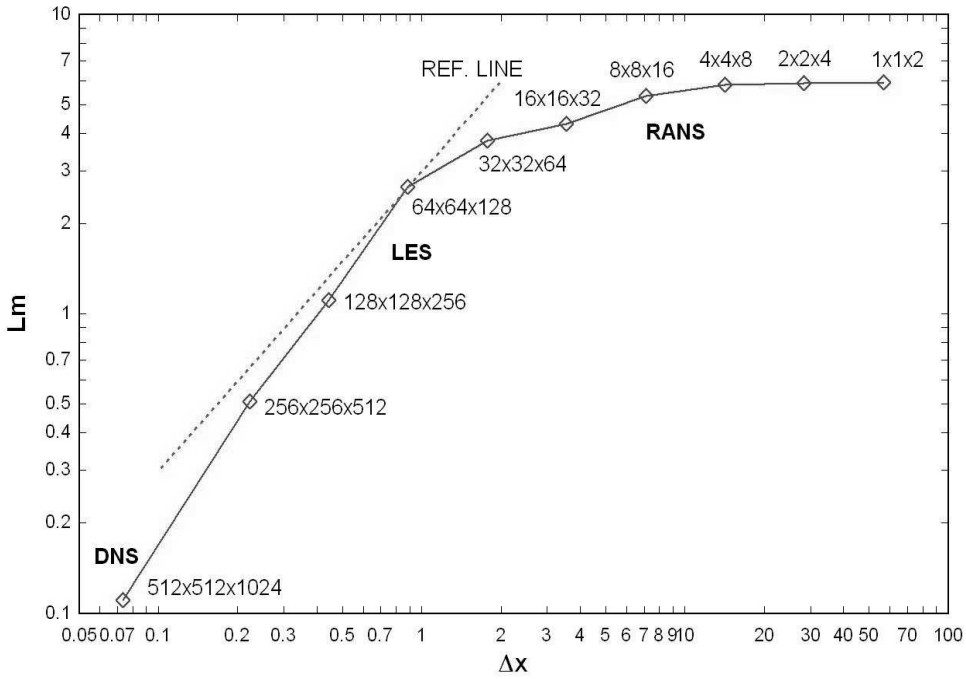

Figure 3.13. Lengthscale behavior at time = 0.5s.

Mesh resolutions smaller than 64x64x128 do not exhibit the linear behavior because now the relevant turbulent lengthscale is no longer related to the mesh size. There is a transition region that one might call VLES(32x32x64), but not full LES. The 8x8x16 mesh might be considered the start of URANS (as shown in Figure 3.4), where the largest scales can just be resolved, and unsteady 3D solutions maintained. These smallest mesh sizes are where the solutions can not sustain a cascade because all the largest turbulence has scales that are smaller than the mesh size. Interestingly, Carati et. al. [1] conducted LES simulations for isotropic decaying turbulence and determined that the smallest mesh size possible for a classic (dynamic model) LES simulation was 48^3. This is in very good agreement with Figure 3.13, and is just where the elbow in the curve occurs.

CHAPTER 4

PARALLEL PROGRAMMING

Even with current technological advances in computer hardware, large simulations can require an extraordinary amount of computer resources. Performing a 256x256x512 simulation for these test cases requires the computer to solve approximately 34 million grid points. Solving this mesh size takes (approx) 2.5 hours per time step with 1 CPU. To compute a solution (ignoring memory limitations for the moment) would take roughly 20 weeks to get data for this large mesh size! It is obvious that a standard PC will not be able to handle the computational requirements for this case anytime soon. In order to decrease simulation time and lower the memory requirements, the code was parallelized. This was accomplished by utilizing the Messaging Passing Interface (MPI) library. MPI is a library of functions for Fortran and C/C++ that distributes information from a single processor to multiple processors. Using more processors allows the larger simulations in this work to be accomplished.

4.1 Boundary Conditions

While MPI facilitates the transfer of data between processors, care must be taken to ensure the correct data is passed between neighboring processors. All simulations performed in this work use periodic boundary conditions as shown on the left side of Figure 4.1.

This figure (left side) shows how data is stored and distributed for periodic boundary conditions on a single processor. All of the interior points are within a domain of Lx and Ly (shaded area) and stored on a single processor. The ghost cells are used for

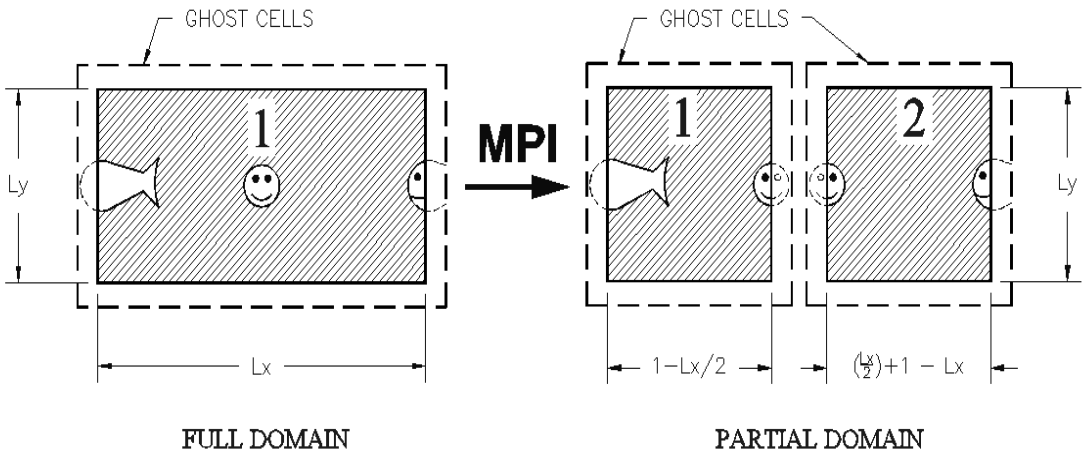

Figure 4.1. Single processor (left side) and MPI (right side) partitioning of the problem.

the periodic boundary conditions and are given by the values of the interior points at each face in this rectangle. Here, Nemo the fish is seen swimming out of the left side of the domain. Because of periodicity, he is wrapped around to the opposite face he just left, and is seen swimming into the domain from the right hand side. Similarly, if he swims out of any other face, he will wrap around to the opposite face he just left. This ensures that Nemo continues swimming inside the periodic domain.

Suppose that the same data set is desired to be split in half, then each processor should get half of the interior points as shown (right side of Figure 4.1). Here the x-direction, with an initial domain of (Lx) has been split in half (Lx/2) for two processors. It is now apparent that care must be taken when determining the correct transfer of data between these two processors. The only way to correctly determine this information is by allowing every processor to locate its neighbor for each of the (cubic) domain's six faces. From Figure 4.1 for the partial domain, interior information must be shared with neighboring processors (given by the smiley face). However, at domain edges, the boundary information should wrap around to the

opposing face, (given by Nemo). An algorithm was developed to do exactly this and is explained in detail in the next section.

4.1.1 Neighboring CPU Data

The algorithm used to compute neighboring CPU information is shown in Appendix B. Figure 4.2 shows a parallel simulation consisting of eight processors. The left side of Figure 4.2 shows the data partitioned together, the right side shows the front and back plane separated for clarity with each of the 8 processors numbered from 0-7. Since each cube of data has six sides it is clear that there will be six neighboring CPU locations calculated for each processor. This algorithm uses simple integer division to allow each processor to use the correct data.

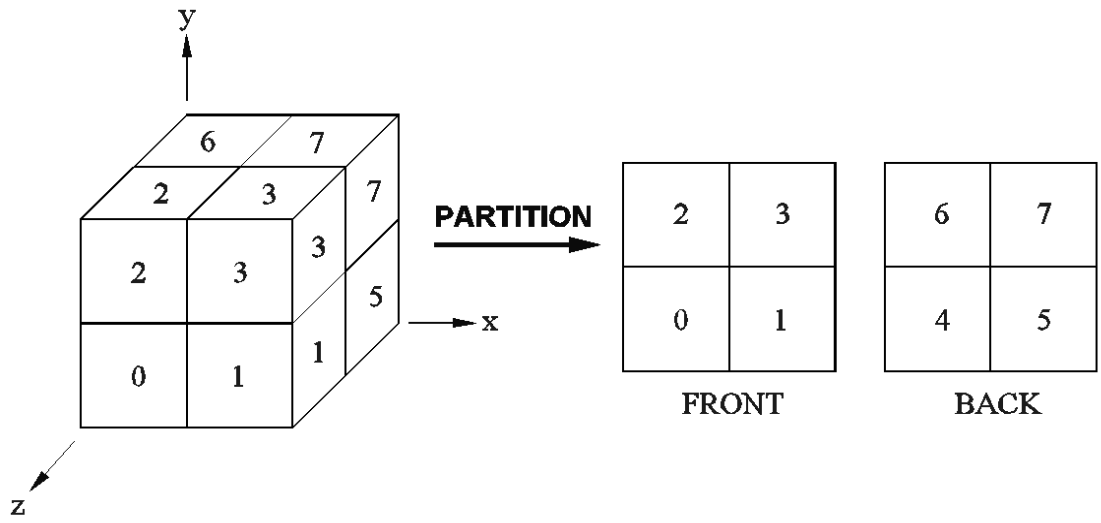

Figure 4.2. Neighboring CPU location.

By visual inspection the processor to the right of 0 is processor 1. There is no processor to the left of 0, however periodicity tells us that the information should wrap around from processor 1. The processor above 0 is processor 2, and below is

also 2 (from periodicity). Finally, the processor in front of 0 is 4 (periodicity) and the processor behind 0 is also processor 4.

This simple reasoning is exactly what the algorithm computes in Appendix B. For the processor in question, the MPI assigned integer 'myid' is 0. Because there are two processors in every direction, $NXP = NYP = NZP = 2$. The values of ix, iy, and iz are calculated as 1. The CPU to the right of processor 0 will be calculated as (CPUinfo1 = 1), the CPU to the left (CPUinfo2 = 1), the CPU at the top and bottom are given by (CPUinfo3 = 2, CPUinfo4 = 2) respectively. Finally, the CPU located at the back and front of the CPU in question are (CPUinfo5 = 4, CPUinfo6 = 4) respectively. Simple integer division based on the variable 'myid' computes neighboring processor locations. The 'If' statements shown in Appendix B are used whenever a processor is on the edge of a domain (such as computing the processor to the left of 0). If any of these processors are on this domain edge, it is computed that boundary information will wrap around to ensure periodicity. It is clear from the above figure that the algorithm has correctly determined the location of the neighboring CPUs relative to processor 0.

It is urged that the extremely interested reader verify that this algorithm (Appendix B) correctly determines all eight CPU locations.

4.1.2 Transfer of Boundary Condition Information

With this background, the actual passing of the boundary information is now explained in detail. The cell indexing convention is shown in Figure 4.3. This shows that the interior points are given by the data values from 1-NX and 1-NY, where NX is the number of points in the x-direction and NY is the number of points in the y-direction. The ghost cells (dashed lines) used for the boundary conditions are placed at index locations around the interior values. The location of cell 1 (0,0), cell 2 (NX+1,0), cell 3 (NX+1,NY+1), and cell 4 (0,NY+1) are the four corners for this

2d example. The correct procedure must be followed in order for these four corner cells to contain the correct data. The z-direction (out of the page) follows the same logic, but for clarity is not shown.

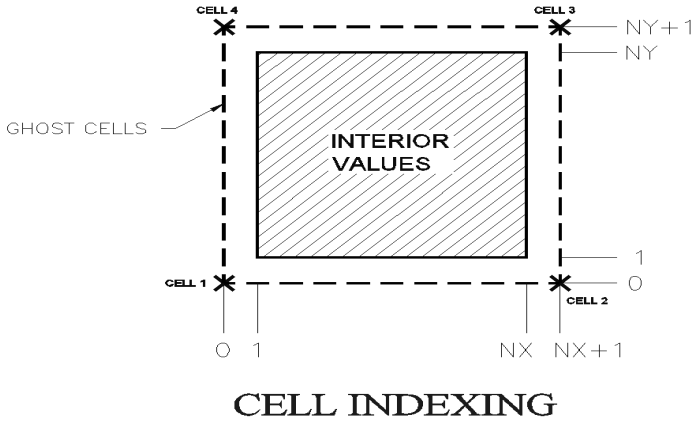

Figure 4.3. Cell indexing convention.

In order to enforce periodic boundary conditions on this cell, there are four update steps for this 2d example. These update steps are shown in Figure 4.4. Step one updates the ghost cells with values from the right side interiors (NX,1-NY) as shown. Because interior values are used, these are correct. The second step takes the top interior values (1-NX,NY) along with ghost cell values at (0,NY) and (NX+1,NY). When transferred to the bottom ghost cell plane, cell 2 now has incorrect information as shown. To correct this cell, the third step is performed. This step takes the left interior values (1,1-NY) along with ghost cell values at (1,0) and (1,NY+1), and transfers these to the ghost cells on the right side. When step three has completed, the incorrect cell 2 value has been replaced with a correct value, but now cell 3 contains bad information as shown. Finally, the fourth step takes the bottom plane (0-NX+1,1) and transfers these corrected values to the top ghost cells at NY+1. Now cell 3 has been replaced with correct data and all ghost cell information is correct. This exact procedure must be followed in this precise order to ensure that all ghost

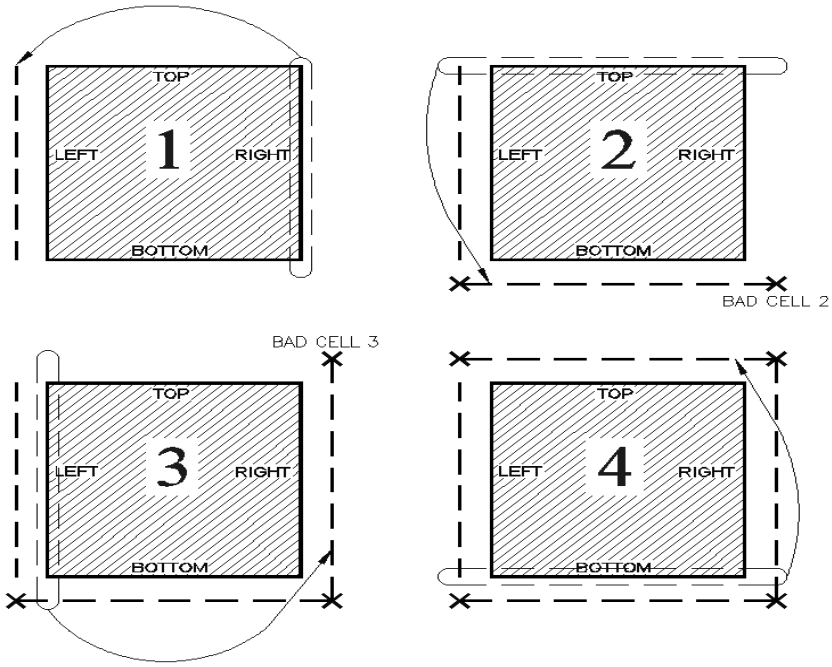

Figure 4.4. Ghost cell update procedure.

cells (especially the corners) are indeed updated with the correct information. For three dimensions (z-direction) would also have two additional update steps (after x and y), following the same logic.

