NPS-ME-93-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AMPHIB: A USERS MANUAL

D. Mukutmoni
M. D. Kelleher
Y. K. Joshi

Prepared for:

Naval Postgraduate School
nterey, California 93943

NDNC
[illegible]
[illegible] - 92-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943

This report was prepared in conjuction with research sponsored in part by the Naval Weapons Support Center, Crane, Indiana.

Reproduction of all part of this report is authorized.

This report was prepared by:

Paul J. Marto
Dean of Research

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassfied	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable) ME	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NWSC, Crane; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Crane, IN 47522	10. SOURCE OF FUNDING NUMBERS: Program Element No; Project No; Task No; Work Unit Accession Number

11. TITLE (Include Security Classification)
AMPHIB: A Users Manual

12. PERSONAL AUTHOR(S) D. Mukutmoni, M. D. Kelleher and Y. K. Joshi

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Technical Report	From Jan. 92 To Dec. 92	January 25, 1993	112

16. SUPPLEMENTARY NOTATION
The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS (continue on reverse if necessary and identify by block number)
Electronic Cooling, Conjugate Heat Transfer, Finite Volume Method, QUICK

19. ABSTRACT (continue on reverse if necessary and identify by block number)

A general purpose three-dimensional code (AMPHIB) that solves electronic cooling problems is documented. In its present structure, the code is set up for computations in liquid immersion cooling of an **m** by **n** array of chips embedded in a substrate in a three-dimensional rectangular enclosure. Nevertheless, it can be modified to solve problems in forced, mixed and natural convection for a wide range of boundary conditions. The subroutines and the input are described in detail. A listing of the code and sample example problems are also included.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Yogendra Joshi	22b. TELEPHONE (Include Area code) 408-656-3400; 22c. OFFICE SYMBOL ME/Ji

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

AMPHIB: A USERS MANUAL

A Documentation of a General Purpose Code for Electronic Cooling Applications

prepared by
Devadatta Mukutmoni, Yogendra K. Joshi and
Matthew D. Kelleher

Department of Mechanical Engineering
Naval Postgraduate School
Monterey, CA 93943

AMPHIB

Abstract

A general purpose three-dimensional code (AMPHIB) that solves electronic cooling problems is documented. In its present structure, the code is set up for computations in liquid immersion cooling of an **m** by **n** array of chips embedded in a substrate in a three-dimensional rectangular enclosure. Nevertheless, it can be modified to solve problems in forced, mixed and natural convection for a wide range of boundary conditions. The subroutines and the input are described in detail. A listing of the code and sample example problems are also included.

TABLE OF CONTENTS

Page

NOMENCLATURE FOR REPORT 4
INTRODUCTION 6
OUTLINE OF ALGORITHM 15
EXAMPLES 21
INPUTS AND SUBROUTINES 33
APPENDICES
A 45
B 49
C 59
REFERENCES 111

Nomenclature for the Report

A_P, A_N, A_S, A_W, A_E, A_F, A_B	Coefficients of the finite difference equation
g	Acceleration due to gravity, $\frac{m^2}{s}$
C	Scaling factor in the convection-diffusion equation
G	Non-dimensional mass flux at the boundaries of a cell
H	Height of the cavity, m
i	x subscript
j	y subscript
k	z subscript
Nu	Nusselt number
P	Non-dimensional pressure
Pr	Prandtl number
q	Volumetric heat generation, $\frac{W}{m^3}$
Ra	Product of Rayleigh and Prandtl number
S	Source term in difference equation
t	Non-dimensional time
T	Non-dimensional temperature
T_C	Cold wall temperature, ^oC
T_H	Hot wall temperature, ^oC
ΔT	Temperature difference, $T_H - T_C$, ^oC
U	Non-dimensional x-direction velocity
V	Non-dimensional y-direction velocity
W	Non-dimensional z-direction velocity
x	Non-dimensional horizontal spatial coordinate
y	Non-dimensional vertical spatial coordinate
z	Non-dimensional spatial coordinate in the direction of depth

Greek Symbols

α	Thermal diffusivity, $\frac{m^2}{s}$

β	Coefficient of volume expansion, $\frac{1}{K}$
ϕ	Generic non-dimensional field variable
κ	Thermal Conductivity, $\frac{W}{mK}$
Δx, Δy, Δz	Non-dimensional mesh or cell size
ν	Dynamic viscosity, $\frac{m^2}{s}$
ρ	Density, $\frac{kg}{m^3}$

Superscripts

*	Estimated quantity
'	Correction to estimated quantity
n	Present time level

Subscripts

c	Refers to the chip
p,n,s,e,w,f,b, ee,ww,ss,nn, bb,ff	Designation of control volume boundaries
P, N, S, E, W,F,B, EE,WW,SS,NN, BB,FF	Node designation of basic grid

INTRODUCTION

1.1 General Remarks

Over the years, computational fluid dynamics has emerged as a powerful tool to solve engineering and scientific problems related to fluid flow. This ranges from the familiar such as solving for flow past a supersonic airplane; flow in an internal combustion engine; flow in an oil reservoir; atmospheric flows and flow in turbo-machines to the esoteric such as geophysical and astrophysical fluid dynamics.

The increase in computing power which roughly doubles every two years has contributed immensely to the feasibility of solving realistic engineering problems. More stunning advances in computing resources are expected in the near future. For instance, massively parallel architecture promises to create a prototypical teraflop machine (which is capable of 10^{12} floating point operations per second) by 1995 (Deng et. al, 1992). This can well be appreciated considering the fact that three-dimensional unsteady problems of moderate complexity can be simulated rather effectively using an RS/6000 workstation which is capable of a 16 megaflops(10^6) performance for a CFD code.

CFD has gradually evolved to become a cost-effective alternative to experimentation with respect to engineering design in many cases. For instance in the aerospace industry, the lead time in design and development of aircrafts has been considerably reduced (Fletcher, 1988), thanks to CFD replacing time-consuming and costly testing procedures in many of the steps. This cost-effectiveness is expected to improve relentlessly, as computing costs decrease.

Nevertheless, CFD must be used in the design process with utmost care. In the first place, CFD can never claim to replace experiments; at least not yet. Experiments form a critical component of the overall process. It is imperative that the CFD code be *validated* by experiments for the particular case. There are several reasons for that. The validity of the boundary conditions, the refinement of the grid or the validity of certain simplifying assumptions can only be checked and confirmed by experiments. There might even be bugs which need to be fixed. For computer codes which include turbulence modelling it is sometimes required to check whether the emperically determined constants are valid for the given case.

The thermal management of electronic components (otherwise known as electronic cooling) can also benefit from these advances in computing power. The design process has in the past relied heavily on empirical testing. Convective heat transfer modelling was expensive due to the complex mathematical equations involved. The heat transfer coefficients were determined from experiments and were then used in simpler conduction calculations to determine the maximum chip temperature and other quantities of interest to an engineer.

With advances in Computational Fluid Dynamics and computing power, the convection and conduction problems can be solved simultaneously in a coupled manner. Computational fluid flow and heat transfer is increasingly being used as part of this design process. A number of general-purpose commercial code available (such as PHOENICS, FLUENT, FIDAP etc) can be used to solve applications in electronic cooling. There are several drawbacks though. The codes are too general (and very expensive!). A considerable amount of effort must be expended in order to tailor them to the specific application. More important, the source code is almost never available. The option of modifying the code does not arise.

The present code AMPHIB attempts to redress these disadvantages. It is a general-purpose code that is exclusively designed to solve problems in electronic cooling. AMPHIB is also fully validated by experiments. It can therefore be used directly in such applications with minimal modifications if at all. To the best of our knowledge, this is the first software package that specifically solves these class of problems. Since the source code is readily available, the code can be modified to solve problems of much greater complexity such as those involving turbulence modelling, mass transfer etc.

1.2 Purpose of the Code

The primary purpose of the code is as a design tool for applications in electronic cooling. As noted earlier, experiments form an integral part of this procedure. The usefulness of the code is best illustrated with examples.

Consider a hypothetical configuration shown in figure 1.1. A power dissipating chip is protruding from a vertical substrate. It is being cooled by a dielectric fluid in direct contact with the chip surface. The dielectric fluid is hermetically sealed in an enclosure which contains the chip and substrate. Heat exchangers at the top and the bottom wall extract heat from the liquid. This configuration is what is commonly known as cooling by liquid immersion.

To check design, experiments may have been carried out to monitor the chip temperature for a case that is considered optimum. The chip symmetrically placed at the center of the substrate would seem to be a reasonably optimal design. Chip temperatures are recorded for different power levels in the chip. However, some nagging doubts still remain. Is this really the optimal configuration? What if the chip was placed in an eccentric position? Would the chip temperature be reduced? Also, would it matter if the substrate was a different material of higher conductivity e.g, ceramic instead of plexiglas. What would be the effect of increasing the spacing between the substrate and the vertical wall opposite to it?

These are legitimate questions. Unfortunately it is very time consuming to respond to these questions experimentally. This is where CFD codes such as AMPHIB fit in. As a first step, the code is used to validate the experiments. The chip temperatures predicted by the program must be made to match those that were experimentally determined within an acceptable bound.

Appropriate models of the boundary conditions must be introduced in the program. To have a well defined problem the geometrical and physical parameters of the setup must be included in the code. This includes the complete dimensions of the enclosure, the dimensions of the chip, the thermo-physical properties of the chip, substrate and fluid. The power dissipated by the chip and the temperature of the heat sinks (the top and bottom wall) and the location of the chip must also be known.

The present code (AMPHIB) can handle the problem so described *without modifications*. If the code predicts reasonably well, it is quite straightforward to change the parameters to predict chip temperatures for a different geometry or different materials. The exact details of the code and implementation are given in a different chapter.

Consider another configuration of a greater complexity shown in figure 1.2. A three by three array of chip protrusions is shown. Experiments were performed with a certain spacing. All the chip dimensions are the same. The chip is so configured that the largest dimension is aligned to the vertical. Experiments were performed for a certain spacing between the substrate and the opposite wall. Unfortunately, due to constraints in the instrumentation, experimentation was possible only for spacings that were larger than the design constraint. Spacing between chips and wall are typically very small since the chip density in real applications are quite high.

The numerical code can be very usefully applied to such a situation. Firstly, the code validates the experimental results for the large spacing. Later, numerical computations are done for the smaller design spacing. It is also possible to check computationally whether a slightly different alignment of chips is a good idea. For instance, if the largest

dimension of the identical chips was aligned to the horizontal rather than the vertical direction. This is a rather trivial thing to do computationally but a rather cumbersome thing to do experimentally. In the latter case, a whole new experimental setup has to be constructed. The optimal approach is to judiciously combine numerics with experimentation in the design process. The configuration shown in figure 1.2 can be handled by AMPHIB with no changes. In fact, the program can handle an $\mathbf{n} \times \mathbf{m}$ array of chips on a substrate where n and m are whole numbers. Theoretically, there are no limits on the number of chips in the computations. Nevertheless, a 5 by 5 chip array would itself require 10^7 bytes of memory for reasonably accurate results in three-dimensional computations. Thus, there are practical limits on the total number that can be handled by the code.

There are other related configurations which can be tackled by the program with minor modifications. One case, which is important from the engineering point of view is shown in figure 1.3. The array of chips shown is cooled by a fluid that is circulated by a pump. This is a mixed convection problem. An important question is should the fluid circulate from the top to the bottom or the other way round. The answer may not be simple. It could depend on the power dissipation rate, the properties as well as the geometry of the configuration. The code can be used. The modifications in the code that are necessary to solve such a problem are discussed in a later chapter.

A more general configuration is shown in figure 1.4. The whole chip assembly and the enclosure is at angle with the vertical and fluid is circulated through it. This problem can be solved with surprisingly little changes with AMPHIB.

1.3 Overview of the Report

In chapter two, the algorithm of the code is presented. The description is brief. Interested readers can look into the references provided in that chapter for additional details. A good readable account of the control volume method applied to heat transfer and fluid flow problems is given in Patankar (1980). Readers interested in the code purely as a user are advised to go to chapter four directly which describes the inputs and subroutines of AMPHIB.

Chapter three describes three example problems that were looked into. One of the example deals with convection and conduction in an enclosure with a three by three array of chips embedded in a substrate. Another example is convection in an enclosure with no substrate or chip. Both these examples are compared with experiments. That is precisely why they were chosen. These two examples underscore the essential correctness and accuracy of the code. The third example is not related to any experiments. It is a

benchmark. It is useful to have one, especially when changes are made in the code. One can always fall back on this case for debugging and checking purposes.

Chapter four provides a detailed and complete account of the inputs to the program (all of them initialized in BLOCK DATA) and the subroutines. Appendix A details the implementation of the boundary conditions. This chapter is important for someone who intends to modify the code and thus extend the scope of AMPHIB. The present version of the code has a fixed type of boundary condition. There are no options or flags in the code which would enable one to chose an arbitrary set of boundary conditions for each wall. Appendix A has easy to follow instructions on how to change the code in order to accommodate a more general set of boundary conditions.

Appendix B provides explicit algebraic expressions for the difference equations and numerical boundary conditions that are implemented in the code. Appendix C gives a listing for AMPHIB.

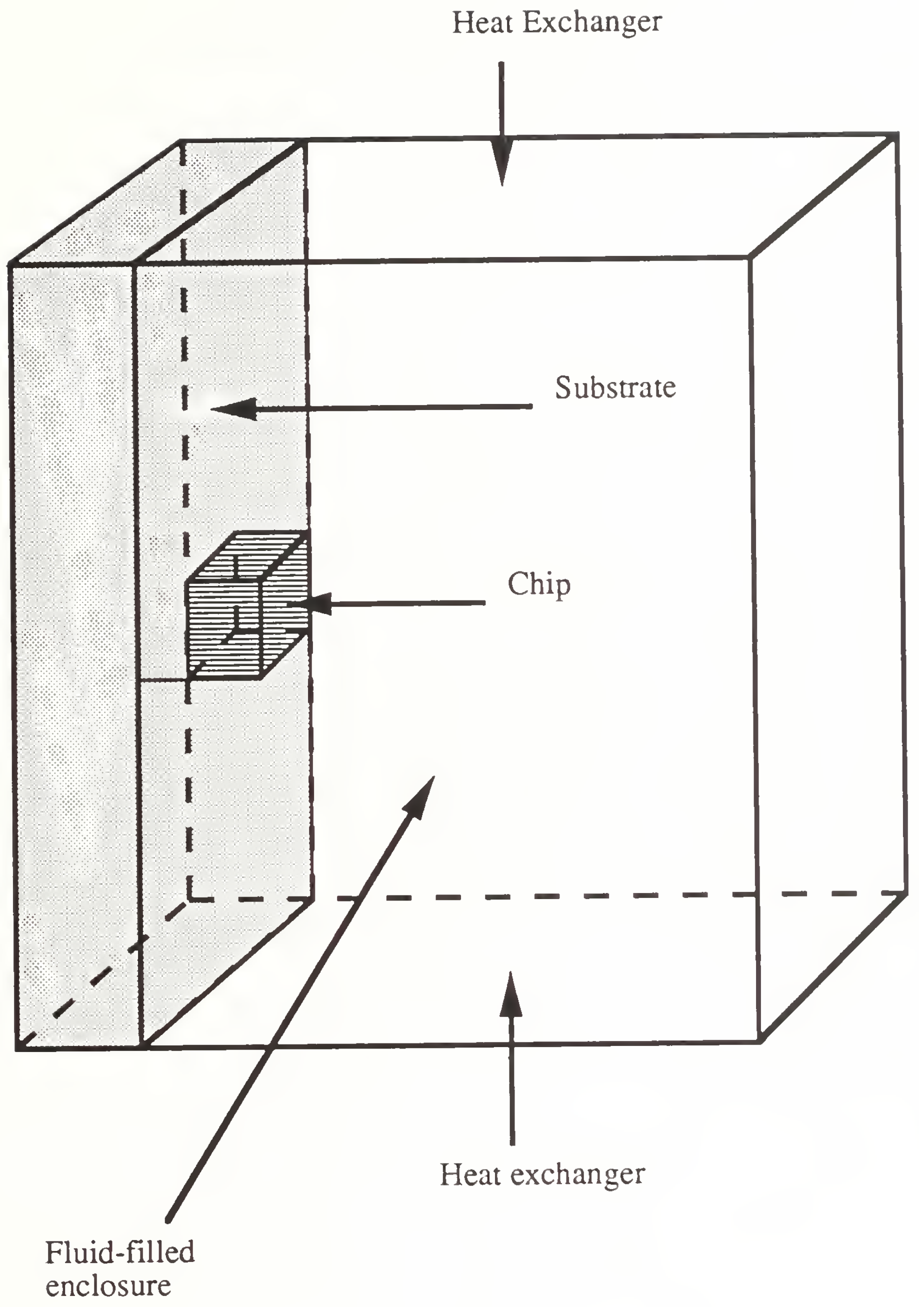


Figure 1.1

In this configuration, a single chip embedded in a substrate is being cooled by a dielectric fluid. The fluid is hermetically sealed in a chamber. The heated fluid is cooled by heat exchangers at the top and bottom walls. The first step is to mathematically model the physical problem. For example, the top and bottom wall can be modelled as isothermal heat sinks. The computer code can then be used to calculate the chip temperatures for different geometrical parameters and transport properties.

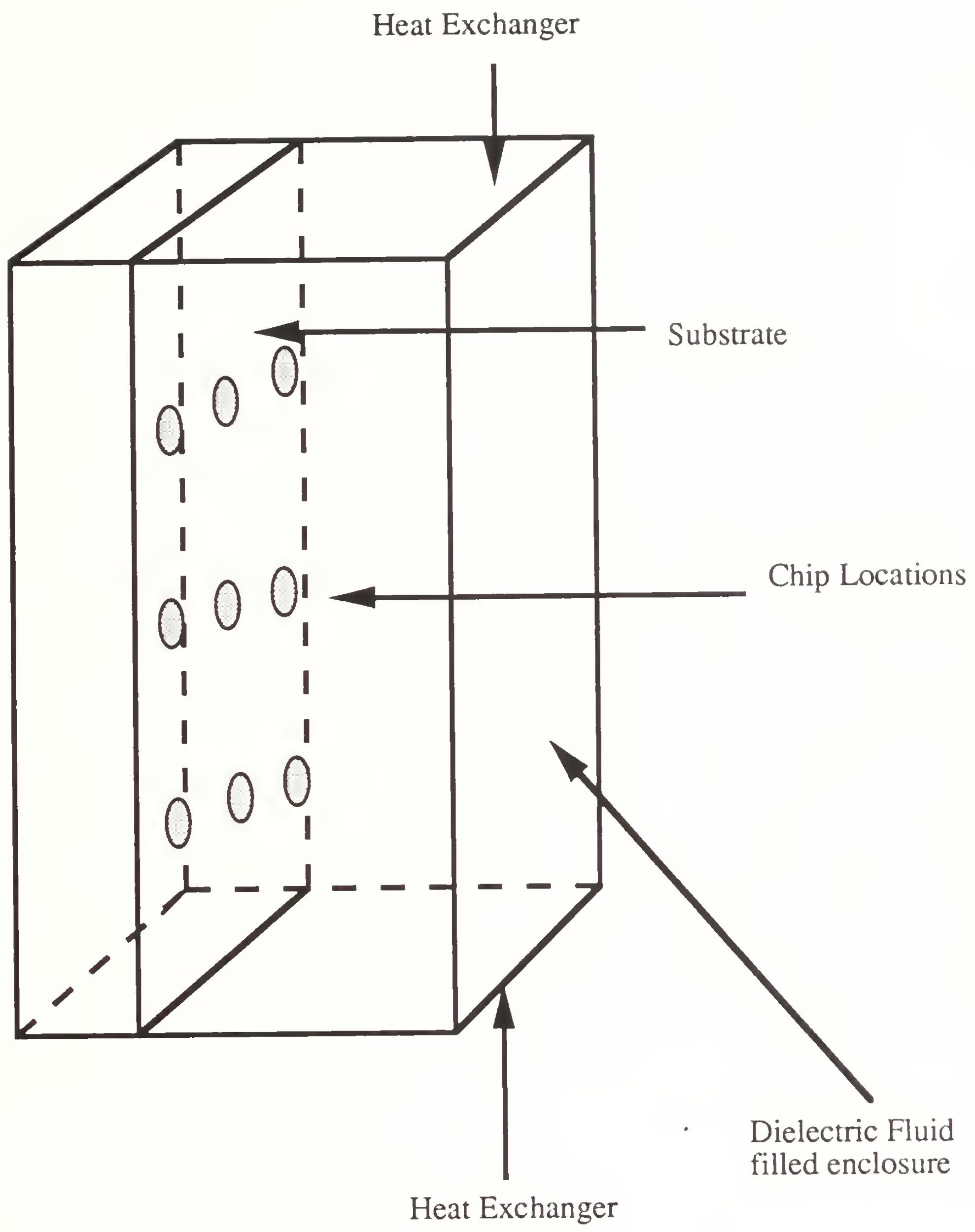


Figure 1.2

This is a chip-substrate configuration of a slightly greater complexity. There are nine chips in a three by three array that is being cooled by a dielectric fluid in an enclosure. This problem can be solved by AMPHIB without changes provided that the chip locations are along a rectangular grid.

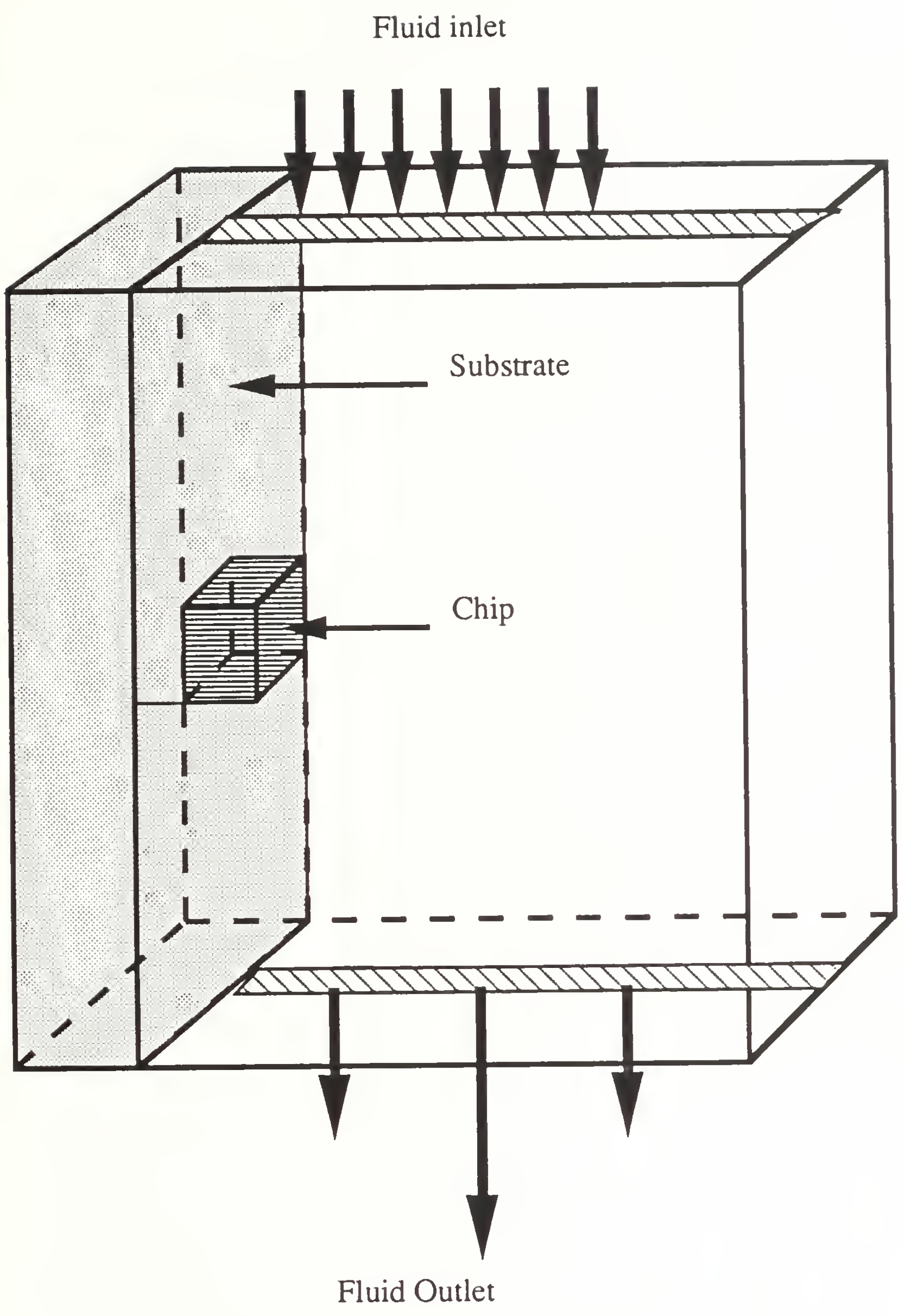


Figure 1.3

This is a configuration similar to figure 1.2, except that cooling is now aided by fluid that is being circulated by a pump. This problem can be solved by AMPHIB only after modifications are being made in the code which is described at length in a later chapter.

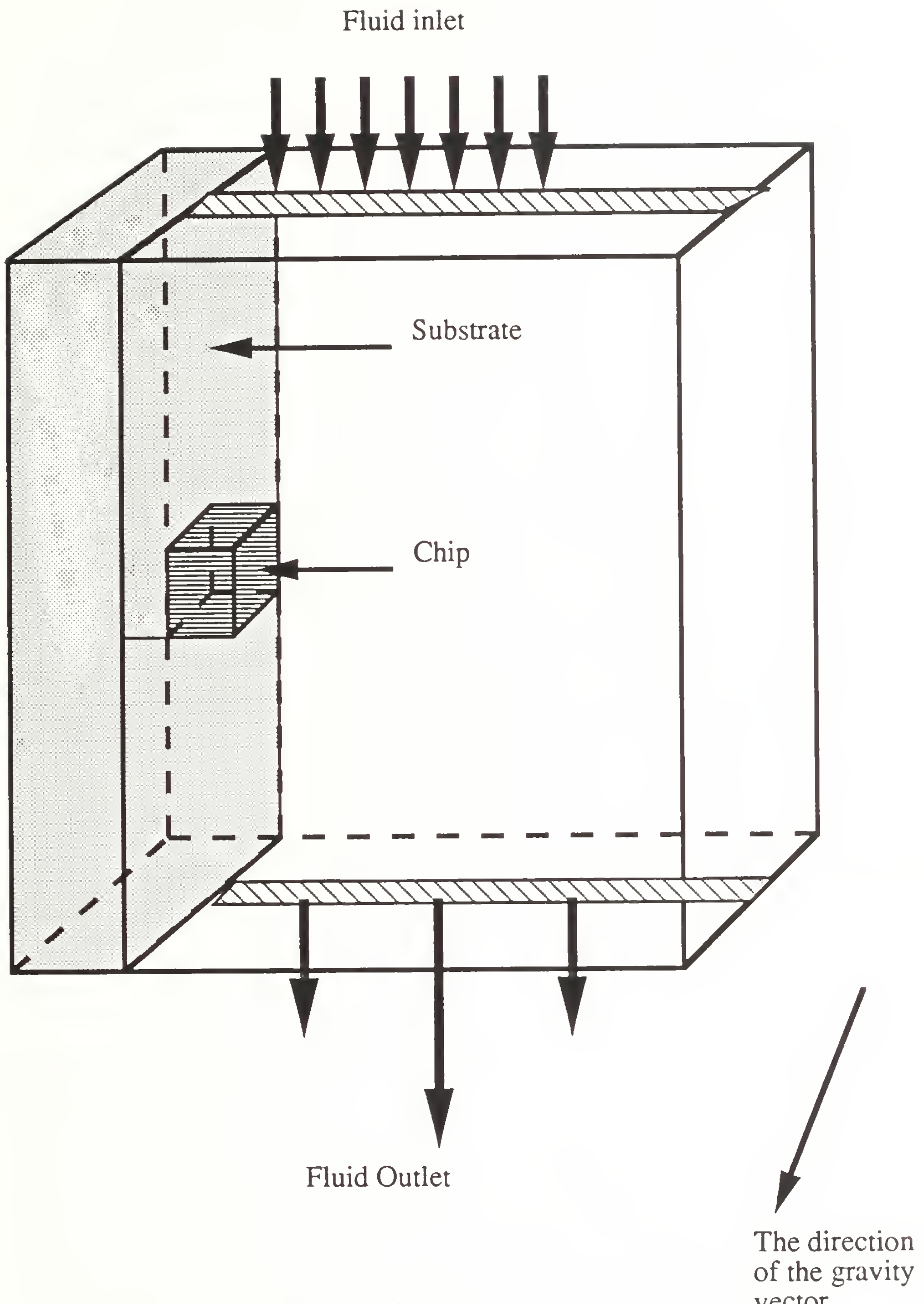


Figure 1.4

This is a chip-substrate configuration that is being cooled in a mixed-convection mode. However, as in many real-life situations, the whole assembly is at an angle with the vertical. Nevertheless, this problem can be solved by AMPHIB with minimal changes in the program.

OUTLINE OF ALGORITHM

2.1 The Scope of the Program

The program is a general purpose finite difference FORTRAN-77 code designed to solve unsteady and steady three-dimensional heat transfer and fluid flow problems in rectangular cartesian coordinates. The code is designed to effectively handle conjugate heat transfer situations, i.e., when conduction and convection is solved simultaneously. However, pure convection or conduction problems can also be solved.

The program uses the control volume method (Patankar, 1980) to discretize the governing Navier-Stokes, continuity and the energy equations in the primitive variable formulation. The SIMPLEX algorithm (Van Doormal and Raithby, 1985) is used to determine the five field variables (the three velocity components, temperature and pressure). It is essentially a more implicit version of SIMPLE (Patankar, 1980). Due to the more implicit nature of the algorithm, larger time steps can be used without encountering numerical instability. This leads to considerable saving of computer resources for steady state computations since a smaller number of time steps would be required. The computational work per time step would be increased somewhat, but is more than compensated by the decrease in the total number of time steps required to reach a converged solution.

The algorithm is for three-dimensional and unsteady heat transfer and fluid flow problems that marches in time. Time stepping is performed by a first order Backward-Euler scheme. The algorithm is therefore implicit in time. The grids are staggered, and the QUICK scheme is used for interpolating the convective terms (Leonard, 1983). Staggered grid means that grid points for the velocities (the so called vector grid) are different from those of pressure and temperature grid points (the scalar grid). More will be said of it later. Generally speaking, the staggered grid leads to more accurate results since the need to interpolate reduces somewhat. It is now an accepted norm for all CFD calculations with finite differences with the primitive variable formulation.

QUICK (quadratic interpolation for convective kinetics) is a third order polynomial interpolation scheme for the convective terms. It is almost as stable as the first order

upwind scheme and vastly more accurate. It does have the twin advantages of being robust as well as accurate compared to the more conventional techniques such as upwinding or central differencing which are the mainstay of most if not all commercially available codes to solve convection problems. It is a well known fact (Patankar, 1988) that although QUICK is very accurate it is not as robust as the upwind scheme. There are times when the QUICK scheme might not converge due to insufficient number of grid points in regions of high velocities. The code therefore has the option of using the upwind scheme when such a situation arises. In particular, the code uses a scheme which is a weighted average of QUICK and upwind. Depending on the case one can have a range of schemes ranging from pure upwind to pure QUICK.

Except for the viscosity, the fluid is assumed to have constant transport properties and is incompressible except for density changes to account for buoyancy. Viscous dissipation and pressure work are neglected. These assumptions are justified for most applications. The harmonic mean formulation for the interfacial diffusivities is used to effectively handle sharp discontinuities in property values in the computational domain that arise in conjugate problems (Patankar, 1980). Such is the case in electronic cooling problems since the thermo-physical properties vary drastically between the fluid and the solid regions. A substrate and package are examples of solid regions where properties might differ sharply.

2.2 Algorithmic Details

The code uses the Control Volume Method to solve five coupled non-linear partial differential equations (governing equations). All the equations can be cast in the form of a three-dimensional convection-diffusion equation of the following type:

$$\frac{\partial \phi}{\partial t} + \frac{\partial}{\partial x}(U\phi) + \frac{\partial}{\partial y}(V\phi) + \frac{\partial}{\partial z}(W\phi) = C\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2}\right) + S \qquad (2.1)$$

The different governing equations can be recovered by setting different values for the canonical variable ϕ and the constant C as well as the source term S which are tabulated below.

Equation	ϕ	C	S

x-momentum	U	Pr	$-\frac{\partial P}{\partial x}$
y-momentum	V	Pr	$-\frac{\partial P}{\partial y} + T\,Pr\,Ra$
z-momentum	W	Pr	$-\frac{\partial P}{\partial z}$
Energy	T	1	0
Continuity	1	0	0

The governing equations are a result of a very specific set of non-dimensionalization of the dependent and independent variables. The x, y and z coordinates are non-dimensionalized by H, the enclosure height. The velocities are scaled by $\frac{\alpha}{H}$, the time is non-dimensionalized by $\frac{H^2}{\alpha}$, the pressure difference by $\frac{\rho\alpha^2}{H^2}$; and the temperature difference by $\frac{qH^2}{k_c}$. q is the volumetric rate of heat generation per chip, α is the thermal diffusivity of the fluid, ρ is the fluid density and the the conductivity of the chip is k_c. For a different set of scales the governing equations will be of the same form as equation (1) but with different values of C and S.

A grid is imposed in the physical domain and the governing equations are discretized on the grid. For the control volume method, the governing equations are first integrated around a volume about the grid point known as the control volume. The problem is now reduced to that of calculating the values of the dependent variables at the grid locations. The generic convection-diffusion equation (equation 1) is reduced to an algebraic equation of the following form:

$$A_p\phi_p = A_E\phi_E + A_W\phi_W + A_N\phi_N + A_S\phi_S + A_E\phi_E + A_B\phi_B + A_F\phi_F + S \qquad (2.2)$$

The scalar variables (pressure and temperature) are in the non-staggered grid. The grid consists of the grid points and control volume surrounding it. In figure 2.1 the shaded region indicates the non-staggered grid. The grid points are indicated by the circular dots. The subscripts refer to the location with respect to the grid point as shown in the same figure.

1. The grid point in question is referred to by the subscript P.
2. The grid point to the right of P is subscripted by E (east). Similarly the grid point to the right of E is the EE grid point as indicated in figure 2.2.
3. The grid point to the left of P is the W (west) point.

4. The grid point above P is N (north).
5. The grid point below P is S (south).
6. The B and F grid points are not shown in the figure. The B (back) grid point is located above the paper immediately above P. This is consistent with a right-handed cartesian system. Similarly, the F (front) grid point is located under the plane of the paper.
7. The grid points WW, NN, SS, BB and FF are similarly defined.

The grids are chosen such that the walls of the enclosure coincide with the control volume faces of the non-staggered grid. The staggered grids are for the vector variables (the three velocities u, v and w). The staggered grids are shifted by half a control volume towards their respective negative directions. In figure 2.1 the grid point for the u-velocity (velocity in the x-direction) are shown as triangular dots. The control volume surrounding the u-velocity grid is highlighted by bold dashed lines.

The u-velocity grid points are thus located at the east and west faces of the non-staggered control volume. The staggered grid points for the v-velocity are portrayed as square dots. The staggered v-velocity control volume is enclosed by bold dashed lines just as the other one. The v-velocity grid points coincide with the north and south face of the scalar grid as shown in the figure.

The coupled fluid flow and heat transfer problem is solved using the SIMPLEX algorithm. Details are given in Van Doormal and Raithby (1985). Briefly, equations are solved for the temperature, the three velocities, the pressure correction and the coupling constants for the three momentum equations. The flow chart is shown in figure 2.2. The temperature, the velocities, the coupling constants, and the pressure corrections. The most updated field variables are used at each stage, e.g., for solving the v-velocity the updated u-velocity and temperature are used and so on.

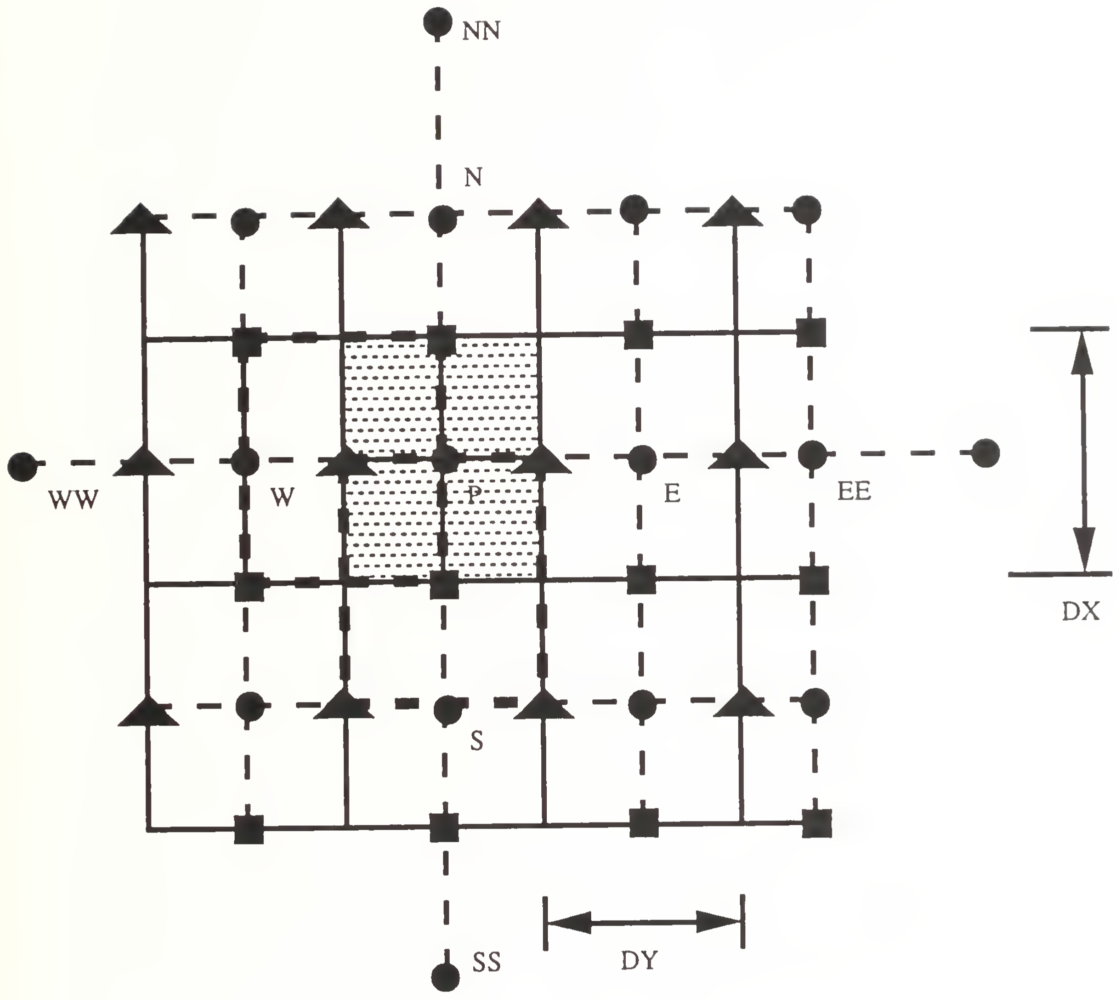


Figure 2.1

The staggered grid,which is one of the cornerstones of the control volume method is shown. The non-staggered control volume is bounded by plain lines. The dotted lines pass through the grid points. The non-staggered grid point which is the location of the temperature and pressure is represented by circular dots. The u-velocity grid points are represented by triangular dots. Similarly, the v-velocity grid points are represented by square dots. The grid in the third direction is analogous.

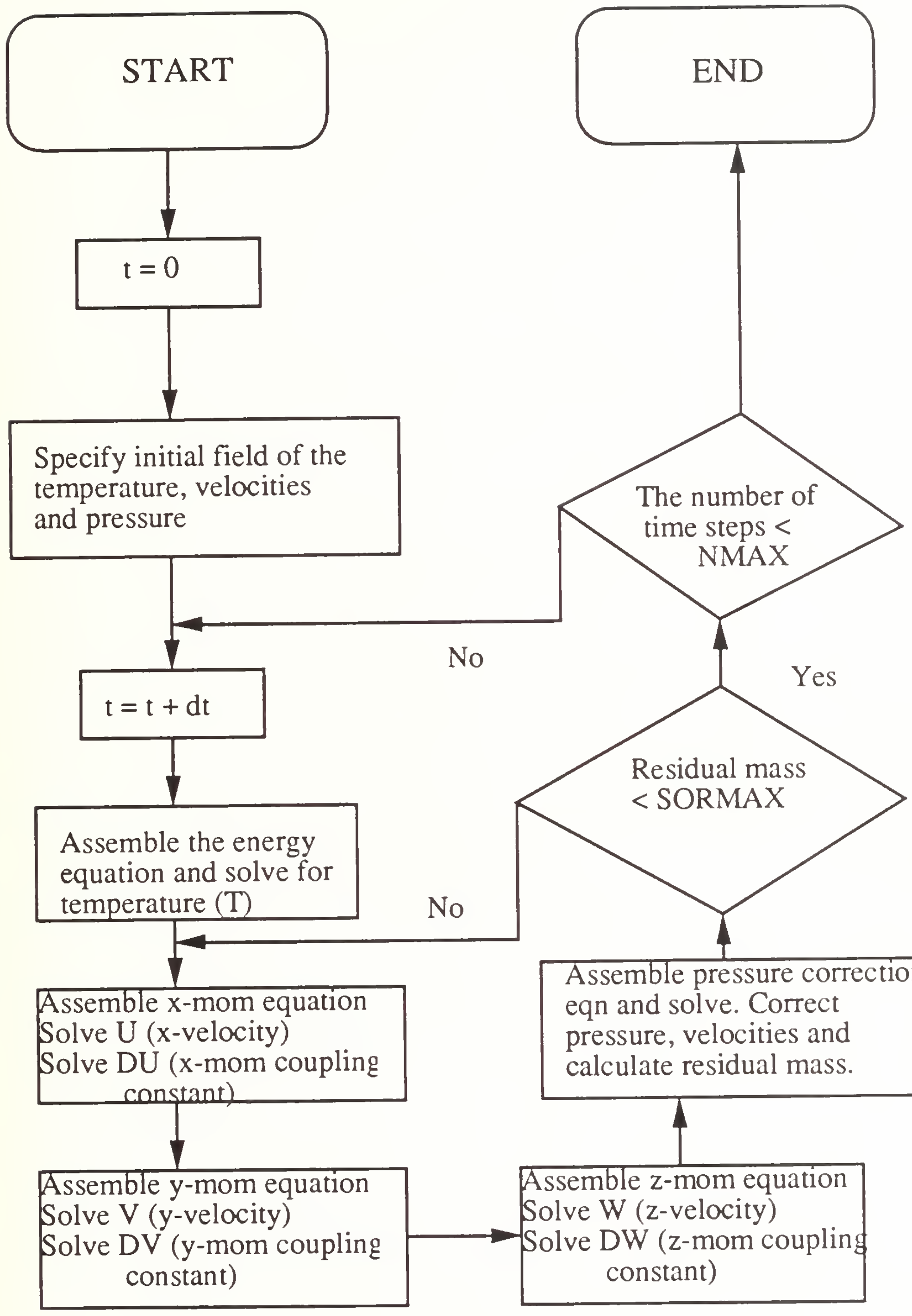

START
END
t = 0
Specify initial field of the temperature, velocities and pressure
The number of time steps < NMAX
No
Yes
t = t + dt
Residual mass < SORMAX
Assemble the energy equation and solve for temperature (T)
No
Assemble pressure correction eqn and solve. Correct pressure, velocities and calculate residual mass.
Assemble x-mom equation Solve U (x-velocity) Solve DU (x-mom coupling constant)
Assemble y-mom equation Solve V (y-velocity) Solve DV (y-mom coupling constant)
Assemble z-mom equation Solve W (z-velocity) Solve DW (z-mom coupling constant)

Figure 2.2

The coupled heat transfer and fluid flow problem is solved by an iterative algorithm known as SIMPLEX. Iterations are necessary, since the problem is non-linear. The algebraic equations are linearized at every iteration step, and are approximate. Each iteration provides an improvement on this approximation. After many iterations, a converged and accurate solution of the non-linear problem is obtained. In contrast, if only conduction is solved for, no iterations are needed since the equations of conduction are linear.

EXAMPLES

3.1 General Remarks

Some specific examples are discussed in this chapter. This serves as a validation for the code for the given applications, namely cooling of a three by three array of chips mounted on a substrate as well as convection without chip or substrate.

3.2 Convection in the absence of Substrate and Chip

The boundary conditions are the following:

1. Top wall cooled and bottom wall heated. THOT = 0.5, TCOOL = - 0.5.
2. All side walls insulated.

The parameters as initialized in BLOCK DATA are the following:

```
DATA NIP2, NIP1, NI,NIM1/37,36,35,34/
DATA NJP2,NJP1,NJ,NJM1/33,32,31,30/
DATA NKP2,NKP1,NK,NKM1/27,26,25,24/
DATA NNMAX,NMAX,IMAX,ITMAX,KRUN,NPRINT/29952,5000,10,5,1,100/
DATA SMALL,EPS,SORMAX/1.0E20,1.0E-8,2.0/
DATA ISUB,ICHIP,JCHIP,KCHIP,NCHP,ICHOICE/2,2,2,2,0,0/
DATA XBR,YBR,ZBR/0.40,0.13,0.30/
DATA H,WTH,BTH/12.3,14.6,25.0/
DATA HCHIP,WCHIP,BCHIP,BSUB/24.0,8.0,6.0,19.5/
DATA ALC,ALS,THOT,TCOOL,TAVG/2823.0,3.5,0.5,- 0.5,19./
DATA RHS,RHC/1.20,0.262/
DATA QQQ,DTIME,RA ,PR,XPER,ROLL/1.5,1.0E-5,1.0E7,130.,2.0,2.0/
DATA IUNFRM/0/
DATA QUICK/1.0/
DATA NJCHIP,NKCHIP/3,3/
DATA YCHIP,ZCHIP/38.0,76.0,114.0,50.8,101.6,152.4/
```

Note that the number of prescribed parameters are in excess of what is required. For example, when NCHP is set to 0 the chip parameters such as ISUB, HCHIP etc are not required and can be set quite arbitrarily.

In figure 3.2, the numerical results are compared with experiments. Both the numerical and experimental results are for a natural convection problem with no substrate or chip in a three-dimensional rectangular enclosure. The heated bottom wall provides the driving force for the flow. Figure 3.1 provides a schematic diagram of the enclosure geometry. Figure 3.2 shows the velocity field across a vertical mid-section as shown in figure 3.1. The results from four different cases are compared with the experiments due to Arroyo and Saviron (1992).

The velocity vectors on the right side are the experiments and on the left are the numerical computations. Each pair corresponds to a particular Rayleigh number. The BLOCK DATA has been initialized for a Rayleigh number of 44,744. The Rayleigh numbers for the five cases starting from the top are the following: 6319.0, 11101.0, 22884.0, 44744.0 and 66604.0. For high Rayleigh number computations, it is generally sound practice to begin at a lower Rayleigh number and work ones way upwards. In other words, use a lower Rayleigh number as an initial condition to start a higher Rayleigh number calculation. Also, note that frequent restarts are necessary even for a particular Rayleigh number since steady-state may not be achieved in a single run.

Figure 3.3 compares the computations (on the left) with the same set of experiments for the first three cases (i.e., 6319.0, 11101.0 and 22884.0). What is now compared are the same set of velocities. However, the data is displayed in terms of pathlines. Since, pathlines can be observed by very standard flow visualization techniques, it is a very useful thing to compare. The degree of match between the two is striking. Also note that the code does not need to be modified. It must be emphasized, that such a good comparison between experiments and numerics are very rare. It is not often that such a comparison is even attempted for three-dimensional flows. Thus, the results do demonstrate the essential correctness of the code.

3.2 Convection with Substrate and Chip

The boundary conditions are the following:

1. The top and bottom wall are isothermal heat sinks. This case is the mathematical modelling of the case shown in figure 1.2. THOT = 0.0, TCOOL = 0.0.
2. All side walls insulated.

The parameters as initialized in BLOCK DATA are the following:

```
DATA NIP2, NIP1, NI,NIM1/21,20,19,18/
DATA NJP2,NJP1,NJ,NJM1/39,38,37,36/
DATA NKP2,NKP1,NK,NKM1/66,65,64,63/
DATA NNMAX,NMAX,IMAX,ITMAX,KRUN,NPRINT/49400,4000,10,5,1,100/
DATA SMALL,EPS,SORMAX/1.0E20,1.0E-8,2.0/
DATA  ISUB,ICHIP,JCHIP,KCHIP,NCHP,ICHOICE/4,4,4,3,1,1/
DATA XBR,YBR,ZBR/0.75,1.00,1.00/
DATA H,WTH,BTH/152.,203.2,49.5/
DATA  HCHIP,WCHIP,BCHIP,BSUB/24.0,8.0,6.0,19.5/
DATA ALC,ALS,THOT,TCOOL,TAVG/3338.0,2.0,0.0,0.0,19.0/
DATA  QQQ,DTIME,PR,RA,XPER,ROLL/1.5,2.0E-8,200.0,1.0E9,0.0,0.0/
DATA IUNFRM/0/
DATA QUICK/1.0/
DATA NJCHIP,NKCHIP/3,3/
DATA  YCHIP,ZCHIP/38.0,76.0,114.0,50.8,101.6,152.4/
```

This corresponds to a case with a three-by-three array of chips that are configured uniformly and symmetrically on one of the vertical sidewalls that includes the substrate as well. The run is for a fairly fine grid (20×38×65). The fluid is the Fluorinert liquid FC71. The viscosity is assumed to be a strong function of temperature. The power dissipation per chip is 1.5W. The flow field for this set of computations is highly three-dimensional and unsteady.

The dimensions of chips and enclosure are shown in figure 3.4. Figure 3.5 shows some of the preliminary data that have been obtained from such a study. For a FC75 fluid it can be seen in figure 3.5(a) that the heat lost to the substrate by conduction from the chip decreases with an increase in the Rayleigh number. This is not surprising, since high Rayleigh number flows will be more convection dominated. For higher Rayleigh numbers an increasing proportion of the heat is transferred directly to the fluid. The substrate is made of plexiglas. Things would of course be different if the substrate was a good conductor such as ceramic.

Figure 3.5(b) shows the chip temperatures for the same fluid and same geometry as a function of the Rayleigh number. It is interesting to note that a straight line is obtained in the log-log scale. It is therefore quite likely that a simple correlation can be deduced which can later be applied for design purposes.

Figure 3.6 compares the experimentally obtained pathlines due to Joshi et al. (1990) with the computations using AMPHIB for the case initialized in BLOCK DATA. The power level is 1.5 Watts per chip and the fluid is FC71. The viscosity varies by more than a factor of 10. This is because the viscosity is a strong function of the temperature for the given fluid. One observes a relatively stagnant flow near the bottom and vertical plumes along the chips. A coarser grid would not suffice for this problem, since the thin boundary layer along the chips and the walls need to be resolved. The results, do indicate that AMPHIB could be usefully employed as a design tool. However, powerful computational resources are required. A number cruncher of the class of a RS/6000 workstation is a must for realistic three-dimensional calculations such as these.

3.4 Convection without Chip or Substrate: Benchmark Study

This example is set up as a benchmark. The boundary conditions are the following:
1. Top wall cooled and bottom wall heated. THOT = 0.5, TCOOL = - 0.5.
2. All side walls insulated.
The parameters as initialized in BLOCK DATA are the following:

```
DATA NIP2, NIP1, NI,NIM1/13,12,11,10/
DATA NJP2,NJP1,NJ,NJM1/13,12,11,10/
DATA NKP2,NKP1,NK,NKM1/13,12,11,10/
DATA NNMAX,NMAX,IMAX,ITMAX,KRUN,NPRINT/1728,500,10,5,0,100/
DATA SMALL,EPS,SORMAX/1.0E20,1.0E-8,2.0/
DATA ISUB,ICHIP,JCHIP,KCHIP,NCHP,ICHOICE/2,2,2,2,0,0/
DATA XBR,YBR,ZBR/0.75,1.00,1.00/
DATA H,WTH,BTH/100.,100.,100./
DATA HCHIP,WCHIP,BCHIP,BSUB/24.0,8.0,6.0,19.5/
DATA ALC,ALS,THOT,TCOOL,TAVG/3338.0,2.0,0.5,-0.5,25.0/
DATA QQQ,DTIME,PR,RA,XPER,ROLL/1.5,2.0E-3,2.5,2.5E4,0.0,0.0/
DATA IUNFRM/1/
DATA QUICK/1.0/
DATA NJCHIP,NKCHIP/3,3/
DATA YCHIP,ZCHIP/38.0,76.0,114.0,50.8,101.6,152.4/
```

After 500 time steps, at the end of the computations, the heat and mass balance statistics looked like the following:

```
NT =     500               TIME =  1.0000
ITER= 1                    SOURCE= 0.001057         SORSUM = 0.000000
NUC = 2.117941             NUH=2.117934
QCHIP = 0.00000            QALL= - 0.000007
QUNS = - 0.001424          QCRT = 0.000002
```

QUNS is relative degree of unsteadiness in the temperature. For steady-state to be reached, QUNS should be about 10^{-6} of NUC.

This doesn't correspond to any experimentally observed case. It is nevertheless a useful benchmark. This case doesn't take much computer time. It runs for about 221 seconds in the CONVEX C240, 90.2 seconds in the AMDAHL 5990/500, and about 66.2 seconds in the CRAY X-MP/216. If large scale modifications have been made in the style and content of the code it will be useful to run the benchmark and check to see if the numbers are unchanged in the output and as such ensures that the changes have been free of errors. It must be emphasized that the numbers after modifications must be unchanged. It is our experience that relatively serious bugs cause minor changes in the output.

Note that the inputs ROLL and XPER will not be described in the next chapter. They are basically not relevant to computations in liquid cooling. XPER must be set to 0. and ROLL can be set arbitrarily.

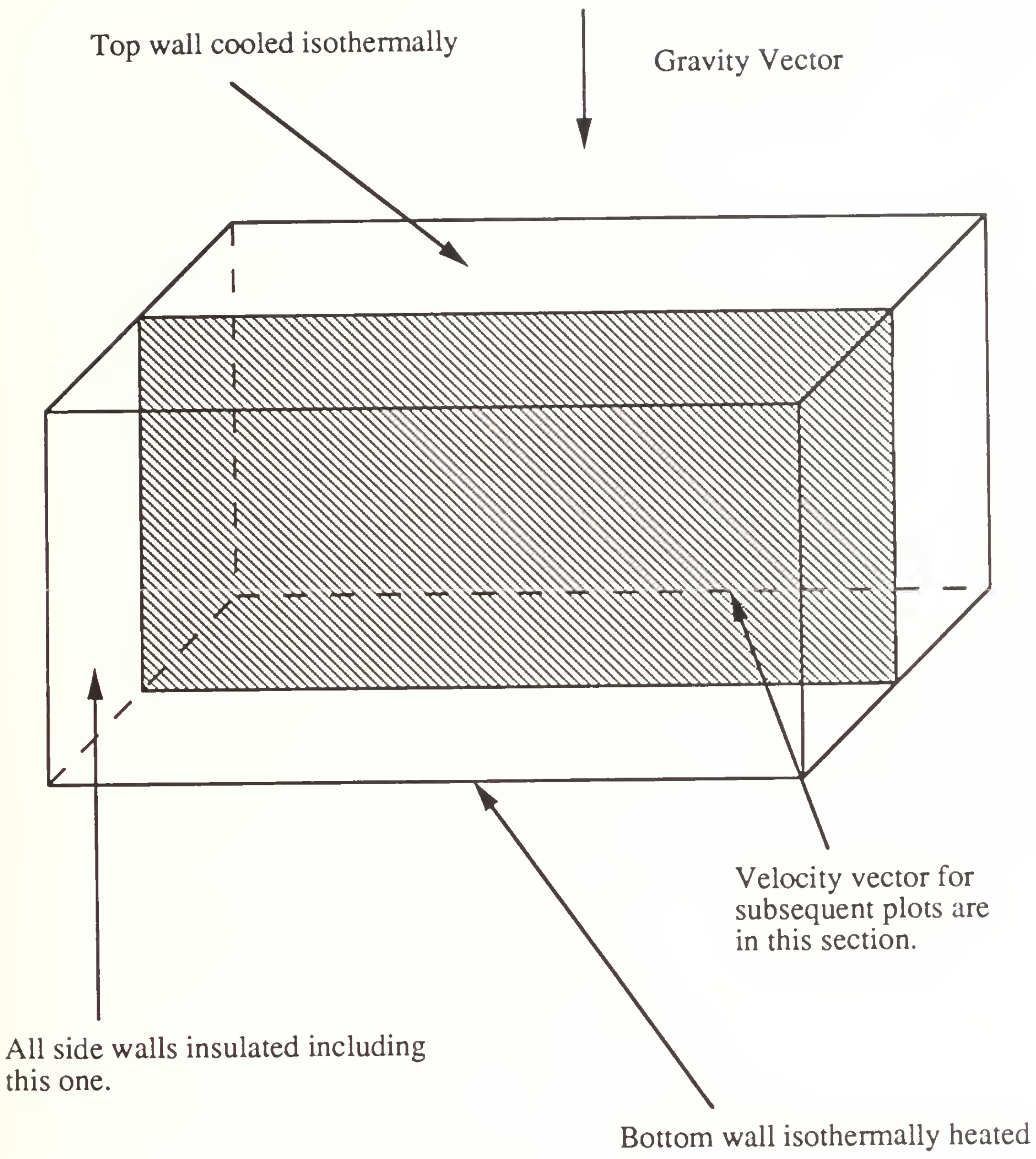

Top wall cooled isothermally
Gravity Vector
Velocity vector for subsequent plots are in this section.
All side walls insulated including this one.
Bottom wall isothermally heated

Figure 3.1

This is an illustration of a problem with a simple geometry but complicated physics, i.e, Rayleigh-Bénard Convection. The driving force is basically the hot bottom wall. The top wall is the heat sink. This is a very well researched and well documented area in literature. That is why one of the example problem compares the computations of AMPHIB with some well established experimental results. The point of comparison is the velocity field at the mid-section indicated by the shaded area. The velocity field in the enclosure is completely three-dimensional.

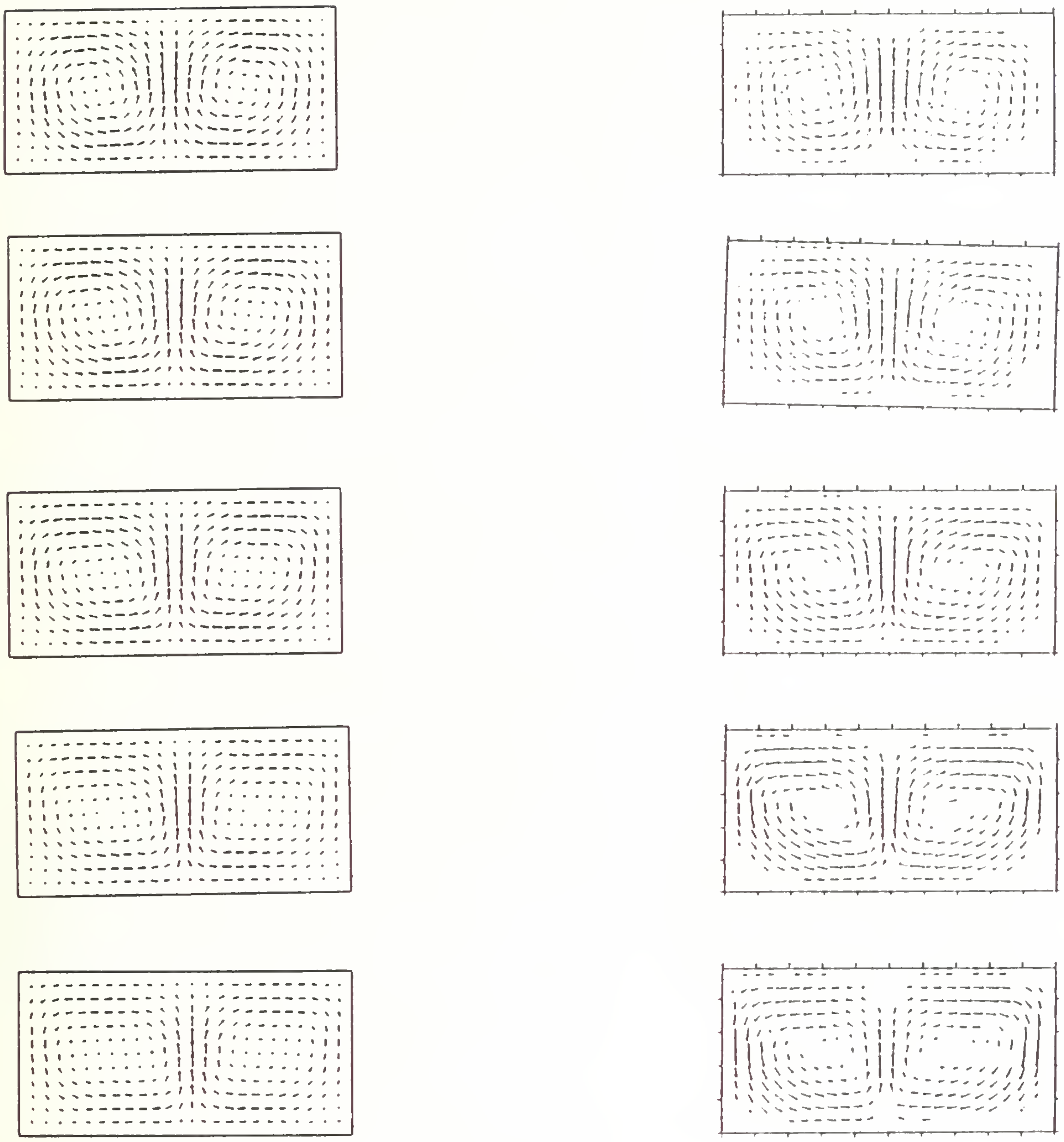


Figure 3.2

The computed and experimental velocity fields are compared. The computed velocity fields are on the left. The experimental ones due to Arroyo and Saviron (1992) are on the right. The box is 25mm wide, 14.6mm broad and 12.3 mm high. The fluid is silicone oil (Pr = 130). From top to bottom, the Rayleigh numbers are respectively 6319.0, 11101.0, 22884.0, 44744.0 and 66604.0. Direct comparison between experiments and simulations for three-dimensional flows are extremely rare. This is, to the best of our knowledge, the first of its kind.

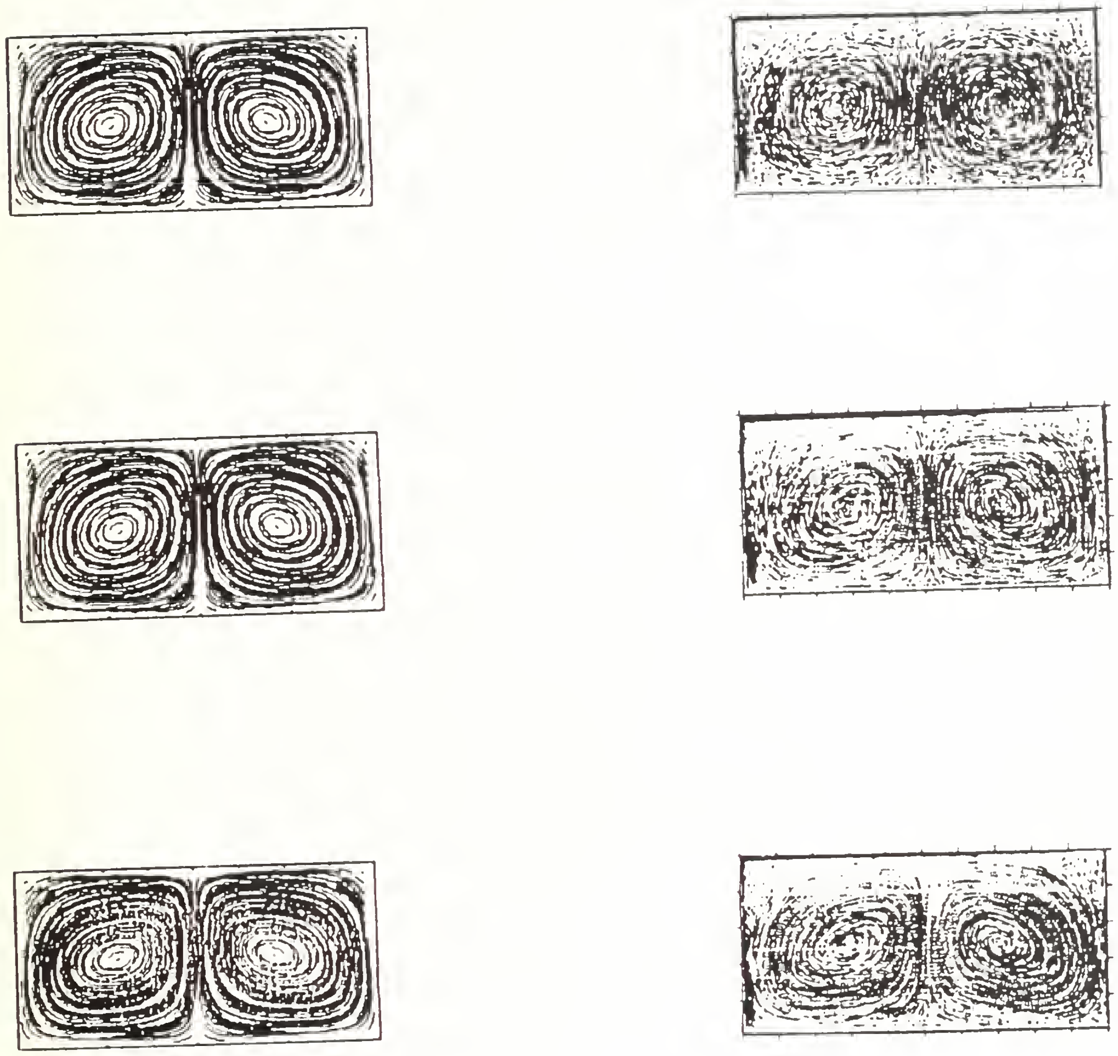


Figure 3.3

The computed and experimental velocities are compared for the same section as in figure 3.2. What is however being compared are the pathlines. There is a distinct advantage in comparing pathlines, since they can be observed in experiments directly. On the other hand, velocity vectors have to be deduced from time-lapse photographs in experiments and are more difficult to do. The experiments are on the right, and the simulations are on the left. The Rayleigh numbers starting from the top are 6319.0,11101.0 and 22884.0. The degree of agreement between the two is striking.