The Fortran code used for the the procedure in Figure 4.4 is given in Appendix C. For brevity only the x-direction will be explained in great detail, it is clear how to apply the same information to the y and z directions. The values of NX, NY, NZ are given by the number of interior points in the domain for each direction. The variable (xx) is a 3D array of data with correct interior domain information, but with incorrect boundary information stored at the ghost cells. The routines Send_BCX and Recv_BCX, are given in Appendix C. These are the procedures that update the ghost cell information.

```
Subroutine pPeriodic_BC(xx)

Call Send_BCX(NX,NY,NZ,xx) !Send B/C info for x dir
Call Recv_BCX(NX,NY,NZ,xx) !Recv B/C info for x dir

Call Send_BCY(NX,NY,NZ,xx) !Send B/C info for y dir
Call Recv_BCY(NX,NY,NZ,xx) !Recv B/C info for y dir

Call Send_BCZ(NX,NY,NZ,xx) !Send B/C info for z dir
Call Recv_BCZ(NX,NY,NZ,xx) !Recv B/C info for z dir

End Subroutine pPeriodic_BC
```

The routine above performs ghost cell updates 1 and 2 with Send/Recv_BCX. Ghost cell updates 3 and 4 are performed with Send/Recv_BCY. The send operation in Send_BCX uses the MPI function call MPI_ISEND and MPI_IRECV (commonly used MPI functions are shown in Appendix D). There are two basic ways to transfer information with MPI, these are blocking and non-blocking operations. A blocking operation (such as MPI_SEND) will perform the function call requested and also halts all processors from continuing any further until the requested process is completed. A non-blocking operation (such as a MPI_ISEND) will send the required data to all processors, but will not halt any processors from doing useful work while the send operation is in progress. This (will be shown shortly) can affect the overall performance of the code and care should be taken to decide which operation to perform. It was decided that the boundary conditions will use non-blocking send/receives and use MPI_WAITALL to guarantee this data is received in the appropriate order of Figure 4.4. The MPI_WAITALL command (Appendix D) allows the programmer to decide when a manual block should be placed in the code. The structure of the pPeriodic_BC routine uses MPI_ISEND and MPI_IRECV to send and receive the correct ghost cell boundary information. Because these are non-blocking, all of this data is

sent at once to all processors. The routine Recv_BCX is just a MPI_WAITALL that will not let any processor continue until all of the send/receives for the x-direction are finished. Because non-blocking operations are performed, if the WAITALL incurs any performance penalty at all, it will be much less than performing a blocking send and a blocking receive. This same methodology is applied to the y and z directions. This strictly guarantees that the x-direction information is sent and received first, y-direction follows second, and the z direction is last.

The reader is now referred to the Send_BCX subroutine in Appendix C. The variables sent into this routine are the NX,NY,NZ, and the 3D array that is to be updated with periodic boundary conditions. A send buffer is created, (sbuffx_r) that contains the values of the y-z plane at fixed interior x-location (beginning (1) or ending (NX)). To take out any ambiguity, the send buffer (x-direction) left plane and send buffer (x-direction) right plane are clearly shown in Figure 4.5, along with orientation axis.

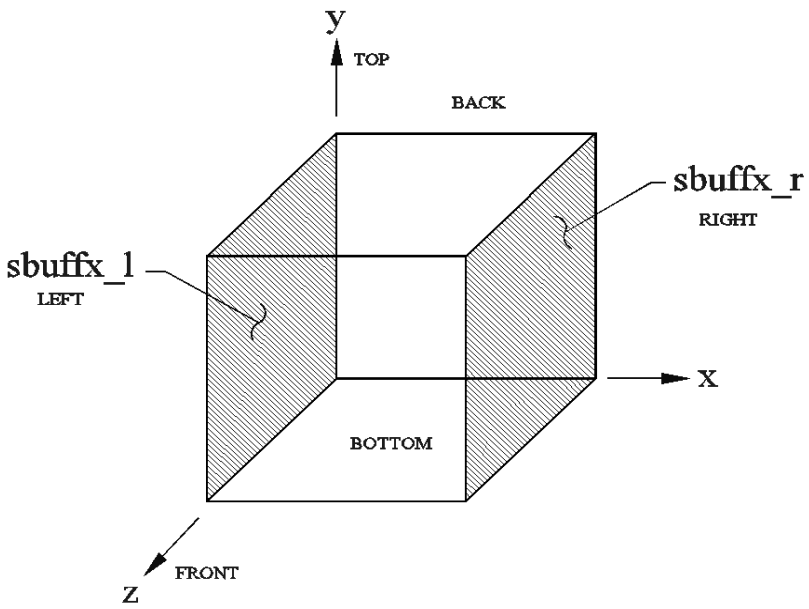

Figure 4.5. Orientation and planar data values used by periodic boundary conditions.

As is shown in Recv_BCX routine these x-direction planar values are now (once the operation is complete) used to update the ghost cell information for the input array that is desired to be periodic. For clarity these values are given by the receive buffer (x-direction) right side (rbuffx_r) and receive buffer (x-direction) left side (rbuffx_l). With the process explained in detail, it should be apparent that the y and z directions follow the exact same procedure outlined above, with only slight changes in orientation and by the naming convention of the send and receive buffers. For example the send buffer (y-direction) top x-z planar data is stored as sbuffy_t, and the sent bottom planar data is stored as sbuffy_b, etc.

4.2 MPI I/O

It may be asked, how is it guaranteed that the CPU order shown in Figure 4.2 is exactly followed? This question is answered by how the initial fields are read into the program with the input routine shown.

```
NXP = DNX/NX ! # processors in x-dir
NYP = DNY/NY ! # processors in y-dir
NZP = DNZ/NZ ! # processors in z-dir

! CPU location indices
kl = myid/(NXP*NYP) + 1
kr = mod(myid,NYP*NXP)
jl = kr/NXP + 1
il = mod(kr,NXP) + 1

! read in the total field and store in a temporary array
! called uJ.
open (unit=503,file=trim(str4),form='formatted')
read(503,*) (((uJ(i,j,k),i=0,DNX+1),j=0,DNY+1),k=0,DNZ+1)

!! now, let every processor get it's own "chunk" of data.
tke(0:NX+1, 0:NY+1, 0:NZ+1) =
uJ( NX*(il-1):(il*NX)+1 , NY*(jl-1):(jl*NY)+1, NZ*(kl-1):(kl*NZ)+1 )
close(unit=503) !close the large field uJ.
```

The input routine is shown for the kinetic energy (tke). All other data fields (u, v, w, eps) are read in exactly the same way. In Fortran the expression mod(n,m) gives the remainder when n is divided by m; it is applied to integers. Examples are mod(8,3) = 2, mod(27,4) = 3, mod(11,2) = 1, mod(20,5) = 0. The number of processors in each direction are determined by dividing the total domain size (DNX, DNY, DNZ) into smaller problem sizes (NX, NY, NZ). Then integer division and the mod(n,m) expression are used to compute three integer variables (il, jl, kl). The problem size plus boundaries is then defined for every processor and the code uses (il, jl, kl) to correctly grab a 'chunk' of data from the total domain. Finally the large (total) array uJ is closed.

Refer to the domain decomposition show in Figure 4.6, where a domain is split in half (x-direction). Assume (for this example) that we wish to run a 64x64x128 simulation on two processors as shown. The total domain is given as (DNX=64, DNY=64, DNZ=128), the problem size for each processor would be (NX=32, NY=64, NZ=128).

MPI will randomly assign the variable 'myid' to each processor it uses. In this case, two processors would be available with 'myid=0' and 'myid=1'. To ensure that the correct processor is located exactly as shown in Figure 4.6, consider the input routine described above. This routine, reads in the total energy array (with boundaries) and stores this in a temporary array (uJ(0:65,0:65,0:129)) on each CPU. The processor with 'myid = 0', computes (il=1, jl=1, kl=1). This stores its 'chunk' of kinetic energy (tke) with information from the total domain uJ(0:33, 0:65, 0:129), exactly as shown in Figure 4.6. Likewise, the processor 'myid = 1', computes (il=2, jl=1, kl=1), which stores it's (tke) from the values uJ(32:65, 0:65, 0:129). Finally, now that the large temporary array (uJ) has been partitioned correctly, it is closed. It is clear from this example, that these are indeed the correct indices for the input arrays. This guarantees that the input data will indeed follow the numbering sequence

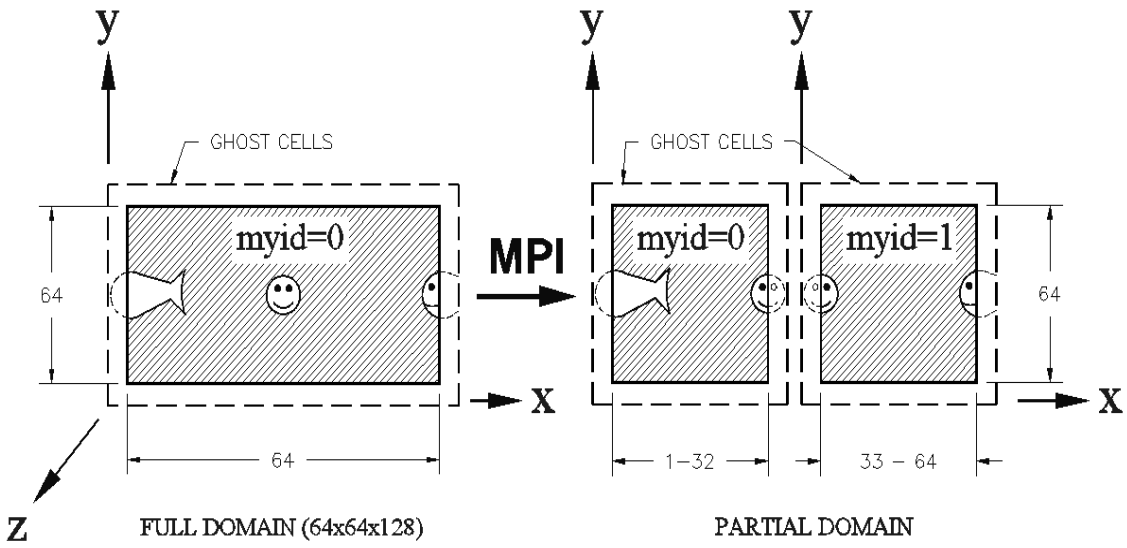

Figure 4.6. Single processor (left side) and MPI (right side) partitioning of the 64x64x128 example problem.

shown in Figure 4.2, with the appropriate interior domain and boundary (ghost cell) information.

4.2.1 Data Output

The output routine (Appendix F) follows exactly the same logic as the input routine, except in reverse order. Although this routine is shown for kinetic energy (tke), the other fields (u, v, w, eps) follow the same procedure. Typically a root processor is chosen, usually 'myid=0', is selected. This root processor will be the processor that collects all data from other CPUs using a (MPI_RECV) and pieces together the total domain by using the values of il, jl ,kl. A simple 'If' statement allows all other processors (not root) to send (MPI_Send) their data along with the corresponding values (myid, il, jl, kl). Once the root processor has received all of the data from each processor and pieced together the total domain, it then writes out

data for post-processing use. The output routine shown does just this, and is clearly explained in Appendix F by the programmer comments.

4.3 Global Calculations

Now that the most difficult part of parallelization has been explained (boundary conditions), there is just one more consideration to take care of before any optimization can begin. This last part deals with global variables used in the calculations of the code. Namely, these are the SUM, MINVAL, and MAXVAL Fortran calls. With the total domain now decomposed and sent to 'N' number of processors there must be a way to figure out the total SUM of data values globally (not just on each processor). Similarly the MINVAL/MAXVAL functions must give values for the total domain and not for each processor. In order to accomplish this, three extra routines were used in conjunction with the MPI_ALLREDUCE function (Appendix D). These three routines are shown in Appendix E. It should be straightforward to see that the ALLREDUCE function simply combines the values from all processors to calculate a desired global value. This global value is then distributed to all processors. For example, if an array is given by (1.0 2.0 3.0 4.0), and each value is sent to a parallel job with four processors, each processor stores the entries 1.0 - 4.0. If a Fortran SUM is performed, processor 1 calculates the total sum of the given array as 1.0, processor 2 calculates the total sum to be 2.0, etc. However with MPI_ALLREDUCE, the actual global sum of $(1.0 + 2.0 + 3.0 + 4.0 = 10.0)$ is calculated and this global value is then sent to every processor.

4.4 Parallel Optimization

With all the pieces mentioned above in place, the code was fully parallelized. Extensive testing and debugging proved that the parallel version of the code gives (almost to machine precision) the same solution as the serial version. The next and

last step was to optimize the code in order to fully take advantage of using multiple processors. Optimization in parallel consists of efficiently sending, receiving, and computing data.

```
Start_MPI

!Non-Optimized
Compute Sqrt(Data B) !computes square root of Data
MPI_SEND(Data A)     !(blocking) send to all processors (takes a while)
!processors sit idle at this point until the send is completed.
Answer = (Data A and Sqrt(Data B))

!Optimized
MPI_ISEND(Data A)     !non-blocking send to all processors (takes a while)
Compute Sqrt(Data B)  !with send in progress, computer continues to work
!once the Sqrt is calculated, the send has most likely been completed
MPI_WAITALL !manual stopping point to make sure Data A was sent
Answer = (Data A and Sqrt(Data B))

End_MPI
```

A simple example is shown, where a non-optimized and an optimized solution are computed with two arrays (Data A, Data B). The non-optimized code computes the square root of (Data B) and then sends (Data A) to all processors. At this point, the answer cannot be computed until this sending operation has been completed. Recall that MPI_SEND is a blocking function call, and will halt all processors until the sending operation is completed. If (Data A) is very large, this send operation can take a long time, and at this point all processors sit idle and wait. When the send is completed the solution (Answer) is calculated. Obviously this will yield a correct answer, but it is not the best use of available resources because now an additional penalty is incurred by the communication time between processors. The optimized code however, will yield the correct answer, and will do so faster! This is because the MPI_ISEND is performed (as shown). Recall that MPI_ISEND is a non-blocking operation. While this information is being sent, there is no reason why the proces-

sors cannot continue to do some useful work. As (Data A) is sent (optimized code), all processors begin computing the square root of (Data B). When the square root has been calculated, most (or ideally all) of (Data A) has been sent. If all of (Data A) was sent during the computation on (Data B), the WAITALL function incurs no communication penalty. Even if all of (Data A) was not sent, the WAITALL still takes less time than the blocking send performed by the non-optimized code. The final answer will be computed much faster. This hides the MPI communication times behind actual computational work in order to develop an ideally optimized code. By letting each processor do as much work as possible during these MPI communication function calls, better parallel performance is obviously obtained. This idea was mentioned above and shown to be effective for the periodic boundary condition routine shown in Section 4.1.2.