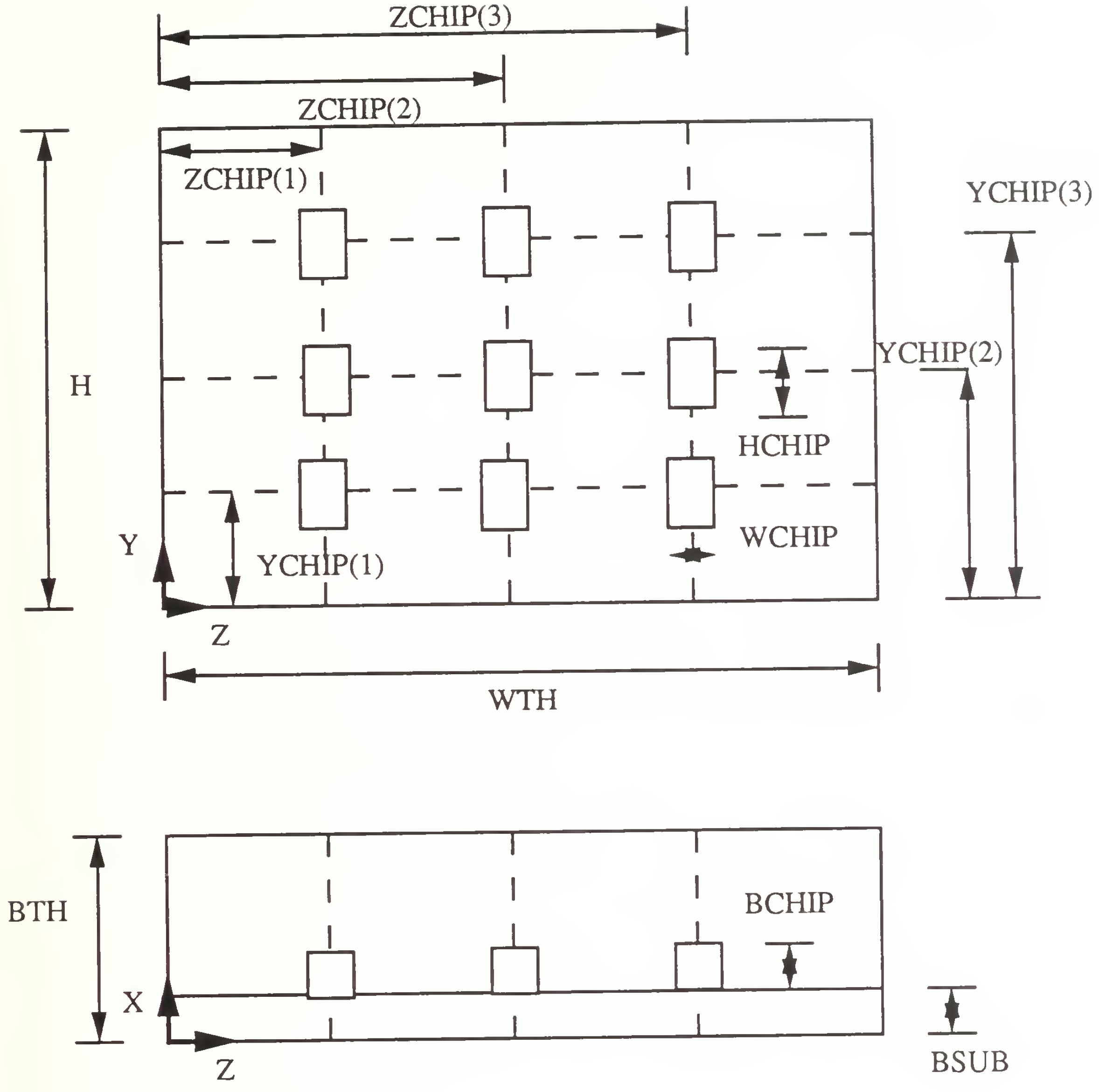
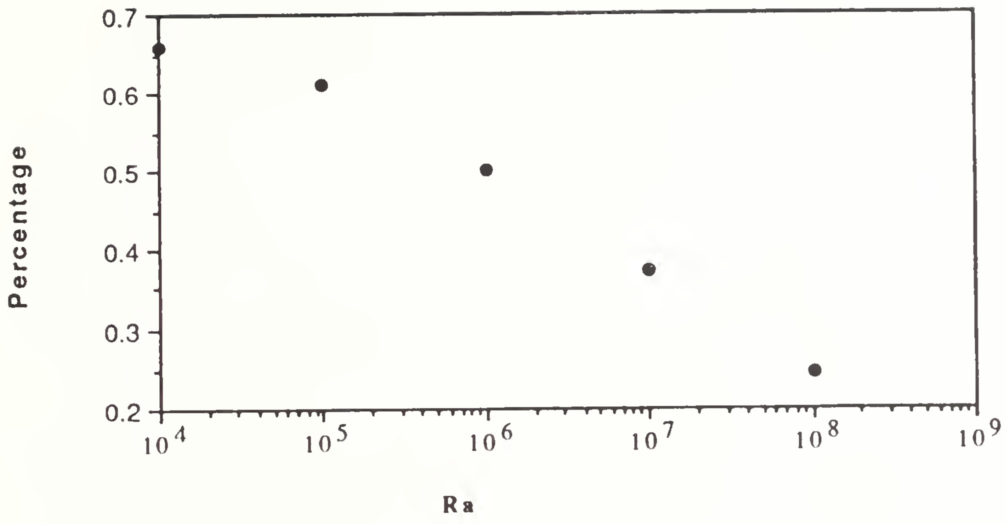


Figure 3.4

The exact dimensions of the example with chip and substrate are shown. This is the problem that was shown in figure 1.2. The dimensions are specified in terms of the parameters that are initialized in BLOCK DATA of AMPHIB. The dimensions in terms of millimeters are the following: H = 152., WTH = 203.2, BTH = 49.5, HCHIP = 24.0, BCHIP = 6.0, WCHIP = 8.0, BSUB = 19.5, YCHIP(1) = 38.0, YCHIP(2) = 76.0, YCHIP(3) = 114.0, ZCHIP(1) = 50.8, ZCHIP(2) = 101.6 and ZCHIP(3) = 152.4.



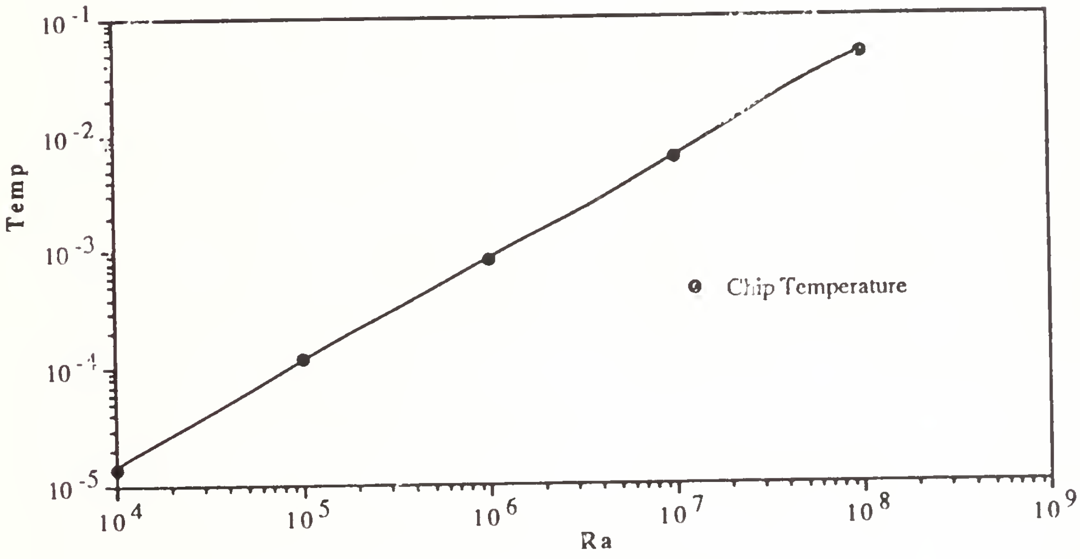


Figure 3.5

The figure at the top plots the percentage heat loss to the substrate as a function of the Rayleigh number. The Rayleigh number is directly proportional to the heat dissipated from the chip. Thus, the plot shows that as the wattage of the chip is increased, more heat as a percentage is lost to the fluid directly. The figure at the bottom compares the average chip temperature as a function of the Rayleigh number. The linear relationship between the two in the log-log scale is interesting. The computations carried out with AMPHIB was for the geometry shown in figure 3.4 for an FC75 fluid (Pr = 30).

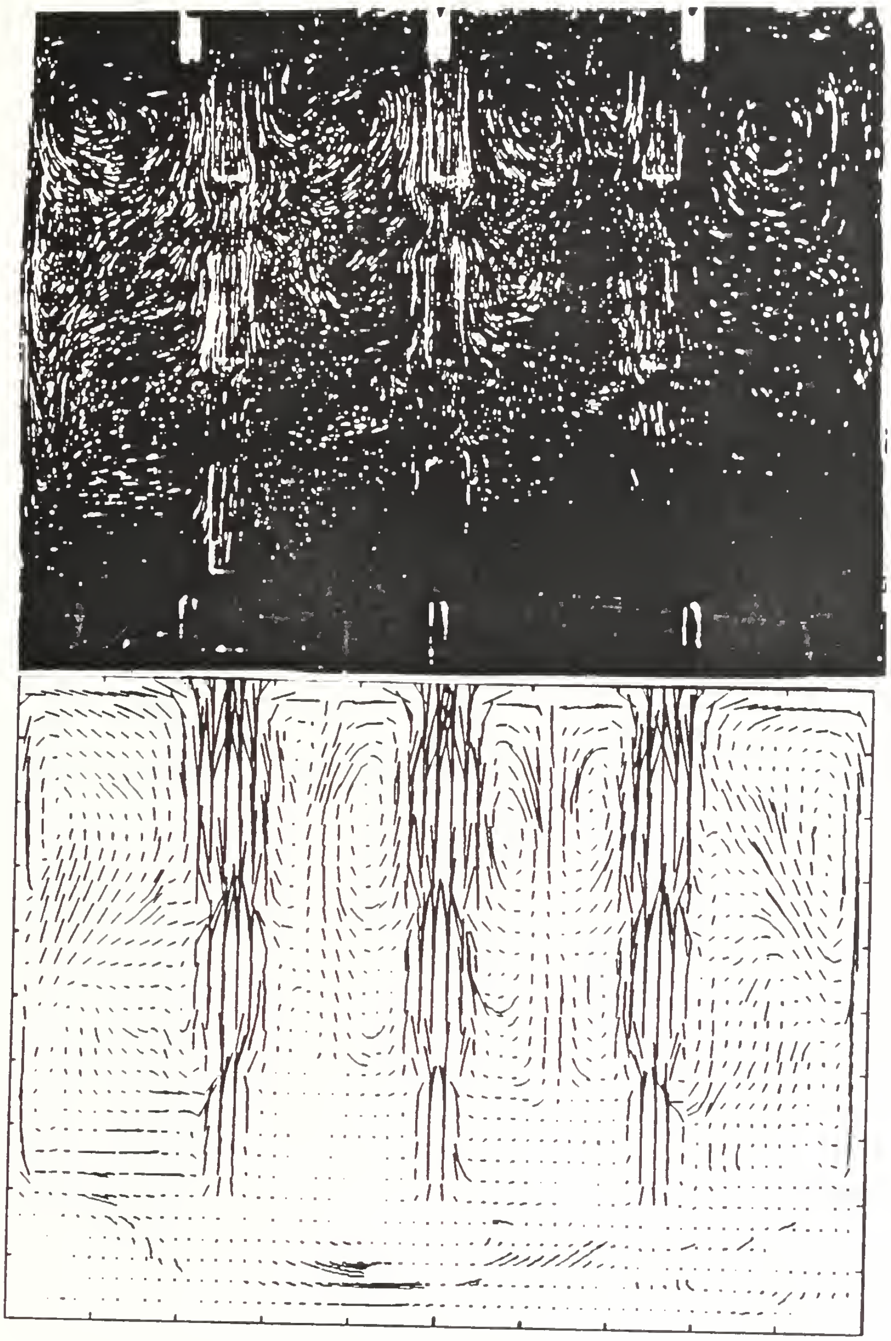
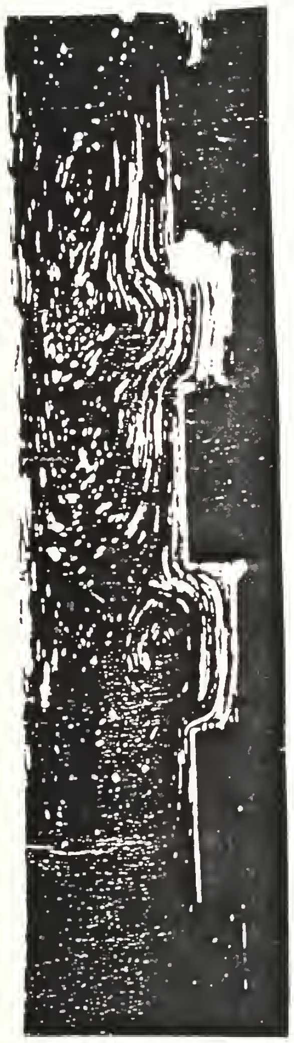


Figure 3.6

The figure at the top represents flow visualization results for an FC71 fluid with a power dissipation level of 1.5W per chip. The viscosity varies dramatically over an order of magnitude. The enclosure geometry is the one shown in figure 3.4. The results are for a section parallel to the substrate at a distance of 0.5 mm from the chip surfaces. The experiments are due to Joshi et. al (1991). The computed pathlines for the same set of parameters are shown at the bottom figure. The computations are for a fairly fine grid size of 65x38x20 and incorporates the sharp changes in temperature dependent viscosity. The observed minor differences in the flow pattern are because the flow is unsteady and the exact time instant has not been matched. One observes a relatively stagnant flow near the bottom, vertical plumes along the chips and recirculating zones between the plumes for both the experiments and computations.



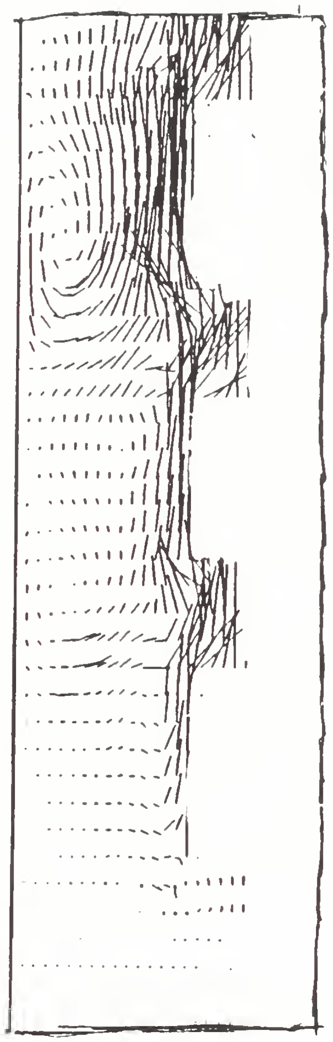


Figure 3.7

The same case as seen in figure 3.6. The flow visualization results are on the left, the computational results are on the right. The results seem to agree even in some of the finer details. There is a recirculating zone in the upper left corner. There is also a weak eddy between the bottom and the middle chip. Flow is noticably stronger in the upper regions. The section in question is along a vertical plane perpendicular to the substrate at the midplane.

INPUTS AND SUBROUTINES

4.1 Inputs to the Program

All the input variables are initialized in BLOCK DATA. The inputs to the problem are described in the order of their appearance in the code.

NIP1:	The number of control volumes spanning the x-direction for the non-staggered grid. The number of control volumes in the computational domain will be two less since one control volume on each end is used to impose boundary conditions.
NIP2:	It is set to NIP1 + 1.
NI:	It is set to NIP1 - 1.
NIM1:	It is set to NIP1 - 2. It is exactly equal to the number of control volumes in the computational domain. The number of control volumes in a realistic simulation depends strongly on the Rayleigh number. The higher the Rayleigh number the greater the number of grid points required. A useful rule of the thumb is that the average resolution for natural convection problems should at least 0.1 (non-dimensional) in the horizontal directions (x and z) and at least 0.05 in the vertical direction. Typically for a 0.3:1:1.3 geometry NIM1 is set to 10. Thus, NI, NIP1 and NIP2 are respectively 11, 12 and 13.
NJP1:	The number of control volumes spanning the y-direction for the non-staggered grid. The number of control volumes in the computational domain will be two less since one control volume on each end is used to impose boundary conditions. NJP1 is set to 22 according to our prescription.
NJP2:	It is set to NJP1 + 1.
NJ:	It is set to NJP1 - 1.
NJM1:	It is set to NJP1 - 2.
NKP1:	The number of control volumes spanning the z-direction for the non-staggered grid. The number of control volumes in the computational domain

will be two less since one control volume on each end is used to impose boundary conditions. It is set to 32 for the geometry discussed.

NKP2:	It is set to NKP1 + 1.
NK:	It is set to NKP1 - 1.
NKM1:	It is set to NKP1 - 2.
NNMAX:	Total number of variables when the matrix is assembled. NNMAX is the product of NIP1, NJP1 and NKP1 and must be initialized accordingly.
NMAX:	Prescribed number of time steps per run. There is no convergence criterion to decide whether steady state is reached, since the algorithm is essentially integrating forward in time. NMAX is basically the only control on the length of a run. An NMAX of 1000 is typical. In practice a complete run is never achieved by a single run and a number of restarts are required.
IMAX:	The maximum number of iterations in the iterative solver, i.e., even if the residual error criterion is not met the number of iterations does not exceed IMAX. IMAX has been set to 10.
ITMAX:	Maximum number of iterations in the pressure loop. Again, if the mass flux criterion is not met, the number of iterations cannot exceed ITMAX. ITMAX is set to 5. If the number of iterations consistently exceed ITMAX, it generally means that the time step DTIME must be reduced.
KRUN:	A flag to determine if the job is a restart or a new run. If KRUN is set to 0 computations start from scratch. Otherwise, an input file containing the field variables is read in to continue the computations if KRUN is set to 1.
NPRINT:	The field variables are written to a file every NPRINT time steps. Also, statistics for the overall mass and energy balance are also printed every NPRINT time steps. This is a checkpointing feature to ensure that the computations are not wasted if the program terminates abnormally. It is typically set to 100.
SMALL:	A variable which is set to an arbitrary high value to force the velocities in the conduction regions to a value very close to zero. Typically SMALL is set to 10^{20}.
EPS:	Minimum prescribed residual in the iterative solver. In the subroutine SIP (the iterative solver) the prescribed square of the error norm is calculated from EPS. i.e., $\|\| \mathbf{Ax} - \mathbf{b} \|\|^2 \leq N \times EPS^2$, where N is the number of variables and the matrix equation is $\mathbf{Ax} = \mathbf{b}$. A value of 10^{-8} with all variables in double precision is quite typical. The error norm is normalized

with the norm of the variable unless the norm is small (less than 10^{-4}) in which case the error norm is absolute and not relative. This is automatically taken care of in the linear solver.

SORMAX: The maximum permissible absolute sum of all mass-fluxes from all control volumes at each time step. This parameter controls the number of iterations in the pressure loop. In other words, if the mass fluxes exceed SORMAX the outer pressure loop is traversed again until the flux reduces to a level below SORMAX. For a run where the final flow field is expected to be steady, the value of SORMAX can be fixed quite arbitrarily since at convergence the mass fluxes reduce to an arbitrarily small value. For unsteady runs (e.g., oscillatory flow) the level must be chosen with care. As the grid size is reduced, SORMAX is increased. SORMAX must not exceed 2.0.

ISUB: The number of control volumes spanning the substrate in the x-direction. It should be at least 2.

ICHIP: The number of control volumes spanning each chip in the x-direction.

JCHIP: The number of control volumes spanning each chip in the y-direction.

KCHIP: The number of control volumes spanning each chip in the z-direction. At the very least, ICHIP, JCHIP and KCHIP should be set to 2.

NCHP: A flag when set to 0 solves an enclosure problem without chips or substrate. When NCHP is set to 1 the conjugate heat transfer problem is solved which now includes conduction in the substrate and chip as well as fluid convection.

ICHOICE: It is a parameter which determines the fluid to be simulated. If ICHOICE is set to 0, the fluid has a constant viscosity. If ICHOICE is set to 1, the fluid is FC75 and if ICHOICE is set to 2, the fluid is FC71. The user can simulate any fluid provided the kinematic viscosity as a function of the temperature is provided. The changes are to be made in the routines PROP and CALT.

XBR: This parameter is required for generating the grid. It is the dimension of the smallest control volume in mm next to the wall in the x-direction. The smallest grid is always next to the wall in order to resolve the boundary layer. Typically, it should be set at a value that is one percent of the enclosure dimension in that direction which is BTH in this case.

YBR: The smallest grid size in the y-direction in mm. Again, one-hundredth of H is suggested. However, for a general case, sensitivity to these

	parameters should be tested.
ZBR:	The smallest grid size in the z-direction in mm. A similar prescription holds.
H:	The height of the enclosure in mm. This is the y-direction.
WTH:	The width of the enclosure in mm. This is the z-direction.
BTH:	The length of the enclosure in mm. This is the x-direction.
HCHIP:	The dimension of an individual chip in the y-direction in mm.
WCHIP:	The dimension of an individual chip in the z-direction in mm.
BCHIP:	The dimension of an individual chip in the x-direction in mm.
BSUB:	The thickness of the substrate in mm (x-direction).
ALC:	Thermal conductivity ratio between chip and fluid.
ALS:	Thermal conductivity ratio between substrate and fluid.
THOT:	The non-dimensional temperature at the bottom wall.
TCOOL:	The non-dimensional temperature at the top wall.
TAVG:	The reference temperature in °C. This is required to evaluate the viscosity as a function of the temperature. In the case when the top and bottom temperatures are equal, THOT and TCOOL is set to zero, the TAVG is set equal to the temperature of the top and bottom walls.
RHC:	The heat capacity (product of density and specific heat) ratio between chip and fluid.
RHS:	The heat capacity (product of density and specific heat) ratio between substrate and fluid.
QQQ:	This is the heat dissipated per chip in Watts.
DTIME:	Non-dimensional time step. The explicit time step for an upwind scheme is calculated by the routine TSTEP. Since the scheme used for the program is QUICK, it is recommended that the time step is about a third of the time step calculated.
PR:	Prandtl number of the fluid. This is a non-dimensional property of the fluid defined as $\frac{\nu}{\alpha}$, where ν is the kinematic viscosity and α is the thermal diffusivity.
RA:	The product of the Prandtl number and the Rayleigh number which is $\frac{g\beta q H^5}{\alpha^2 \kappa_c}$, where g is the acceleration due to gravity; β is the coefficient of volume expansion; L is the length scale (height of the enclosure in this case); and κ_c is the thermal conductivity of the chip; q is the volumetric rate of heat generation for each chip. This must be specified only if the it is a

	pure enclosure problem (when NCHP = 0). Otherwise, the product of the Rayleigh and Prandtl numbers are automatically calculated using QQQ.
IUNFRM:	This parameter is used in the grid. When set to 0, a non-uniform grid is generated. Otherwise, a uniform grid is created.
QUICK:	It is a number between 0 and 1. If QUICK is set to 0. the upwind scheme is used. If QUICK is set to 1. the quick scheme is used. Any number between the two is a weighted average of the two schemes. It is recommended that for initial runs especially for high Rayleigh numbers (greater than 10^7) QUICK be set to 0. After a converged solution is got QUICK should be set to 1. However, if there are convergence problems QUICK should be set to zero again. A failure to converge to a steady state (when experiments clearly indicate steady-state) implies that the number of grid points are insufficient. Therefore, NI etc must be increased.
NJCHIP:	The number of chips in the vertical y-direction.
NKCHIP:	The number of chips in the vertical z-direction.
YCHIP:	A one-dimensional array of length NJCHIP. These are the locations of the chip centers in the y-direction.
ZCHIP:	A one-dimensional array of length NKCHIP. These are the locations of the chip centers in the z-direction.

The schematic of a typical geometry that is solved is shown in figure 1.1. The substrate and the chip protrusions are shaded. In terms of the variables in the program, the geometry is marked in figure 3.4 as an example for a 3 by 3 three array. In the most general situation, the protrusions are located in a non-uniformly spaced NJCHIP by NKCHIP array. All protrusions have the same dimensions, which is HCHIP$\times$ WCHIP$\times$ BCHIP.

4.2 Subroutines

PROGRAM AMPHIB:

This is the main program. It consists of five subroutine calls in sequence. The first subroutine call is for the subroutine OPENF.

SUBROUTINE OPENF

This subroutine opens the files that handle the output and input for the code. The files are the following:

UNIT 8:	This is the input data for the field variables (tod(i,j,k), uod(i,j,k),vod(i,j,k), wod(i,j,k) and pod(i,j,k)) i.e., the initial temperature, velocities and pressure. The field variables are read in only if KRUN is set to 1.
UNIT 10:	This is an output file of the u,v and w velocities at a specific location for all the time steps. This was used to gauge the dynamical behavior of the system i.e., whether it is steady-state, oscillatory etc.
UNIT 11:	This is output file for the field variables updated at specific instances which is determined by NPRINT. This is a checkpointing procedure.
UNIT 12:	This is the output file for the mean field variables over the entire run, i.e., over NMAX time steps.
UNIT 13:	This is the file where the grid for the given run is stored.

SUBROUTINE GRID:

This subroutine generates the three-dimensional grid for the problem. The subroutine non-dimensionalizes the dimensions first in terms of the height of the cavity, h. It is therefore sufficient to input the data in consistent units (say inches) rather than in millimeters. The subroutine generates a non-uniform grid x of the following type

$$x = \frac{\mathrm{Tanh}\,(HKr)}{K} \qquad (4.1)$$

This is known as the Robert's transformation (Robert, 1970). The grid distribution ensures that the boundary layers are resolved near the wall. At the same time, the grid is spaced as uniformly as possible away from the wall. There are two grid parameters (H and K) that need to be determined. The variable r represents the uniform grid. Equation 4.1 thus maps a uniform grid (r) to a non-uniform one (x).

The grid distribution generated for the NJCHIP by NKCHIP enclosure is a little more complicated. The non-uniform hyperbolic tangent distribution of the grid is confined to a region near the walls. Otherwise, the grid is made as uniform as possible, away from the wall. The same thing applies to the y and z directions. As an example, consider a 3 by 3 array that was portrayed in figure 1.2. Two perpendicular sections of the non-staggered control volumes for a 20×38×65 grid is shown in figure 4.1. The x-y plane is on the left

and the z-y plane is shown on the right. The grid point is located at the centroid of the control volume for the non-staggered grid.

SUBROUTINE GRID1

This subroutine is called by GRID to calculate the grid parameters for the non-uniform grid (H, K). The routine solves transcendental equations by the bisection and the Newton-Raphson method. Note that the users could supply their own grids in which case GRID and GRID1 are not required. The whole program can only handle grids that are rectangular. The non-uniformity is restricted to their respective directions only.

SUBROUTINE PROP

In this routine the conductivity ratios and the heat capacity ratios are assigned for the chip, substrate and fluid. The heat capacity is the product of the specific heat and density. The parameter ICHOICE determines the fluid chosen. In the present program, the choice is limited to FC71 and FC75. Using the properties, the product of the Rayleigh and Prandtl numbers (sometimes referred to as the Boussinesq number) is calculated using QQQ, the heat dissipated per chip. The user will need to modify this subroutine accordingly for a different set of fluids.

SUBROUTINE INITIO

In this subroutine all variables are initialized. If the computations are started from scratch, the field variables are all initialized to zero. For a restart job (KRUN = 1), the field variables are read in from UNIT 8. If KRUN is set to 0, and NCHP set to zero as well, a problem of pure natural convection is solved.

SUBROUTINE PLOOP

This subroutine incorporates the essence of the SIMPLEX algorithm. It also includes an error control routine (Liu, 1979). It calls the following subroutines:

SUBROUTINE CALT

This subroutine calculates the coefficients of the algebraic equation for solving the finite-difference equation for temperature which is of the following form:

$$A_pT_p = A_ET_E + A_WT_W + A_NT_N + A_ST_S + A_ET_E + A_BT_B + A_FT_F + S \tag{4.2}$$

The explicit expressions for the coefficients (i.e., A_P etc) as well as the derivation of the equations from scratch are given in appendix B for the interested reader.

The coefficients must be modified for accommodating the boundary conditions. This is also accomplished in the subroutine. The computational details are available in appendix B.

The interfacial thermal conductivities are calculated in this routine. It uses the harmonic mean formulation as suggested by Patankar (1980). Essentially the conductivities are calculated in the following manner:

$$\kappa_e = \kappa_i \kappa_{i+1} \left(\frac{\Delta x_i + \Delta x_{i+1}}{\Delta x_i \kappa_{i+1} + \Delta x_{i+1} \kappa_i} \right) \qquad (4.3)$$

κ_e is the thermal conductivity of the east face. In this manner, sharp changes in conductivities can be handled accurately. The formulation is necessary since properties vary dramatically in a conjugate heat transfer problem such as this. For instance, in a problem with aluminum protrusions, plexiglas substrate and FC75 liquid the conductivity ratio between aluminum and FC75 is almost 2800. For the heat capacity ratios, the interfacial heat capacities are calculated by linear interpolation.

SUBROUTINE CALU

The finite difference analog of the x-momentum equation is assembled in this subroutine. The equations are linearized and the unknown to be solved is the U-velocity at every grid point. The equation is of the following form:

$$A_P U_P = A_E U_E + A_W U_W + A_N U_N + A_S U_S + A_E U_E + A_B U_B + A_F U_F + S \qquad (4.4)$$

The boundary conditions are also incorporated by suitable modifications of the coefficients. All details are provided for in appendix B. The equation for solving the coupling constants (to be discussed later) of the U-velocity for each grid point is also calculated. The coupling constants will be used later in the equation for the pressure correction.

Similarly, the subroutines CALV and CALW are used to assemble the equations and apply boundary conditions for the V and W velocities respectively. The equation for solving the coupling constants are also assembled in these two subroutines.

In all the velocity routines, the interfacial kinematic viscosity for the fluid region is calculated by a harmonic mean formulation.

SUBROUTINE CALP

The pressure correction equation is derived from the continuity equation and enforces mass balance by correcting the pressure and velocities. The fundamental relationships are the following:

$$U^*_e = U' - du_e\,(P'_E - P'_P) \tag{4.5}$$

$$V^*_n = V' - dv_n\,(P'_N - P'_P) \tag{4.6}$$

$$W^*_f = W' - dw_f\,(P'_F - P'_P) \tag{4.7}$$

The starred quantities are corrected velocities that satisfy the continuity equations. The primed quantities for pressure are the pressure correction. The coupling constants du_e, dv_n, and dw_f were determined respectively in CALU, CALV and CALW. A pressure correction equation similar in form to equation (6) is set up in CALP.

SUBROUTINE NU

This subroutine calculates the average Nusselt numbers at the cold and hot walls and the generation of internal energy. It does an overall heat balance. The mass balance and the heat balance statistics are printed in this subroutine. NU is called by PLOOP every NPRINT time steps.

SUBROUTINE SIP

This subroutine is called by CALU, CALV and CALW twice every time step for solving the generic linear equations that have been assembled (equation 2.2). The first call is to solve the velocity equations and the second call is made to solve for the coupling constants as dictated by the SIMPLEX algorithm. SIP is called by CALT and CALP once to solve the temperature and pressure correction equations respectively. SIP is an iterative solver. It is controlled by two parameters EPS and IMAX. EPS determines the relative error norm. EPS is generally set to 10^{-8} for double precision calculations. Double precision calculations are recommended and are in fact the default (using the IMPLICIT statement in every subroutine).