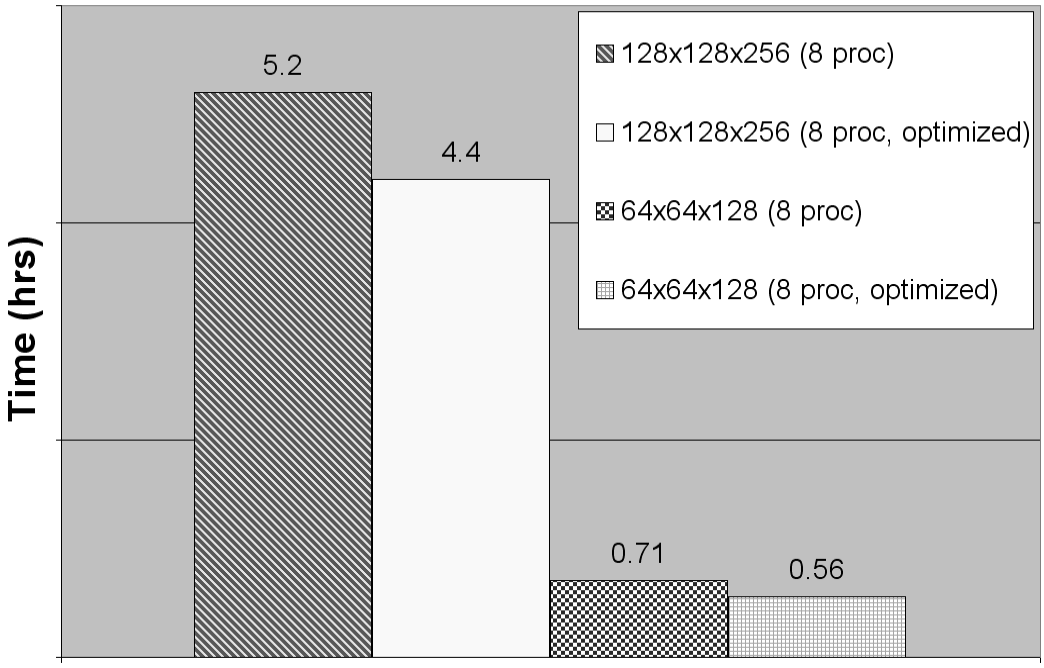

Figure 4.7. Total simulation time (hrs) for 64x64x128 and 128x128x256 before and after optimization on the UMass cluster.

Figure 4.7 shows the total simulation time for a 128x128x256 and a 64x64x128 simulation before and after MPI optimization. This figure shows that once parallelized, the total simulation time for a 128x128x256 simulation was 5.2 hours and 0.71 hours for the 64x64x128 case, using eight processors. After the code was optimized by hiding as much of the communication time as possible, the total simulation time for both cases (again with 8 processors) dropped by roughly 20 percent. This reduction with the optimized code saves valuable supercomputing time.

4.5 MPI Performance

Figure 4.8 shows the parallel performance of the developed code as recorded on the Arctic Region Supercomputing center IBM/P690 nicknamed 'Iceberg' (symbols with dashed lines), and on the UMass cluster 'von Karman' (symbols with dotted lines). The speed-up factor is given by the total simulation time on a single processor divided by the simulation time for multiple processors.

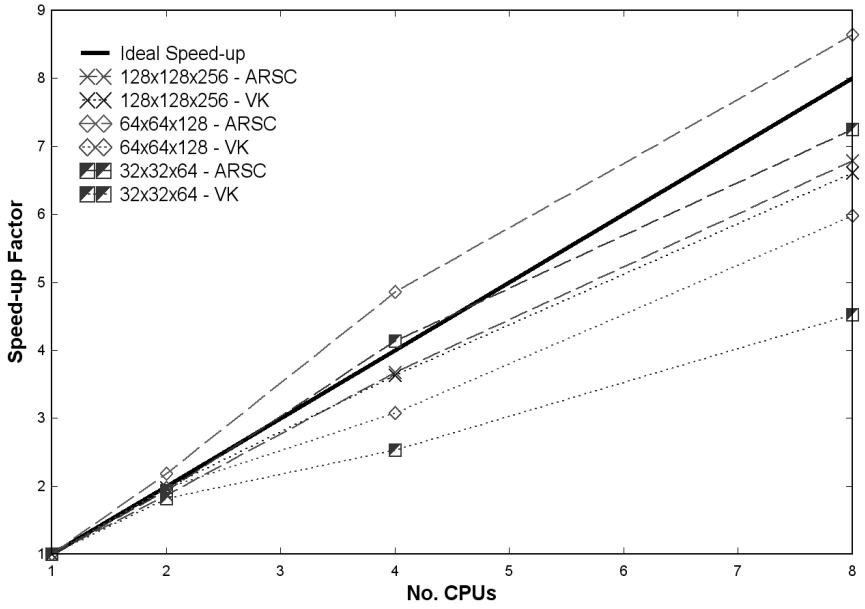

Figure 4.8. Speed-up factor vs. number of processors (UMass and ARSC).

It is not necessarily a measure of how fast a given CPU is, but rather, a measure of the relative communication time between processors for any given architecture. The ideal performance is also shown (solid line) and shows a linear relationship. For example, using twice as many processors, should give (ideally) a speed-up factor of two, etc.

Because von Karman has a slower interconnect than the DoD computer, it is not surprising to see that bottom three curves correspond to speed-up values for this machine. It is shown that using multiple processors (1 per node) increase the speed-up factor and therefore decrease computational time to a solution. It is even a bit surprising that for the 128x128x256 mesh resolution the speed-up factor is almost the same as what was observed on Iceberg. It was noted during the testing of MPI performance that von Karman gives poor results when both processors on a node are used. This is because there is only one bus per node. If both CPUs are utilized, the memory access is at 1/2 the speed over this shared bus. Because of this, all MPI simulations run on von Karman use only one processor per node to achieve the best performance as shown.

It is clear from Figure 4.8 increasing the number of processors on Iceberg immediately results in a faster simulation time. This is shown by (dashed lines - symbols) the close behavior of the simulations on Iceberg to the ideal speed-up. At 32x32x64 Iceberg gives a speed-up of 7.3 compared to 4.5 for von Karman with 8 processors. The 64x64x128 on Iceberg yields 8.5 compared to 6.0 for von Karman again with 8 processors. Lastly, the 128x128x256 mesh has a speed-up that is almost the same for both machines giving a 6.5 out of 8.0 speed-up factor. It should be mentioned that the increased speed-up factor for the 64x64x128 simulation on Iceberg has nothing to do with MPI or optimization. This has to do with the particular architecture the code is running on. In this case, Iceberg appears to handle the 64x64x128 extremely well (fits well into cache) and gives a better than ideal speed-up factor. With the

available information on parallel performance, all of the large simulations 32x32x64 - 256x256x512 were performed with multiple processors at ARSC.

CHAPTER 5

REYNOLDS STRESS TRANSPORT MODEL

The turbulence modeling approach presented thus far is applicable to any RANS transport equation system. Since the well known two-equation model has been presented and shown to work well, a more complex equation system was investigated. Reynolds stress transport (RST) models are the next step in complexity for turbulence modeling. Additional physics are contained in the six equations used to represent the evolution of the Reynolds stresses (R_{ij}) that are not in two-equation models. In particular, backscatter happens naturally in a Reynolds stress model and therefore no explicit function for backscatter is needed.

5.1 RST Model Overview

The general form of the Reynolds stress transport equation is shown below.

$$\frac{DR_{ij}}{Dt} = \underbrace{P_{ij}}_{\text{exact}} + \underbrace{D^v_{ij}}_{\text{exact}} + \underbrace{D_{ij}}_{\text{modeled}} - \underbrace{\varepsilon_{ij}}_{\text{modeled}} + \underbrace{\Pi_{ij}}_{\text{modeled}} + \underbrace{\Omega_{ij}}_{\text{exact}}$$

This equation states that: the material derivative of R_{ij} equals the rate of production (P_{ij}), plus transport by diffusion ($D^v_{ij} + D_{ij}$), minus the rate of dissipation (ϵ_{ij}), plus redistribution due to turbulent pressure-strain interaction (Π_{ij}), plus redistribution due to rotation (Ω_{ij}). The known quantities are labeled 'exact' and the unknown quantities require models to solve this system. Major advantages with this approach are that the production term (usually a dominant term) is now exact, along with any rotational effects. Because transport equations are solved for the evolution of R_{ij},

the eddy viscosity hypothesis is not needed and thus any deficiencies with the eddy viscosity model are avoided. RST models also have a mechanism for the backscatter of energy unlike the classic k/ϵ model, therefore no ad-hoc function such as α, is required. Finally, since most of the computational expense for this finite volume code is spent on solving a Poisson equation for pressure, additional transport equations may well prove to be worth the minimal increase in computational requirements.

5.2 Mathematical Review

As mentioned in Chapter 2, Germano [8] revealed that the exact, but unclosed governing equations for RANS and LES (and URANS, VLES and DNS) are mathematically identical. From the assumptions mentioned in Chapter 2, the equations for turbulence evolution are (from Appendix A) shown once again.

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{u_i}\,\overline{u_j}) = -\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u_i}}{\partial x_j^2} - \frac{\partial R_{ij}}{\partial x_j} \tag{5.1}$$

Where $\overline{u_i}$ and $\overline{p}$ are the computed velocity and pressure and $R_{ij} = \overline{u_i u_j} - \overline{u_i}\,\overline{u_j}$ is the unknown turbulent stress tensor. The exact (but unclosed) evolution equation for this stress tensor is,

$$\begin{aligned} \frac{\partial R_{ij}}{\partial t} + \overline{u_k}\frac{\partial R_{ij}}{\partial x_k} &= \nu \frac{\partial^2 R_{ij}}{\partial x_k^2} - (R_{jk}\frac{\partial \overline{u_i}}{\partial x_k} + R_{ik}\frac{\partial \overline{u_j}}{\partial x_k}) - \frac{\partial}{\partial x_k} T_{ijk} \\ &\quad -(< \frac{\partial p}{\partial x_i}\,, u_j > + < \frac{\partial p}{\partial x_j}\,, u_i >) - 2\nu < \frac{\partial u_i}{\partial x_k}\,, \frac{\partial u_j}{\partial x_k} > \end{aligned} \tag{5.2}$$

where the bracket operation is given by $< a_i\,, b_j > \equiv \overline{a_i b_j} - \overline{a_i}\,\overline{b_j}$ and the turbulent transport is defined by $T_{ijk} \equiv \overline{u_i u_j u_k} - \bar{u}_i R_{jk} - \bar{u}_j R_{ik} - \bar{u}_k R_{ij} - \bar{u}_i \bar{u}_j \bar{u}_k$. The turbulent transport and bracketed terms require a model if the system is to be solved. To accurately model some of the terms in 5.1 and 5.2, a third 'scale' equation is often postulated that captures the turbulent energetic length or timescale. The epsilon

equation [12] will be used once again. The reader is to keep in mind that other forms such as omega (inverse timescale) [26] and lengthscale equations [22, 16] also are possible. In order to implement a Reynolds stress transport model, a modeled form of equation 5.2 is needed and is given in Appendix G.

5.2.1 Second Moment Closure Self-Adapting LES Model

The self-adapting LES model will use the following transport equations,

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i\, u_j) = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} - \frac{\partial R_{ij}}{\partial x_j} \tag{5.3}$$

$$\begin{aligned} \frac{\partial \overline{R}_{mn}}{\partial t} + \frac{\partial (u_j \overline{R}_{mn})}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[(\nu + \nu_T)\frac{\partial \overline{R}_{mn}}{\partial x_j}\right] + \overline{P}_{mn} - \overline{P}\,\overline{R}_{mn} \\ &+ C^S_{p2}(S_{mj}\overline{R}_{jn} + S_{nj}\overline{R}_{jm} + \overline{P}\,\overline{R}_{mn}) \\ &+ C^W_{p2}(W_{mj}\overline{R}_{jn} + W_{nj}\overline{R}_{jm}) \\ &+ C_{p1}(T_i)(\frac{2}{3}\delta_{mn} - \overline{R}_{mn}) \\ &+ C^*_{p2} S_{mn} + 2\nu_T \left(\frac{1}{k}\frac{\partial k}{\partial x_j}\frac{\partial \overline{R}_{mn}}{\partial x_j}\right) \end{aligned} \tag{5.4}$$

$$\frac{\partial k}{\partial t} + \frac{\partial}{\partial x_j}(k u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_k})\frac{\partial k}{\partial x_j}] + P - \epsilon \tag{5.5}$$

$$\frac{\partial \epsilon}{\partial t} + \frac{\partial}{\partial x_j}(\epsilon u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_\epsilon})\frac{\partial \epsilon}{\partial x_j}] + \frac{\epsilon}{k} C_{\epsilon 1} P - \frac{\hat{\epsilon}}{k} C_{\epsilon 2}\epsilon \tag{5.6}$$

where the overbar on the velocity and pressure have been dropped for convenience. Equation 5.4 is the closed form of the normalized Reynolds stress tensor $\overline{R_{ij}} = R_{ij}/k$. The derivation of this transport equation is provided in Appendix G. Equations 5.5 and 5.6 are provided for closure of 5.4 and will be explained shortly.

Equation 5.4 shows the time derivative and convective term of the Reynolds stress tensor $\overline{R_{mn}}$ on the left hand side. The first term on the right hand side shows viscous and turbulent diffusion together. The viscous diffusion is given exactly and the

turbulent diffusion has been modeled with an eddy viscosity (ν_T) term from the well known gradient-diffusion model. The eddy viscosity is given as $\nu_T = \frac{C_\mu F k}{T_i}(\frac{k}{k+k_r})$ with $C_\mu = 0.28$.

The next two terms ($\overline{P_{mn}} - \overline{P}\,\overline{R_{mn}}$) are the exact production terms given from Appendix G. As mentioned previously, this term is typically a very large and dominant term that no longer requires a model. The RST production term is given by $\overline{P}_{ij} = -\Big(\overline{R}_{ik}\,u_{j,k} + \overline{R}_{jk}\,u_{i,k}\Big)$. In addition $\overline{P} = \frac{1}{2}\overline{P}_{ii}$ and $P = \overline{P}k$.

The remaining terms on the right hand side of equation 5.4 correspond to pressure-strain interactions. Various models exist in literature and can be found to almost any order desired [13, 19] however, there is little benefit to going beyond first order. Typically the pressure-strain models can be thought of in three parts, the fast pressure-strain interactions, the slow pressure-strain interactions, and any wall blockage terms. The fast pressure-strain terms are those in which a model is based upon the gradients in the flow field where $u_{i,j} = S_{ij} + W_{ij}$. The strain tensor is given as $S_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$ and the vorticity tensor is given by $W_{ij} = \frac{1}{2}(u_{i,j} - u_{j,i})$. Thus any terms involving velocity gradients (or components thereof) such as the C^S_{p2}, C^W_{p2}, and C^*_{p2} terms are designated fast pressure-strain (or rapid) terms as they respond instantaneously to gradients in the velocity. The fast pressure-strain constants are given as $C^S_{p2} = C^W_{p2} = 0.6$. The value of $C^*_{p2} = -0.2F^2(\frac{k}{k+k_r})^2$, where F is a realizability constraint given as $F = \det\Big(\frac{3}{2}\overline{R}_{ij}\Big)$.

The slow pressure-strain term comes from the assumption that (in the absence of any production terms) pressure fluctuations should force any turbulence to isotropy. Hence the slow pressure strain term is a linear return-to-isotropy model as was first investigated by Rotta [20]. This well known model is given by $C_{p1}\,T_i(\frac{2}{3}\delta_{mn} - \overline{R_{mn}})$ as shown in the right hand side of equation 5.4. The inverse of the turbulent time scale is $T_i = \epsilon/k$. The slow pressure-strain term is defined as $C_{p1} = 0.6[2.5 - 1.5(\frac{k}{k+k_r})]$. This varies C_{p1} from an average value of 0.6 at RANS to 1.5 in the DNS regime.

The last term in equation 5.4 looks like a dissipation term, but is in fact, a pressure-strain interaction term. The derivation of the normalized Reynolds stress tensor ($\overline{R_{ij}} = R_{ij}/k$) from Appendix G cancels dissipation exactly but leaves behind this term. This last term can be considered a wall blocking term as it only contributes if gradients in turbulent kinetic energy (k) or $\overline{R_{ij}}$ become large (as they would near walls). Because the test cases presented are on a periodic domain with no walls this term has minimal contribution to the overall transport equation in this work.

5.2.2 Closure Equations

Equations 5.5 and 5.6 are similar to the previously shown k/ϵ model. However, these equations are now used for closure of the RST equation with a few key differences. The production term $P = \overline{P}k$ and turbulent viscosity are now given exactly by the RST equation. The near wall term $\hat{\epsilon} = \epsilon - 2(\nu + \frac{\nu_T}{\sigma_{\epsilon 2}})(\frac{\partial k^{1/2}}{\partial x_j})^2$ from Perot and Natu [17] is now included in the dissipation equation. The constants are fairly standard model constants, $C_{\epsilon 1} = 1.55$, $\sigma_\epsilon = 1.2$, $\sigma_{\epsilon 2} = 0.11$, $\sigma_k = 1.0$, $C_\mu = 0.28$. As with the k/ϵ model, the parameter $C_{\epsilon 2} = \frac{11}{6}f + \frac{25}{Re_T}f^2$ is sensitive to the local turbulent Reynolds number $Re_T = \frac{k^2}{\nu\epsilon}$ of the modeled turbulence via the function $f = \frac{Re_T}{30}(\sqrt{1 + \frac{60}{Re_T}} - 1)$ as per the analysis of Perot and de Bruyn Kops [16]. Once again, this will vary $C_{\epsilon 2}$ from its theoretical limits of 11/6 at high Reynolds numbers to 3/2 at low Reynolds numbers. Any Reynolds number dependent $C_{\epsilon 2}$ would probably be sufficient. For incompressible flow, the pressure in 5.3 is determined from the constraint $u_{j,j} = 0$.

CHAPTER 6

RST SIMULATION RESULTS

A number of simulations were performed with the RST model using the same isotropic decaying turbulence (Re=640) initial conditions used for the two-equation universal k/ϵ model. The previous finite volume code was utilized with the addition of the six transport equations needed for the evolution of the Reynolds stresses. A staggered Cartesian grid (previously shown in Figure 3.1) was again implemented.

6.1 High Re Isotropic Decay

The high Re isotropic decaying turbulence test case was again chosen for validation of the RST code. Simulations were performed on a 1x1x2 to 128x128x256 mesh size and total kinetic energy predictions are shown in Figure 6.1.

The ordinate shows total kinetic energy ($k + k_r$) normalized by the initial total kinetic energy at time t=0.0, the abscissa τ, is given by simulation time (seconds) non-dimensionalized by the inverse time scale ($\frac{\epsilon_t}{k_t}$) at time t=0.0 (where $_t$ indicates total quantities). Since the same initial conditions were used for the RST code as the k/ϵ model, the initial turbulent Reynolds number is once again, $Re_T = \frac{k_t^2}{\nu\epsilon_t} = 640$. The DNS results given by de Bruyn Kops [6] on a 768x768x1536 mesh using a Fourier spectral method is given by the large circles. The RST code gives very similar results to what was obtained for the k/ϵ model. The simulations in Figure 6.1 show that at any mesh resolution, the model predicts the decay of the turbulence well. The very lowest mesh resolution is clearly a RANS (or URANS) simulation and the largest

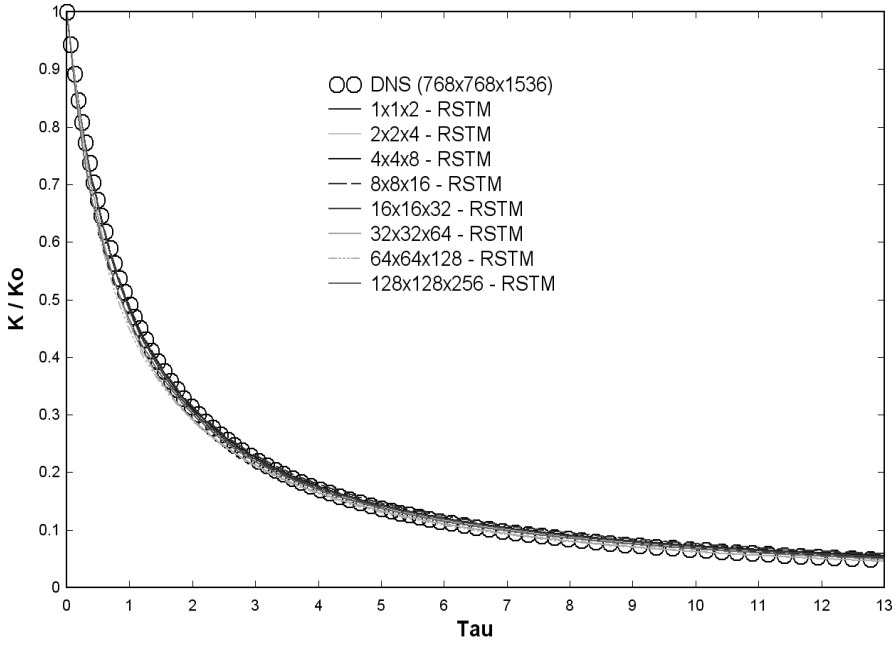

Figure 6.1. RST model total kinetic energy predictions for (initial Re=640) isotropic decaying turbulence.

mesh, 128x128x256, is a LES simulation (with the mesh size roughly in the inertial range). The intermediate resolutions might be considered URANS, VLES, or LES.

6.1.1 Turbulent Kinetic Energy Ratio

Of more importance than total energy prediction is the investigation of how the turbulence is behaving. This is shown in Figure 6.2.

The 1x1x2 solution is the top curve, with all its energy contained in the model (giving a ratio of 1.0) and the largest RSTM simulation (128x128x256) is the bottom line, with the smallest ratio of modeled kinetic energy ($\approx$ 20 percent). As was mentioned for the two-equation model, the desired behavior is for these curves to stay relatively constant but decrease slightly with time. As is observed, the curves indeed follow the desired behavior and are almost identical to the previous k/ϵ results. It must be noted that the desired behavior is now obtained without an explicit energy

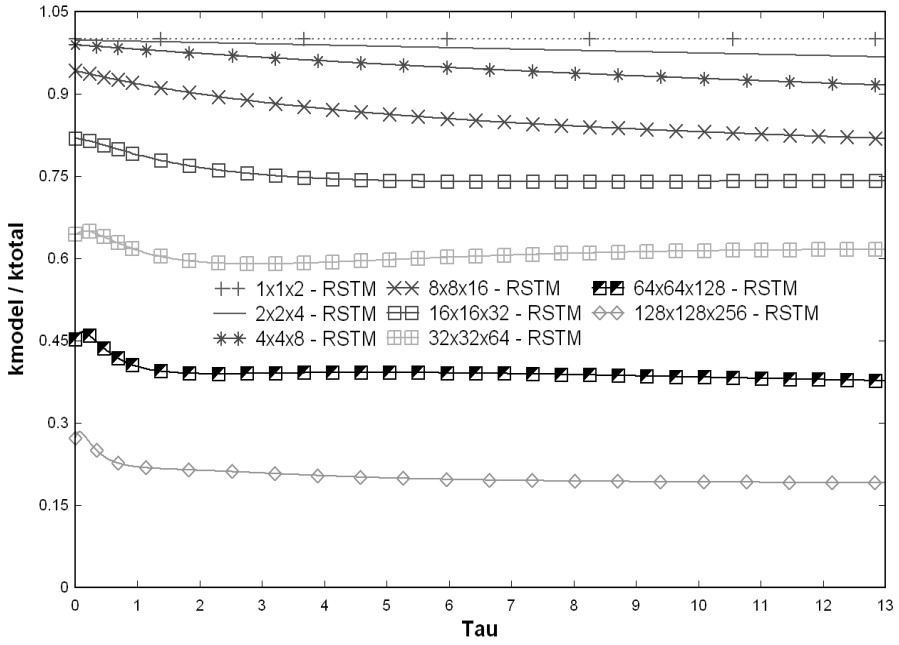

Figure 6.2. RST model ratio of modeled kinetic energy to total kinetic energy, Re=640.

transfer function such as α. The simplistic k/ϵ model needed such a term to displace energy appropriately (backscatter). It is clear that the energy is already being transferred appropriately by the pressure-strain Reynolds stress terms. Therefore adding an extra ad-hoc function is not needed.

6.1.2 Perturbation of Initial Conditions

The final validation of the RST code is to see if it has a self-adaptive character. A truly adaptive model is able to obtain the correct behavior from incorrect initial conditions. This was previously tested in Chapter 3 for the k/ϵ model. Once again, initial conditions were either smoothed or sharpened using a filtering operation. The filter used to alter the initial conditions was a nearest neighbor averaging procedure, $u_{ijk}^{filtered} = \beta u_{ijk} + (1-\beta)(u_{i+1jk} + u_{i-1jk} + u_{ij+1k} + u_{ij-1k} + u_{ijk+1} + u_{ijk-1})\frac{1}{6}$.

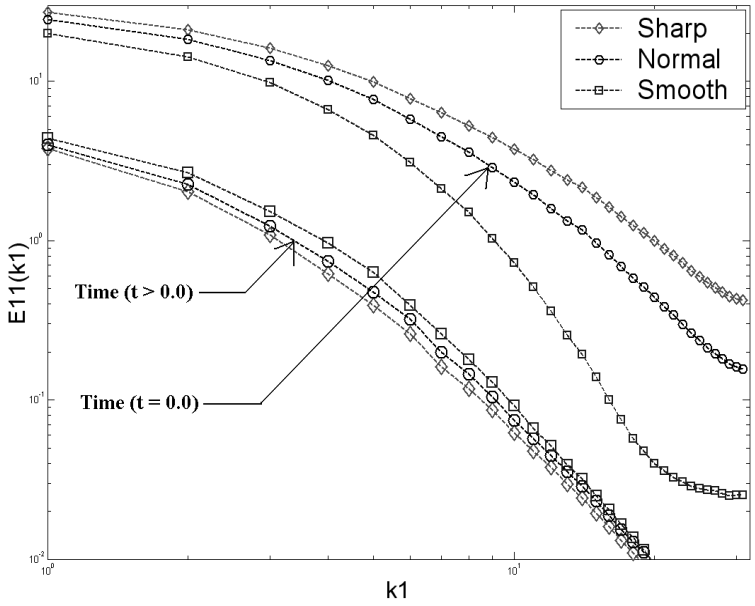

Figure 6.3. 1D RSTM energy spectra for 64x64x128 case, Re=640 (Sharp, Normal, Smooth).

For smoothing, $\beta = 0$ was used. This replaces the value at a mesh point by the average of its nearest neighbors. This type of filter removes energy primarily from the highly oscillatory modes with wavelengths close to the mesh size. In spectral terms it damps the spectra in the region just above the cutoff wave number. Sharpening the velocity field is performed by using $\beta = 1.5$. This adds energy to the existing high frequency modes. The effect is shown in Figure 6.3, which shows the original initial spectra (t=0.0) for the 64x64x128 simulation (normal, smooth, sharpened) as the top three curves. The bottom three curves are the normal, smooth, and sharpened spectra at some later time.

Figure 6.4 shows the affect of smoothing and sharpening the initial conditions on the kinetic energy ratio. When smoothing is used, energy is removed from the resolved modes. In order to keep the total kinetic energy the same, the model now must start with more energy. The ratio therefore starts higher than before. As time

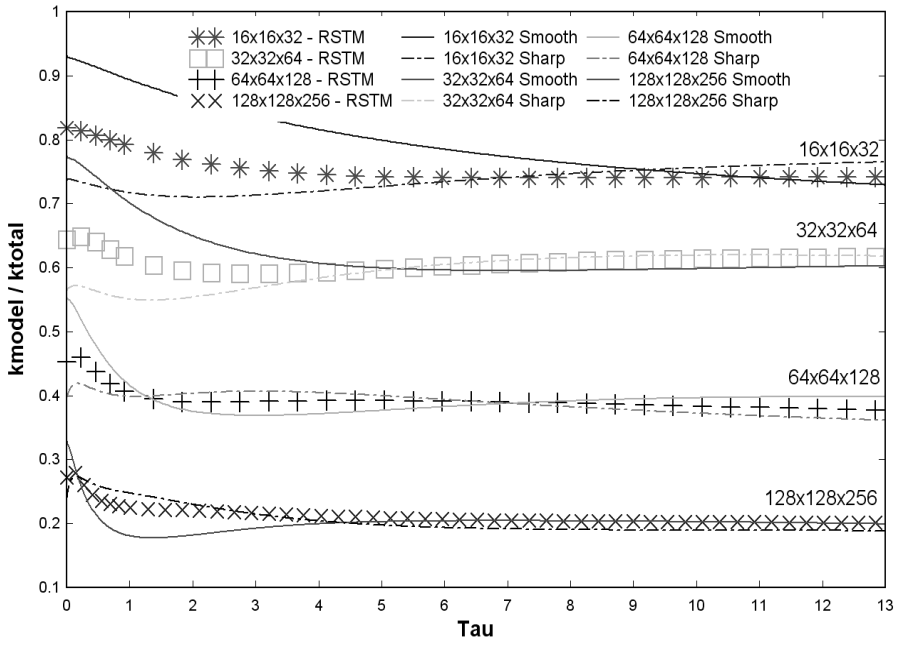

Figure 6.4. RSTM ratio of modeled turbulent kinetic energy to total kinetic energy, Re=640, for perturbed initial conditions.

proceeds the model achieves the same ratio irrespective of the initial conditions. At early times, the smoothed solution has less error and therefore backscatters somewhat more than the unperturbed initial condition. This removes energy from the model and makes the ratio decrease faster, so that it approaches its original state.

A similar (but opposite) process happens when the spectra is sharpened. In this case, the model senses that the mesh can not support the input resolved fluctuations, and quickly sends that energy to the model. Note that the rate at which the model adjusts to perturbed initial conditions depends on the mesh resolution. The higher mesh resolutions adjust more rapidly. It is hypothesized that the time it takes to transfer the energy scales on the timescale of the turbulence at the cutoff (transfer) lengthscale (k/ϵ).

Although the ratio of ($\frac{k}{k+k_r}$) indeed shows the correct behavior, a more definitive test would be to see if the energy spectra for all three cases (sharp, normal, smooth)

which are clearly not similar become similar at some later time. These spectra were previously shown in Figure 6.3. As mentioned, the top three curves show the initial energy spectra at (t=0.0) and the bottom three curves correspond to the spectra at some later time, where the code is trying to correct itself. These spectra (almost right on top of each other) along with the turbulent energy ratio clearly show that the model is self-adapting and does indeed correct itself for perturbed initial conditions.

6.1.3 RST Scaling

The use of the mesh size to infer the lengthscale is why classic LES fails outside of the inertial range (close to the DNS or RANS regimes), and one reason why the self-adaptive model (which predicts the lengthscale and does not infer it) can operate at any mesh size. However, the current model should still obtain the classical LES behavior in its range of applicability. The lengthscale predicted by the model ($L_m = k^{3/2}/\epsilon$) should be proportional to the mesh size, when the model is operating in the LES regime. This was previously shown to be true in Chapter 3 for the k/ϵ model. Figure 6.5 looks at predicted lengthscale (in log scale) at a fixed time t=0.5 for the various meshes from the k/ϵ and RSTM simulations.