IMAX is the maximum number of iterations allowed in the iterative solver. IMAX is typically set to a value of 10. The subroutine calls the subroutines RES and XL. RES calculates the error vector i.e., **Ax - b.** XL performs the forward and back substitution for

the incomplete LU decomposition which is an integral part of the solver. The solver is known as Strongly Implicit Procedure and is credited to Stone (1968). The reader can substitute any solver in its place. SIP was found to be the most cost-effective for natural convection problems that were studied. Note that iterative solvers are generally more efficient than direct solvers for three-dimensional problems since the matrices that are assembled are generally speaking ill-conditioned. The condition number (ratio of the largest to the smallest eigen value of the matrix) increases for finer grids. If the smallest eigen value of the matrix is exactly zero, then the matrix is singular.

The input (arguments) to the subroutine are IST, JST, KST, ISP,JSP and KSP and the variable to be solved (the field variable). IST, JST and KST are the starting indices of the computational domain for the x, y and z direction. ISP, JSP and KSP are the last indices for the respective directions.

SUBROUTINE TSTEP

This subroutine calculates the maximum allowable time step for the difference equations of the temperature and the velocities if the scheme were explicit. This information, i.e., the minimum time step is printed every NPRINT time steps. This routine is called by PLOOP. This provides a useful guideline for the time step that is specified by the variable DTIME. The explicit time step is calculated using the CFL criteria and for an upwind interpolation scheme. The criteria for the QUICK scheme will be more stringent. It is therefore recommended that the time step be assigned a value that is between a third or half the computed time step. In this way, numerically induced oscillations can be avoided.

SUBROUTINE CHIPTEMP

This subroutine calculates the average temperature rise of the chips. The temperature is calculated by volume average of temperatures of the control volumes that are contained in the chip. The average temperatures are then suitably scaled to give the temperature rise in degree celsius. The relevant information is then printed. The subroutine is called by PLOOP every NPRINT time steps.

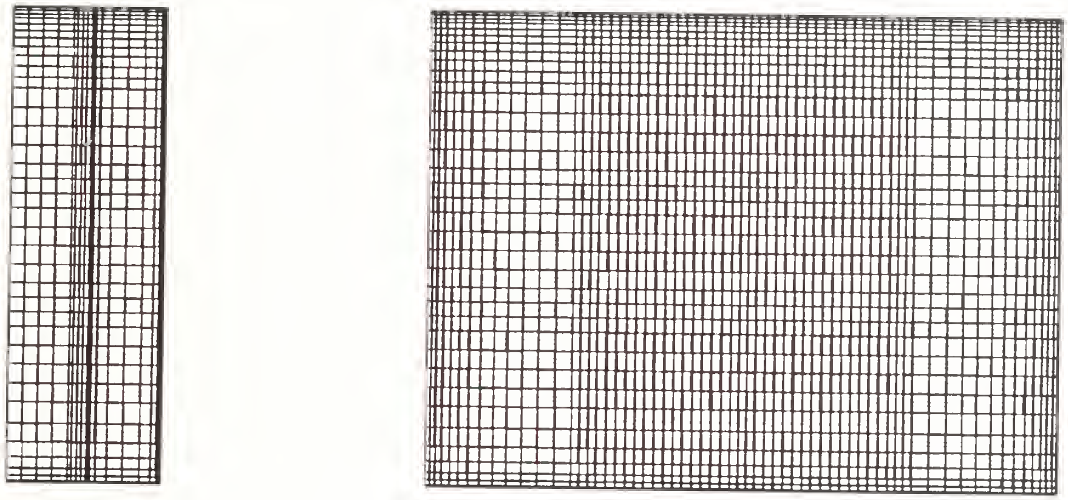


Figure 4.1

The figure on the left and right are two perpendicular views of the grid generated for a fine mesh (20x38x65). The control volume sections of the non-staggered grid points are what is shown. The grid generated corresponds to a 3x3 array shown in figure 3.4. Some of the results produced for the above grid are shown in figures 3.6 and 3.7.

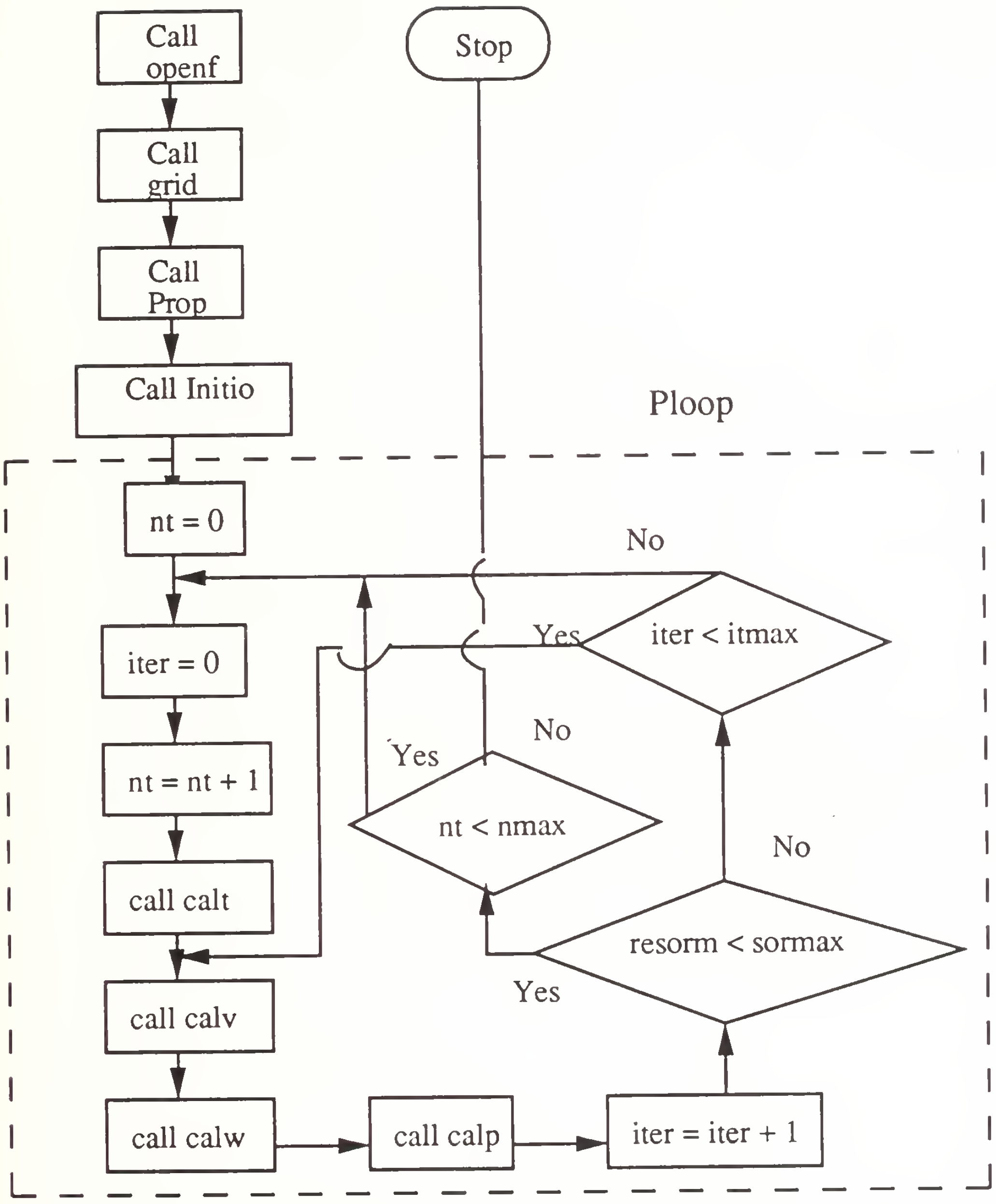


Figure 4.2

This is a schematic flow chart of the subroutines in the code AMPHIB. The main program has five subroutine calls; OPENF, GRID,PROP, INITIO and PLOOP. The subroutine PLOOP is blown up to reveal more details.

APPENDIX A

5.1 Boundary Conditions

5.1.1 The Defaults

The temperature boundary conditions have been set up in the program as follows:

Top wall:	Isothermal (perpendicular to y axis)
Bottom wall:	Isothermal (perpendicular to y axis)
Left wall:	Adiabatic (perpendicular to x axis)
Right wall:	Adiabatic (perpendicular to x axis)
Back wall:	Adiabatic (perpendicular to z axis)
Front wall:	Adiabatic (perpendicular to z axis)

The velocity boundary conditions are all non-slip.

The temperature boundary conditions are imposed in the subroutine CALT. The scope of the program can be expanded by imposing different combinations of boundary conditions. The three commonly used boundary conditions are: (1) Adiabatic (2) Isothermal, (3) Convective or mixed and (4) specified heat flux. Each of these will now be described in detail. The velocity boundary conditions imposed in the program are all non-slip.

5.1.2 Adiabatic Boundary Conditions

This corresponds to the condition of having the heat flux set to zero at the wall. For example, $\frac{\partial T}{\partial x}$ (at the left wall) = 0. The numerical boundary condition corresponding to this for the control volume next to the left wall (i = 2) is $T_W = T_P$. Substituting this condition in the algebraic equation (equation 2.2), the coefficients must be modified in the following manner:

$$A_P = A_P - A_W \text{ and } A_W = 0.$$

For the control volume that is one away from the wall (i = 3), the numerical boundary condition is $T_{WW} = T_W$. This corresponds to $S = T_W A_{WW}$ in the program.

5.1.3 Isothermal Boundary Conditions

This corresponds to the condition where the temperature at the wall is specified. For example, T(at the bottom wall) = THOT . The numerical boundary condition corresponding to this for the control volume next to the bottom wall (j = 2) is $T_S + T_P = 2$ THOT. Substituting this condition in the algebraic equation (equation 2.2), the coefficients must be modified in the following manner:

$$A_P = A_P + A_S \text{ , } S = 2 \text{ THOT } A_S \text{ and } A_S = 0.$$

For the control volume that is one away from the wall (j = 3), the numerical boundary condition is $T_{SS} = 2 \text{ THOT} - T_S$. This corresponds to $S = (2 \text{ THOT} - T_S) A_{WW}$ in the program.

5.1.4 Convective Boundary Condition

This corresponds to the case when the temperature at the wall satisfies a boundary condition of the type $\frac{\partial T}{\partial z}$ + (HCON) T = 0, for example at the back wall. The numerical boundary condition corresponding to this for the control volume next to the bottom wall (k=2) is $\text{HCON}(\frac{T_P + T_B}{2}) + \frac{T_P - T_B}{\Delta z} = 0$. Substituting this condition in the algebraic equation (equation 2.2), the coefficients must be modified in the following manner:

$$A_P = A_P - \text{CONST } A_B \text{ , where CONST} = \frac{\frac{1}{\Delta z} + \frac{H}{2}}{\frac{1}{\Delta z} - \frac{H}{2}}$$

For the control volume that is one away from the wall (k = 3), the numerical boundary condition is T_{BB} = CONST T_B. This corresponds to S = (CONST)T_B A_{BB} in the program.

5.1.4 Heat Flux Boundary Condition

This corresponds to the condition of having the heat flux specified at the wall. For example, $\frac{\partial T}{\partial x}$ (at the left wall) = C. The numerical boundary condition corresponding to this for the control volume next to the left wall (i = 2) is $T_W = T_P$ - Const, where Const =

$C\Delta x$. Substituting this condition in the algebraic equation (eqn 5), the coefficients must be modified in the following manner:

$$A_P = A_P - A_W\ ,\ S = -\ A_W \text{Const} \ \text{ and } A_W = 0.$$

For the control volume that is one away from the wall ($i = 3$), the numerical boundary condition is $T_{WW} = T_W$ - Const. This corresponds to $S = (T_W - \text{Const})\ A_{WW}$ in the program. Note that the adiabatic condition is a special case, when $C = 0$.

5.2 Velocity Boundary Conditions

All velocities i.e., the U, V and W velocities are set to zero (non-slip condition) for all six walls in the program. The most commonly used boundary conditions and their implementation will now be described.

5.2.1 Non-Slip Conditions

The numerical implementation is complicated by the staggered nature of the grid. Consider the left wall as an example. For the u-velocity the grid point coincides with the left wall due to stagger). Hence, in equation 2.2 we simply set $A_W = 0$ (since $U_W = 0$). For the V and W velocities for the control volume next to the wall ($i = 2$ for V and W, and $i = 3$ for U) we have $V_P = -\ V_W$ and $W_P = -\ W_W$. Hence $A_P = A_P + A_W$ and $A_W = 0$ in each of the cases.

For the control volume one away from the wall ($i = 3$ for V and W, and $i = 4$ for U) we have U_{WW} as identically zero. Hence $A_{WW} = 0$ for the U-velocity grid. For the V and W velocities we have $V_W = -\ V_{WW}$ and $W_W = -\ W_{WW}$. Hence, $S = -\ A_{WW}V_W$ and $A_{WW} = 0$ for the V velocity equation and $S = -\ A_{WW}W_W$ and $A_{WW} = 0$ for the W velocity equation.

5.2.2 Slip Wall Conditions

Again consider the left wall as an example. These boundary conditions are employed to generally impose a symmetry. For instance, if the flow consists of two symmetric vortices, a slip wall can be imposed on one of them and calculations can be carried out for only half the computational domain. The slip wall is the numerical implementation of zero shear stress at the wall. For the left wall it would mean that $\frac{\partial V}{\partial x} = 0$

and $\frac{\partial W}{\partial x} = 0$. Also U = 0. The numerical boundary conditions for the U velocity is the same as the non-slip conditions at the wall i.e. $A_W = 0$ (i = 3) and $A_{WW} = 0$ (i = 4). The boundary conditions for the V and W are identical to the adiabatic condition for the temperature i.e., $A_P = A_P - A_W$ and $A_W = 0$ (i = 2) and $S = T_W A_{WW}$ and $A_{WW} = 0$ (i = 3).

5.2.3 Velocity Prescribed

This is a very important boundary condition. Using this one can solve problems in mixed or forced convection as in figure 1.3. As an example, let the left wall have an inlet with a U-velocity of U_0 (consistently non-dimensionalized). Also the V and W velocity are zero. For that portion of the wall, the boundary conditions for the U velocity are $S = U_0 A_W$ and $A_W = 0$ (i = 3) and $S = U_0 A_{WW}$ and $A_{WW} = 0$ (i = 4). The V and W velocities have the usual non-slip boundary condition.

For the control volume next to the wall (i = 2 for V and W, and i = 3 for U), we have $V_P = - V_W$ and $W_P = - W_W$. Hence $A_P = A_P + A_W$ and $A_W = 0$ in each of the cases. For the control volume one away from the wall (i = 3 for V and W), we have $V_W = - V_{WW}$ and $W_W = - W_{WW}$. Hence, $S = - A_{WW} V_W$ and $A_{WW} = 0$ for both.

If there is an inlet there must be an outlet. For the outlet the velocities cannot be prescribed. In fact, only approximate boundary conditions can be used. This is the so called 'natural boundary condition'. For instance, if the outlet is at a portion of the east wall we numerically implement $\frac{\partial U}{\partial x} = 0$, $\frac{\partial V}{\partial x} = 0$ and $\frac{\partial W}{\partial x} = 0$ at the east wall. The numerical implementation of these conditions have already been discussed and are quite straightforward. Basically, this means that we are not sure what the boundary conditions are but assume that they don't change much near the outlet. Overall mass balance is automatically satisfied.

APPENDIX B

6.1 Difference Equations

In this section the finite difference equations are presented for the equations of the velocities and temperatures. The details of the derivation are skipped. Only the final form of the equations are stated. For additional details of the control volume method the reader is referred to Patankar (1980).

6.1.1 Energy Equation and linear equation for Temperature

The QUICK scheme for estimating temperature fluxes at the east and west faces are the following:

$$\begin{aligned} G_eT_e &= G_e\left(\frac{\Delta x_{i+1}T_P + \Delta x_i T_E}{2\Delta x_e}\right) \\ &+ \left(\frac{|G_e| - G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e \Delta x_{ee}}\right)\left(\frac{T_P\Delta x_{ee}}{\Delta x_e + \Delta x_{ee}} + \frac{T_{EE}\Delta x_e}{\Delta x_e + \Delta x_{ee}} - T_E\right) \\ &- \left(\frac{|G_e| + G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e \Delta x_w}\right)\left(\frac{T_E\Delta x_w}{\Delta x_w + \Delta x_e} + \frac{T_W\Delta x_e}{\Delta x_e + \Delta x_w} - T_P\right) \end{aligned} \tag{6.1}$$

$$\begin{aligned} G_wT_w &= G_w\left(\frac{\Delta x_{i-1}T_P + \Delta x_i T_W}{2\Delta x_w}\right) \\ &+ \left(\frac{|G_w| - G_w}{8}\right)\left(\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_e \Delta x_w}\right)\left(\frac{T_E\Delta x_w}{\Delta x_e + \Delta x_w} + \frac{T_W\Delta x_e}{\Delta x_e + \Delta x_w} - T_P\right) \\ &- \left(\frac{|G_w| + G_w}{8}\right)\left(\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww} \Delta x_w}\right)\left(\frac{T_{WW}\Delta x_w}{\Delta x_w + \Delta x_{ww}} + \frac{T_P\Delta x_{ww}}{\Delta x_w + \Delta x_{ww}} - T_w\right) \end{aligned} \tag{6.2}$$

The coefficients of the energy equation are the following:

$$\begin{aligned} A_E = &- \frac{G_e\Delta x_i}{2\Delta x_e} + \frac{\Delta y_j \Delta z_k}{\Delta x_e} + \left(\frac{|G_e| + G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e(\Delta x_w + \Delta x_e)}\right) \\ &+ \left(\frac{|G_e| - G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e \Delta x_{ee}}\right) + \left(\frac{|G_w| - G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_e(\Delta x_w + \Delta x_e)} \end{aligned} \tag{6.3}$$

$$\begin{aligned} A_W = {} & \frac{G_w \Delta x_i}{2\Delta x_w} + \frac{\Delta y_j \Delta z_k}{\Delta x_w} + \left(\frac{|G_w| + G_w}{8}\right)\left(\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w \Delta x_{ww}}\right) \\ & + \left(\frac{|G_w| - G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w(\Delta x_w + \Delta x_e)} + \left(\frac{|G_e| + G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_w(\Delta x_w + \Delta x_e)} \end{aligned} \tag{6.4}$$

$$\begin{aligned} A_N = {} & -\frac{G_n \Delta y_j}{2\Delta y_n} + \frac{\Delta x_i \Delta z_k}{\Delta y_n} + \left(\frac{|G_n| + G_n}{8}\right)\left(\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n(\Delta y_s + \Delta y_n)}\right) \\ & + \left(\frac{|G_n| - G_n}{8}\right)\left(\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n \Delta y_{nn}}\right) + \left(\frac{|G_s| - G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_n(\Delta y_s + \Delta y_n)} \end{aligned} \tag{6.5}$$

$$\begin{aligned} A_S = {} & \frac{G_s \Delta y_j}{2\Delta y_s} + \frac{\Delta x_i \Delta z_k}{\Delta y_s} + \left(\frac{|G_s| + G_s}{8}\right)\left(\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s \Delta y_{ss}}\right) \\ & + \left(\frac{|G_s| - G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s(\Delta y_s + \Delta y_n)} + \left(\frac{|G_n| + G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_s(\Delta y_s + \Delta y_n)} \end{aligned} \tag{6.6}$$

$$\begin{aligned} A_F = {} & -\frac{G_f \Delta z_k}{2\Delta z_f} + \frac{\Delta x_i \Delta y_j}{\Delta z_f} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f(\Delta z_b + \Delta z_f)} \\ & + \left(\frac{|G_f| - G_f}{8}\right)\left(\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f \Delta z_{ff}}\right) + \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_f(\Delta z_b + \Delta z_f)} \end{aligned} \tag{6.7}$$

$$\begin{aligned} A_B = {} & \frac{G_b \Delta z_k}{2\Delta z_b} + \frac{\Delta x_i \Delta y_j}{\Delta z_b} + \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b \Delta z_{bb}} \\ & + \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b(\Delta z_b + \Delta z_f)} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_b(\Delta z_b + \Delta z_f)} \end{aligned} \tag{6.8}$$

$$\begin{aligned} S = {} & \frac{\Delta x_i \Delta y_j \Delta z_k}{\Delta t} T_P^{\,n-1} - \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)} T_{EE} \\ & - \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)} T_{WW} \\ & - \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)} T_{NN} \\ & - \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)} T_{SS} \\ & - \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)} T_{FF} \\ & - \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} T_{BB} \end{aligned} \tag{6.9}$$

$$\begin{aligned} A_P = {} & A_E + A_W + A_N + A_S + A_F + A_B \\ & - \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)} \\ & - \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)} \\ & - \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)} \\ & - \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)} \\ & - \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)} \\ & - \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} + \frac{\Delta x_i \Delta y_j \Delta z_k}{\Delta t} \end{aligned} \tag{6.10}$$

The coefficient A_p etc have their usual meaning.

6.1.2 x-Momentum Equation and Linear Equation for u

For the x-momentum equation, the velocities must be interpolated differently due to the staggered nature of the grid. The QUICK scheme results in the following for the east and west faces of the control volume:

$$\begin{aligned} G_e u_e &= G_e \left(\frac{u_P + u_E}{2}\right) \\ &+ \left(\frac{|G_e| - G_e}{16}\right)\left(\frac{\Delta x_i^2}{\Delta x_e}\right)\left(\frac{u_{EE} - u_E}{\Delta x_{i+1}} - \frac{u_E - u_P}{\Delta x_i}\right) \\ &- \left(\frac{|G_e| + G_e}{16}\right)\left(\frac{\Delta x_i^2}{\Delta x_w}\right)\left(\frac{u_E - u_P}{\Delta x_i} - \frac{u_P - u_W}{\Delta x_{i-1}}\right) \end{aligned} \qquad (6.11)$$

$$\begin{aligned} G_w u_w &= G_w \left(\frac{u_P + u_W}{2}\right) \\ &+ \left(\frac{|G_w| - G_w}{16}\right)\left(\frac{\Delta x_{i-1}^2}{\Delta x_w}\right)\left(\frac{u_E - u_P}{\Delta x_i} - \frac{u_P - u_W}{\Delta x_{i-1}}\right) \\ &- \left(\frac{|G_w| + G_w}{16}\right)\left(\frac{\Delta x_{i-1}^2}{\Delta x_{ww}}\right)\left(\frac{u_P - u_W}{\Delta x_{i-1}} - \frac{u_W - u_{WW}}{\Delta x_{i-2}}\right) \end{aligned} \qquad (6.12)$$

The coefficients of the discretized x-momentum equations are the following:

$$\begin{aligned} A_E = &- \frac{G_e}{2} + Pr\frac{\Delta y_j \Delta z_k}{\Delta x_i} + \left(\frac{|G_e| + G_e}{16}\right)\frac{\Delta x_i}{\Delta x_w} \\ &+ \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i}{\Delta x_{i+1}} + \left(\frac{|G_w| - G_w}{16}\right)\frac{\Delta x_{i-1}^2}{\Delta x_w \Delta x_i} \end{aligned} \qquad (6.13)$$

$$\begin{aligned} A_W = &\frac{G_w}{2} + Pr\frac{\Delta y_j \Delta z_k}{\Delta x_{i-1}} + \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_{i-1}}{\Delta x_{i-2}} \\ &+ \left(\frac{|G_w| - G_w}{16}\right)\frac{\Delta x_{i-1}}{\Delta x_w} + \left(\frac{|G_e| + G_e}{16}\right)\frac{\Delta x_i^2}{\Delta x_w \Delta x_{i-1}} \end{aligned} \qquad (6.14)$$

$$\begin{aligned} A_N = &- \frac{G_n \Delta y_j}{2\Delta y_n} + Pr\frac{\Delta x_w \Delta z_k}{2\Delta y_n} + \left(\frac{|G_n| + G_n}{8}\right)\left(\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n(\Delta y_s + \Delta y_n)}\right) \\ &+ \left(\frac{|G_n| - G_n}{8}\right)\left(\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n \Delta y_{nn}}\right) + \left(\frac{|G_s| - G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_n(\Delta y_s + \Delta y_n)} \end{aligned} \qquad (6.15)$$

$$\begin{aligned} A_S = &\frac{G_s \Delta y_j}{2\Delta y_s} + Pr\frac{\Delta x_w \Delta z_k}{2\Delta y_s} + \left(\frac{|G_s| + G_s}{8}\right)\left(\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s \Delta y_{ss}}\right) \\ &+ \left(\frac{|G_s| - G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s(\Delta y_s + \Delta y_n)} + \left(\frac{|G_n| + G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_s(\Delta y_s + \Delta y_n)} \end{aligned} \qquad (6.16)$$

$$A_F = -\frac{G_f \Delta z_k}{2\Delta z_f} + Pr\frac{\Delta x_w \Delta y_j}{2\Delta z_f} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f(\Delta z_b + \Delta z_f)}$$
$$+ \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f \Delta z_{ff}} + \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_f(\Delta z_b + \Delta z_f)} \qquad (6.17)$$

$$A_B = \frac{G_b \Delta z_k}{2\Delta z_b} + Pr\frac{\Delta x_w \Delta y_j}{2\Delta z_b} + \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b \Delta z_{bb}}$$
$$+ \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b(\Delta z_b + \Delta z_f)} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_b(\Delta z_b + \Delta z_f)} \qquad (6.18)$$

$$S = \frac{\Delta x_w \Delta y_j \Delta z_k}{\Delta t} u_P^{\,n-1} - \left(\frac{|G_e| - G_e}{16}\right)\frac{\Delta x_i^2}{\Delta x_e \Delta x_{i+1}} u_{EE}$$
$$- \left(\frac{|G_w| + G_w}{16} \frac{\Delta x_{i-1}^2}{\Delta x_{ww}\Delta x_{i-2}}\right) u_W$$
$$- \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)} u_{NN}$$
$$- \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)} u_{SS}$$
$$- \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)} u_{FF}$$
$$- \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} u_{BB}$$
$$- (P_P - P_W)\Delta y_j \Delta z_k \qquad (6.19)$$

$$A_P = A_E + A_W + A_N + A_S + A_F + A_B$$
$$- \left(\frac{|G_e| - G_e}{16}\right)\frac{\Delta x_i^2}{\Delta x_e \Delta x_{i+1}}$$
$$- \left(\frac{|G_w| + G_w}{16} \frac{\Delta x_{i-1}^2}{\Delta x_{ww}\Delta x_{i-2}}\right)$$
$$- \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)}$$
$$- \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)}$$
$$- \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)}$$
$$- \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} + \frac{\Delta x_i \Delta y_j \Delta z_k}{\Delta t} \qquad (6.20)$$

6.1.3 y-Momentum Equation and Linear Equation for v

The coefficients of the y-momentum equations are the following:

$$A_N = -\frac{G_n}{2} + Pr\frac{\Delta x_i \Delta z_k}{\Delta y_j} + \left(\frac{|G_n| + G_n}{16}\right)\frac{\Delta y_j}{\Delta y_s}$$

$$+ \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j}{\Delta y_{j+1}} + \left(\frac{|G_s| - G_s}{16}\right)\frac{\Delta y_{j-1}^2}{\Delta y_s \Delta y_j} \tag{6.21}$$

$$\begin{aligned} A_S = {} & \frac{G_s}{2} + Pr\frac{\Delta x_i \Delta z_k}{\Delta y_{j-1}} + \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_{j-1}}{\Delta y_{j-2}} \\ & + \left(\frac{|G_s| - G_s}{16}\right)\frac{\Delta y_{j-1}}{\Delta y_s} + \left(\frac{|G_n| + G_n}{16}\right)\frac{\Delta y_j^2}{\Delta y_s \Delta y_{j-1}} \end{aligned} \tag{6.22}$$

$$\begin{aligned} A_E = {} & -\frac{G_e \Delta x_i}{2\Delta x_e} + Pr\frac{\Delta y_s \Delta z_k}{2\Delta x_e} + \left(\frac{|G_e| + G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e(\Delta x_e + \Delta x_w)}\right) \\ & + \left(\frac{|G_e| - G_e}{8}\right)\left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e \Delta x_{ee}}\right) + \left(\frac{|G_w| - G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_e(\Delta x_e + \Delta x_w)} \end{aligned} \tag{6.23}$$

$$\begin{aligned} A_W = {} & \frac{G_w \Delta x_i}{2\Delta x_w} + Pr\frac{\Delta y_s \Delta z_k}{2\Delta x_w} + \left(\frac{|G_w| + G_w}{8}\right)\left(\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w \Delta x_{ww}}\right) \\ & + \left(\frac{|G_w| - G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w(\Delta x_e + \Delta x_w)} + \left(\frac{|G_e| + G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_w(\Delta x_e + \Delta x_w)} \end{aligned} \tag{6.24}$$

$$\begin{aligned} A_F = {} & -\frac{G_f \Delta z_k}{2\Delta z_f} + Pr\frac{\Delta x_i \Delta y_s}{2\Delta z_f} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f(\Delta z_b + \Delta z_f)} \\ & + \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_f \Delta z_{ff}} + \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_f(\Delta z_b + \Delta z_f)} \end{aligned} \tag{6.25}$$

$$\begin{aligned} A_B = {} & \frac{G_b \Delta z_k}{2\Delta z_b} + Pr\frac{\Delta x_i \Delta y_s}{2\Delta z_b} + \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b \Delta z_{bb}} \\ & + \left(\frac{|G_b| - G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_b(\Delta z_b + \Delta z_f)} + \left(\frac{|G_f| + G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_b(\Delta z_b + \Delta z_f)} \end{aligned} \tag{6.26}$$

$$\begin{aligned} S = {} & \frac{\Delta x_i \Delta y_s \Delta z_k}{\Delta t} v_P^{n-1} - \left(\frac{|G_n| - G_n}{16}\right)\frac{\Delta y_j^2}{\Delta y_n \Delta y_{j+1}} v_{NN} \\ & - \left(\frac{|G_s| + G_s}{16}\right)\frac{\Delta y_{j-1}^2}{\Delta y_{ss} \Delta y_{j-2}} \\ & - \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)} v_{EE} \\ & - \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)} v_{WW} \\ & - \left(\frac{|G_f| - G_f}{8}\right)\frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)} v_{FF} \\ & - \left(\frac{|G_b| + G_b}{8}\right)\frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} v_{BB} \\ & - (P_P - P_S)\Delta x_i \Delta z_k + \frac{T_s \Delta y_j + T_s \Delta y_{j-1}}{2\Delta y_s}\Delta x_i \Delta y_s \Delta z_k \end{aligned} \tag{6.27}$$

$$\begin{aligned} A_P = {} & A_E + A_W + A_N + A_S + A_F + A_B \\ & - \left(\frac{|G_n| - G_n}{16}\right)\frac{\Delta y_j^2}{\Delta y_n \Delta y_{j+1}} \end{aligned}$$

$$- \left(\frac{|G_s| + G_s}{16} \frac{\Delta y_{j-1}^2}{\Delta y_{nn} \Delta y_{j-2}}\right)$$
$$- \left(\frac{|G_e| - G_e}{8}\right) \frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)}$$
$$- \left(\frac{|G_w| + G_w}{8}\right) \frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)}$$
$$- \left(\frac{|G_f| - G_f}{8}\right) \frac{\Delta z_k \Delta z_{k+1}}{\Delta z_{ff}(\Delta z_{ff} + \Delta z_f)}$$
$$- \left(\frac{|G_b| + G_b}{8}\right) \frac{\Delta z_k \Delta z_{k-1}}{\Delta z_{bb}(\Delta z_{bb} + \Delta z_b)} + \frac{\Delta x_i \Delta y_s \Delta z_k}{\Delta t} \qquad (6.28)$$