At small values of Δx (large number of mesh points) it is obvious that the lengthscale is indeed a function of grid spacing as would be expected with classic LES. A reference line has been added (dotted line) with a slope of 1.0 for comparison. Previously mentioned was that the k/ϵ model did not exactly match this reference line. The RST model however, does match this reference line due to the fact that a simplistic model (eddy viscosity model) is no longer used. The figure clearly shows how the turbulent lengthscale is not related to the mesh spacing outside the LES regime.

Mesh resolutions smaller than 64x64x128 do not exhibit the linear behavior because now the relevant turbulent lengthscale is no longer related to the mesh size for both systems. There is a transition region that one might call VLES(32x32x64),

but not full LES. The 8x8x16 mesh for both systems might be considered the start of URANS, where the largest scales can just be resolved, and unsteady 3D solutions maintained. These smallest mesh sizes are where the solutions can not sustain a cascade because all the largest turbulence has scales that are smaller than the mesh size. Again it is pointed out that both systems show that near the 32x32x64 mesh

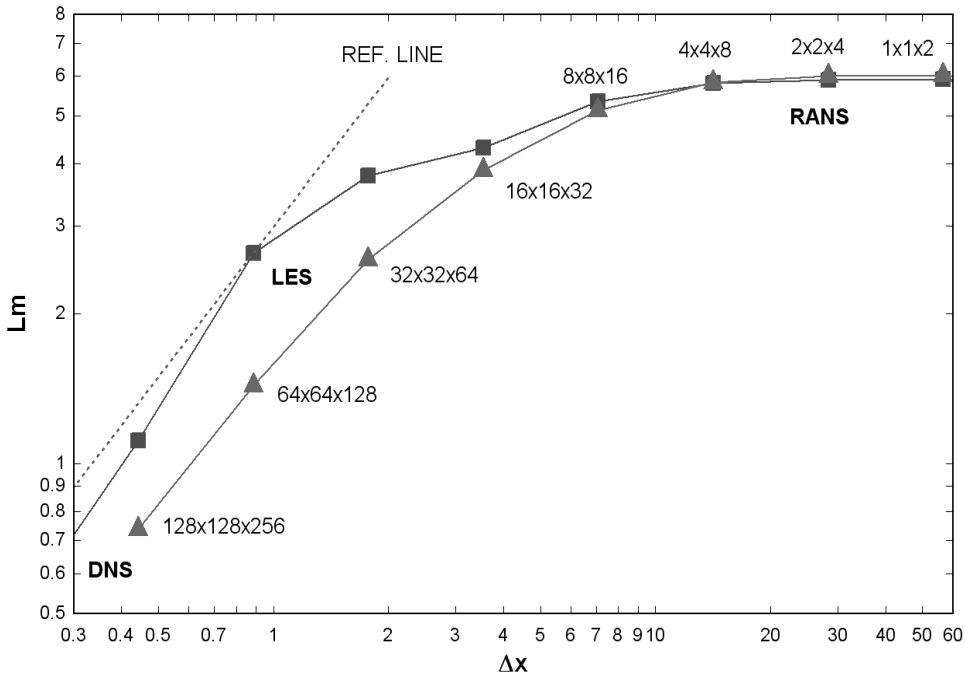

Figure 6.5. RSTM (diamonds) and k/ϵ (squares) lengthscale behavior at time = 0.5s.

resolution is a transition from VLES to an LES simulation. This supports the finding by Carati et. al. [1] that the smallest mesh size possible for a classic (dynamic model) LES simulation is 48^3.

6.2 Realizability Constraints

The Reynolds stresses are components of a second order tensor which is symmetric. The diagonal stresses correspond to normal stresses while the off diagonal components

represent shear stresses. This tensor is symmetric positive definite and must always be so. Due to the nature of Reynolds stress transport modeling, enforcing the solution to a set of partial differential equations to be symmetric positive definite can cause many problems. This issue is called realizability. The methods used to enforce realizability of the stress tensor are described below.

6.2.1 Turbulence Initial Conditions

The initial conditions used for modeled turbulent kinetic energy (k), dissipation (ϵ), and velocity (u, v, w) were calculated as previously mentioned from the DNS fields. The same initial fields used for the k/ϵ model are again used for the RST model. The

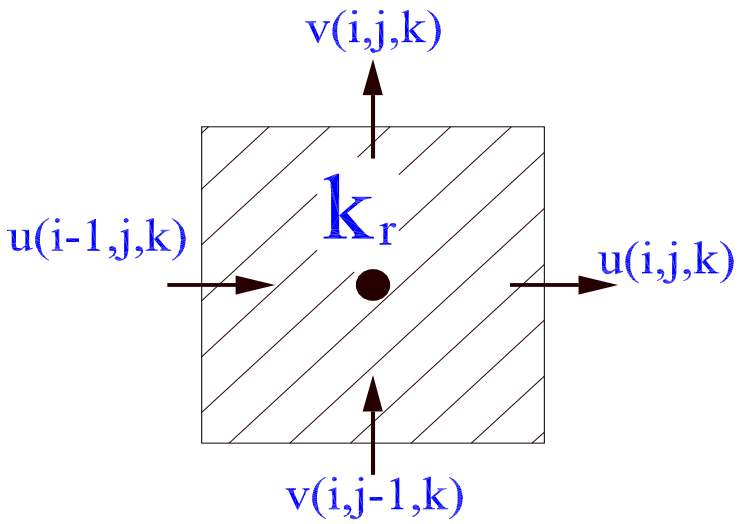

Figure 6.6. Calculation of resolved turbulent kinetic energy on a Cartesian grid.

calculation of the resolved turbulent kinetic energy is given by $\frac{1}{2}(u^2 + v^2 + w^2)$ and is shown in Figure 6.6. In order to calculate this cell centered quantity, each velocity component was first averaged to the cell center and then squared. For example, to calculate resolved turbulent kinetic energy (k_r), average $\frac{1}{2}(u_{i,j} + u_{i-1,j})$ to get u_{avg} as cell centered (repeat for v, w), then take $1/2(u_{avg}^2 + v_{avg}^2 + w_{avg}^2) = k_r$. This

averaging procedure for the resolved turbulent kinetic energy was used for both the k/ϵ and RSTM system. It was found that averaging the velocity to the cell center first and then squaring the result was optimal over squaring the velocity first and then averaging to cell centers.

6.2.2 Reynolds Stress Initial Conditions

The initial conditions for the Reynolds stress tensor are a vital role for enforcing realizability. Recall that this thesis uses a normalized Reynolds stress tensor where $\overline{R_{ij}} = R_{ij}/k$. The normalized Reynolds stress tensor is shown as

$$\overline{R_{ij}} = \begin{bmatrix} \overline{R_{11}} & \overline{R_{12}} & \overline{R_{13}} \\ \overline{R_{21}} & \overline{R_{22}} & \overline{R_{23}} \\ \overline{R_{31}} & \overline{R_{32}} & \overline{R_{33}} \end{bmatrix}$$

because of symmetry ($\overline{R_{12}} = \overline{R_{21}}$) etc., only six of the nine terms are needed to construct this tensor. The main problem is to insure that the eigenvalues of this tensor are positive or zero (never negative). There are two parts to guarantee this. First the initial condition must be realizable and secondly the solutions to the transport equation must guarantee that this tensor stays realizable.

Unfortunately, starting the initial condition for the Reynolds stress tensor using the eddy viscosity hypothesis did not guarantee realizability. Recall that the eddy viscosity hypothesis is $\overline{R_{ij}} = \frac{2}{3}\delta_{ij} - \nu_T S_{ij}/k$. Although this approach works well for the k/ϵ model, it did not guarantee realizability of the Reynolds stress tensor initially. Modifications to this approach to force the tensor to be realizable where to no avail. Another approach was to start the Reynolds stress tensor from the isotropic initial condition. This is accomplished by setting the diagonal terms to 2/3 and off diagonal terms to zero. Although this approach starts off realizable, it is a very poor initial condition to start with.

The best approach is to solve the steady-state normalized Reynolds stress equations ($\frac{D\overline{R_{ij}}}{Dt} = 0$) to get an accurate initial condition. This assumes quasi-equilibrium and guarantees that the normalized stress tensor is initially realizable. This equation was solved for on all mesh sizes from 1x1x2 to 128x128x256.

6.2.3 Reynolds Stress Averaging

With the initial conditions specified for the Reynolds stress code, it was found that using different averaging procedures could also enforce or violate realizability. The storage locations for the six Reynolds stresses are shown in Figure 6.7. The staggered arrangement of the Reynolds stresses was chosen because it enforced realizability more so than a collocated arrangement. Therefore all source / sink terms related to the Reynolds stress transport equation were averaged to the locations shown in Figure 6.7.

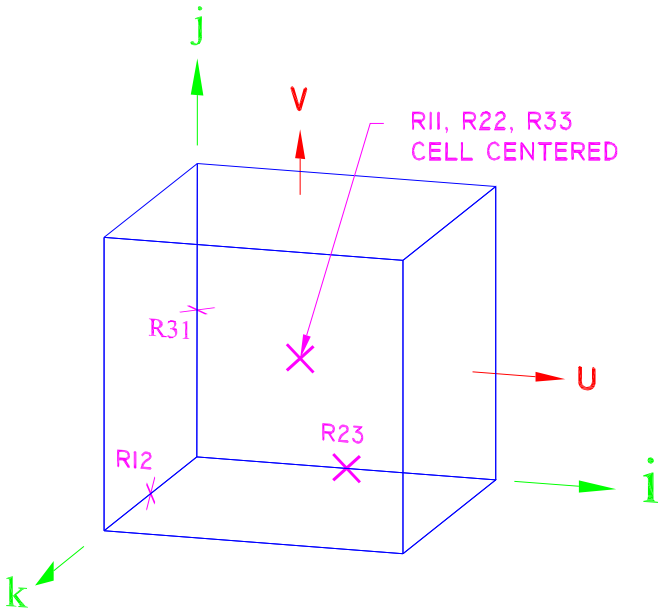

Figure 6.7. Location of the Reynolds stresses.

Having an arrangement as shown was beneficial in the calculation of the gradients computed for the evolution of velocity. This equation is shown once again below.

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i\, u_j) = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} - \frac{\partial R_{ij}}{\partial x_j} \tag{6.1}$$

Cell centered diagonal components ($\overline{R_{11}}, \overline{R_{22}}, \overline{R_{33}}$) will lie exactly on velocity control volume faces where they are needed. Figure 6.8 illustrates this point. It is shown (u velocity) that any cell centered quantity such as pressure (p), turbulent kinetic energy (k), and diagonal stresses ($\overline{R_{11}}, \overline{R_{22}}, \overline{R_{33}}$) are indeed located at control volume faces (where they should be). Therefore no averaging or interpolation is needed. Any gradients associated with equation 6.1 are also computed at control volume faces. For example, fluxes through the left face of the control volume around u(i,j,k) will have a gradient ($\frac{\partial u}{\partial x}$) which is calculated (Figure 6.8) automatically at the control volume face ($\frac{u_i - u_{i-1}}{\Delta x}$). Similarly any off diagonal terms will fall on control volume faces (where they should be). The only difference for the off diagonal terms are that any cell centered quantities such as (p) and (k) from equation 6.1 will no longer be located on a control volume face. In order to calculate $R_{12} = \overline{R_{12}}k$, the model kinetic energy (k) must be averaged to the $\overline{R_{12}}$ location. This is easy to correct by simply averaging the 4 nearest cell centered quantities from their cell centered locations to a control volume face.

It should be mentioned that care must be taken when calculating any source or sink term with the staggered $\overline{R_{ij}}$ arrangement. Particularly the RSTM production term presents some difficulties in the averaging procedure. The exact production term is given as $\overline{P}_{ij} = -\left(\overline{R}_{ik}\, u_{j,k} + \overline{R}_{jk}\, u_{i,k}\right)$. Simply expanding this term out for the $\overline{P}_{11}$ case will yield terms much like $\overline{R}_{11}\frac{\partial u}{\partial x}$. Figure 6.9 shows velocity (u, v) and cell centered (CC) locations. The $\overline{R_{12}}$ stress is also shown in the lower right corner of the cell. It is obvious that the $\overline{R}_{11}\frac{\partial u}{\partial x}$ term is easy to compute. The $\overline{R_{11}}$ is already a cell centered quantity and the derivative $\frac{\partial u}{\partial x} = \frac{u_{i,j} - u_{i-1,j}}{\Delta x}$ is also cell centered. The

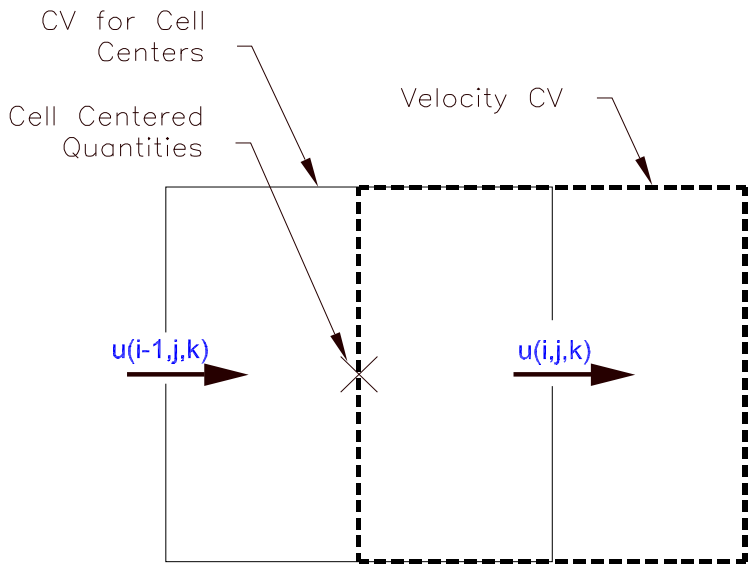

Figure 6.8. Velocity (u) control volume averaging.

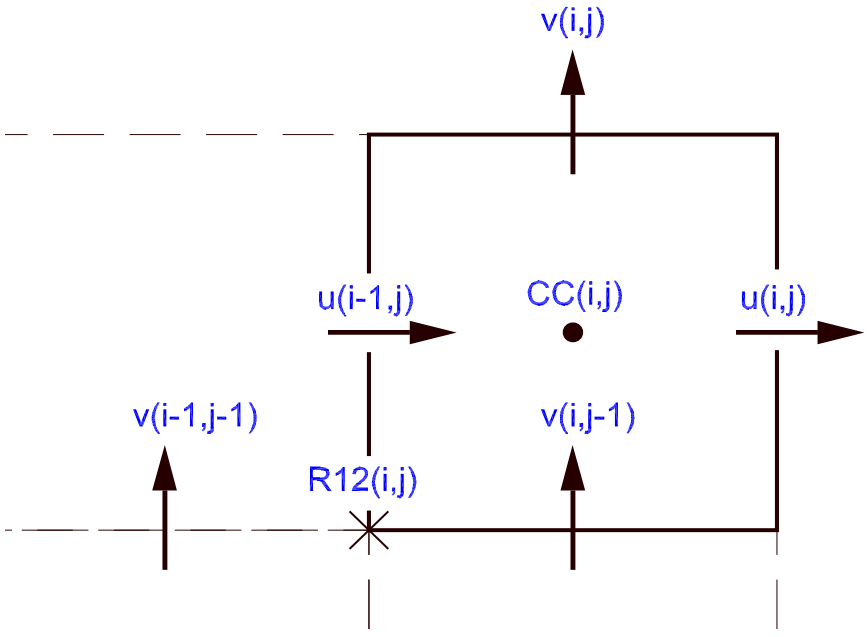

Figure 6.9. Control volume averaging for RST production terms.

problem arises when any off diagonal terms are calculated such as $\overline{P}_{12}$. For example one component of the $\overline{P}_{12}$ term would be $\overline{R}_{11}\frac{\partial v}{\partial x}$. Since the (v) velocity derivative is calculated at the $\overline{R}_{12}$ location, the cell centered $\overline{R}_{11}$ must be averaged there also. As mentioned above, this is accomplished by taking the average of the 4 neighboring cell centered $\overline{R}_{11}$ values around $\overline{R}_{12}$.

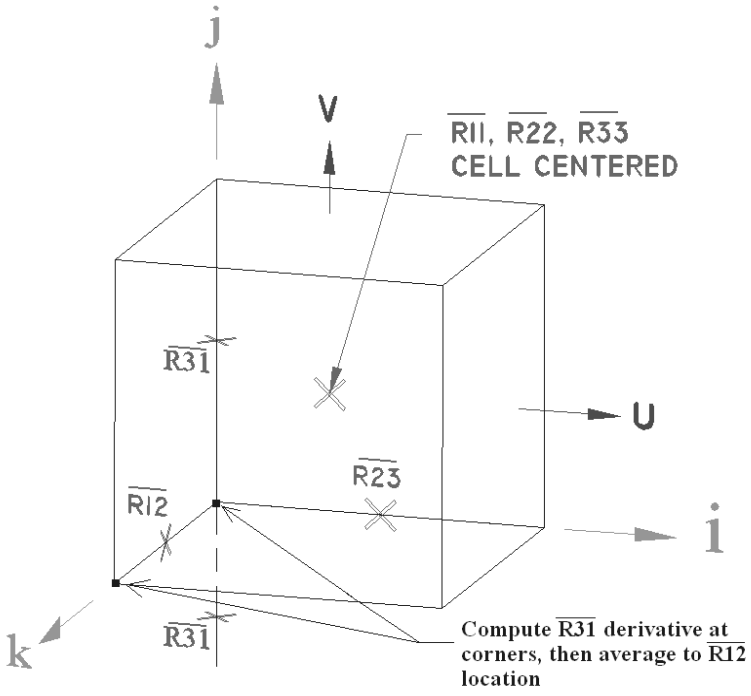

Figure 6.10. Averaging procedure for RSTM code.

The most difficult part of this averaging procedure is when an off diagonal stress derivative is needed at another off diagonal stress location. For example, if the derivative $\frac{\partial \overline{R_{31}}}{\partial y}$ is required at the $\overline{R}_{12}$ location as shown in Figure 6.10. A central difference is used to compute this derivative to the corners of the cell. Once the derivative has been computed on each side of $\overline{R}_{12}$, it is then averaged to the correct location. Needless to say, great care must be taken to average these quantities correctly.

CHAPTER 7

CONCLUSIONS

The goal of this thesis is to demonstrate a new turbulence modeling approach where RANS systems are used for subgrid-scale modeling in LES simulations at any mesh resolution. The implication being that subgrid-scale turbulence is not fundamentally any different than regular turbulence. It can be modeled using the same RANS ideas. The following is a summary of the important ideas presented in this thesis.

- Existing RANS turbulence models can indeed be applied to LES simulations to model subgrid-scale turbulence.
- Simplistic RANS models like a two-equation (k/ϵ) model will need a backscatter term to appropriately displace energy from the resolved to unresolved fields (vice-versa).
- More complex RANS models such as the RST model already have the ability to backscatter energy appropriately and do not require ad-hoc modifications.
- Both systems exhibited a self-adaptive nature where the code had the ability to correct itself from incorrect initial conditions without any user input.
- Both systems can operate from a RANS regime to LES to (with enough mesh) a DNS solution with the user specifying only the mesh size.
- As future RANS turbulence models are developed, they can easily extend to this framework in hopes of a more accurate and cost effective solution.

The results shown indicate that this new modeling approach has the ability to give fairly accurate predictions at any mesh resolution. The self-adaptive nature observed in this method is certainly advantageous for any industrial applications. The code is given the desired mesh resolution and will compute the best answer based on this criteria alone. The user is not required to have advanced knowledge of the flow field (as in LES) nor is the user restricted to modeling the entire flow (as in RANS). Instead the approach presented finds the best solution possible based solely off the mesh size the user can afford.

APPENDIX A

DERIVATION OF THE EVOLUTION EQUATIONS

The incompressible Navier-Stokes equations are:

$$u_{k,k} = 0 \tag{A.1}$$

$$u_{i,t} + (u_i u_k)_{,k} = -p_{,i} + \sigma_{ik,k} \tag{A.2}$$

Following the ideas of Germano [8], the moment of A.2 is taken with u_j, another moment of this same equation with indices interchanged yields A.3 and A.4.

$$u_j u_{i,t} + u_j (u_i u_k)_{,k} = -p_{,i} u_j + u_j \sigma_{ik,k} \tag{A.3}$$

$$u_i u_{j,t} + u_i (u_j u_k)_{,k} = -p_{,j} u_i + u_i \sigma_{jk,k} \tag{A.4}$$

Adding together A.3 and A.4 and using a little calculus, it can be shown that the resulting equation can be written as:

$$(u_i u_j)_{,t} + (u_i u_j u_k)_{,k} = -\left[p u_i \delta_{jk} + p u_j \delta_{ik} - \nu \left(u_i u_j\right)_{,k}\right]_{,k} + 2 p s_{ij} - 2\nu u_{i,k} u_{j,k} \tag{A.5}$$

if a Newtonian fluid $\sigma_{ij} = \nu u_{i,j}$ is assumed. Recall that the turbulent stress tensor was previously defined as $R_{ij} = \overline{u_i u_j} - \overline{u_i}\,\overline{u_j}$, the double bracket was $< a_i, b_j > \equiv \overline{a_i b_j} - \bar{a}_i \bar{b}_j$, and the turbulent transport term was defined by $T_{ijk} \equiv \overline{u_i u_j u_k} - \bar{u}_i R_{jk} - \bar{u}_j R_{ik} - \bar{u}_k R_{ij} - \bar{u}_i \bar{u}_j \bar{u}_k$. Simply averaging equations A.1, A.2, and A.5 and

substituting for the stress tensor, double bracket, and turbulent transport will recover the equations below:

$$\overline{u}_{k,k} = 0 \tag{A.6}$$

$$\overline{u}_{i,t} + (\overline{u_i u_j})_{,j} = -\overline{p}_{,i} + \nu \overline{u}_{i,jj} - R_{ij,j} \tag{A.7}$$

$$\begin{aligned} R_{ij,t} + \bar{u}_k R_{ij,k} = \nu R_{ij,kk} - (R_{jk}\bar{u}_{i,k} + R_{ik}\bar{u}_{j,k}) \\ - T_{ijk,k} - (< p_{,i}, u_j > + < p_{,j}, u_i >) - 2\nu < u_{i,k}, u_{j,k} > \end{aligned} \tag{A.8}$$

Hence it is shown that the averaging invariance procedure of Germano does indeed provide evolution equations that are exactly the same as the Reynolds stress transport (RST) equations.

APPENDIX B

CPU NEIGHBORS

```
**********************************************************************
Subroutine CPU_neigh()
Implicit None
Integer :: iz,iy,ix, tmp

!!location in CPU grid
iz = myid/(NXP*NYP) + 1
tmp = myid-(iz-1)*NXP*NYP
iy = tmp/NXP + 1
ix = tmp -(iy-1)*NXP + 1

!Compute neighboring CPU locations below

CPU_info(1) = myid+1 !x-dir right
if ( ix == NXP) CPU_info(1) = CPU_info(1) - NXP

CPU_info(2) = myid-1 !x-dir left
if ( ix == 1  ) CPU_info(2) = CPU_info(2) + NXP

CPU_info(3) = myid+NXP !y-dir top
if ( iy == NYP) CPU_info(3) = CPU_info(3) - NXP*NYP

CPU_info(4) = myid-NXP !y-dir bottom
if ( iy == 1  ) CPU_info(4) = CPU_info(4) + NXP*NYP

CPU_info(5) = myid+NXP*NYP !z-dir back
if ( iz == NZP) CPU_info(5) = CPU_info(5) - NXP*NYP*NZP

CPU_info(6) = myid-NXP*NYP !z-dir front
if ( iz == 1  ) CPU_info(6) = CPU_info(6) + NXP*NYP*NZP

End Subroutine CPU_neigh
**********************************************************************
```

Integer 'myid', is a variable MPI assigns to each processor in use. If eight processors are used in parallel 'myid' would range from 0 - 7. The variables NXP, NYP,

and NZP correspond to the number of processors in each direction (x,y,z). The calculated integer, CPUinfo1, is the CPU located to the right of the processor in question. CPUinfo2 computes the CPU to the left of the processor in question. CPUinfo3 and CPUinfo4 correspond to the top and bottom CPUs respectively. Lastly, CPUinfo5 and CPUinfo6 are the back and front CPUs located relative to the CPU in question. Further clarification by (top, bottom, etc.) was clearly shown in Figure 4.2.

APPENDIX C

PERIODIC BOUNDARY CONDITION DATA TRANSFER

```
**********************************************************************
! Nonblocking MPI send x

Subroutine Send_BCX(NX,NY,NZ,var)
Implicit None
Integer :: NX,NY,NZ
Real, Intent(IN) :: var(0:NX+1,0:NY+1,0:NZ+1)
Integer :: num, tag1,tag2, cpu1,cpu2
Real :: tmp1,tmp2

num = (NY+2)*(NZ+2)
cpu1 = CPU_info(1)
cpu2 = CPU_info(2)
tag1 = tag+1
tag2 = tag+2

!Sending right y-z plane
sbuffx_r(0:NY+1,0:NZ+1) = var(NX,0:NY+1,0:NZ+1)
call mpi_isend (sbuffx_r,num,datasize,cpu1,tag1,MPI_COMM_WORLD,
                handle_sx(1),ierr)

!Sending left y-z plane
sbuffx_l(0:NY+1,0:NZ+1) = var(1,0:NY+1,0:NZ+1)
call mpi_isend (sbuffx_l,num,datasize,cpu2,tag2,MPI_COMM_WORLD,
                handle_sx(2),ierr)

!Receiving right y-z plane
call mpi_irecv (rbuffx_r,num,datasize,cpu1,tag2,MPI_COMM_WORLD,
                handle_rx(1),ierr)

!Receiving left y-z plane
call mpi_irecv (rbuffx_l,num,datasize,cpu2,tag1,MPI_COMM_WORLD,
                handle_rx(2),ierr)

End Subroutine Send_BCX
**********************************************************************
! Nonblocking MPI recieve - x
```

```
Subroutine Recv_BCX(NX,NY,NZ,var)
Implicit None
Integer :: NX,NY,NZ , i
Real :: var(0:NX+1,0:NY+1,0:NZ+1)


! wait for recv and send to complete
call mpi_waitall(2,handle_sx,status_s,ierr) !wait for sends
call mpi_waitall(2,handle_rx,status_r,ierr) !wait for recvs

!from the left (high #)
!move data values from buffers into ghost cells
var(0,0:NY+1,0:NZ+1) = rbuffx_l(0:NY+1,0:NZ+1)

!from the right CPU (low #)
!move data values from buffers into ghost cells
var(NX+1,0:NY+1,0:NZ+1) = rbuffx_r(0:NY+1,0:NZ+1)

End Subroutine Recv_BCX
**********************************************************************
```

APPENDIX D

TYPICAL MPI FUNCTIONS USED

```
Great detail explaining MPI function calls can be found at:
http://www-unix.mcs.anl.gov/

Below are a selection of the main function calls used in this thesis.
**********************************************************************
MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag,
           MPI_Comm comm, MPI_Request *request )

MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request )


Input Parameters:
buf:       initial address of send/recieve buffer (choice).
count:     number of elements in send/receive buffer (integer).
datatype:  datatype of each send/receive buffer element (handle).
source:    rank of source (integer).
dest:      rank of destination (integer).
tag:       message tag (integer).
comm:      communicator (handle).

Output Parameter
request: communication request (handle)

All MPI routines in Fortran have an additional argument ierr at the end of
the argument list.
**********************************************************************
MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm )

MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Status *status )


Input Parameters:
buf:        initial address of send/receive buffer (choice)
count:      number of elements in send buffer (nonnegative integer)
datatype:   datatype of each send/receive buffer element (handle)
```

```
dest:       rank of destination (integer)
tag:        message tag (integer)
comm:       communicator (handle)

Output Parameters:
buf:        initial address of receive buffer (choice)
status:     status object (Status)
**********************************************************************
MPI_Waitall(
        int count,
        MPI_Request array_of_requests[],
        MPI_Status array_of_statuses[] )

Input Parameters:
count:               lists length (integer).
array_of_requests:   array of requests (array of handles)

Output Parameter:    array_of_statuses
array of status objects (array of Status). May be MPI_STATUSES_IGNORE
**********************************************************************
MPI_Allreduce ( void *sendbuf, void *recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm )

Input Parameters:
sendbuf:    starting address of send buffer (choice)
count:      number of elements in send buffer (integer)
datatype:   data type of elements of send buffer (handle)
op:         operation (handle)
comm:       communicator (handle)

Output Parameter:
recvbuf:
starting address of receive buffer (choice)
**********************************************************************
```

APPENDIX E

GLOBAL VALUES

```
************************************************************************
Real Function pSUM(var)
Implicit None
Real :: num, var(:,:,:)
Real :: data_G

num = SUM(var)

if (numprocs > 1) then
Call MPI_Allreduce(num,data_G,1,datasize,MPI_SUM,MPI_COMM_WORLD,
                   ierr) !MPI_REAL
  pSUM = data_G
else
  pSUM = num
end if

End Function pSUM
************************************************************************
Real Function pMAXVAL(var)
Implicit None
Real :: num, var(:,:,:)
Real :: data_G

    num = MAXVAL(var)

if (numprocs > 1) then
Call MPI_Allreduce(num,data_G,1,datasize,MPI_MAX,MPI_COMM_WORLD,
                 ierr) !MPI_REAL
  pMAXVAL = data_G
else
  pMAXVAL = num
end if

End Function pMAXVAL
************************************************************************
Real Function pMINVAL(var)
Implicit None
Real :: num, var(:,:,:)
```

```
Real :: data_G

    num = MINVAL(var)

if (numprocs > 1) then
Call MPI_Allreduce(num,data_G,1,datasize,MPI_MIN,MPI_COMM_WORLD,
                 ierr) !MPI_REAL
  pMINVAL = data_G
else
  pMINVAL = num
end if

End Function pMINVAL
**************************************************************************
```