6.1.4 <u>z-Momentum Equation and linear equation for w</u>

The coefficients of the z-momentum equations are the following:

$$A_F = - \frac{G_f}{2} + Pr\frac{\Delta x_i \Delta y_j}{\Delta z_k} + \left(\frac{|G_f| + G_f}{16}\right) \frac{\Delta z_k}{\Delta z_b} + \left(\frac{|G_f| - G_f}{8}\right) \frac{\Delta z_k}{\Delta z_{k+1}} + \left(\frac{|G_b| - G_b}{16}\right) \frac{\Delta z_{k-1}^2}{\Delta z_b \Delta z_k} \qquad (6.29)$$

$$A_B = \frac{G_b}{2} + Pr\frac{\Delta x_i \Delta y_j}{\Delta z_{k-1}} + \left(\frac{|G_b| + G_b}{8}\right) \frac{\Delta z_{k-1}}{\Delta z_{k-2}} + \left(\frac{|G_b| - G_b}{16}\right) \frac{\Delta z_{k-1}}{\Delta z_b} + \left(\frac{|G_n| + G_n}{16}\right) \frac{\Delta z_k^2}{\Delta z_b \Delta z_{k-1}} \qquad (6.30)$$

$$A_E = - \frac{G_e \Delta x_i}{2\Delta x_e} + Pr\frac{\Delta y_j \Delta z_b}{\Delta x_e} + \left(\frac{|G_e| + G_e}{8}\right) \left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e(\Delta x_e + \Delta x_w)}\right) + \left(\frac{|G_e| - G_e}{8}\right) \left(\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_e \Delta x_{ee}}\right) + \left(\frac{|G_w| - G_w}{8}\right) \frac{\Delta x_i \Delta x_{i-1}}{\Delta x_e(\Delta x_e + \Delta x_w)} \qquad (6.31)$$

$$A_W = \frac{G_w \Delta x_i}{2\Delta x_w} + Pr\frac{\Delta y_j \Delta z_b}{2\Delta x_w} + \left(\frac{|G_w| + G_w}{8}\right) \left(\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w \Delta x_{ww}}\right) + \left(\frac{|G_w| - G_w}{8}\right) \frac{\Delta x_i \Delta x_{i-1}}{\Delta x_w(\Delta x_e + \Delta x_w)} + \left(\frac{|G_e| + G_e}{8}\right) \frac{\Delta x_i \Delta x_{i+1}}{\Delta x_w(\Delta x_e + \Delta x_w)} \qquad (6.32)$$

$$A_N = - \frac{G_n \Delta y_j}{2\Delta y_n} + Pr\frac{\Delta x_i \Delta z_b}{\Delta y_n} + \left(\frac{|G_n| + G_n}{8}\right) \frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n(\Delta y_s + \Delta y_n)} + \left(\frac{|G_n| - G_n}{8}\right) \frac{\Delta y_j \Delta y_{j+1}}{\Delta y_n \Delta y_{nn}} + \left(\frac{|G_s| - G_s}{8}\right) \frac{\Delta y_j \Delta y_{j-1}}{\Delta y_n(\Delta y_n + \Delta y_s)} \qquad (6.33)$$

$$A_S = \frac{G_s \Delta y_j}{2\Delta y_s} + Pr\frac{\Delta x_i \Delta z_b}{\Delta y_s} + \left(\frac{|G_s| + G_s}{8}\right) \frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s \Delta y_{ss}} + \left(\frac{|G_s| - G_s}{8}\right) \frac{\Delta y_j \Delta y_{j-1}}{\Delta y_s(\Delta y_s + \Delta y_n)} + \left(\frac{|G_n| + G_n}{8}\right) \frac{\Delta y_j \Delta y_{j+1}}{\Delta y_s(\Delta y_s + \Delta y_n)} \qquad (6.34)$$

$$S = \frac{\Delta x_i \Delta y_j \Delta z_b}{\Delta t} w_P^{n-1} - \left(\frac{|G_f| - G_f}{16}\right)\frac{\Delta z_k^2}{\Delta z_f \Delta z_{k+1}} w_{FF}$$
$$- \left(\frac{|G_b| + G_b}{16}\right)..\frac{\Delta z_{k-1}^2}{\Delta z_{bb}\,\Delta z_{k+1}} w_{BB}$$
$$- \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)} w_{EE}$$
$$- \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)} w_{WW}$$
$$- \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)} w_{NN}$$
$$- \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)} w_{SS}$$
$$- (P_P - P_B)\,\Delta x_i \Delta y_j \qquad (6.35)$$

$$A_P = A_E + A_W + A_N + A_S + A_F + A_B$$
$$- \left(\frac{|G_f| - G_f}{16}\right)\frac{\Delta z_k^2}{\Delta z_f \Delta z_{k+1}}$$
$$- \left(\frac{|G_b| + G_b}{16}\,\frac{\Delta z_{k-1}^2}{\Delta z_{ff} \Delta z_{k-2}}\right)$$
$$- \left(\frac{|G_e| - G_e}{8}\right)\frac{\Delta x_i \Delta x_{i+1}}{\Delta x_{ee}(\Delta x_{ee} + \Delta x_e)}$$
$$- \left(\frac{|G_w| + G_w}{8}\right)\frac{\Delta x_i \Delta x_{i-1}}{\Delta x_{ww}(\Delta x_{ww} + \Delta x_w)}$$
$$- \left(\frac{|G_n| - G_n}{8}\right)\frac{\Delta y_j \Delta y_{j+1}}{\Delta y_{nn}(\Delta y_{nn} + \Delta y_n)}$$
$$- \left(\frac{|G_s| + G_s}{8}\right)\frac{\Delta y_j \Delta y_{j-1}}{\Delta y_{ss}(\Delta y_{ss} + \Delta y_s)} + \frac{\Delta x_i \Delta y_j \Delta z_b}{\Delta t} \qquad (6.36)$$

6.1.5 <u>The Pressure Correction Equation</u>

The pressure correction equation is derived from the discretized continuity equation. The continuity equation is the following:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \qquad (6.37)$$

Integrating around the control volume

$$(u_e - u_w)\Delta y_j \Delta z_k + (v_n - v_s)\Delta x_i \Delta z_k + (w_f - w_b)\Delta x_i \Delta y_j$$
$$= 0 \qquad (6.38)$$

A pressure correction relationships given before are the following:

$u_e' = du_e (P_P' - P_E')$;

$u_w' = du_w (P_W' - P_P')$;

$v_n' = dv_n (P_P' - P_N')$;

$v_s' = dv_s\,(P_S' - P_P')$;

$w_f' = dw_f\,(P_P' - P_F')$;

$w_b' = dw_b\,(P_B' - P_P')$ (6.39)

The coupling constants are calculated using the SIMPLEX algorithm. Further details are given in Van Doormal and Raithby (1985). also,

$$u_e = u_e^* + u_e'$$
$$u_w = u_w^* + u_w'$$
$$v_n = v_n^* + v_n'$$
$$v_s = v_s^* + v_s'$$
$$w_f = w_f^* + w_f'$$
$$w_b = w_b^* + w_b' \qquad (6.40)$$

The starred quantities represent the velocities which satisfy the momentum equations but not the continuity equation. The primed quantities are the velocity corrections which results in velocities that satisfy the continuity equation. Hence, using equations 6.39 and 6.40 in 6.38 we obtain a pressure correction equation of the following type:

$$A_P P_P' = A_E P_E' + A_W P_W' + A_N P_N' + A_S P_S' + A_F P_F' + A_B P_B' + S$$

Where,

$A_E = du_e \Delta y_j \Delta z_k$;

$A_W = du_w \Delta y_j \Delta z_k$;

$A_N = dv_n \Delta x_i \Delta z_k$;

$A_S = dv_s \Delta x_i \Delta z_k$;

$A_F = dw_f \Delta x_i \Delta y_j$;

$A_B = dw_b \Delta x_i \Delta y_j$;

$A_P = A_E + A_W + A_N + A_S + A_F + A_B$

$S = - u_e^* \Delta y_j \Delta z_k + u_w^* \Delta y_j \Delta z_k - v_n^* \Delta x_i \Delta z_k + v_s^* \Delta x_i \Delta z_k$
$\quad - w_f^* \Delta x_i \Delta y_j + w_b^* \Delta x_i \Delta y_j$ (6.41)

After solving the equations for pressure correction, the velocities and pressures are corrected.

6.2 Numerical Boundary Conditions

All the coefficients and the relations shown so far are valid for interior points. Near the boundary some special treatment is necessary. For example, the central differencing

used to estimate the diffusive fluxes cannot be used for the wall since there is no grid point beyond the wall. There are basically two approaches open to us:

1. Use different interpolation and differencing scheme as we approach the wall. For instance, a one sided three point differencing scheme could be used at the wall.

2. Use the same interpolation and differencing scheme but create additional fictitious grid points or control volumes beyond the boundary and assign suitable values to the grid points for the different variables. Both approaches are equivalent.

The second approach was used for this problem. The advantage being that it is a lot simpler to implement since the same scheme is used with modifications. To that end, additional control volumes were used outside the calculation domain to impose the boundary condition. The one guideline we follow is that the boundary conditions imposed should be second order or higher.

6.2.1 Velocity Boundary Conditions

The no slip boundary conditions are to be imposed at all walls. For the x-momentum equation, the u-velocity grid is staggered in the x-direction the grid point coincides with the wall for the east and west walls perpendicular to the x-direction as shown in figure. In the other directions the control volume surfaces coincide with the walls as shown in figure. For the staggered direction the computational grid extends up to the point i as shown. Point i+1 is the wall. i+2 is the pseudo-point beyond the wall. Up to i-1 no special treatment is necessary. For the equation of point i a special value needs to be assigned to i+2. Noting that the wall is a plane of anti-symmetry for a scheme of second order accuracy we impose,

$$u_{i+2} = -u_i$$

u_{i+1} is of course set equal to zero. Consequently, substituting the values in equation A-20 for the point i ,

$A_E = 0$, the source term S is affected also because $u_{EE} = -u_P$. Similarly, A_P is suitably modified. The west wall is similarly dealt with.

For the other walls only the control volume next to the wall needs to be dealt with hence $u_{i+1} = -u_i$ or,

$u_P = - u_E$

therefore, $A_P = A_P + A_E$ and $A_E = 0$

The v-velocity and the w-velocity boundary conditions are similarly dealt with. What must be kept in mind is that the v-velocity grid is staggered in the y direction and that the w-velocity grid is staggered in the z direction.

In cases where symmetry was exploited $u_{i+1} = u_i$ was used instead.

6.2.2 Temperature Boundary Conditions

There are two types of boundary conditions to be considered; adiabatic and isothermal. The temperature grid is not staggered. The adiabatic boundary condition is identical to the symmetry condition hence for the grid point next to the east wall

$T_P = T_E$

therefore, $A_P = A_P - A_E$ and $A_E = 0$.

For the isothermal boundary condition the value of the temperature grid point outside the domain was assigned by a linear extrapolation. Hence, we have the following;

$$T_E = 2T_{wall} - T_p$$

The symmetry conditions are identical to the adiabatic conditions.

6.2.3 Pressure Correction Boundary Conditions

The velocity corrections at the wall must all be zero. This is because as a result of the no-slip conditions the velocities at the wall are identically zero. Hence,

$$P_P' = P_E'$$

Therefore, in the governing equations;

$$A_P = A_P - A_E \text{ and } A_E = 0$$

Similar modifications will have to be made at the other walls.

APPENDIX C

6.1 General Remarks

The listing of the code is provided in this appendix. The code is written in a manner that facilitates comprehension. The indentation of all DO loops and space between operands all add to the clarity. There are plenty of comments inserted in the code. In fact, it is quite possible to understand most of the code, without actually going through the documentation.

Note that all three-dimensional variables are subscripted (NIP1,NJP1,NKP1). A higher number can also be dimensioned, but that is a waste of computer resources. It is assumed that the user has access to some full-screen editors, therefore it is quite straightforward to change indices for the arrays. That may be required when the grid sizes are changed or if a different physical problem is being attempted. All one-dimensional variables in the subroutine SIP must be dimensioned NNMAX which is a product of NIP1, NJP1 and NKP1. In the specific example of the listing, NIP1 = 12, NJP1 = 16, NKP1 = 16 and NNMAX = 3072. This problem requires about 1.2 Mbytes of memory in a RS/6000 workstation. The listing follows.

```
      PROGRAM AMPHIB
c

c
      CALL OPENF
c
      call grid
c
      call prop
c
      call initio
c
      call ploop
c
      end
c***************************************************************************
      BLOCK DATA
      implicit real*8 (a-h,o-z)
      common/blayer/xbr,ybr,zbr
      common/unfrm/iunfrm
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/count/nt,nmax,imax,itmax,krun,nprint
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/diff/alc,als,thot,tcool,tavg
      common/scheme/quick,upwind
      common/array/njchip,nkchip,ichoice
      common/spaced/ychip(3),zchip(3)
      COMMON/RHOCP/RHS,RHC
      COMMON/POWER/QQQ,QCOND
c
      DATA NIP2,NIP1,NI,NIM1/13,12,11,10/
      DATA NJP2,NJP1,NJ,NJM1/17,16,15,14/
      DATA NKP2,NKP1,NK,NKM1/17,16,15,14/
      DATA NNMAX,NMAX,IMAX,ITMAX,KRUN,NPRINT/3072,4000,10,5,
     &                                        1,100/
      DATA SMALL,EPS,SORMAX/1.0e20,1.0E-8,2./
      DATA ISUB,ICHIP,JCHIP,KCHIP,NCHP,ICHOICE/2,2,2,2,1,1/
      DATA XBR,YBR,ZBR/1.50,1.00,1.50/
      DATA H,WTH,BTH/140.,140.0,100./
      DATA HCHIP,WCHIP,BCHIP,BSUB/24.,8.,6.,19.5/
      DATA ALC,ALS,THOT,TCOOL,TAVG/3338.,2.0,0.0,0.0,19.0/
      DATA RHC,RHS/1.20,0.262/
      DATA QQQ,DTIME,pr,ra,XPER,ROLL/2.0e-03,2.0E-7,28.,1.0e6,0.0,0.0/
      DATA IUNFRM/1/
      DATA QUICK/1./
      DATA NJCHIP,NKCHIP/3,3/
      DATA YCHIP,ZCHIP/38.0,76.0,114.0,50.8,101.6,152.4/
      END
c ************************************************************************
      SUBROUTINE CALT
C
C     THIS SUBROUTINE ASSEMBLES THE DISCRETE ANALOG OF THE ENERGY
C     EQUATION, APPLIES THE BOUNDARY CONDITION AND THEN SOLVES FOR
C     THE TEMPERATURE.
C
```

```
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      COMMON/POWER/QQQ,QCOND
      COMMON/SCHEME/QUICK,UPWIND
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip,ichoice
c
c         CALCULATE COEFFICIENTS
c
      do 80 k=2,nk
         do 80 j=2,nj
            do 80 I=2,ni
C
               VOLDT = DXX(I) * DYY(J) * DZZ(K) / DTIME
     &              * RHO(I,J,K)
C
C       CALCULATE INTERFACIAL THERMAL DIFFUSIVITIES
C          USING THE HARMONIC MEAN FORMULATION
C
               CONDW = ALPHA(I,J,K) * ALPHA(I-1,J,K)
     &              * (DXX(I-1) + DXX(I)) / (DXX(I-1)
     &              * ALPHA(I,J,K) + DXX(I) * ALPHA(I-1,J,K))
               CONDE = ALPHA(I,J,K) * ALPHA(I+1,J,K)
     &              * (DXX(I+1) + DXX(I)) / (DXX(I+1)
     &              * ALPHA(I,J,K) + DXX(I) * ALPHA(I+1,J,K))
               CONDS = ALPHA(I,J,K) * ALPHA(I,J-1,K)
     &              * (DYY(J-1) + DYY(J)) / (DYY(J-1)
     &              * ALPHA(I,J,K) + DYY(J) * ALPHA(I,J-1,K))
               CONDN = ALPHA(I,J,K) * ALPHA(I,J+1,K)
     &              * (DYY(J+1) + DYY(J)) / (DYY(J+1)
     &              * ALPHA(I,J,K) + DYY(J) * ALPHA(I,J+1,K))
               CONDB = ALPHA(I,J,K) * ALPHA(I,J,K-1)
     &              * (DZZ(K-1) + DZZ(K)) / (DZZ(K-1)
     &              * ALPHA(I,J,K) + DZZ(K) * ALPHA(I,J,K-1))
               CONDF = ALPHA(I,J,K) * ALPHA(I,J,K+1)
     &              * (DZZ(K+1) + DZZ(K)) / (DZZ(K+1)
     &              * ALPHA(I,J,K) + DZZ(K) * ALPHA(I,J,K+1))
C
               CONDN1 = DZZ(K) * DXX(I) / DYYS(J+1) * CONDN
               CONDS1 = DZZ(K) * DXX(I) / DYYS(J) * CONDS
               CONDF1 = DXX(I) * DYY(J) / DZZS(K+1) * CONDF
               CONDB1 = DXX(I) * DYY(J) / DZZS(K) * CONDB
```

```
              CONDE1 = DYY(J)  * DZZ(K)  / DXXS(I+1)  * CONDE
              CONDW1 = DYY(J)  * DZZ(K)  / DXXS(I)  * CONDW
C
              GP = RHO(I,J,K)
              GW = RHO(I-1,J,K)
              GE = RHO(I+1,J,K)
              GS  = RHO(I,J-1,K)
              GN = RHO(I,J+1,K)
              GB = RHO(I,J,K-1)
              GF = RHO(I,J,K+1)
C
              RW = (GP * DXX(I-1)  + GW * DXX(I))
     &           / (DXX(I-1)  + DXX(I))
              RE = (GP * DXX(I+1)  + GE * DXX(I))
     &           / (DXX(I+1)  + DXX(I))
              RS = (GP * DYY(J-1)  + GS * DYY(J))
     &           / (DYY(J-1)  + DYY(J))
              RN = (GP * DYY(J+1)  + GN * DYY(J))
     &           / (DYY(J+1)  + DYY(J))
              RB = (GP * DZZ(K-1)  + GB * DZZ(K))
     &           / (DZZ(K-1)  + DZZ(K))
              RF = (GP * DZZ(K+1)  + GF * DZZ(K))
     &           / (DZZ(K+1)  + DZZ(K))
C
              CE = U(I+1,J,K)  * DYY(J)  * DZZ(K)  * RE
              CW = U(I,J,K)  * DYY(J)  * DZZ(K)  * RW
              CN = V(I,J+1,K)  * DZZ(K)  * DXX(I)  * RN
              CS = V(I,J,K)  * DZZ(K)  * DXX(I)  * RS
              CF = W(I,J,K+1)  * DXX(I)  * DYY(J)  * RF
              CB = W(I,J,K)  * DXX(I)  * DYY(J)  * RB
C
       AE(I,J,K)  = CONDE1 + QUICK
     &            * (- 0.5 * CE * DXX(I)  / DXXS(I+1)
     &            + 0.125 * (abs(ce)  + ce)  * dxx(i)  * dxx(i+1)
     &            / (dxxs(i+1)  * (dxxs(i)  + dxxs(i+1)))
     &            + 0.125 * (abs(ce)  - ce)  * dxx(i)  * dxx(i+1)
     &            / (dxxs(i+1)  * dxxs(i+2))
     &            + 0.125 * (abs(cw)  - cw)  * dxx(i-1)  * dxx(i)
     &            / (DXXS(I+1)  * (DXXS(I)  + DXXS(I+1))))
     &            + UPWIND * ( 0.5 * (ABS(CE)  - CE))
       AW(I,J,K)  = CONDW1 + QUICK
     &            * (0.5 * CW * DXX(I)  / DXXS(I)
     &            + 0.125 * (abs(cw)  + cw)  * dxx(i)  * dxx(i-1)
     &            / (dxxs(i-1)  * dxxs(i))
     &            + 0.125 * (abs(cw)  - cw)  * dxx(i-1)  * dxx(i)
     &            / (dxxs(i)  * (dxxs(i)  + dxxs(i+1)))
     &            + 0.125 * (abs(ce)  + ce)  * dxx(i+1)  * dxx(i)
     &            / (DXXS(I)  * (DXXS(I)  + DXXS(I+1))))
     &            + UPWIND * ( 0.5 * (ABS(CW)  + CW))
       AN(I,J,K)  = CONDN1 + QUICK
     &            * (- 0.5 * CN * DYY(J)  / DYYS(J+1)
     &            + 0.125 * (abs(cn)  + cn)  * dyy(j)  * dyy(j+1)
     &            / (dyys(j+1)  * (dyys(j)  + dyys(j+1)))
     &            + 0.125 * (abs(cn)  - cn)  * dyy(j)  * dyy(j+1)
     &            / (dyys(j+1)  * dyys(j+2))
     &            + 0.125 * (abs(cs)  - cs)  * dyy(j-1)  * dyy(j)
     &            / (DYYS(J+1)  * (DYYS(J)  + DYYS(J+1))))
     &            + UPWIND * ( 0.5 * (ABS(CN)  - CN))
       AS(I,J,K)  = CONDS1 + QUICK
     &            * (0.5 * CS * DYY(J)  / DYYS(J)
```

```
     &           + 0.125 * (abs(cs) + cs) * dyy(j) * dyy(j-1)
     &           / (dyys(j-1) * dyys(j))
     &           + 0.125 * (abs(cs) - cs) * dyy(j-1) * dyy(j)
     &           / (dyys(j) * (dyys(j) + dyys(j+1)))
     &           + 0.125 * (abs(cn) + cn) * dyy(j+1) * dyy(j)
     &           / (DYYS(J) * (DYYS(J) + DYYS(J+1))))
     &           + UPWIND * ( 0.5 * (ABS(CS) + CS))
      AF(I,J,K) = CONDF1 + QUICK
     &           * (- 0.5 * CF * DZZ(K) / DZZS(K+1)
     &           + 0.125 * (abs(cf) + cf) * dzz(k) * dzz(k+1)
     &           / (dzzs(k+1) * (dzzs(k) + dzzs(k+1)))
     &           + 0.125 * (abs(cf) - cf) * dzz(k) * dzz(k+1)
     &           / (dzzs(k+1) * dzzs(k+2))
     &           + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &           / (DZZS(K+1) * (DZZS(K) + DZZS(K+1))))
     &           + UPWIND * ( 0.5 * (ABS(CF) - CF))
      AB(I,J,K) = CONDB1 + QUICK
     &           * (0.5 * CB * DZZ(K) / DZZS(K)
     &           + 0.125 * (abs(cb) + cb) * dzz(k) * dzz(k-1)
     &           / (dzzs(k-1) * dzzs(k))
     &           + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &           / (dzzs(k) * (dzzs(k) + dzzs(k+1)))
     &           + 0.125 * (abs(cf) + cf) * dzz(k+1) * dzz(k)
     &           / (DZZS(K) * (DZZS(K) + DZZS(K+1))))
     &           + UPWIND * ( 0.5 * (ABS(CB) + CB))
c
      AEE(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CE) - CE) * DXX(I) * DXX(I+1)
     &             / (DXXS(I+2) * (DXXS(I+1) + DXXS(I+2)))
      AWW(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CW) + CW) * DXX(I) * DXX(I-1)
     &             / (DXXS(I-1) * (DXXS(I-1) + DXXS(I)))
      ANN(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CN) - CN) * DYY(J) * DYY(J+1)
     &             / (DYYS(J+2) * (DYYS(J+1) + DYYS(J+2)))
      ASS(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CS) + CS) * DYY(J) * DYY(J-1)
     &             / (DYYS(J-1) * (DYYS(J-1) + DYYS(J)))
      AFF(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CF) - CF) * DZZ(K) * DZZ(K+1)
     &             / (DZZS(K+2) * (DZZS(K+1) + DZZS(K+2)))
      ABB(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CB) + CB) * DZZ(K) * DZZ(K-1)
     &             / (DZZS(K-1) * (DZZS(K-1) + DZZS(K)))
c
      ap(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     &          + AF(I,J,K) + AB(I,J,K) + AWW(I,J,K)
     &          + AEE(I,J,K) + ASS(I,J,K) + ANN(I,J,K)
     &          + ABB(I,J,K) + AFF(I,J,K) + VOLDT
      SU(I,J,K) = VOLDT * TOD(I,J,K)
   80 continue
C
C     CALCULATION OF THE SOURCE TERM
C
      DO 81 I=4,NI
         DO 81 J=2,NJ
            DO 81 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AWW(I,J,K) * TPD(I-2,J,K)
   81 CONTINUE
```

```
C
      DO 82 I=2,NI-2
         DO 82 J=2,NJ
            DO 82 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AEE(I,J,K) * TPD(I+2,J,K)
   82 CONTINUE
C
      DO 83 I=2,NI
         DO 83 J=4,NJ
            DO 83 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ASS(I,J,K) * TPD(I,J-2,K)
   83 CONTINUE
C
      DO 84 I=2,NI
         DO 84 J=2,NJ-2
            DO 84 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ANN(I,J,K) * TPD(I,J+2,K)
   84 CONTINUE
C
      DO 85 I=2,NI
         DO 85 J=2,NJ
            DO 85 K=4,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ABB(I,J,K) * TPD(I,J,K-2)
   85 CONTINUE
C
      DO 86 I=2,NI
         DO 86 J=2,NJ
            DO 86 K=2,NK-2
               SU(I,J,K) = SU(I,J,K)
     &                   + AFF(I,J,K) * TPD(I,J,K+2)
   86 CONTINUE
C
      if (nchp .ne. 0) then
C
C       CHIP HEAT GENERATION
C
         do 90 m=1,njchip
            do 90 n=1,nkchip
               do 90 i=ibgn,iend
                  do 90 j=jbgn(m),jend(m)
                     do 90 k=kbgn(n),kend(n)
                        su(i,j,k) = alc * dxx(i) * dyy(j)
     &                            * dzz(k) + su(i,j,k)
   90    continue
      endif
C
C       BOUNDARY CONDITIONS (TOP AND BOTTOM,ISOTHERMAL)
c
      do 101 k=2,nk
         do 101 i=2,ni
            AP(I,2,K) = AP(I,2,K) + AS(I,2,K) - ASS(I,2,K)
            su(i,2,k) = su(i,2,k) + 2. * as(i,2,k) * thot
            as(i,2,k) = 0.
            SU(I,3,K) = SU(I,3,K) + ASS(I,3,K)
     &                * (2. * THOT - TPD(I,2,K))
            SU(I,NJM1,K) = SU(I,NJM1,K) + ANN(I,NJM1,K)
```

```
     &                     * (2. * TCOOL - TPD(I,NJ,K))
             AP(I,NJ,K) = AP(I,NJ,K) + AN(I,NJ,K) - ANN(I,NJ,K)
             su(i,nj,k) = su(i,nj,k) + 2. * an(i,nj,k) * tcool
             an(i,nj,k) = 0.
  101 continue
c
c        ADIABATIC WALLS (LEFT AND RIGHT)
c
       do 102 k=2,nk
          do 102 j=2,nj
             AP(2,J,K) = AP(2,J,K) - AW(2,J,K) - AWW(2,J,K)
             aw(2,j,k) = 0.
             SU(3,J,K) = SU(3,J,K) + AWW(3,J,K) * TPD(2,J,K)
             AP(NI,J,K) = AP(NI,J,K) - AE(NI,J,K) - AEE(NI,J,K)
             SU(NIM1,J,K) = SU(NIM1,J,K)
     &                    + AEE(NIM1,J,K) * TPD(NI,J,K)
             ae(ni,j,k) = 0.
  102 continue
c
c        ADIABATIC WALLS (BACK AND FRONT)
c
       do 103 j=2,nj
          do 103 i=2,ni
             AP(I,J,2) = AP(I,J,2) - AB(I,J,2) - ABB(I,J,2)
             ab(i,j,2) = 0.
             SU(I,J,3) = SU(I,J,3) + ABB(I,J,3) * TPD(I,J,2)
             AP(I,J,NK) = AP(I,J,NK) - AF(I,J,NK) - AFF(I,J,NK)
             SU(I,J,NKM1) = SU(I,J,NKM1)
     &                    + AFF(I,J,NKM1) * TPD(I,J,NK)
             af(i,j,nk) = 0.
  103 continue
C
C        SOLVE DIFFERENCE EQUATION
C
       call sip (2,2,2,ni,nj,nk,t)
C
C      PRESCRIBE WALL TEMPERATURES CONSISTENT WITH THE
C                  BOUNDARY CONDITIONS
C              (RIGHT AND LEFT WALL)
C
       do 310 k=2,nk
          do 310 j=2,nj
             t(nip1,j,k) = t(ni,j,k)
             t(1,j,k) = t(2,j,k)
  310 continue
c
c              FRONT AND BACK WALL
c
       do 320 j=2,nj
          do 320 i=2,ni
             t(i,j,nkp1) = t(i,j,nk)
             t(i,j,1) = t(i,j,2)
  320 continue
c
c             TOP AND BOTTOM WALL
c
       do 330 k=2,nk
          do 330 i=2,ni
             t(i,1,k) = 2. * thot - t(i,2,k)
             t(i,njp1,k) = 2. * tcool - t(i,nj,k)
```

```
  330 continue
c
c     THE KINEMATIC VISCOSITY IS CALCULATED, WHICH IS A STRONG
c     FUNCTION OF TEMPERATURE
c
      do 350 i=2,ni
         do 350 j=2,nj
            do 350 k=2,nk
               ttmp = tavg + qcond * t(i,j,k)
               if (ichoice .eq. 0) then
                   gnu = pr
               endif
               if (ichoice .eq. 1) then
                   gnu = (1.4074 - 2.96e-2 * ttmp
     &                   + 3.8018e-4 * ttmp**2
     &                   - 2.731e-6 * ttmp**3
     &                   + 8.168e-9 * ttmp**4) * 29.088
               endif
               if (ichoice .eq. 2) then
                   gnu = (251.62 - 13.723 * tavg
     &                   + 3.056e-1 * tavg**2
     &                   - 3.1704e-3 * tavg**3
     &                   + 1.2668e-5 * tavg**4) * 28.666
               endif
               visco(i,j,k) = gnu
  350 continue
c
      end
C**********************************************************************
      SUBROUTINE CALU
C
C     THIS SUBROUTINE ASSEMBLES THE U-MOMENTUM EQUATION AND
C     BOUNDARY CONDITION AND SOLVES FOR THE U-VELOCITY (X-DIRECTION)
C
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/scheme/quick,upwind
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip,ichoice
```

```
      common/tol/small,eps,sormax
      common/ifirst/nust,nvst,nwst,npst
c
c        CALCULATE COEFFICIENTS
c
      do 100 k=2,nk
         do 100 j=2,nj
            do 100 i=nust,ni
               voldt = dxxs(i) * dyy(j) * dzz(k) / dtime
C
C       CALCULATE INTERFACIAL KINEMATIC VISCOSITIES
C          USING THE HARMONIC MEAN FORMULATION
C
               visw = visco(i-1,j,k)
               vise = visco(i,j,k)
               viss = visco(i,j,k) * visco(i,j-1,k)
     &              * (dyy(j-1) + dyy(j)) / (dyy(j-1)
     &              * visco(i,j,k) + dyy(j) * visco(i,j-1,k))
               visn = visco(i,j,k) * visco(i,j+1,k)
     &              * (dyy(j+1) + dyy(j)) / (dyy(j+1)
     &              * visco(i,j,k) + dyy(j) * visco(i,j+1,k))
               visb = visco(i,j,k) * visco(i,j,k-1)
     &              * (dzz(k-1) + dzz(k)) / (dzz(k-1)
     &              * visco(i,j,k) + dzz(k) * visco(i,j,k-1))
               visf = visco(i,j,k) * visco(i,j,k+1)
     &              * (dzz(k+1) + dzz(k)) / (dzz(k+1)
     &              * visco(i,j,k) + dzz(k) * visco(i,j,k+1))
c
               dxys = dxxs(i) * dyy(j)
               dyzs = dyy(j) * dzz(k)
               dzxs = dzz(k) * dxxs(i)
               zxoyn = dzxs / dyys(j+1)
               zxoys = dzxs / dyys(j)
               xyozf = dxys / dzzs(k+1)
               xyozb = dxys / dzzs(k)
               yzoxe = dyzs / dxx(i)
               yzoxw = dyzs / dxx(i-1)
c
               GN = V(I,J+1,K)
               GNW = V(I-1,J+1,K)
               GS = V(I,J,K)
               GSW = V(I-1,J,K)
c
               GE = U(I+1,J,K)
               GP = U(I,J,K)
               GW = U(I-1,J,K)
c
               GF = W(I,J,K+1)
               GFW = W(I-1,J,K+1)
               GB = W(I,J,K)
               GBW = W(I-1,J,K)
c
               cn = (gn * dxx(i-1) + gnw * dxx(i))
     &            / (dxx(i-1) + dxx(i)) * dzxs
               cs = (gs * dxx(i-1) + gsw * dxx(i))
     &            / (dxx(i-1) + dxx(i)) * dzxs
               ce = 0.5 * (ge + gp) * dyzs
               cw = 0.5 * (gp + gw) * dyzs
               cf = (gf * dxx(i-1) + gfw * dxx(i))
     &            / (dxx(i-1) + dxx(i)) * dxys
```

```
          cb = (gb * dxx(i-1) + gbw * dxx(i))
     &         / (dxx(i-1) + dxx(i)) * dxys
c
          vise1 = yzoxe * vise
          visw1 = yzoxw * visw
          visn1 = zxoyn * visn
          viss1 = zxoys * viss
          visf1 = xyozf * visf
          visb1 = xyozb * visb
c
      AE(I,J,K) = VISE1 + (- 0.5 * CE + 0.0625 * (ABS(CE) + CE)
     &            * dxx(i) / dxxs(i) + 0.125 * (abs(ce) - ce)
     &            * dxx(i) / dxx(i+1) + 0.0625 * (abs(cw) - cw)
     &            * DXX(I-1)**2 / (DXX(I) * DXXS(I))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CE) - CE))
      AW(I,J,K) = VISW1 + (0.5 * CW + 0.125 * (ABS(CW) + CW)
     &            * dxx(i-1) / dxx(i-2) + 0.0625 * (abs(cw) - cw)
     &            * dxx(i-1) / dxxs(i) + 0.0625 * (abs(ce) + ce)
     &            * DXX(I)**2 / (DXX(I-1) * DXXS(I))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CW) + CW))
      AN(I,J,K) = VISN1 + (- 0.5 * CN * DYY(J) / DYYS(J+1)
     &            + 0.125 * (abs(cn) + cn) * dyy(j) * dyy(j+1)
     &            / (dyys(j+1) * (dyys(j) + dyys(j+1)))
     &            + 0.125 * (abs(cn) - cn) * dyy(j) * dyy(j+1)
     &            / (dyys(j+1) * dyys(j+2))
     &            + 0.125 * (abs(cs) - cs) * dyy(j-1) * dyy(j)
     &            / (DYYS(J+1) * (DYYS(J) + DYYS(J+1)))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CN) - CN))
      AS(I,J,K) = VISS1 + (0.5 * CS * DYY(J) / DYYS(J)
     &            + 0.125 * (abs(cs) + cs) * dyy(j) * dyy(j-1)
     &            / (dyys(j-1) * dyys(j))
     &            + 0.125 * (abs(cs) - cs) * dyy(j-1) * dyy(j)
     &            / (dyys(j) * (dyys(j) + dyys(j+1)))
     &            + 0.125 * (abs(cn) + cn) * dyy(j+1) * dyy(j)
     &            / (DYYS(J) * (DYYS(J) + DYYS(J+1)))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CS) + CS))
      AF(I,J,K) = VISF1 + (- 0.5 * CF * DZZ(K) / DZZS(K+1)
     &            + 0.125 * (abs(cf) + cf) * dzz(k) * dzz(k+1)
     &            / (dzzs(k+1) * (dzzs(k) + dzzs(k+1)))
     &            + 0.125 * (abs(cf) - cf) * dzz(k) * dzz(k+1)
     &            / (dzzs(k+1) * dzzs(k+2))
     &            + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &            / (DZZS(K+1) * (DZZS(K) + DZZS(K+1)))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CF) - CF))
      AB(I,J,K) = VISB1 + (0.5 * CB * DZZ(K) / DZZS(K)
     &            + 0.125 * (abs(cb) + cb) * dzz(k) * dzz(k-1)
     &            / (dzzs(k-1) * dzzs(k))
     &            + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &            / (dzzs(k) * (dzzs(k) + dzzs(k+1)))
     &            + 0.125 * (abs(cf) + cf) * dzz(k+1) * dzz(k)
     &            / (DZZS(K) * (DZZS(K) + DZZS(K+1)))) * QUICK
     &            + UPWIND * ( 0.5 * (ABS(CB) + CB))
c
      AEE(I,J,K) = - 0.0625 * QUICK
     &             * (ABS(CE) - CE) * DXX(I)**2
     &             / (DXX(I+1) * DXXS(I+1))
      AWW(I,J,K) = - 0.0625 * QUICK
     &             * (ABS(CW) + CW) * DXX(I-1)**2
     &             / (DXX(I-2) * DXXS(I-1))
      ANN(I,J,K) = - 0.125 * QUICK
```

```
     &             * (ABS(CN) - CN) * DYY(J)  * DYY(J+1)
     &             / (DYYS(J+2)  * (DYYS(J+1)  + DYYS(J+2)))
      ASS(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CS) + CS) * DYY(J)  * DYY(J-1)
     &             / (DYYS(J-1)  * (DYYS(J-1)  + DYYS(J)))
      AFF(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CF) - CF) * DZZ(K)  * DZZ(K+1)
     &             / (DZZS(K+2)  * (DZZS(K+1)  + DZZS(K+2)))
      ABB(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CB) + CB) * DZZ(K)  * DZZ(K-1)
     &             / (DZZS(K-1)  * (DZZS(K-1)  + DZZS(K)))
C
      AP(I,J,K) = AE(I,J,K) + AW(I,J,K) + AN(I,J,K) + AS(I,J,K)
     &            + AF(I,J,K) + AB(I,J,K) + AWW(I,J,K)
     &            + AEE(I,J,K) + ASS(I,J,K) + ANN(I,J,K)
     &            + ABB(I,J,K) + AFF(I,J,K) + VOLDT
      SU(I,J,K) = VOLDT * UOD(I,J,K)
     &            + DYZS * (P(I-1,J,K) - P(I,J,K))
  100 continue
C
C     CALCULATION OF THE SOURCE TERM
C
      DO 101 I=NUST+1,NI
         DO 101 J=2,NJ
            DO 101 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AWW(I,J,K) * UPD(I-2,J,K)
 101  CONTINUE
C
      DO 102 I=NUST,NI-1
         DO 102 J=2,NJ
            DO 102 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AEE(I,J,K) * UPD(I+2,J,K)
 102  CONTINUE
C
      DO 103 I=NUST,NI
         DO 103 J=4,NJ
            DO 103 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ASS(I,J,K) * UPD(I,J-2,K)
 103  CONTINUE
C
      DO 104 I=NUST,NI
         DO 104 J=2,NJ-2
            DO 104 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ANN(I,J,K) * UPD(I,J+2,K)
 104  CONTINUE
C
      DO 105 I=NUST,NI
         DO 105 J=2,NJ
            DO 105 K=4,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ABB(I,J,K) * UPD(I,J,K-2)
 105  CONTINUE
C
      DO 106 I=NUST,NI
         DO 106 J=2,NJ
            DO 106 K=2,NK-2
```

```
                 SU(I,J,K)  = SU(I,J,K)
     &                      + AFF(I,J,K) * UPD(I,J,K+2)
 106    CONTINUE
C
C          BOUNDARY CONDITIONS (TOP AND BOTTOM)
C
        DO 121 K=2,NK
           DO 121 I=NUST,NI
              AP(I,2,K)  = AP(I,2,K) + AS(I,2,K) - ASS(I,2,K)
              AS(I,2,K)  = 0.
              SU(I,3,K)  = SU(I,3,K) - ASS(I,3,K) * UPD(I,2,K)
              SU(I,NJM1,K)  = SU(I,NJM1,K)
     &                      - ANN(I,NJM1,K) * UPD(I,NJ,K)
              AP(I,NJ,K) = AP(I,NJ,K) + AN(I,NJ,K) - ANN(I,NJ,K)
              AN(I,NJ,K) = 0.
  121   CONTINUE
C
C          LEFT AND RIGHT WALL
C
        DO 122 K=2,NK
           DO 122 J=2,NJ
              AW(NUST,J,K)  = 0.
              SU(NUST,J,K)  = SU(NUST,J,K)
     &                      - AWW(NUST,J,K) * UPD(NUST,J,K)
              SU(NI,J,K)  = SU(NI,J,K)
     &                    - AEE(NI,J,K) * UPD(NI,J,K)
              AE(NI,J,K)  = 0.
  122   CONTINUE
C
C          FRONT AND BACK WALL
C
        DO 123 J=2,NJ
           DO 123 I=NUST,NI
              AP(I,J,2)  = AP(I,J,2) + AB(I,J,2) - ABB(I,J,2)
              AB(I,J,2)  = 0.
              SU(I,J,3)  = SU(I,J,3) - ABB(I,J,3) * UPD(I,J,2)
              SU(I,J,NKM1)  = SU(I,J,NKM1)
     &                      - AFF(I,J,NKM1) * UPD(I,J,NK)
              AP(I,J,NK)  = AP(I,J,NK) + AF(I,J,NK) - AFF(I,J,NK)
              AF(I,J,NK)  = 0.
  123   CONTINUE
C
        IF (NCHP .NE. 0) THEN
C
C          U-VELOCITY SET TO ZERO IN CHIP
C
           DO 130 M=1,NJCHIP
              DO 130 N=1,NKCHIP
                 DO 130 I=IBGN+1,IEND+1
                    DO 130 J=JBGN(M),JEND(M)
                       DO 130 K=KBGN(N),KEND(N)
                          AP(I,J,K)  = small
                          AW(I,J,K)  = 0.
                          AE(I,J,K)  = 0.
                          AS(I,J,K)  = 0.
                          AN(I,J,K)  = 0.
                          AB(I,J,K)  = 0.
                          AF(I,J,K)  = 0.
                          SU(I,J,K)  = 0.
  130      CONTINUE
```

```
C
C       CHIP BOUNDARY CONDITIONS (TOP AND BOTTOM)
C
        DO 131 M=1,NJCHIP
           DO 131 N=1,NKCHIP
              DO 131 I=IBGN+1,IEND+1
                 DO 131 K=KBGN(N),KEND(N)
                    JJS = JEND(M)
                    JJS1 = JEND(M) + 1
                    JJS2 = JEND(M) + 2
                    JJN = JBGN(M)
                    JJN1 = JBGN(M) - 1
                    JJN2 = JBGN(M) - 2
                    AP(I,JJS1,K) = AP(I,JJS1,K)
     &                           + AS(I,JJS1,K) - ASS(I,JJS1,K)
                    AS(I,JJS1,K) = 0.
                    SU(I,JJS2,K) = SU(I,JJS2,K)
     &                           - ASS(I,JJS2,K) * UPD(I,JJS,K)
     &                           - ASS(I,JJS2,K) * UPD(I,JJS1,K)
                    SU(I,JJN2,K) = SU(I,JJN2,K)
     &                           - ANN(I,JJN2,K) * UPD(I,JJN,K)
     &                           - ANN(I,JJN2,K) * UPD(I,JJN1,K)
                    AP(I,JJN1,K) = AP(I,JJN1,K)
     &                           + AN(I,JJN1,K) - ANN(I,JJN1,K)
                    AN(I,JJN1,K) = 0.
  131 CONTINUE
C
C       FRONT AND BACK WALL
C
        DO 132 M=1,NJCHIP
           DO 132 N=1,NKCHIP
              DO 132 I=IBGN+1,IEND+1
                 DO 132 J=JBGN(M),JEND(M)
                    KKB = KEND(N)
                    KKB1 = KEND(N) + 1
                    KKB2 = KEND(N) + 2
                    KKF = KBGN(N)
                    KKF1 = KBGN(N) - 1
                    KKF2 = KBGN(N) - 2
                    AP(I,J,KKB1) = AP(I,J,KKB1)
     &                           + AB(I,J,KKB1) - ABB(I,J,KKB1)
                    AB(I,J,KKB1) = 0.
                    SU(I,J,KKB2) = SU(I,J,KKB2)
     &                           - ABB(I,J,KKB2) * UPD(I,J,KKB)
     &                           - ABB(I,J,KKB2) * UPD(I,J,KKB1)
                    SU(I,J,KKF2) = SU(I,J,KKF2)
     &                           - AFF(I,J,KKF2) * UPD(I,J,KKF)
     &                           - AFF(I,J,KKF2) * UPD(I,J,KKF1)
                    AP(I,J,KKF1) = AP(I,J,KKF1)
     &                           + AF(I,J,KKF1) - AFF(I,J,KKF1)
                    AF(I,J,KKF1) = 0.
  132 CONTINUE
C
C                 RIGHT WALL
C
        DO 133 M=1,NJCHIP
           DO 133 N=1,NKCHIP
              DO 133 J=JBGN(M),JEND(M)
                 DO 133 K=KBGN(N),KEND(N)
                    II2 = IEND + 2
```

```
                        AW(II2,J,K)  = 0.
                        SU(II2,J,K)  = SU(II2,J,K)
     &                                 - AWW(II2,J,K)  * UPD(IEND,J,K)
     &                                 - AWW(II2,J,K)  * UPD(II2,J,K)
  133    CONTINUE
      ENDIF
C
C        SOLVE THE LINEARISED U MOMENTUM EQUATIONS
C
      CALL SIP (NUST,2,2,NI,NJ,NK,U)
C
C        CALCULATE DU NEEDED FOR PRESSURE CORRECTION AND SOLVE
C
      DO 200 K=2,NK
         DO 200 J=2,NJ
            DO 200 I=NUST,NI
               SU(I,J,K)  = DYY(J)  * DZZ(K)
  200 CONTINUE
C
      CALL SIP (NUST,2,2,NI,NJ,NK,DU)
C
      END
C***********************************************************************
      SUBROUTINE CALV
C
C     THIS SUBROUTINE ASSEMBLES THE V-MOMENTUM EQUATION AND
C     BOUNDARY CONDITION AND SOLVES FOR THE V-VELOCITY (Y DIRECTION)
C
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &           isub,ichip,jchip,kchip
      COMMON/SCHEME/QUICK,UPWIND
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip,ichoice
      common/tol/small,eps,sormax
      common/ifirst/nust,nvst,nwst,npst
C
C        CALCULATE COEFFICIENTS
C
      do 100 k=2,nk
```

```
      do 100 j=3,nj
         do 100 i=nvst,ni
            voldt = dxx(i) * dyys(j) * dzz(k) / dtime
c
c     CALCULATE INTERFACIAL KINEMATIC VISCOSITIES
c        USING THE HARMONIC MEAN FORMULATION
c
            visw = visco(i,j,k) * visco(i-1,j,k)
     &           * (dxx(i-1) + dxx(i)) / (dxx(i-1)
     &           * visco(i,j,k) + dxx(i) * visco(i-1,j,k))
            vise = visco(i,j,k) * visco(i+1,j,k)
     &           * (dxx(i+1) + dxx(i)) / (dxx(i+1)
     &           * visco(i,j,k) + dxx(i) * visco(i+1,j,k))
            viss = visco(i,j-1,k)
            visn = visco(i,j,k)
            visb = visco(i,j,k) * visco(i,j,k-1)
     &           * (dzz(k-1) + dzz(k)) / (dzz(k-1)
     &           * visco(i,j,k) + dzz(k) * visco(i,j,k-1))
            visf = visco(i,j,k) * visco(i,j,k+1)
     &           * (dzz(k+1) + dzz(k)) / (dzz(k+1)
     &           * visco(i,j,k) + dzz(k) * visco(i,j,k+1))
c
            dxys = dxx(i) * dyys(j)
            dyzs = dyys(j) * dzz(k)
            dzxs = dzz(k) * dxx(i)
            zxoyn = dzxs / dyy(j)
            zxoys = dzxs / dyy(j-1)
            xyozf = dxys / dzzs(k+1)
            xyozb = dxys / dzzs(k)
            yzoxe = dyzs / dxxs(i+1)
            yzoxw = dyzs / dxxs(i)
c
            GN = V(I,J+1,K)
            GP = V(I,J,K)
            GS = V(I,J-1,K)
            GE = U(I+1,J,K)
            GSE = U(I+1,J-1,K)
            GW = U(I,J,K)
            GSW = U(I,J-1,K)
            GF = W(I,J,K+1)
            GSF = W(I,J-1,K+1)
            GB = W(I,J,K)
            GSB = W(I,J-1,K)
c
            cn = 0.5 * (gn + gp) * dzxs
            cs = 0.5 * (gp + gs) * dzxs
            ce = (ge * dyy(j-1) + gse * dyy(j))
     &         / (dyy(j-1) + dyy(j)) * dyzs
            cw = (gw * dyy(j-1) + gsw * dyy(j))
     &         / (dyy(j-1) + dyy(j)) * dyzs
            cf = (gf * dyy(j-1) + gsf * dyy(j))
     &         / (dyy(j-1) + dyy(j)) * dxys
            cb = (gb * dyy(j-1) + gsb * dyy(j))
     &         / (dyy(j-1) + dyy(j)) * dxys
c
            vise1 = yzoxe * vise
            visw1 = yzoxw * visw
            visn1 = zxoyn * visn
            viss1 = zxoys * viss
            visf1 = xyozf * visf
```

```
      VISB1 = XYOZB * VISB
C
      AE(I,J,K)  = VISE1 + (- 0.5 * CE * DXX(I) / DXXS(I+1)
     &             + 0.125 * (abs(ce) + ce) * dxx(i) * dxx(i+1)
     &             / (dxxs(i+1) * (dxxs(i) + dxxs(i+1)))
     &             + 0.125 * (abs(ce) - ce) * dxx(i) * dxx(i+1)
     &             / (dxxs(i+1) * dxxs(i+2))
     &             + 0.125 * (abs(cw) - cw) * dxx(i-1) * dxx(i)
     &             / (DXXS(I+1) * (DXXS(I) + DXXS(I+1)))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CE) - CE))
      AW(I,J,K)  = VISW1 + (0.5 * CW * DXX(I) / DXXS(I)
     &             + 0.125 * (abs(cw) + cw) * dxx(i) * dxx(i-1)
     &             / (dxxs(i-1) * dxxs(i))
     &             + 0.125 * (abs(cw) - cw) * dxx(i-1) * dxx(i)
     &             / (dxxs(i) * (dxxs(i) + dxxs(i+1)))
     &             + 0.125 * (abs(ce) + ce) * dxx(i+1) * dxx(i)
     &             / (DXXS(I) * (DXXS(I) + DXXS(I+1)))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CW) + CW))
      AN(I,J,K)  = VISN1 + (- 0.5 * CN + 0.0625 * (ABS(CN) + CN)
     &             * dyy(j) / dyys(j) + 0.125 * (abs(cn) - cn)
     &             * dyy(j) / dyy(j+1) + 0.0625 * (abs(cs) - cs)
     &             * DYY(J-1)**2 / (DYY(J) * DYYS(J))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CN) - CN))
      AS(I,J,K)  = VISS1 + (0.5 * CS + 0.125 * (ABS(CS) + CS)
     &             * dyy(j-1) / dyy(j-2) + 0.0625 * (abs(cs) - cs)
     &             * dyy(j-1) / dyys(j) + 0.0625 * (abs(cn) + cn)
     &             * DYY(J)**2 / (DYY(J-1) * DYYS(J))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CS) + CS))
      AF(I,J,K)  = VISF1 + (- 0.5 * CF * DZZ(K) / DZZS(K+1)
     &             + 0.125 * (abs(cf) + cf) * dzz(k) * dzz(k+1)
     &             / (dzzs(k+1) * (dzzs(k) + dzzs(k+1)))
     &             + 0.125 * (abs(cf) - cf) * dzz(k) * dzz(k+1)
     &             / (dzzs(k+1) * dzzs(k+2))