APPENDIX F

DATA OUTPUT

```
**********************************************************************
Subroutine TKE_OUT(dir)
Implicit None
Real   :: uJ(0:DNX+1,0:DNY+1,0:DNZ+1)
Real,dimension(0:NX+1,0:NY+1,0:NZ+1) :: rbuff
Real  :: intsend(1:3),intrecv(1:3)
Integer :: sendcount,recvcount,root,tag,tag2,ierr,ss(MPI_STATUS_SIZE)

!(NXP, NYP, NZP, il, jl ,kl) are computed exactly as in input
root = 0 !assign root processor to be myid = 0.
recvcount = ((NX+2)*(NY+2)*(NZ+2)) !size of data to be sent
sendcount = recvcount !size of data to be received.

If (myid == 0) Then

!Store root processor data section into larger array (uJ).
uJ( NX*(il-1):(il*NX)+1 , NY*(jl-1):(jl*NY)+1, NZ*(kl-1):(kl*NZ)+1 ) =
tke(0:NX+1, 0:NY+1, 0:NZ+1)

!After storing root information, receive the values (myid, il, jl, kl)
!from all other processors

do i = 1, numprocs-1 !loop over all processors (exept root)

!Now receive data from processor "i"
Call MPI_RECV(rbuff,recvcount,datasize,i,tag,MPI_COMM_WORLD,ss,ierr)

!Now receive (il,jk,kl) from the processor that just sent data
Call MPI_RECV(intrecv,3,datasize,i,tag2,MPI_COMM_WORLD,ss,ierr)

!with myid known, along with (il,jl,kl) reconstruct the large array (uJ)
!from the receive buffer.
uJ( NX*(il-1):(il*NX)+1 , NY*(jl-1):(jl*NY)+1, NZ*(kl-1):(kl*NZ)+1 )
= rbuff(0:NX+1, 0:NY+1, 0:NZ+1)

end do !stops loop on root
```

```
!!!!! now write out total domain to binary file.
  open (unit=500,file=trim(str1),form='binary')
  write(500) (((uJ(i,j,k),i=1,DNX),j=1,DNY),k=1,DNZ)
  close(unit=500)

 Else

!If processor is not root, then send your tke data to root.
Call MPI_SEND(tke,sendcount,datasize,root,tag,MPI_COMM_WORLD,ierr)

!store il,jl,kl in an array called (intsend)
intsend(1) = real(il)
intsend(2) = real(jl)
intsend(3) = real(kl)

!Now send values of (il,jl,kl) to root.
Call MPI_SEND(intsend,3,datasize,root,tag2,MPI_COMM_WORLD,ierr)

End If

Return
End Subroutine TKE_OUT
**********************************************************************
```

APPENDIX G

NORMALIZED RST MODEL DERIVATION

The transport equations for k and ϵ are shown below.

$$\frac{\partial k}{\partial t} + \frac{\partial}{\partial x_j}(k u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_k})\frac{\partial k}{\partial x_j}] + P - \epsilon \tag{G.1}$$

$$\frac{\partial \epsilon}{\partial t} + \frac{\partial}{\partial x_j}(\epsilon u_j) = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_T}{\sigma_\epsilon})\frac{\partial \epsilon}{\partial x_j}] + \frac{\epsilon}{k}C_{\epsilon 1}P - \frac{\hat{\epsilon}}{k}C_{\epsilon 2}\epsilon \tag{G.2}$$

The RST model used in this thesis is given below.

$$\begin{aligned}
\frac{\partial R_{mn}}{\partial t} + \frac{\partial (u_j R_{mn})}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[(\nu + \nu_T)\frac{\partial R_{mn}}{\partial x_j}\right] + P_{mn} \\
&+ C^S_{p2}(S_{mj}R_{jn} + S_{nj}R_{jm} + \frac{P}{k}R_{mn}) \\
&+ C^W_{p2}(W_{mj}R_{jn} + W_{nj}R_{jm}) \\
&+ C_{p1}(\frac{2}{3}\epsilon\delta_{mn} - T_i R_{mn}) - T_i R_{mn} \\
&+ C^*_{p2} k S_{mn} - 2\nu\left(\frac{\partial k}{\partial x_j}\frac{\partial R_{mn}}{\partial x_j}\right)
\end{aligned} \tag{G.3}$$

The normalized Reynolds stress tensor is defined as $\overline{R_{mn}} = \frac{R_{mn}}{k}$. Using a normalized Reynolds stress tensor is advantageous because the simulation will require less mesh resolution near walls. Although there are no walls presented in this thesis, future work would most certainly incorporate walls into the existing code for channel flow. The evolution of the normalized stress tensor can be shown as $\frac{D}{Dt}\left(\frac{R_{mn}}{k}\right) = \frac{1}{k}\frac{D(R_{mn})}{Dt} - \frac{R_{mn}}{k^2}\frac{D(k)}{Dt}$. Substituting equations G.1 and G.3 into this expression and utilizing a little calculus yields the evolution of the normalized Reynolds stresses ($\overline{R_{ij}}$).

$$
\begin{aligned}
\frac{\partial \overline{R}_{mn}}{\partial t} + \frac{\partial (u_j \overline{R}_{mn})}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[(\nu + \nu_T)\frac{\partial \overline{R}_{mn}}{\partial x_j}\right] + \overline{P}_{mn} - \overline{P}\,\overline{R}_{mn} \\
&+ C^S_{p2}(S_{mj}\overline{R}_{jn} + S_{nj}\overline{R}_{jm} + \overline{P}\,\overline{R}_{mn}) \\
&+ C^W_{p2}(W_{mj}\overline{R}_{jn} + W_{nj}\overline{R}_{jm}) \\
&+ C_{p1}(T_i)(\frac{2}{3}\delta_{mn} - \overline{R}_{mn}) \\
&+ C^*_{p2}S_{mn} + 2\nu_T\left(\frac{1}{k}\frac{\partial k}{\partial x_j}\frac{\partial \overline{R}_{mn}}{\partial x_j}\right) \qquad \text{(G.4)}
\end{aligned}
$$

The above equation predicts the evolution of $\overline{R}_{ij}$ where $\overline{R}_{ij} = \frac{R_{ij}}{k}$. The constants and functions used for equations G.1, G.2, and G.4 now given. The typical model constants used are $\sigma_k = 1.0$, $\sigma_\epsilon = 1.2$, and $C_{\epsilon 1} = 1.55$. The parameter $C_{\epsilon 2} = \frac{11}{6}f + \frac{25}{Re_T}f^2$ is sensitive to the local turbulent Reynolds number $Re_T = \frac{k^2}{\nu\epsilon}$ of the modeled turbulence via the function $f = \frac{Re_T}{30}(\sqrt{1 + \frac{60}{Re_T}} - 1)$ as per [16].

The turbulent viscosity is given as $\nu_T = \frac{C_\mu F k}{T_i}(\frac{k}{k+k_r})$ where $C_\mu = 0.28$. The RST production term is given by $\overline{P}_{ij} = -\left(\overline{R}_{ik}\,u_{j,k} + \overline{R}_{jk}\,u_{i,k}\right)$. The value of $\overline{P} = \frac{1}{2}\overline{P}_{ii}$ is calculated for the RST and closure equations where $P = \overline{P}k$. Gradients in the flow field are $u_{i,j} = S_{ij} + W_{ij}$. The strain tensor is given as $S_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$ and the vorticity tensor is given by $W_{ij} = \frac{1}{2}(u_{i,j} - u_{j,i})$. The high pressure strain constants are given as $C^S_{p2} = C^W_{p2} = 0.6$. The value of $C^*_{p2} = -0.2F^2(\frac{k}{k+k_r})^2$, where F is a realizability constraint given as $F = \det\left(\frac{3}{2}\overline{R}_{ij}\right)$. The inverse of the turbulent time scale is $T_i = \epsilon/k$. The slow pressure strain term is defined as $C_{p1} = 0.6[2.5 - 1.5(\frac{k}{k+k_r})]$.

APPENDIX H

K/ϵ MODEL CONSTANTS

Numerous testing of the k/ϵ code revealed two constants that can greatly influence the turbulent energy predictions. These two constants C_μ and C^* are included in the energy transfer variable (α). Below is a brief description each has on the energy predictions for the 16x16x32 mesh resolution.

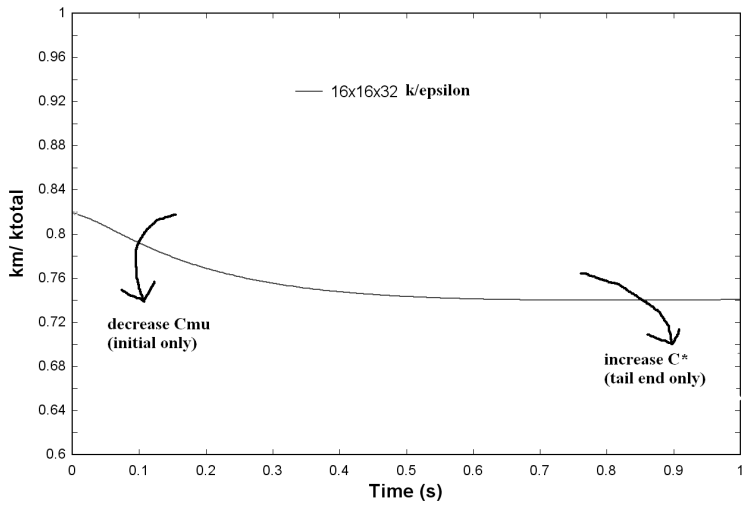

Figure H.1. Effect of changing C_μ and C^* on the k ratio.

Figure H.1 shows the combined results for each constant. The first constant C_μ is shown to push the initial transient on the ratio's down when it is decreased. The opposite is observed when it is increased. This constant also has the effect of raising the total predictions upward when C_μ is decreased, and brings the total predictions down when C_μ is increased (Total predictions are not shown).

Figure H.1 also shows that the constant C^* has more of an effect on the tail end of the energy ratio. Increasing this constant will push the tail end of these curves downward. The opposite is observed when it is decreased. This constant has almost no effect on the total energy predictions.

APPENDIX I

RSTM CONSTANTS

Numerous testing of the RSTM code resulted in six constants/functions that appeared to have more of an effect on the turbulent kinetic energy ratio's and totals. These six constants/functions are C_{p1}, C_{p2}^{S}, C_{p2}^{W}, C_{p}^{*}, $\sigma_{\epsilon 2}$, and C_{μ}. Below is a brief description each has on the energy predictions for the 16x16x32 mesh resolution.

I.1 C_{p1} Contribution

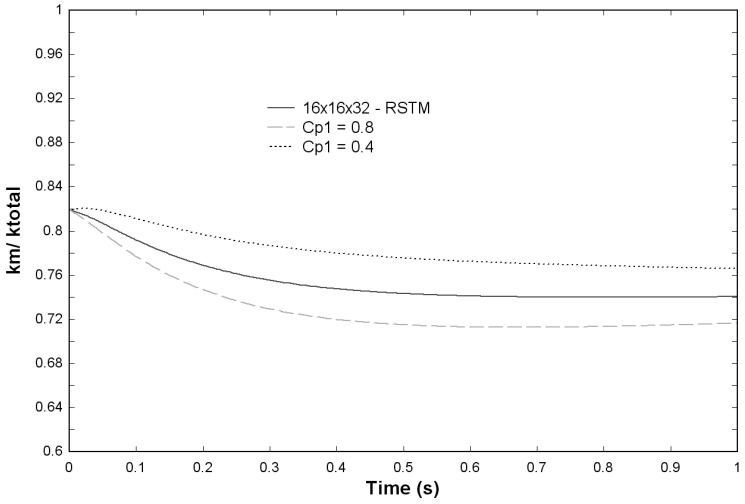

Figure I.1. Effect of changing RSTM C_{p1} function on k ratio.

The function $C_{p1} = 0.6(2.5 - 1.5\frac{k}{k+k_r})$, Figure I.1 and I.2 shows what happens when the 0.6 constant in this function is varied. Increasing the overall value of C_{p1} drives the ratio downward and brings the total slightly upward. The opposite is observed when C_{p1} is decreased.

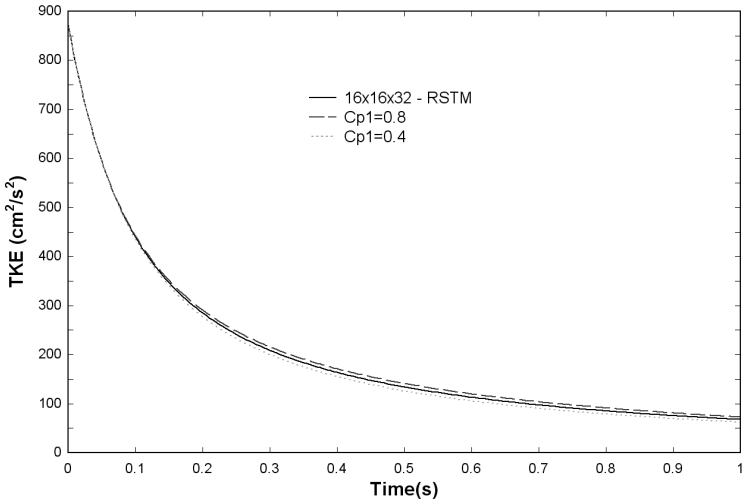

Figure I.2. Effect of changing RSTM C_{p1} function on k total.

I.2 C_{p2}^S and C_{p2}^W Contribution

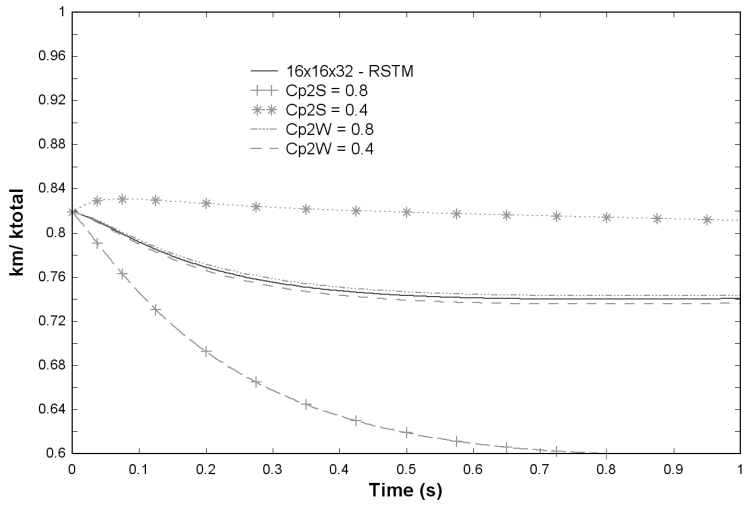

Figure I.3. Effect of changing RSTM constants C_{p2}^S, C_{p2}^W on k ratio.

Figure I.3 shows what happens when the constants C_{p2}^S and C_{p2}^W are varied on the ratio. Increasing C_{p2}^S drives the ratio down very quickly and decreasing C_{p2}^S brings the ratio upwards as shown. The constant C_{p2}^W has an insignificant effect on the ratio.

The total predictions are shown in Figure I.4. The constant C_{p2}^{W} is not shown here because it has no contribution to the total predictions. However, the C_{p2}^{S} term raises the total when it is increased and lowers the total when it is decreased.