     &             + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &             / (DZZS(K+1) * (DZZS(K) + DZZS(K+1)))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CF) - CF))
      AB(I,J,K)  = VISB1 + (0.5 * CB * DZZ(K) / DZZS(K)
     &             + 0.125 * (abs(cb) + cb) * dzz(k) * dzz(k-1)
     &             / (dzzs(k-1) * dzzs(k))
     &             + 0.125 * (abs(cb) - cb) * dzz(k-1) * dzz(k)
     &             / (dzzs(k) * (dzzs(k) + dzzs(k+1)))
     &             + 0.125 * (abs(cf) + cf) * dzz(k+1) * dzz(k)
     &             / (DZZS(K) * (DZZS(K) + DZZS(K+1)))) * QUICK
     &             + UPWIND * ( 0.5 * (ABS(CB) + CB))
C
      AEE(I,J,K)  = - 0.125 * QUICK
     &              * (ABS(CE) - CE) * DXX(I) * DXX(I+1)
     &              / (DXXS(I+2) * (DXXS(I+1) + DXXS(I+2)))
      AWW(I,J,K)  = - 0.125 * QUICK
     &              * (ABS(CW) + CW) * DXX(I) * DXX(I-1)
     &              / (DXXS(I) * (DXXS(I-1) + DXXS(I)))
      ANN(I,J,K)  = - 0.0625 * QUICK
     &              * (ABS(CN) - CN) * DYY(J)**2
     &              / (DYY(J+1) * DYYS(J+1))
      ASS(I,J,K)  = - 0.0625 * QUICK
     &              * (ABS(CS) + CS) * DYY(J-1)**2
     &              / (DYY(J-2) * DYYS(J-1))
      AFF(I,J,K)  = - 0.125 * QUICK
     &              * (ABS(CF) - CF) * DZZ(K) * DZZ(K+1)
     &              / (DZZS(K+2) * (DZZS(K+1) + DZZS(K+2)))
```

```
      ABB(I,J,K) = - 0.125 * QUICK
     &             * (ABS(CB) + CB) * DZZ(K) * DZZ(K-1)
     &             / (DZZS(K) * (DZZS(K-1) + DZZS(K)))
      AP(I,J,K) = AE(I,J,K) + AW(I,J,K) + AN(I,J,K) + AS(I,J,K)
     &          + AF(I,J,K) + AB(I,J,K) + AWW(I,J,K)
     &          + AEE(I,J,K) + ASS(I,J,K) + ANN(I,J,K)
     &          + ABB(I,J,K) + AFF(I,J,K) + VOLDT
      SU(I,J,K) = VOLDT * VOD(I,J,K)
     &          + DZXS * (P(I,J-1,K) - P(I,J,K))
     &          + RA * (T(I,J,K) * DYY(J-1) + T(I,J-1,K)
     &          * DYY(J)) / (DYY(J) + DYY(J-1))
     &          * DXX(I) * DYYS(J) * DZZ(K)
  100 CONTINUE
C
C     CALCULATION OF THE SOURCE TERM
C
      DO 101 I=NVST+2,NI
         DO 101 J=3,NJ
            DO 101 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AWW(I,J,K) * VPD(I-2,J,K)
 101  CONTINUE
C
      DO 102 I=NVST,NI-2
         DO 102 J=3,NJ
            DO 102 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AEE(I,J,K) * VPD(I+2,J,K)
 102  CONTINUE
C
      DO 103 I=NVST,NI
         DO 103 J=4,NJ
            DO 103 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ASS(I,J,K) * VPD(I,J-2,K)
 103  CONTINUE
C
      DO 104 I=NVST,NI
         DO 104 J=3,NJM1
            DO 104 K=2,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ANN(I,J,K) * VPD(I,J+2,K)
 104  CONTINUE
C
      DO 105 I=NVST,NI
         DO 105 J=3,NJ
            DO 105 K=4,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ABB(I,J,K) * VPD(I,J,K-2)
 105  CONTINUE
C
      DO 106 I=NVST,NI
         DO 106 J=3,NJ
            DO 106 K=2,NK-2
               SU(I,J,K) = SU(I,J,K)
     &                   + AFF(I,J,K) * VPD(I,J,K+2)
 106  CONTINUE
C
C        BOUNDARY CONDITIONS (TOP AND BOTTOM WALL)
C
```

```
      DO 121 K=2,NK
         DO 121 I=NVST,NI
            AS(I,3,K) = 0.
            SU(I,3,K) = SU(I,3,K)
     &                - ASS(I,3,K) * VPD(I,3,K)
            SU(I,NJ,K) = SU(I,NJ,K)
     &                 - ANN(I,NJ,K) * VPD(I,NJ,K)
            AN(I,NJ,K) = 0.
  121 CONTINUE
C
C        LEFT AND RIGHT WALL
C
      DO 122 K=2,NK
         DO 122 J=3,NJ
            AP(NVST,J,K) = AP(NVST,J,K) + AW(NVST,J,K)
     &                   - AWW(NVST,J,K)
            AW(NVST,J,K) = 0.
            SU(NVST+1,J,K) = SU(NVST+1,J,K)
     &                     - AWW(NVST+1,J,K) * VPD(NVST,J,K)
            SU(NIM1,J,K) = SU(NIM1,J,K)
     &                   - AEE(NIM1,J,K) * VPD(NI,J,K)
            AP(NI,J,K) = AP(NI,J,K) + AE(NI,J,K)
     &                 - AEE(NI,J,K)
            AE(NI,J,K) = 0.
  122 CONTINUE
C
C        FRONT AND BACK WALL
C
      DO 123 J=3,NJ
         DO 123 I=NVST,NI
            AP(I,J,2) = AP(I,J,2) + AB(I,J,2) - ABB(I,J,2)
            SU(I,J,3) = SU(I,J,3)
     &                - ABB(I,J,3) * VPD(I,J,2)
            SU(I,J,NKM1) = SU(I,J,NKM1)
     &                   - AFF(I,J,NKM1) * VPD(I,J,NK)
            AB(I,J,2) = 0.
            AP(I,J,NK) = AP(I,J,NK) + AF(I,J,NK) - AFF(I,J,NK)
            AF(I,J,NK) = 0.
  123 CONTINUE
C
      IF (NCHP .NE. 0) THEN
C
C        V-VELOCITY SET TO ZERO IN CHIP
C
         DO 130 M=1,NJCHIP
            DO 130 N=1,NKCHIP
               DO 130 I=IBGN,IEND
                  DO 130 J=JBGN(M),JEND(M)+1
                     DO 130 K=KBGN(N),KEND(N)
                        AP(I,J,K) = small
                        AW(I,J,K) = 0.0
                        AE(I,J,K) = 0.0
                        AS(I,J,K) = 0.0
                        AN(I,J,K) = 0.0
                        AB(I,J,K) = 0.0
                        AF(I,J,K) = 0.0
                        SU(I,J,K) = 0.0
  130    CONTINUE
C
C        CHIP BOUNDARY CONDITIONS (TOP AND BOTTOM)
```

```
C
         DO 131 M=1,NJCHIP
            DO 131 N=1,NKCHIP
               DO 131 I=IBGN,IEND
                  DO 131 K=KBGN(N),KEND(N)
                     JJS = JEND(M)
                     JJS2 = JEND(M) + 2
                     JJN1 = JBGN(M) - 1
                     JJP1 = JBGN(M) + 1
                     AS(I,JJS2,K) = 0.
                     SU(I,JJS2,K) = SU(I,JJS2,K)
     &                            - ASS(I,JJS2,K) * VPD(I,JJS,K)
     &                            - ASS(I,JJS2,K) * VPD(I,JJS2,K)
                     SU(I,JJN1,K) = SU(I,JJN1,K)
     &                            - ANN(I,JJN1,K) * VPD(I,JJP1,K)
     &                            - ANN(I,JJN1,K) * VPD(I,JJN1,K)
                     AN(I,JJN1,K) = 0.
  131 CONTINUE
C
C        FRONT AND BACK WALL
C
         DO 132 M=1,NJCHIP
            DO 132 N=1,NKCHIP
               DO 132 I=IBGN,IEND
                  DO 132 J=JBGN(M),JEND(M)+1
                     KKB = KEND(N)
                     KKB1 = KEND(N) + 1
                     KKB2 = KEND(N) + 2
                     KKF = KBGN(N)
                     KKF1 = KBGN(N) - 1
                     KKF2 = KBGN(N) - 2
                     AP(I,J,KKB1) = AP(I,J,KKB1)
     &                            + AB(I,J,KKB1) - ABB(I,J,KKB1)
                     AB(I,J,KKB1) = 0.
                     SU(I,J,KKB2) = SU(I,J,KKB2)
     &                            - ABB(I,J,KKB2) * VPD(I,J,KKB)
     &                            - ABB(I,J,KKB2) * VPD(I,J,KKB1)
                     SU(I,J,KKF2) = SU(I,J,KKF2)
     &                            - AFF(I,J,KKF2) * VPD(I,J,KKF)
     &                            - AFF(I,J,KKF2) * VPD(I,J,KKF1)
                     AP(I,J,KKF1) = AP(I,J,KKF1)
     &                            + AF(I,J,KKF1) - AFF(I,J,KKF1)
                     AF(I,J,KKF1) = 0.
  132 CONTINUE
C
C                 RIGHT WALL
C
         DO 133 M=1,NJCHIP
            DO 133 N=1,NKCHIP
               DO 133 J=JBGN(M),JEND(M)
                  DO 133 K=KBGN(N),KEND(N)
                     II = IEND
                     II1 = IEND + 1
                     II2 = IEND + 2
                     AP(II1,J,K) = AP(II1,J,K)
     &                           + AW(II1,J,K) - AWW(II1,J,K)
                     SU(II2,J,K) = SU(II2,J,K)
     &                           - AWW(II2,J,K) * VPD(II,J,K)
     &                           - AWW(II2,J,K) * VPD(II1,J,K)
                     AW(II1,J,K) = 0.
```

```
  133    CONTINUE
       ENDIF
C
C        SOLVE THE LINEARISED V MOMENTUM EQUATION
C
       CALL SIP (NVST,3,2,NI,NJ,NK,V)
C
C        SET UP SU FOR CALCULATING DV AND SOLVE
C
       DO 200 K=2,NK
          DO 200 J=3,NJ
             DO 200 I=NVST,NI
                SU(I,J,K) = DZZ(K) * DXX(I)
  200  CONTINUE
C
       CALL SIP (NVST,3,2,NI,NJ,NK,DV)
C
       END
C***************************************************************************
       SUBROUTINE CALW
C
C      THIS SUBROUTINE ASSEMBLES THE W-MOMENTUM EQUATION AND
C      BOUNDARY CONDITION AND SOLVES FOR THE W-VELOCITY (Z DIRECTION)
C
       implicit real*8 (a-h,o-z)
       common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
       common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
       common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
       common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
       common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
       common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
       common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
       common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
       COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
       common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
       common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
       COMMON/SCHEME/QUICK,UPWIND
       common/diff/alc,als,thot,tcool,tavg
       common/array/njchip,nkchip,ichoice
       common/tol/small,eps,sormax
       common/ifirst/nust,nvst,nwst,npst
C
C              CALCULATE COEFFICIENTS
C
       do 100 k=3,nk
          do 100 j=2,nj
             do 100 i=nwst,ni
                voldt = dxx(i) * dyy(j) * dzzs(k) / dtime
C
```

```
C      CALCULATE INTERFACIAL KINEMATIC VISCOSITIES
C         USING THE HARMONIC MEAN FORMULATION
C
            visw = visco(i,j,k) * visco(i-1,j,k)
     &           * (dxx(i-1) + dxx(i)) / (dxx(i-1)
     &           * visco(i,j,k) + dxx(i) * visco(i-1,j,k))
            vise = visco(i,j,k) * visco(i+1,j,k)
     &           * (dxx(i+1) + dxx(i)) / (dxx(i+1)
     &           * visco(i,j,k) + dxx(i) * visco(i+1,j,k))
            viss = visco(i,j,k) * visco(i,j-1,k)
     &           * (dyy(j-1) + dyy(j)) / (dyy(j-1)
     &           * visco(i,j,k) + dyy(j) * visco(i,j-1,k))
            visn = visco(i,j,k) * visco(i,j+1,k)
     &           * (dyy(j+1) + dyy(j)) / (dyy(j+1)
     &           * visco(i,j,k) + dyy(j) * visco(i,j+1,k))
            visb = visco(i,j,k-1)
            visf = visco(i,j,k)
c
            dzxs = dzzs(k) * dxx(i)
            dxys = dxx(i) * dyy(j)
            dyzs = dyy(j) * dzzs(k)
            zxoyn = dzxs / dyys(j+1)
            zxoys = dzxs / dyys(j)
            xyozf = dxys / dzz(k)
            xyozb = dxys / dzz(k-1)
            yzoxe = dyzs / dxxs(i+1)
            yzoxw = dyzs / dxxs(i)
c
            GN = V(I,J+1,K)
            GNB = V(I,J+1,K-1)
            GS = V(I,J,K)
            GSB = V(I,J,K-1)
            GE = U(I+1,J,K)
            GEB = U(I+1,J,K-1)
            GW = U(I,J,K)
            GWB = U(I,J,K-1)
            GF = W(I,J,K+1)
            GP = W(I,J,K)
            GB = W(I,J,K-1)
c
            cn = (gn * dzz(k-1) + gnb * dzz(k))
     &         / (dzz(k-1) + dzz(k)) * dzxs
            cs = (gs * dzz(k-1) + gsb * dzz(k))
     &         / (dzz(k-1) + dzz(k)) * dzxs
            ce = (ge * dzz(k-1) + geb * dzz(k))
     &         / (dzz(k-1) + dzz(k)) * dyzs
            cw = (gw * dzz(k-1) + gwb * dzz(k))
     &         / (dzz(k-1) + dzz(k)) * dyzs
            cf = 0.5 * (gf + gp) * dxys
            cb = 0.5 * (gb + gp) * dxys
c
            vise1 = yzoxe * vise
            visw1 = yzoxw * visw
            visn1 = zxoyn * visn
            viss1 = zxoys * viss
            visf1 = xyozf * visf
            visb1 = xyozb * visb
c
      AE(I,J,K) = VISE1 + (- 0.5 * CE * DXX(I) / DXXS(I+1)
     &          + 0.125 * (abs(ce) + ce) * dxx(i) * dxx(i+1)
```

```
     &          / (dxxs(i+1) * (dxxs(i) + dxxs(i+1)))
     &          + 0.125 * (abs(ce) - ce) * dxx(i) * dxx(i+1)
     &          / (dxxs(i+1) * dxxs(i+2))
     &          + 0.125 * (abs(cw) - cw) * dxx(i-1) * dxx(i)
     &          / (DXXS(I+1) * (DXXS(I) + DXXS(I+1)))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CE) - CE))
      AW(I,J,K) = VISW1 + (0.5 * CW * DXX(I) / DXXS(I)
     &          + 0.125 * (abs(cw) + cw) * dxx(i) * dxx(i-1)
     &          / (dxxs(i-1) * dxxs(i))
     &          + 0.125 * (abs(cw) - cw) * dxx(i-1) * dxx(i)
     &          / (dxxs(i) * (dxxs(i) + dxxs(i+1)))
     &          + 0.125 * (abs(ce) + ce) * dxx(i+1) * dxx(i)
     &          / (DXXS(I) * (DXXS(I) + DXXS(I+1)))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CW) + CW))
      AN(I,J,K) = VISN1 + (- 0.5 * CN * DYY(J) / DYYS(J+1)
     &          + 0.125 * (abs(cn) + cn) * dyy(j) * dyy(j+1)
     &          / (dyys(j+1) * (dyys(j) + dyys(j+1)))
     &          + 0.125 * (abs(cn) - cn) * dyy(j) * dyy(j+1)
     &          / (dyys(j+1) * dyys(j+2))
     &          + 0.125 * (abs(cs) - cs) * dyy(j-1) * dyy(j)
     &          / (DYYS(J+1) * (DYYS(J) + DYYS(J+1)))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CN) - CN))
      AS(I,J,K) = VISS1 + (0.5 * CS * DYY(J) / DYYS(J)
     &          + 0.125 * (abs(cs) + cs) * dyy(j) * dyy(j-1)
     &          / (dyys(j-1) * dyys(j))
     &          + 0.125 * (abs(cs) - cs) * dyy(j-1) * dyy(j)
     &          / (dyys(j) * (dyys(j) + dyys(j+1)))
     &          + 0.125 * (abs(cn) + cn) * dyy(j+1) * dyy(j)
     &          / (DYYS(J) * (DYYS(J) + DYYS(J+1)))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CS) + CS))
      AF(I,J,K) = VISF1 + (- 0.5 * CF + 0.0625 * (ABS(CF) + CF)
     &          * dzz(k) / dzzs(k) + 0.125 * (abs(cf) - cf)
     &          * dzz(k) / dzz(k+1) + 0.0625 * (abs(cb) - cb)
     &          * DZZ(K-1)**2 / (DZZ(K) * DZZS(K))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CF) - CF))
      AB(I,J,K) = VISB1 + (0.5 * CB + 0.125 * (ABS(CB) + CB)
     &          * dzz(k-1) / dzz(k-2) + 0.0625 * (abs(cb) - cb)
     &          * dzz(k-1) / dzzs(k) + 0.0625 * (abs(cf) + cf)
     &          * DZZ(K)**2 / (DZZ(K-1) * DZZS(K))) * QUICK
     &          + UPWIND * ( 0.5 * (ABS(CB) + CB))
c
      AEE(I,J,K) = - 0.125 * QUICK
     &           * (ABS(CE) - CE) * DXX(I) * DXX(I+1)
     &           / (DXXS(I+2) * (DXXS(I+1) + DXXS(I+2)))
      AWW(I,J,K) = - 0.125 * QUICK
     &           * (ABS(CW) + CW) * DXX(I) * DXX(I-1)
     &           / (DXXS(I-1) * (DXXS(I-1) + DXXS(I)))
      ANN(I,J,K) = - 0.125 * QUICK
     &           * (ABS(CN) - CN) * DYY(J) * DYY(J+1)
     &           / (DYYS(J+2) * (DYYS(J+1) + DYYS(J+2)))
      ASS(I,J,K) = - 0.125 * QUICK
     &           * (ABS(CS) + CS) * DYY(J) * DYY(J-1)
     &           / (DYYS(J-1) * (DYYS(J-1) + DYYS(J)))
      AFF(I,J,K) = - 0.0625 * QUICK
     &           * (ABS(CF) - CF) * DZZ(K)**2
     &           / (DZZ(K+1) * DZZS(K+1))
      ABB(I,J,K) = - 0.0625 * QUICK
     &           * (ABS(CB) + CB) * DZZ(K-1)**2
     &           / (DZZ(K-2) * DZZS(K-1))
      AP(I,J,K) = AE(I,J,K) + AW(I,J,K) + AN(I,J,K) + AS(I,J,K)
```

```
     &          + AF(I,J,K) + AB(I,J,K) + AWW(I,J,K)
     &          + AEE(I,J,K) + ASS(I,J,K) + ANN(I,J,K)
     &          + ABB(I,J,K) + AFF(I,J,K) + VOLDT
      SU(I,J,K) = VOLDT * WOD(I,J,K)
     &          + DXYS * (P(I,J,K-1) - P(I,J,K))
  100 continue
C
C     CALCULATION OF THE SOURCE TERM
C
      DO 101 I=NWST+2,NI
         DO 101 J=2,NJ
            DO 101 K=3,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AWW(I,J,K) * WPD(I-2,J,K)
 101  CONTINUE
C
      DO 102 I=NWST,NI-2
         DO 102 J=2,NJ
            DO 102 K=3,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + AEE(I,J,K) * WPD(I+2,J,K)
 102  CONTINUE
C
      DO 103 I=NWST,NI
         DO 103 J=4,NJ
            DO 103 K=3,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ASS(I,J,K) * WPD(I,J-2,K)
 103  CONTINUE
C
      DO 104 I=NWST,NI
         DO 104 J=2,NJ-2
            DO 104 K=3,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ANN(I,J,K) * WPD(I,J+2,K)
 104  CONTINUE
C
      DO 105 I=NWST,NI
         DO 105 J=2,NJ
            DO 105 K=4,NK
               SU(I,J,K) = SU(I,J,K)
     &                   + ABB(I,J,K) * WPD(I,J,K-2)
 105  CONTINUE
C
      DO 106 I=NWST,NI
         DO 106 J=2,NJ
            DO 106 K=3,NKM1
               SU(I,J,K) = SU(I,J,K)
     &                   + AFF(I,J,K) * WPD(I,J,K+2)
 106  CONTINUE
C
C         BOUNDARY CONDITIONS (TOP AND BOTTOM)
C
      DO 121 K=3,NK
         DO 121 I=NWST,NI
            AP(I,2,K) = AP(I,2,K) + AS(I,2,K) - ASS(I,2,K)
            SU(I,3,K) = SU(I,3,K) - ASS(I,3,K) * WPD(I,2,K)
            SU(I,NJM1,K) = SU(I,NJM1,K)
     &                   - ANN(I,NJM1,K) * WPD(I,NJ,K)
            AS(I,2,K) = 0.
```

```
          AP(I,NJ,K) = AP(I,NJ,K) + AN(I,NJ,K) - ANN(I,NJ,K)
          AN(I,NJ,K) = 0.
  121 CONTINUE
C
C          LEFT AND RIGHT WALL
C
      DO 122 K=3,NK
         DO 122 J=2,NJ
          AP(NWST,J,K) = AP(NWST,J,K) + AW(NWST,J,K)
     &                 - AWW(NWST,J,K)
          AW(NWST,J,K) = 0.
          SU(NWST+1,J,K) = SU(NWST+1,J,K)
     &                   - AWW(NWST+1,J,K) * WPD(NWST,J,K)
          SU(NIM1,J,K) = SU(NIM1,J,K)
     &                 - AEE(NIM1,J,K) * WPD(NI,J,K)
          AP(NI,J,K) = AP(NI,J,K) + AE(NI,J,K) - AEE(NI,J,K)
          AE(NI,J,K) = 0.
  122 CONTINUE
C
C          FRONT AND BACK WALL
C
      DO 123 J=2,NJ
         DO 123 I=NWST,NI
          AB(I,J,3) = 0.
          SU(I,J,3) = SU(I,J,3) - ABB(I,J,3) * WPD(I,J,3)
          SU(I,J,NK) = SU(I,J,NK)
     &               - AFF(I,J,NK) * WPD(I,J,NK)
          AF(I,J,NK) = 0.
 123  CONTINUE
C
      if (nchp .ne. 0) then
C
C         W-VELOCITY SET TO ZERO IN CHIP
C
         DO 130 M=1,NJCHIP
            DO 130 N=1,NKCHIP
               DO 130 I=IBGN,IEND
                  DO 130 J=JBGN(M),JEND(M)
                     DO 130 K=KBGN(N),KEND(N)+1
                        AP(I,J,K) = small
                        AW(I,J,K) = 0.0
                        AE(I,J,K) = 0.0
                        AS(I,J,K) = 0.0
                        AN(I,J,K) = 0.0
                        AB(I,J,K) = 0.0
                        AF(I,J,K) = 0.0
                        SU(I,J,K) = 0.0
  130    CONTINUE
C
C        CHIP BOUNDARY CONDITIONS (TOP AND BOTTOM)
C
         DO 131 M=1,NJCHIP
            DO 131 N=1,NKCHIP
               DO 131 I=IBGN,IEND
                  DO 131 K=KBGN(N),KEND(N)+1
                     JJS = JEND(M)
                     JJS1 = JEND(M) + 1
                     JJS2 = JEND(M) + 2
                     JJN = JBGN(M)
                     JJN1 = JBGN(M) - 1
```

```
                           JJN2 = JBGN(M) - 2
                           AP(I,JJS1,K) = AP(I,JJS1,K)
     &                                  + AS(I,JJS1,K) - ASS(I,JJS1,K)
                           AS(I,JJS1,K) = 0.
                           SU(I,JJS2,K) = SU(I,JJS2,K)
     &                                  - ASS(I,JJS2,K) * WPD(I,JJS,K)
     &                                  - ASS(I,JJS2,K) * WPD(I,JJS1,K)
                           SU(I,JJN2,K) = SU(I,JJN2,K)
     &                                  - ANN(I,JJN2,K) * WPD(I,JJN,K)
     &                                  - ANN(I,JJN2,K) * WPD(I,JJN1,K)
                           AP(I,JJN1,K) = AP(I,JJN1,K)
     &                                  + AN(I,JJN1,K) - ANN(I,JJN1,K)
                           AN(I,JJN1,K) = 0.
 131 CONTINUE
C
C         FRONT AND BACK WALL
C
          DO 132 M=1,NJCHIP
             DO 132 N=1,NKCHIP
                DO 132 I=IBGN,IEND
                   DO 132 J=JBGN(M),JEND(M)
                      KKB = KEND(N)
                      KKB2 = KEND(N) + 2
                      KKP1 = KBGN(N) + 1
                      KKF1 = KBGN(N) - 1
                      AB(I,J,KKB2) = 0.
                      SU(I,J,KKB2) = SU(I,J,KKB2)
     &                             - ABB(I,J,KKB2) * WPD(I,J,KKB)
     &                             - ABB(I,J,KKB2) * WPD(I,J,KKB2)
                      SU(I,J,KKF1) = SU(I,J,KKF1)
     &                             - AFF(I,J,KKF1) * WPD(I,J,KKP1)
     &                             - AFF(I,J,KKF1) * WPD(I,J,KKF1)
                      AF(I,J,KKF1) = 0.
 132      CONTINUE
C
C                  RIGHT WALL
C
          DO 133 M=1,NJCHIP
             DO 133 N=1,NKCHIP
                DO 133 J=JBGN(M),JEND(M)
                   DO 133 K=KBGN(N),KEND(N)+1
                      II = IEND
                      II1 = IEND + 1
                      II2 = IEND + 2
                      AP(II1,J,K) = AP(II1,J,K)
     &                            + AW(II1,J,K) - AWW(II1,J,K)
                      SU(II2,J,K) = SU(II2,J,K)
     &                            - AWW(II2,J,K) * WPD(II,J,K)
     &                            - AWW(II2,J,K) * WPD(II1,J,K)
                      AW(II1,J,K) = 0.
 133      CONTINUE
        ENDIF
C
C       SOLVE THE LINEARISED W MOMENTUM EQUATIONS
C
        CALL SIP (NWST,2,3,NI,NJ,NK,W)
C
C       SET UP SU FOR CALCULATING DW AND SOLVE
C
        DO 200 K=3,NK
```

```
         DO 200 J=2,NJ
            DO 200 I=NWST,NI
               su(i,j,k) = dxx(i) * dyy(j)
  200 CONTINUE
c
      CALL SIP (NWST,2,3,NI,NJ,NK,DW)
c
      END
c************************************************************************
      SUBROUTINE CALP
C
C     THIS SUBROUTINE ASSEMBLES THE PRESSURE CORRECTION EQUATION.
C     THE PRESSURE CORRECTION IS SET TO ZERO AT ALL THE BOUNDARIES
C
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
      common/mean/t_mean(12,16,16),u_mean(12,16,16),
     &            v_mean(12,16,16),w_mean(12,16,16),
     &            p_mean(12,16,16)
      COMMON/SCHEME/QUICK,UPWIND
      common/array/njchip,nkchip,ichoice
      common/ifirst/nust,nvst,nwst,npst
c
c          SET UP COEFFICIENTS
c
      do 100 k=2,nk
         do 100 j=2,nj
            do 100 i=npst,ni
               dxys = dxx(i) * dyy(j)
               dyzs = dyy(j) * dzz(k)
               dzxs = dzz(k) * dxx(i)
               an(i,j,k) = dzxs * dv(i,j+1,k)
               as(i,j,k) = dzxs * dv(i,j,k)
               ae(i,j,k) = dyzs * du(i+1,j,k)
               aw(i,j,k) = dyzs * du(i,j,k)
               af(i,j,k) = dxys * dw(i,j,k+1)
               ab(i,j,k) = dxys * dw(i,j,k)
               cn = v(i,j+1,k) * dzxs
               cs = v(i,j,k) * dzxs
               ce = u(i+1,j,k) * dyzs
```

```
                cw = u(i,j,k) * dyzs
                cf = w(i,j,k+1) * dxys
                cb = w(i,j,k) * dxys
                smp(i,j,k) = - ce + cw - cn + cs - cf + cb
                su(i,j,k) = smp(i,j,k)
  100 continue
c
c        BOUNDARY CONDITIONS (TOP AND BOTTOM)
c
      do 101 k=2,nk
         do 101 i=npst,ni
            as(i,2,k) = 0.
            an(i,nj,k) = 0.
  101 continue
c
c         LEFT AND RIGHT WALL
c
      do 102 k=2,nk
         do 102 j=2,nj
            aw(npst,j,k) = 0.
            ae(ni,j,k) = 0.
  102 continue
c
c         FRONT AND BACK WALL
c
      do 103 i=npst,ni
         do 103 j=2,nj
            ab(i,j,2) = 0.
            af(i,j,nk) = 0.
 103  continue
c
c       CALCULATE AP
c
      do 200 j=2,nj
         do 200 i=npst,ni
            do 200 k=2,nk
            ap(i,j,k) = an(i,j,k) + as(i,j,k) + ae(i,j,k)
     &                + aw(i,j,k) + af(i,j,k) + ab(i,j,k)
  200 continue
c
c       SOLVE THE PRESSURE CORRECTION EQUATION
c
      call sip (npst,2,2,ni,nj,nk,pp)
c
c       U VELOCITY CORRECTION
c
      do 201 i=nust,ni
         do 201 j=2,nj
            do 201 k=2,nk
         U(I,J,K) = U(I,J,K) + DU(I,J,K) * (PP(I-1,J,K) - PP(I,J,K))
  201 continue
c
c       V VELOCITY CORRECTION
c
      do 202 j=3,nj
         do 202 k=2,nk
            do 202 i=nvst,ni
         V(I,J,K) = V(I,J,K) + DV(I,J,K) * (PP(I,J-1,K) - PP(I,J,K))
  202 continue
c
```

```
c       W VELOCITY CORRECTION
c
      do 203 k=3,nk
         do 203 i=nwst,ni
            do 203 j=2,nj
         W(I,J,K) = W(I,J,K) + DW(I,J,K) * (PP(I,J,K-1) - PP(I,J,K))
 203  continue
c
c       PRESSURE CORRECTION
c
      do 204 j=2,nj
         do 204 i=npst,ni
            do 204 k=2,nk
               p(i,j,k) = p(i,j,k) + pp(i,j,k)
               PP(I,J,K) = 0.
  204 continue
c
c       RECALCULATE MASS FLUX ERRORS AFTER U,V,W,P,CORRECTIONS
c
      sorsum = 0.
      resorm(iter) = 0.
      do 205 j=2,nj
         do 205 i=npst,ni
            do 205 k=2,nk
               dxys = dxx(i) * dyy(j)
               dyzs = dyy(j) * dzz(k)
               dzxs = dzz(k) * dxx(i)
               cn = v(i,j+1,k) * dzxs
               cs = v(i,j,k) * dzxs
               ce = u(i+1,j,k) * dyzs
               cw = u(i,j,k) * dyzs
               cf = w(i,j,k+1) * dxys
               cb = w(i,j,k) * dxys
               smp(i,j,k) = - ce + cw - cn + cs - cf + cb
               sorsum = sorsum + smp(i,j,k)
               resorm(iter) = resorm(iter) + abs(smp(i,j,k))
  205 continue
      end
C ************************************************************************
      subroutine nu
C
C    THIS SUBROUTINES DETERMINES THE OVERALL HEAT BALANCE FOR THE
C    ENTIRE ENCLOSURE
C
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/count/nt,nmax,imax,itmax,krun,nprint
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &               nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &                wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &               w(12,16,16),p(12,16,16)
```

```
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip,ichoice
c
      RNUC = 0.
      RNUH = 0.
      QCHIP = 0.
      quns = 0.
      QCRT = 0.
      do 10 i=1,ni
         do 10 j=1,nj
            do 10 k=1,nk
               quns = quns + (t(i,j,k) - tod(i,j,k)) * dxx(i)
     &                    * dyy(j) * dzz(k) / dtime
               QCRT = QCRT + T(I,J,K) * SMP(I,J,K)
   10 continue
      a1 = 1. / dyys(2)
      a2 = 1. / dyys(njp1)
      DO 20 I=2,NI
        DO 20 K=2,NK
          DTDYH = a1 * (T(I,2,K) - T(I,1,K)) * ALPHA(I,2,K)
          DTDYC = a2 * (T(I,NJP1,K) - T(I,NJ,K)) * ALPHA(I,NJ,K)
          RNUC = RNUC - DTDYC * dzz(k) * dxx(i)
          RNUH = RNUH - DTDYH * dzz(k) * dxx(i)
   20 CONTINUE
C
C     CALCULATE HEAT GENERATED FROM CHIP
C
      do 30 m=1,njchip
         do 30 n=1,nkchip
            do 30 i=ibgn,iend
               do 30 j=jbgn(m),jend(m)
                  do 30 k=kbgn(n),kend(n)
                     qchip = qchip + alc * dxx(i) * dyy(j) * dzz(k)
   30 continue
      AA = 1. / (bth * wth)
      RNUH = RNUH * AA
      RNUC = RNUC * AA
      QCHIP = QCHIP * AA
      QCRT = QCRT * AA
      QALL = QCHIP + RNUH - RNUC
      WRITE(6,500) NT,XTIME,ITER,RESORM(ITER),SORSUM,
     &             RNUC,RNUH,QCHIP,QALL,QUNS,QCRT
  500 FORMAT(1X, 'NT =',I9,5X,'TIME=',f7.4,/,
     &       1X, 'ITER=',I2,  5X,'SOURCE=',F9.6,5X,'SORSUM=',F9.6,/,
     &       1X, 'NUC=',F10.6, 5X,'NUH=',F10.6,/,
     &       1X, 'QCHIP=',F10.6,5X,'QALL=',F10.6,/,
     &       1X, 'QUNS=',F10.6,5X,'QCRT=',F10.6,/)
      END
c************************************************************************
      SUBROUTINE SIP(IST,JST,KST,ISP,JSP,KSP,PHI)
      implicit real*8 (a-h,o-z)
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
```

```
     &              af(12,16,16),su(12,16,16)
      common/abc/kmax,n1,k1
      common/count/nt,nmax,imax,itmax,krun,nprint
      COMMON/ MAT/A1(3072),A2(3072),A3(3072),A4(3072),A5(3072),
     &             A6(3072),A7(3072),B(3072),D(3072)
      DIMENSION X1(3072),X2(3072),XR1(3072),
     &          S(12,16,16),PHI(12,16,16)
C
      N1   = ISP - IST + 1
      K1   = (ISP - IST + 1) * (JSP - JST + 1)
      KMAX = (ISP - IST + 1) * (JSP - JST + 1) * (KSP - KST + 1)
      EPS2 = KMAX * EPS**2
C
C    DEFINING A SCRATCH ARRAY FOR FURTHER MODIFICATIONS
C
      DO 9 K=KST,KSP
         DO 9 J=JST,JSP
            DO 9 I=IST,ISP
                 S(I,J,K) = SU(I,J,K)
    9 CONTINUE
C
C    EXTRACT MATRIX A, RHS VECTOR B AND VECTOR XU-AN INITIAL
C                      GUESS FROM GIVEN DATA
C
      DO 2 K=KST,KSP
         DO 2 J=JST,JSP
            DO 2 I=IST,ISP
               M   = I - IST + 1 + (J - JST) * N1 + (K - KST) * K1
               A1(M)  = AP(I,J,K)
               A2(M)  = -AE(I,J,K)
               A3(M)  = -AW(I,J,K)
               A4(M)  = -AN(I,J,K)
               A5(M)  = -AS(I,J,K)
               A6(M)  = -AF(I,J,K)
               A7(M)  = -AB(I,J,K)
               B(M)  = S(I,J,K)
               X1(M)  = PHI(I,J,K)
    2 CONTINUE
C
C        THE DIAGONAL ELEMENT OF L IN THE INCOMPLETE LU
C                      DECOMPOSITION
C
      D(1)  = A1(1)
      DO 113 I=2,N1
         D(I)  = A1(I)  - A2(I-1)  * A3(I)  / D(I-1)
  113 CONTINUE
      DO 114 I=N1+1,K1
         D(I)  = A1(I)  - A2(I-1)  * A3(I)  / D(I-1)  - A4(I-N1)
     &           * A5(I)  / D(I-N1)
  114 CONTINUE
      DO 115 I=K1+1,KMAX
         D(I)  = A1(I)  - A2(I-1)  * A3(I)  / D(I-1)
     &                  - A4(I-N1)  * A5(I)  / D(I-N1)
     &                  - A6(I-K1)  * A7(I)  / D(I-K1)
  115 CONTINUE
      DO 116 I=1,KMAX
         D(I)  = 1. / D(I)
  116 CONTINUE
      ITR = 0
C
```

```
c               THE MAIN ITERATION LOOP BEGINS
c
   7  CONTINUE
      ITR = ITR + 1
      CALL RES (X1,XR1)
      CALL XL (XR1,X2)
      DO 117 I=1,KMAX
         X2(I) = X2(I) + X1(I)
  117 CONTINUE
      V2 = 0.
      DO 55 I=1,KMAX
         V2 = V2 + XR1(I)**2
   55 CONTINUE
      X2MAG = 0.
      DO 56 I=1,KMAX
         X2MAG = X2MAG + X2(I)**2
   56 continue
      if (x2mag .gt. 0.0001) v2 = v2 / x2mag

c
c             RELABELING THE VECTORS BEFORE NEXT ITERATION
c
      do 50 i=1,kmax
         X1(I) = X2(I)
  50  continue
      IF (V2 .GT. EPS2 .AND. ITR .LT. IMAX) GO TO 7
c
c     RECOVER THE SOLUTION VECTOR AND RETURN THE AVERAGE
c             PHI AS WELL AS THE RMS ERROR
c
      IF (MOD(NT,nprint) .EQ. 0) PRINT *, ITR,V2
      do 3 k=kst,ksp
         do 3 j=jst,jsp
            do 3 i=ist,isp
               m  = i - ist + 1 + (j - jst) * n1 + (k - kst) * k1
               phi(i,j,k) = x1(m)
    3 continue
      end
c*****************************************************************************
      SUBROUTINE RES(X1,X2)
      implicit real*8 (a-h,o-z)
      COMMON/ABC/KMAX,N1,K1
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/ mat/a1(3072),a2(3072),a3(3072),a4(3072),a5(3072),
     &            a6(3072),a7(3072),b(3072),d(3072)
      dimension x1(3072),x2(3072)
C
C
C     CALCULATE Ax1
C
      x2(1) = a1(1) * x1(1) + a2(1) * x1(2)
     &      + a4(1) * x1(1+n1) + a6(1) * x1(1+k1)
      do 10 i=2,n1
         x2(i) = a1(i) * x1(i) + a2(i) * x1(i+1)
     &         + a4(i) * x1(i+n1) + a6(i) * x1(i+k1)
     &         + a3(i) * x1(i-1)
   10 continue
      do 20 i=n1+1,k1
         x2(i) = a1(i) * x1(i) + a2(i) * x1(i+1)
```

```
     &          + a4(i) * x1(i+n1) + a5(i) * x1(i-n1)
     &          + a3(i) * x1(i-1) + a6(i) * x1(i+k1)
 20   continue
      DO 30 I=K1+1,KMAX-K1
         x2(i) = a1(i) * x1(i) + a2(i) * x1(i+1)
     &          + a4(i) * x1(i+n1) + a5(i) * x1(i-n1)
     &          + a3(i) * x1(i-1) + a6(i) * x1(i+k1)
     &          + a7(i) * x1(i-k1)
 30   continue
      DO 32 I=KMAX-K1+1,KMAX-N1
         x2(i) = a1(i) * x1(i) + a2(i) * x1(i+1)
     &          + a4(i) * x1(i+n1) + a5(i) * x1(i-n1)
     &          + A3(I) * X1(I-1) + A7(I) * X1(I-K1)
 32   CONTINUE
      DO 34 I=KMAX-N1+1,KMAX-1
         x2(i) = a1(i) * x1(i) + a2(i) * x1(i+1)
     &          + A5(I) * X1(I-N1) + A3(I) * X1(I-1)
     &          + A7(I) * X1(I-K1)
 34   CONTINUE
      X2(KMAX) = A1(KMAX) * X1(KMAX) + A5(KMAX) * X1(KMAX-N1)
     &          + A3(KMAX) * X1(KMAX-1) + A7(KMAX) * X1(KMAX-K1)