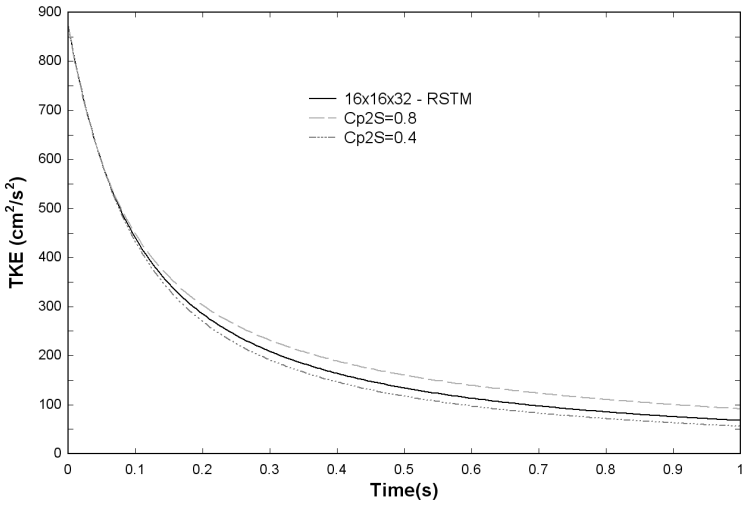

Figure I.4. Effect of changing RSTM constants C_{p2}^{S}, C_{p2}^{W} on k total.

I.3 C_p^* Contribution

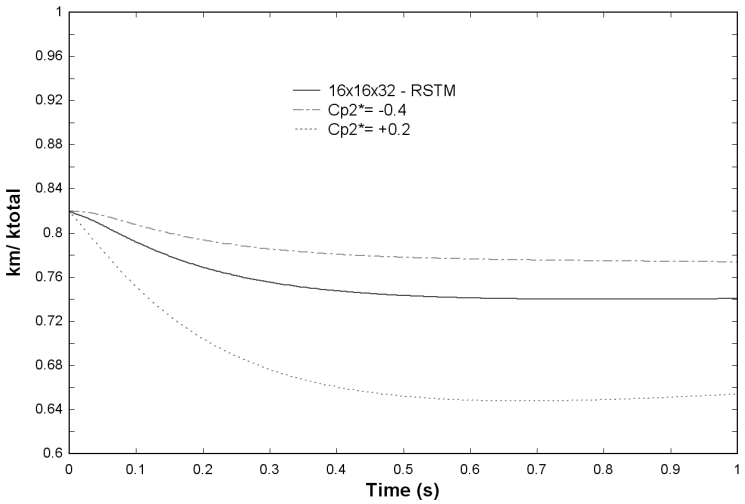

Figure I.5. Effect of changing RSTM function C_p^* on k ratio.

The function $C_{p2}^* = -0.2F^2(\frac{k}{k+k_r})^2$ has the ability to backscatter. It was observed that positive values of C_p^* backscatter more and negative values incur more forward scatter. The C_p^* term has a -0.2 constant out front. Figure I.5 shows the affect of changing this constant. Here it is shown that decreasing this constant to a more negative number -0.4 drives the ratio upwards. Increasing the constant to a positive number $+0.2$ drives the ratio downwards (much the same as C_{p2}^S). Looking at Figure I.6 it is clear that negative values for C_p^* push the total downward and positive values bring the total prediction up.

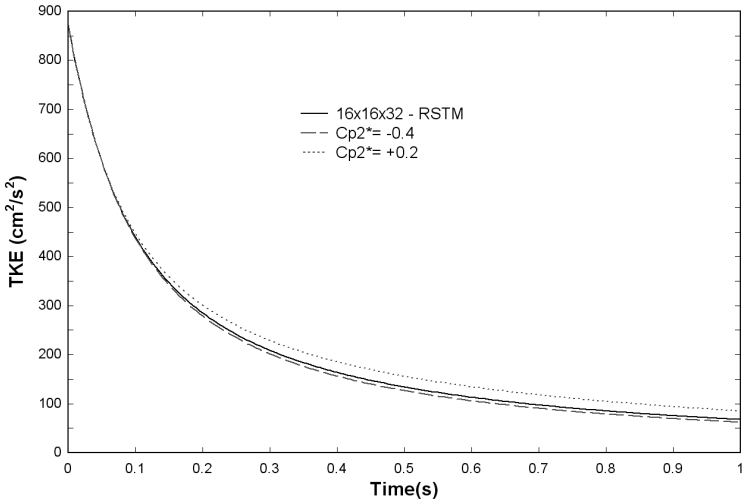

Figure I.6. Effect of changing RSTM function C_p^* on k total.

I.4 $\sigma_{\epsilon 2}$ Contribution

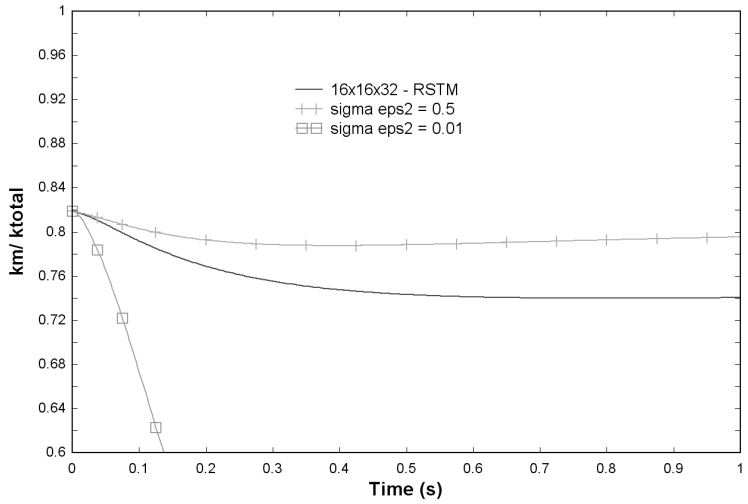

Figure I.7. Effect of changing RSTM constant $\sigma_{\epsilon 2}$ on k ratio.

The constant $\sigma_{\epsilon 2} = 0.11$ is included in the near wall dissipation term. Because this constant is in the dissipation equation, it has the effect of greatly varying the energy predictions from the code. Figure I.7 shows that increasing this constant will

raise the energy ratio and decreasing significantly pushes the ratio downwards. Figure I.8 shows the effect this constant has on the total predictions. By increasing $\sigma_{\epsilon 2}$, the total is brought upwards. Decreasing $\sigma_{\epsilon 2}$ pushes the totals down.

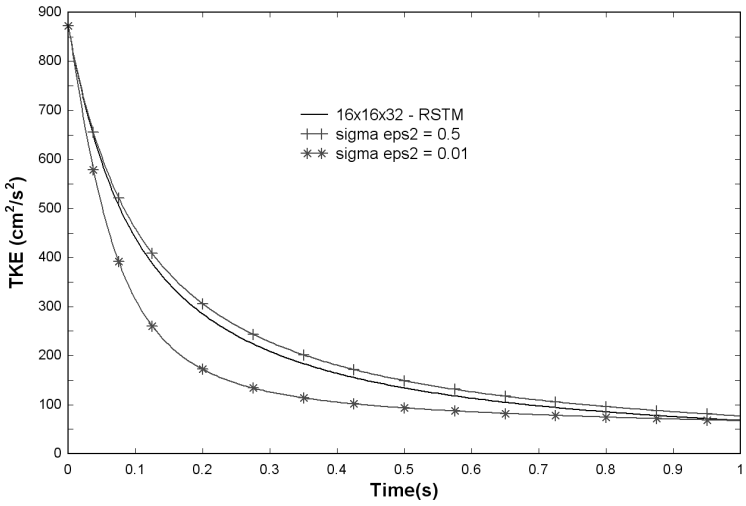

Figure I.8. Effect of changing RSTM constant $\sigma_{\epsilon 2}$ on k total.

I.5 C_μ Contribution

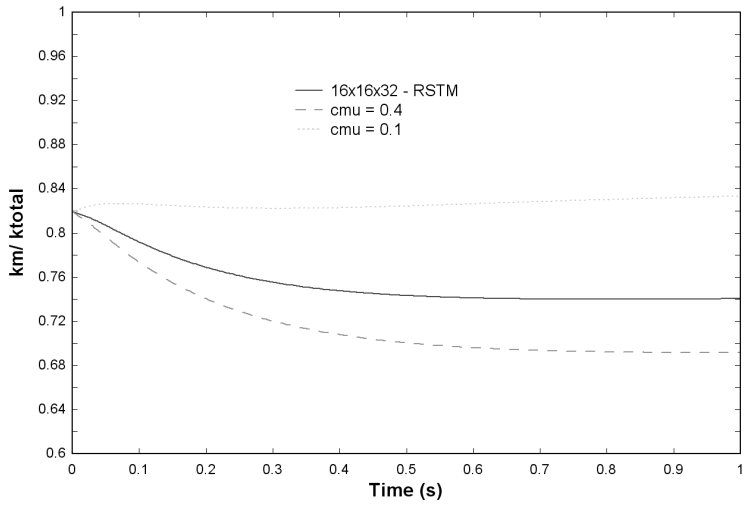

Figure I.9. Effect of changing RSTM constant C_μ on k ratio.

The constant C_μ is used for the calculation of the turbulent viscosity. Figure I.9 shows that decreasing this constant will bring the ratio's upward. Increasing this constant will push the ratio down. Figure I.10 shows that increasing C_μ will raise the totals slightly and decreasing C_μ will bring the totals down slightly. However, C_μ has most of its effect on the ratio's as the effect on the total energy prediction is hardly noticeable.

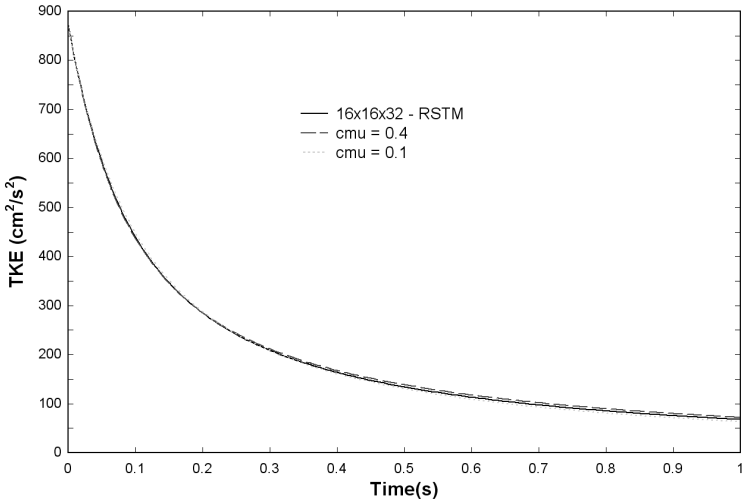

Figure I.10. Effect of changing RSTM constant C_μ on k total.

I.6 Final Remarks on RSTM Constants

These six constants appeared to have the most effect on the overall results from the RSTM code. However, it should be noted that other model constants also have some effect on the results. The figures shown were all demonstrated with a 16x16x32 mesh resolution. Other mesh sizes will produce similar (but slightly different) results. Hence, great care must be taken when trying to balance these constants to give the correct energy predictions for all mesh sizes.

BIBLIOGRAPHY

[1] Carati, D., Ghosal, S., and Moin, P. On the representation of backscatter in dynamic localization models. *Physics of Fluids 07* (1995), 606–616.

[2] Chang, W., Giraldo, F., and Perot, J.B. Analysis of an exact fractional step method. *Journal of Computational Physics 180* (2002), 183–199.

[3] Chasnov, J. Simulation of the kolmogorov inertial subrange using an improved subgrid model. *Physics of Fluids A 3* (1991), 188–200.

[4] Comte-Bellot, G., and Corrsin, S. Simple eulerian time correlation of full and narrow-band velocity signals in grid-generated isotropic turbulence. *Journal of Fluid Mechanics 48* (1971), 273–337.

[5] de Bruyn Kops, S.M., and Riley, J.J. Direct numerical simulation of laboratory experiments in isotropic turbulence. *Physics of Fluids 10* (1998), 2125–2127.

[6] de Bruyn Kops, S.M., Riley, J.J., and Kos'aly, G. Direct numerical simulation of reacting scalar mixing layers. *Physics of Fluids 13* (2001), 1450–1465.

[7] Deardorff, J.W. The use of subgrid transport equations in a three-dimensional model of atmospheric turbulence. *ASME J. Fluids Engineering 95* (1973), 429.

[8] Germano, M. Turbulence: the filtering approach. *Journal of Fluid Mechanics 238* (1992), 325–336.

[9] Ghosal, S., Lund, T.S., Moin, P., and Akselvoll, K. A dynamic localization model for large-eddy simulation of turbulent flows. *Journal of Fluid Mechanics 286* (1995), 229–255.

[10] Girimaji, S., Sreenivasan, R., and Jeong, R. Pans turbulence model for seamless transition between rans,les: fixed-point analysis and preliminary results. In *Proceedings of ASME-JSME Joint Fluids Engineering Conferences* (2003).

[11] Harlow, F.H., and Welch, J.E. Numerical calculation of time dependent viscous incompressible flow of fluid with free surface. *Physics of fluids 8* (1965), 2182.

[12] Jones, W.P., and Launder, B.E. The prediction of laminarization with a two-equation model of turbulence. *International Journal of Heat and Mass transfer 15* (1972), 301–314.

[13] Launder, B.E., and Sandham, N.D. *Closure Strategies for Turbulent and Transitional Flows.* Cambridge University Press, 2002.

[14] Menter, F.R., Kuntz, M., and Bender, R. A scale-adaptive simulation model for turbulent flow predictions. *ASME Wind Energy Symposium, 41st AIAA Aerospace Meeting 2003-767* (2003).

[15] Nikitin, N.V., Nicoud, F., Wasistho, B., Squires, K.D., and Spalart, P.R. An approach to wall modeling in large-eddy simulations. *Physics of Fluids 12* (2000), 1629–1632.

[16] Perot, J.B., and de Bruyn Kops, S. Modeling turbulent dissipation at low and moderate reynolds numbers. *Journal of Turbulence 07* (2006).

[17] Perot, J.B., and Natu, S. A model for the dissipation tensor in inhomogeneous and anisotropic turbulence. *Physics of fluids 16* (2004), 4053–4065.

[18] Piomelli, U., Balaras, E., Pasinato, H., Squires, K., and Spallart, P. The inner-outer layer interface in large-eddy simulations with wall-layer models. *International Journal of Heat and Fluid Flow 24* (2003), 538–550.

[19] Pope, S.B. *Turbulent Flows.* Cambridge University Press, 2000.

[20] Rotta, J.C. Statistische theorie nichthomogener turbulenz. *Z. Phys. 129* (1951), 547–572.

[21] Schumann, U. Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli. *Journal of Computational Physics 18* (1975), 376–401.

[22] Smith, B.R. A near wall model for the k-l two equation turbulence model. *AIAA 94-2386* (1994).

[23] Spalart, P., and Allmaras, S. A one-equation turbulence model for aerodynamic flows. *Recherche Aerospatiale 1* (1994), 5–21.

[24] Spalart, P., Jou, W., Streetlets, M., and Allmaras, S. Comments on the feasibility of les for wings, and on a hybrid rans/les approach. *First AFOSR International Conference on DNS/LES, Ruston, Louisiana, USA* (2001).

[25] Speziale, C.G. Turbulence modeling for time-dependant rans and vles: A review. *AIAA Journal 36* (1998), 173–184.

[26] Wilcox, D.C. *Turbulence modeling for CFD.* La Canada, CA: DCW Industries, 1993.