      do 40 i=1,kmax
         X2(I) = B(I) - X2(I)
 40   continue
      end
c***************************************************************************
      subroutine xl(x1,x3)
      implicit real*8 (a-h,o-z)
      common/abc/kmax,n1,k1
      common/ mat/a1(3072),a2(3072),a3(3072),a4(3072),a5(3072),
     &            a6(3072),a7(3072),b(3072),d(3072)
      dimension x1(3072),x2(3072),x3(3072)
      x2(1) = x1(1) * d(1)
      do 31 i=2,n1
         X2(I) = D(I) * (X1(I) - A3(I) * X2(I-1))
 31   continue
      do 32 i=n1+1,k1
         X2(I) = D(I) * (X1(I) - A3(I) * X2(I-1)
     &          - A5(I) * X2(I-N1))
 32   continue
      do 33 i=k1+1,kmax
         X2(I) = D(I) * (X1(I) - A3(I) * X2(I-1)
     &          - A5(I) * X2(I-N1) - A7(I) * X2(I-K1))
 33   continue
      x3(kmax) = x2(kmax)
      do 67 i=kmax-1,kmax-n1+1,-1
         X3(I) = X2(I) - D(I) * A2(I) * X3(I+1)
 67   continue
      do 68 i=kmax-n1,kmax-k1+1,-1
         X3(I) = X2(I) - D(I)*(A2(I)*X3(I+1) + A4(I)*X3(I+N1))
 68   continue
      do 69 i=kmax-k1,1,-1
         X3(I) = X2(I) - D(I)*(A2(I)*X3(I+1) + A4(I)*X3(I+N1)
     &                        + A6(I)*X3(I+K1))
 69   continue
      end
c***************************************************************************
      SUBROUTINE PROP
C
C     IN THIS SUBROUTINE THE INTERFACIAL THERMAL DIFFUSIVITIES ARE
```

```
C       CALCULATED.
C
        implicit real*8 (a-h,o-z)
        common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &            dxxs(40),dyys(40),dzzs(40)
        common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
        common/tol/small,eps,sormax
        common/dims/h,wth,bth,hchip,wchip,bchip,bsub
        common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &                nip2,njp2,nkp2,iter,nnmax
        common/count/nt,nmax,imax,itmax,krun,nprint
        common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &              isub,ichip,jchip,kchip
        common/diff/alc,als,thot,tcool,tavg
        COMMON/RHOCP/RHS,RHC
        common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
        COMMON/POWER/QQQ,QCOND
        common/array/njchip,nkchip,ichoice
c
        if (ichoice .eq. 1) then
            cond = 0.065 - 7.895E-5 * tavg
            rh = (1.825 - 0.00246 * tavg) * 1000.
            cp = 4186. * (0.2411 + 3.7037E-4 * tavg)
            alp = cond / (rh * cp)
            beta = 0.00246 / (1.825 - 0.00246 * tavg)
            PRINT *,'******** FLUID IS FC75 ************'
        endif
        if (ichoice .eq. 2) then
            cond = 0.071
            rh = (2.002 - 0.00224 * tavg) * 1000.
            cp = 4186. * (0.2411 + 3.7037E-4 * tavg)
            alp = cond / (rh * cp)
            beta = 0.00224 / (2.002 - 0.00224 * tavg)
            PRINT *,'******** FLUID IS FC71 ************'
        endif
C
        if (ichoice .ne. 0) then
            ra = 9.81 * beta * qqq * h**2 * 1.0E-6
     &          / (hchip * bchip * wchip * alc * cond * alp * alp)
            qcond = qqq * 1.0E3
     &          / (h * hchip * bchip * wchip * alc * cond)
            PRINT *,'THE POWER DISSIPATED PER CHIP IS',QQQ
            PRINT *,'THE BOUSSINESQ NUMBER IS',RA
        endif
c


C
C        PRINT THE INPUT PARAMETERS
C
        if (ichoice .eq. 0) then
            print *, 'A CONSTANT VISCOSITY FLUID IS ASSUMED'
            print *,'WHOSE PRANDTL NUMBER IS',pr
            print *,'AND WHOSE BOUSSINESQ NUMBER IS',ra
        endif
c
        PRINT *,'THE ASPECT RATIOS ARE ',BTH,WTH
        PRINT *,'THE TIME STEP IS',DTIME
        PRINT *,'NUMBER OF TIME STEPS',NMAX
        if (krun .eq. 1) then
            print *,'*********THIS IS A RESTART*********'
```

```
      else
          print *,'*********THIS IS NOT A RESTART*****'
      endif
      if (nchp .eq. 1) then
          print *,'*****CHIP AND SUBSTRATE PRESENT*****'
      else
          print *,'****NO CHIP OR SUBSTRATE***'
      endif
C
C           SET UP THE THERMAL DIFFUSIVITY
C            FOR DIFFERENT CONTROL VOLUMES
C
C         FLUID DIFFUSIVITY
C
      DO 10 I=1,NIP1
         DO 10 J=1,NJP1
            DO 10 K=1,NKP1
               ALPHA(I,J,K) = 1.
               RHO(I,J,K) = 1.
   10 CONTINUE
      if (nchp .ne. 0) then
C
C           SUBSTRATE DIFFUSIVITY
C
          DO 15 I=1,IBGN-1
             DO 15 J=1,NJP1
                DO 15 K=1,NKP1
                   ALPHA(I,J,K) = ALS
                   RHO(I,J,K) = RHS
   15     CONTINUE
C
C          CHIP DIFFUSIVITY FOR CHIPS
C
          DO 20 M=1,NJCHIP
             DO 20 N=1,NKCHIP
                do 20 i=ibgn,iend
                   do 20 j=jbgn(m),jend(m)
                      do 20 k=kbgn(n),kend(n)
                         ALPHA(I,J,K) = ALC
                         RHO(I,J,K) = RHC
   20     continue
      endif
      end
c************************************************************************
      subroutine openf
c
c     THIS SUBROUTINE OPENS ALL FILES
c
c     UNIT 8: THIS IS THE INPUT FILE FOR THE FIELD VARIABLES. IT IS
c             USED FOR A RESTART JOB.
c     UNIT 10: THIS FILE STORES THE U,V AND W VELOCITIES FOR A GIVEN
c              LOCATION.
c     UNIT 11: THIS FILE STORES THE FIELD VARIABLES AT PRESCRIBED
c              INTERVALS. THIS IS A CHECKPOINTING PROCEDURE.
c     UNIT 12: THIS FILE STORES THE MEAN FIELD VARIABLES.
c
      OPEN(8,FILE='c21.data',STATUS='UNKNOWN',FORM='UNFORMATTED')
      OPEN(10,FILE='c2v.data',STATUS='UNKNOWN')
      OPEN(11,FILE='c2tmp.data',STATUS='UNKNOWN',FORM='UNFORMATTED')
      OPEN(12,FILE='c2m.data',STATUS='UNKNOWN',FORM='UNFORMATTED')
```

```
      OPEN(13,FILE='c2grd.data',STATUS='UNKNOWN')
      end
c***************************************************************************
      subroutine initio
c
c     IN THIS SUBROUTINE ALL VARIABLES ARE INITIALIZED TO DEFAULTS
c     OR THE FIELD VARIABLES ARE READ.

      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      common/mean/t_mean(12,16,16),u_mean(12,16,16),
     &            v_mean(12,16,16),w_mean(12,16,16),
     &            p_mean(12,16,16)
      common/count/nt,nmax,imax,itmax,krun,nprint
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/scheme/quick,upwind
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip,ichoice
      common/ifirst/nust,nvst,nwst,npst
      DATA PI/3.14159265359/
C
C        SETTING UP THE WEIGHTED AVERAGE
C
      UPWIND = 1.0 - QUICK
c
      qck = 100. * quick
      uwd = 100. * upwind
      print *,'THE SCHEME IS',qck,'PERCENTAGE QUICK AND'
      print *,uwd,'PERCENTAGE UPWIND'
c
c     SET THE I-DIRECTION INDEX FOR CALU,CALV AND CALW
c
      if (nchp .eq. 0) then
          nust = 3
          nvst = 2
          nwst = 2
          npst = 2
      else
          nust = ibgn + 1
          nvst = ibgn
          nwst = ibgn
```

```
            npst = ibgn
        endif
c
c          INITIALIZE ALL VARIABLES TO ZERO OR DEFAULTS
c          BEFORE START OF COMPUTATIONS
c
      DO 205 I=1,NIP1
         DO 205 J=1,NJP1
            DO 205 K=1,NKP1
                 UOD(I,J,K) = 0.
                 U(I,J,K) = 0.
                 UPD(I,J,K) = 0.
                 VOD(I,J,K) = 0.
                 V(I,J,K) = 0.
                 VPD(I,J,K) = 0.
                 W(I,J,K) = 0.
                 WPD(I,J,K) = 0.
                 WOD(I,J,K) = 0.
                 POD(I,J,K) = 0.
                 P(I,J,K) = 0.
                 PPD(I,J,K) = 0.
                 DU(I,J,K) = 0.
                 DV(I,J,K) = 0.
                 DW(I,J,K) = 0.
                 SU(I,J,K) = 0.
                 PP(I,J,K) = 0.
                 AP(I,J,K) = 0.
                 AW(I,J,K) = 0.
                 AE(I,J,K) = 0.
                 AN(I,J,K) = 0.
                 AS(I,J,K) = 0.
                 SMP(I,J,K) = 0.
                 T_MEAN(I,J,K) = 0.
                 U_MEAN(I,J,K) = 0.
                 V_MEAN(I,J,K) = 0.
                 W_MEAN(I,J,K) = 0.
                 P_MEAN(I,J,K) = 0.
  205 CONTINUE
c
C     RESTART JOB IF KRUN IS SET TO ONE
c
      IF (KRUN .EQ. 1) THEN
         READ(8) TOD,UOD,VOD,WOD,POD
      ELSE
         DO 220 J=1,NJP1
           DO 220 I=1,NIP1
              DO 220 K=1,NKP1
                 TOD(I,J,K) = THOT - (THOT - TCOOL)
     &                      * (y(j) + 0.5 * dyy(j))
c                tod(i,j,k) = 0.
                 T(I,J,K) = TOD(I,J,K)
                 TPD(I,J,K) = TOD(I,J,K)
  220         CONTINUE
c
c          AMPLITUDE OF PERTURBATIONS
c
                 A1 = 1. / bth
                 YPER = ROLL * XPER / bth
c
c          U VELOCITY PERTURBATIONS
```

```
c
       do 230 k=2,nk
          DO 230 J=2,NJ
             DO 230 I=3,NI
                U(I,J,K) = - xper * cos(pi * (y(j) + 0.5 * dyy(j)))
     &                   * sin(roll * pi * a1 * x(i))
                UOD(I,J,K) = U(I,J,K)
                UPD(I,J,K) = U(I,J,K)
 230  CONTINUE
c
c        V VELOCITY PERTURBATIONS
c
       do 235 k=2,nk
          DO 235 J=3,NJ
             DO 235 I=2,NI
                V(I,J,K) = yper * sin(pi * y(j))
     &                   * COS(ROLL * PI * A1 * (x(i) + 0.5 * dxx(i)))
                VOD(I,J,K) = V(I,J,K)
                VPD(I,J,K) = V(I,J,K)
 235   CONTINUE
      ENDIF
C
C         SET THE VELOCITIES OUTSIDE THE COMPUTATIONAL
C               DOMAIN TO ZERO
C
      do 505 i=1,nip1
         DO 505 J=1,NJP1
            UOD(I,J,1) = 0.
            UOD(I,J,NKP1) = 0.
            VOD(I,J,1) = 0.
            VOD(I,J,NKP1) = 0.
            WOD(I,J,1) = 0.
            WOD(I,J,2) = 0.
            WOD(I,J,NKP1) = 0.
  505 CONTINUE
      DO 510 I=1,NIP1
         DO 510 K=1,NKP1
            UOD(I,1,K) = 0.
            UOD(I,NJP1,K) = 0.
            VOD(I,1,K) = 0.
            VOD(I,2,K) = 0.
            VOD(I,NJP1,K) = 0.
            WOD(I,1,K) = 0.
            WOD(I,NJP1,K) = 0.
  510 CONTINUE
      DO 520 J=1,NJP1
         DO 520 K=1,NKP1
            UOD(1,J,K) = 0.
            UOD(2,J,K) = 0.
            UOD(NIP1,J,K) = 0.
            VOD(1,J,K) = 0.
            VOD(NIP1,J,K) = 0.
            WOD(1,J,K) = 0.
            WOD(NIP1,J,K) = 0.
  520 CONTINUE
c
c        SET THE VELOCITIES AND PRESSURE IN THE SUBSTRATE TO ZERO
c
      if (nchp .ne. 0) then
          do 530 i=1,ibgn
```

```
            do 530 j=1,njp1
               do 530 k=1,nkp1
                  uod(i,j,k) = 0.
  530     continue
c
         do 540 i=1,ibgn-1
            do 540 j=1,njp1
               do 540 k=1,nkp1
                  vod(i,j,k) = 0.
                  wod(i,j,k) = 0.
                  pod(i,j,k) = 0.
  540     continue
      endif
C
C        INITIALISE U,V,W,T,P
C
      DO 210 K=1,NKP1
         DO 210 J=1,NJP1
            DO 210 I=1,NIP1
               T(I,J,K) = TOD(I,J,K)
               U(I,J,K) = UOD(I,J,K)
               V(I,J,K) = VOD(I,J,K)
               W(I,J,K) = WOD(I,J,K)
               P(I,J,K) = POD(I,J,K)
  210 CONTINUE
      end
c***************************************************************************
      subroutine ploop
c
c     THIS SUBROUTINE INCLUDES THE PRESSURE LOOP. IT INCORPORATES THE
c     SIMPLEX ALGORITHM AND THE INFAMOUS ERROR CONTROL ROUTINE DUE TO
c     THE VENERABLE VINCENT LIU OF THE ARGONNE NATIONAL LABS WHO DEVISED
c     IT AS A HOBBY WHILE AT THE UNIV OF NOTRE DAME.
c
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/fv_int/tpd(12,16,16),upd(12,16,16),vpd(12,16,16),
     &              wpd(12,16,16),ppd(12,16,16)
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/coeff/ap(12,16,16),aw(12,16,16),ae(12,16,16),
     &             as(12,16,16),an(12,16,16),ab(12,16,16),
     &             af(12,16,16),su(12,16,16)
      COMMON/COEF2/AWW(12,16,16),AEE(12,16,16),ASS(12,16,16),
     &             ANN(12,16,16),ABB(12,16,16),AFF(12,16,16)
      common/mean/t_mean(12,16,16),u_mean(12,16,16),
     &            v_mean(12,16,16),w_mean(12,16,16),
     &            p_mean(12,16,16)
      common/count/nt,nmax,imax,itmax,krun,nprint
```

```
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/array/njchip,nkchip,ichoice
      common/diff/alc,als,thot,tcool,tavg
c
      DIMENSION UU(5000),VV(5000),WW(5000)
c       CALL CPUTIME (BEGIN,IIR)
      XTIME = 0.
      NT = 0
c
c    THE MAIN TIME MARCHING LOOP BEGINS HERE
c
  300 continue
c
      NT = NT + 1
c
c    UPDATE THE MEAN VELOCITIES,TEMPERATURE AND
c                PRESSURE
c
      C1 = 1. / FLOAT(NT)
      do 310 k=1,nkp1
         DO 310 J=1,NJP1
            DO 310 I=1,NIP1
               U_MEAN(I,J,K) = (1. - C1) * U_MEAN(I,J,K)
     &                       + C1 * U(I,J,K)
               V_MEAN(I,J,K) = (1. - C1) * V_MEAN(I,J,K)
     &                       + c1 * v(i,j,k)
               W_MEAN(I,J,K) = (1. - C1) * W_MEAN(I,J,K)
     &                       + c1 * w(i,j,k)
               T_MEAN(I,J,K) = (1. - C1) * T_MEAN(I,J,K)
     &                       + c1 * t(i,j,k)
  310 continue
c
c      STORE THE FIELD VARIABLES EVERY 1000 TIME STEPS
c             AS A KIND OF CHECKPOINTING
c
      IF (MOD(NT,nprint) .EQ. 0) THEN
          WRITE(11) T,U,V,W,P
          REWIND 11
      ENDIF
c
c     STOP COMPUTATIONS AT THE MAXIMUM PRESCRIBED
c             TIME STEP
c
      IF(NT .GT. NMAX) THEN
         DO 10 I=1,NMAX
            WRITE(10,*) UU(I),VV(I),WW(I)
   10    CONTINUE
         WRITE(12) T_MEAN,U_MEAN,V_MEAN,W_MEAN,P_MEAN
c          CALL CPUTIME (END,IIR)
          TT = (END - BEGIN) * 1.E-06
          PRINT *,'THE CPUTIME USED WAS',TT
         STOP
      ENDIF
C
      XTIME = XTIME + DTIME
C
C        START CALCULATIONS
C
      ITER = 0
```

```
      JTERM = 0
      JJTERM = 0
C
C     DEFINE THE UPDATED TPD(I,J,K), UPD(I,J,K) AND VPD(I,J,K)
C                               FOR  SU(I,J,K)
C
      DO 48 K=1,NKP1
         DO 48 J=1,NJP1
            DO 48 I=1,NIP1
               TPD(I,J,K) = T(I,J,K)
               UPD(I,J,K) = U(I,J,K)
               VPD(I,J,K) = V(I,J,K)
               WPD(I,J,K) = W(I,J,K)
   48 CONTINUE
c
   29 CONTINUE
      JTERM = JTERM + 1
c
      CALL CALT
C
      DO 2220 I=1,NIP1
         DO 2220 K=1,NKP1
            DO 2220 J=2,NJ
               IF (T(I,J,K) .LT. TCOOL) T(I,J,K) = TCOOL
 2220 CONTINUE
C
C        PRESSURE CORRECTION LOOP
C
  301 CONTINUE
C
      ITER = ITER + 1
C
      CALL CALU
c
      CALL CALV
c
      CALL CALW
c
      CALL CALP
c
      if(resorm(iter) .le. sormax) go to 49
      if(iter .eq. 1) go to 302
      if(resorm(iter) .le. resorm(iter-1)) go to 302
      go to 304
  302 if(jterm .ge. 2) go to 37
      source=resorm(iter)
      go to 39
   37 if(resorm(iter) .le. source) go to 38
      go to 304
   38 source=resorm(iter)
   39 continue
      do 23 k=1,nkp1
         do 23 j=1,njp1
            do 23 i=1,nip1
               tpd(i,j,k) = t(i,j,k)
               upd(i,j,k) = u(i,j,k)
               vpd(i,j,k) = v(i,j,k)
               wpd(i,j,k) = w(i,j,k)
               ppd(i,j,k) = p(i,j,k)
   23 continue
```

```
      jjterm=0
      if(iter .eq. itmax) go to 49
      if(jterm .eq. 2) go to 35
      if(iter .eq. 4) go to 29
   35 continue
      if(jterm .eq. 3) go to 58
      if(iter .eq. 7) go to 29
   58 continue
      jjterm=0
      go to 301
  304 continue
      jjterm=jjterm+1
      if(jterm .eq. 1) go to 41
      if(jterm .eq. 2 .and. jjterm .eq. 1 .and. iter .ne. 5) go to 41
      go to 82
   41 continue
      do 40 k=1,nkp1
         do 40 j=1,njp1
            do 40 i=1,nip1
               u(i,j,k) = upd(i,j,k)
               v(i,j,k) = vpd(i,j,k)
               w(i,j,k) = wpd(i,j,k)
               p(i,j,k) = ppd(i,j,k)
   40 continue
      if(iter .eq. itmax) go to 49
      go to 29
   82 continue
      do 43 k=1,nkp1
         do 43 j=1,njp1
            do 43 i=1,nip1
               t(i,j,k) = tpd(i,j,k)
               u(i,j,k) = upd(i,j,k)
               v(i,j,k) = vpd(i,j,k)
               w(i,j,k) = wpd(i,j,k)
               p(i,j,k) = ppd(i,j,k)
   43 continue
      if(iter .eq. itmax) go to 49
      if((jterm .eq. 3 .and. iter .ne. 8) .or. jjterm .eq. 2) go to 49
      go to 301
   49 continue
c
      II = NIP1 * 2 / 3
      JJ = NJP1 * 2 / 3
      KK = NKP1 * 2 / 3
      UU(NT) = U(II,JJ,KK)
      VV(NT) = V(II,JJ,KK)
      WW(NT) = W(II,JJ,KK)
C
C       PRINT ENERGY AND MASS BALANCE STATISTICS AT REGULAR
C                        INTERVALS
C
      IF (MOD(NT,nprint) .EQ. 0) THEN
         CALL NU
         call chiptemp
         call tstep
      ENDIF
c
c        UPDATE VARIABLES FOR THE NEXT TIME STEP
c
      DO 305 K=1,NKP1
```

```
         DO 305 J=1,NJP1
            DO 305 I=1,NIP1
               TOD(I,J,K) = T(I,J,K)
               UOD(I,J,K) = U(I,J,K)
               VOD(I,J,K) = V(I,J,K)
               WOD(I,J,K) = W(I,J,K)
  305 CONTINUE
      GO TO 300
      end
c***************************************************************************
      subroutine tstep
      implicit real*8 (a-h,o-z)
c
c     THE MAXIMUM TIME STEP FOR AN EXPLICIT MARCH IS CALCULATED.
c     THIS IS USEFUL FOR DECIDING THE TIME STEP THAT NEEDS
c     TO BE SPECIFIED.
c     ALSO, THE VOLUME AVERAGED PRANDTL NUMBER IS CALCULATED.
c
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/ifirst/nust,nvst,nwst,npst
c
      tmin = 0.
      do 10 i=2,ni
         do 10 j=2,nj
            do 10 k=2,nk
               tmp1 = 0.5 * ((u(i,j,k) + u(i+1,j,k)) / dxx(i)
     &                     + (v(i,j,k) + v(i,j+1,k)) / dyy(j)
     &                     + (w(i,j,k) + w(i,j,k+1)) / dzz(k))
               tmp2 = 1. / dxx(i)**2 + 1. / dyy(j)**2
     &              + 1. / dzz(k)**2
c               tmp = max (tmp1,tmp2)
               tmp = tmp1
               if (tmp .gt. tmin) tmin = tmp
  10  continue
      tmin = 1. / tmin
c
      umin = 0.
      do 20 i=nust,ni
         do 20 j=2,nj
            do 20 k=2,nk
               tmp1 = 0.5 * (2. * u(i,j,k) / dxxs(i)
     &              + (v(i,j,k) + v(i,j+1,k)) / dyy(j)
     &              + (w(i,j,k) + w(i,j,k+1)) / dzz(k))
               tmp2 = visco(i,j,k) * (1. / dxxs(i)**2
     &              + 1. / dyy(j)**2 + 1. / dzz(k)**2)
c              tmp = max (tmp1,tmp2)
               tmp = tmp1
               if (tmp .gt. umin) umin = tmp
  20  continue
      umin = 1. / umin
c
      vmin = 0.
```

```
      do 30 i=nvst,ni
         do 30 j=3,nj
            do 30 k=2,nk
               tmp1 = 0.5 * (2. * v(i,j,k) / dyys(j)
     &                    + (u(i,j,k) + u(i+1,j,k)) / dxx(i)
     &                    + (w(i,j,k) + w(i,j,k+1)) / dzz(k))
               tmp2 = visco(i,j,k) * (1. / dxx(i)**2
     &                    + 1. / dyys(j)**2 + 1. / dzz(k)**2)
c               tmp = max (tmp1,tmp2)
               tmp = tmp1
               if (tmp .gt. vmin) vmin = tmp
   30 continue
      vmin = 1. / vmin
c
      wmin = 0.
      do 40 i=nwst,ni
         do 40 j=2,nj
            do 40 k=3,nk
               tmp1 = 0.5 * (2. * w(i,j,k) / dzzs(k)
     &                    + (v(i,j,k) + v(i,j+1,k)) / dyy(j)
     &                    + (u(i,j,k) + u(i+1,j,k)) / dxx(i))
               tmp2 = visco(i,j,k) * (1. / dxx(i)**2
     &                    + 1. / dyy(j)**2 + 1. / dzzs(k)**2)
c              tmp = max (tmp1,tmp2)
               tmp = tmp1
               if (tmp .gt. wmin) wmin = tmp
   40 continue
      wmin = 1. / wmin
      tt = min (tmin,umin,vmin,wmin)
      print *,'THE EXPLICIT TIME STEP IS',tt
c
c     THE VOLUME AVERAGED PRANDTL NUMBER IS CALCULATED.
c
      vavgpr = 0.
      do 50 i=nust,ni
         do 50 j=2,nj
            do 50 k=2,nk
               vavgpr = vavgpr + dxx(i) * dyy(j) * dzz(k)
     &                      * visco(i,j,k)
   50 continue
      vavgpr = vavgpr / (wth * bth)
      print *,'THE VOLUME AVERAGED PRANDTL NUMBER IS',vavgpr
      end
c**********************************************************************
      subroutine grid
c
c     THIS SUBROUTINE GENERATES THE GRID. THE ROBERTS TRANSFORMATION
c     IS USED TO RESOLVE BOUNDARY LAYER NEAR THE WALL.
c
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/blayer/xbr,ybr,zbr
      common/unfrm/iunfrm
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2
      common/array/njchip,nkchip
```

```
      common/spaced/ychip(3),zchip(3)
c
      dimension kgap(4),jgap(4),ygap(4),zgap(4),
     &          dygap(4),dzgap(4)
c
      print *,'THE HEIGHT OF THE ENCLOSURE IS',h,'mm'
      print *,'THE WIDTH OF THE ENCLOSURE IS',wth,'mm'
      print *,'THE LENGTH OF THE ENCLOSURE IS',bth,'mm'
c
      wth = wth / h
      bth = bth / h
      xbr = xbr / h
      ybr = ybr / h
      zbr = zbr / h
c
      print *,'THE CHIP DIMENSIONS ARE THE FOLLOWING:'
      print *,'CHIP HEIGHT (Y-DIRECTION)',hchip,'mm'
      print *,'CHIP WIDTH (Z-DIRECTION)',wchip,'mm'
      print *,'CHIP LENGTH(X-DIRECTION)',bchip,'mm'
c
c     NONDIMENSIONALIZE THE LENGTH PARAMETERS WHICH ARE GIVEN IN MM
c
      hchip = hchip / h
      wchip = wchip / h
      bchip = bchip / h
      bsub = bsub / h
c
      if (iunfrm .eq. 0) then
      if (nchp .ne. 0) then
c
      do 10 i=1,njchip
         ychip(i) = ychip(i) / h
   10 continue
c
      do 20 i=1,nkchip
         zchip(i) = zchip(i) / h
   20 continue
c
      dx = bchip / float(ichip)
      dy = hchip / float(jchip)
      dz = wchip / float(kchip)
c
      zgap(1) = zchip(1) - 0.5 * wchip
      zgap(nkchip+1) = wth - zchip(nkchip) - 0.5 * wchip
      ygap(1) = ychip(1) - 0.5 * hchip
      ygap(njchip+1) = 1.0 - ychip(njchip) - 0.5 * hchip
c
      kgap(1) = int(float(kchip) * zgap(1) / wchip) - 4
      dzgap(1) = zgap(1) / float(kgap(1))
      jgap(1) = int(float(jchip) * ygap(1) / hchip) + 3
      dygap(1) = ygap(1) / float(jgap(1))
c
      do 25 i=2,nkchip
         zgap(i) = zchip(i) - zchip(i-1) - wchip
         kgap(i) = int(float(kchip) * zgap(i) / wchip) - 2
         dzgap(i) = zgap(i) / float(kgap(i))
   25 continue
      do 30 i=2,njchip
         ygap(i) = ychip(i) - ychip(i-1) - hchip
         jgap(i) = int(float(jchip) * ygap(i) / hchip) + 1
```

```
         dygap(i) = ygap(i) / float(jgap(i))
 30   continue
c
      ktotal = 0
      jtotal = 0
      do 35 i=1,nkchip
         ktotal = ktotal + kgap(i)
 35   continue
      do 40 i=1,njchip
         jtotal = jtotal + jgap(i)
 40   continue
      ktotal = ktotal + nkchip * kchip
      jtotal = jtotal + njchip * jchip
c
      kgap(nkchip+1) = nkm1 - ktotal
      dzgap(nkchip+1) = zgap(nkchip+1) / float(kgap(nkchip+1))
      jgap(njchip+1) = njm1 - jtotal
      dygap(njchip+1) = ygap(njchip+1) / float(jgap(njchip+1))
c
      if ((nkm1 - ktotal) .lt. 4) then
          print *,'INSUFFICIENT # OF POINTS IN THE Z-DIRECTION'
          stop
      endif
c
      if ((njm1 - jtotal) .lt. 6) then
          print *,'INSUFFICIENT # OF POINTS IN THE Y-DIRECTION'
          stop
      endif
c
      call grid1 (ygap(1),ybr,jgap(1),yk1,yh1)
      call grid1 (ygap(njchip+1),ybr,jgap(njchip+1),yk2,yh2)
      call grid1 (zgap(1),zbr,kgap(1),zk1,zh1)
      call grid1 (zgap(nkchip+1),zbr,kgap(nkchip+1),zk2,zh2)
c
c     GRID GENERATION IN THE Y AND Z-DIRECTION (TOP AND BOTTOM)
c
      do 45 j=2,jgap(1)+2
         y(j) = tanh (yk1 * yh1 * dygap(1) * float(j - jgap(1) - 2))
     &           / yk1 + ygap(1)
 45   continue
      y(1) = - y(3)
c
      do 50 k=2,kgap(1)+2
         z(k) = tanh (zk1 * zh1 * dzgap(1) * float(k - kgap(1) - 2))
     &           / zk1 + zgap(1)
 50   continue
      z(1) = - z(3)
c
      jj = 2 + jtotal
      yy = ychip(njchip) + 0.5 * hchip
      do 55 j=1,jgap(njchip+1)+1
         y(j+jj) = tanh (yk2 * yh2 * dygap(njchip+1)
     &                * float(j)) / yk2 + yy
 55   continue
      y(njp2) = 1.0 + y(njp1) - y(nj)
c
      kk = 2 + ktotal
      zz = zchip(nkchip) + 0.5 * wchip
      do 60 k=1,kgap(nkchip+1)+1
         z(k+kk) = tanh (zk2 * zh2 * dzgap(nkchip+1)
```

```
     &            * float(k)) / zk2 + zz
  60  continue
      z(nkp2) = wth + z(nkp1) - z(nk)
c
      yy = - hchip
      jj = 2 - jchip
      do 65 i=1,njchip
         jj = jj + jchip + jgap(i)
         yy = yy + hchip + ygap(i)
         do 65 j=1,jchip
            y(jj+j) = dy * float(j) + yy
  65  continue
c
      zz = - wchip
      kk = 2 - kchip
      do 70 i=1,nkchip
         kk = kk + kchip + kgap(i)
         zz = zz + wchip + zgap(i)
         do 70 k=1,kchip
            z(kk+k) = dz * float(k) + zz
  70  continue
c
      yy = 0.
      jj = 2
      do 75 i=2,njchip
         jj = jj + jchip + jgap(i-1)
         yy = yy + hchip + ygap(i-1)
         do 75 j=1,jgap(i)
            y(jj+j) = dygap(i) * float(j) + yy
  75  continue
c
      zz = 0.
      kk = 2
      do 80 i=2,nkchip
         kk = kk + kchip + kgap(i-1)
         zz = zz + zgap(i-1) + wchip
         do 80 k=1,kgap(i)
            z(kk+k) = dzgap(i) * float(k) + zz
  80  continue
c
c     GRID GENERATION IN THE X-DIRECTION
c
      ii = (nim1 - ichip - isub) / 2
      ii = 2 * ii
      inc = nim1 - ichip -isub - ii
      isub = isub + inc
      dsub = bsub / float(isub)
      do 160 i=1,isub+2
         x(i) = dsub * float(i - 2)
 160  continue
c
      dx = bchip / float(ichip)
      xx = bsub
      ii = isub + 2
      do 170 i=1,ichip
         x(i+ii) = dx * float(i) + xx
 170  continue
c
      ii = isub + ichip + 2
      bb = 0.5 * (bth - bsub - bchip)
```

```
      xx = bsub + bchip + bb
      ii1 = (nim1 - ichip - isub) / 2
      call grid1 (bb,xbr,ii1,xk2,xh2)
      ii2 = 2 * ii1
      dr = bb / float(ii1)
      do 180 i=1,ii2
         x(i+ii) = tanh (xh2 * xk2 * dr * float (i - ii1)) / xk2
     &             + xx
  180 continue
      x(nip1+1) = bth + x(nip1) - x(ni)
c
      ibgn = isub + 2
      iend = isub + ichip + 1
c
      jbgn(1) = jgap(1) + 2
      jend(1) = jbgn(1) + jchip - 1
      do 200 i=2,njchip
         jbgn(i) = jbgn(i-1) + jchip + jgap(i)
         jend(i) = jbgn(i) + jchip - 1
  200 continue
c
      kbgn(1) = kgap(1) + 2
      kend(1) = kbgn(1) + kchip - 1
      do 220 i=2,nkchip
         kbgn(i) = kbgn(i-1) + kchip + kgap(i)
         kend(i) = kbgn(i) + kchip - 1
  220 continue
c
      else
      print *,'THE HEIGHT OF THE ENCLOSURE IS',h,'mm'
      print *,'THE WIDTH OF THE ENCLOSURE IS',wth,'mm'
      print *,'THE LENGTH OF THE ENCLOSURE IS',bth,'mm'
c
c     NONDIMENSIONALIZE THE LENGTH PARAMETERS WHICH ARE GIVEN IN MM
c
c
c     GRID GENERATION IN THE Z-DIRECTION
c
      z1 = wth * 0.5
      kz1 = (nkp1 - 2) / 2
      dr1 = z1 / float(kz1)
      call grid1 (z1,zbr,kz1,zk1,zh1)
c
      z(1) = - zbr
      ii = 2 * kz1 + 1
      do 181 i=1,ii
         z(i+1) = tanh (zh1 * zk1 * dr1 * float (i - kz1 - 1)) / zk1
     &             + z1
  181 continue
      z(nkp1+1) = wth + zbr
c
c     GRID GENERATION IN THE Y-DIRECTION
c
      y1 = 0.5
      jy1 = (njp1 - 2) / 2
      dr1 = y1 / float(jy1)
      call grid1 (y1,ybr,jy1,yk1,yh1)
c
      y(1) = - ybr
      ii = 2 * jy1 + 1
```

```
      do 182 i=1,ii
         y(i+1) = tanh (yh1 * yk1 * dr1 * float (i - jy1 - 1)) / yk1
     &           + y1
  182 continue
      y(njp1+1) = 1.0 + ybr
c
c     GRID GENERATION IN THE X-DIRECTION
c
      x1 = 0.5 * bth
      ix1 = (nip1 - 2) / 2
      dr1 = x1 / float(ix1)
      call grid1 (x1,xbr,ix1,xk1,xh1)
c
      x(1) = - xbr
      ii = 2 * ix1 + 1
      do 183 i=1,ii
         x(i+1) = tanh (xh1 * xk1 * dr1 * float (i - ix1 - 1)) / xk1
     &           + x1
  183 continue
      x(nip1+1) = bth + xbr
      endif
      else
      print *,'THE GRID IS UNIFORM'
c
      dx = bth / float(nim1)
      dy = 1.0 / float(njm1)
      dz = wth / float(nkm1)
c
c       UNIFORM GRID
c
      do 185 i=1,nip2
         x(i) = dx * float(i - 2)
  185 continue
c
      do 186 j=1,njp2
         y(j) = dy * float(j - 2)
  186 continue
c
      do 187 k=1,nkp2
         z(k) = dz * float(k - 2)
  187 continue
      IF (NCHP .NE. 0) THEN
          IBGN = 4
          IEND = 5
          JBGN(1) = 4
          JEND(1) = 5
          JBGN(2) = 8
          JEND(2) = 9
          JBGN(3) = 12
          JEND(3) = 13
          KBGN(1) = 4
          KEND(1) = 5
          KBGN(2) = 8
          KEND(2) = 9
          KBGN(3) = 12
          KEND(3) = 13
       ENDIF
       endif
c
c      CALCULATE THE DIMENSIONS OF THE CONTROL VOLUMES
```

```
c
      do 188 i=1,nip1
         dxx(i) = x(i+1) - x(i)
  188 continue
      do 189 i=1,njp1
         dyy(i) = y(i+1) - y(i)
  189 continue
      do 191 i=1,nkp1
         dzz(i) = z(i+1) - z(i)
  191 continue
      dxx(nip2) = dxx(nip1)
      dyy(njp2) = dyy(njp1)
      dzz(nkp2) = dzz(nkp1)
c
c     CALCULATE THE DIMENSIONS OF THE STAGGERED CONTROL VOLUMES
c
      do 245 i=2,nip1
         dxxs(i) = 0.5 * (dxx(i-1) + dxx(i))
  245 continue
      dxxs(1) = dxxs(2)
      do 250 i=2,njp1
         dyys(i) = 0.5 * (dyy(i-1) + dyy(i))
  250 continue
      dyys(1) = dyys(2)
      do 255 i=2,nkp1
         dzzs(i) = 0.5 * (dzz(i-1) + dzz(i))
  255 continue
      dzzs(1) = dzzs(2)
      dxxs(nip2) = dxxs(nip1)
      dyys(njp2) = dyys(njp1)
      dzzs(nkp2) = dzzs(nkp1)
c
c     CALCULATE THE AREAS AND VOLUMES TO
c     CHECK ACCURACY OF GRID
c
      svol = 0.
      do 260 i=2,nip1-1
         do 260 j=2,njp1-1
            do 260 k=2,nkp1-1
               svol = svol + dxx(i) * dyy(j) * dzz(k)
  260 continue
      svol = svol * h**3
      print *,'THE TOTAL VOLUME IS',svol,'cubic mm'
      svol = 0.
      do 261 i=2,nip1-1
         do 261 j=2,njp1-1
            svol = svol + dxx(i) * dyy(j)
  261 continue
      svol = svol * h**2
      print *,'THE TOTAL XY AREA IS',svol,'square mm'
      svol = 0.
      do 262 j=2,njp1-1
         do 262 k=2,nkp1-1
            svol = svol + dyy(j) * dzz(k)
  262 continue
      svol = svol * h**2
      print *,'THE TOTAL YZ AREA IS',svol,'square mm'
      svol = 0.
      do 263 k=2,nkp1-1
         do 263 i=2,nip1-1
```

```
          svol = svol + dzz(k) * dxx(i)
  263 continue
      svol = svol * h**2
      print *,'THE TOTAL ZX AREA IS',svol,'square mm'
      INDIC = MAX0 (NIP2,NJP2,NKP2)
      DO 270 I=1,INDIC
         write(6,1000) i,dxx(i),x(i),dyy(i),y(i),dzz(i),z(i)
  270 continue
 1000 format(1x,i5,2x,6(f9.5,1x))
      PRINT *,IBGN,IEND,JBGN,JEND,KBGN,KEND
      write(13,2000) x,y,z,dxx,dyy,dzz
 2000 format(4(f17.7))
      end
c*************************************************************************
      subroutine grid1 (bb,dx,ngrid,x1,hk)
      implicit real*8 (a-h,o-z)
c
c     THIS SUBROUTINE CALCULATES THE TWO GRID PARAMETERS
c
      data epps,nmax/1.0e-8,1000/
      dr = bb / float(ngrid)
      c = 1. / (float(nmax) * bb)
      yy = 1. - dr / bb
      y1 = yy - 1.
c
c     NARROW DOWN THE ZERO USING THE BISECTION METHOD
c
      i = 0
      ii = 0
    8 continue
      i = i + 1
      x1 = c * float(i)
      f1 = (1. + x1 * (bb - dx)) * (1. - x1 * bb)**yy
     &   - (1. - x1 * (bb - dx)) * (1. + x1 * bb)**yy
      x2 = c * float(i+1)
      f2 = (1. + x2 * (bb - dx)) * (1. - x2 * bb)**yy
     &   - (1. - x2 * (bb - dx)) * (1. + x2 * bb)**yy
      if (f1 * f2 .lt. 0. .or. i .eq. (nmax - 1)) then
          go to 9
      endif
      go to 8
    9 continue
      x3  = 0.5 * (x1 + x2)
      ii = ii + 1
      f3 = (1. + x3 * (bb - dx)) * (1. - x3 * bb)**yy
     &   - (1. - x3 * (bb - dx)) * (1. + x3 * bb)**yy
      if (f3 * f1 .lt. 0.) then
          x2 = x3
      else
          x1 = x3
      endif
      if (ii. lt. 5) go to 9
      xinit = 0.5 * (x1 + x2)
c
c      THE GRID PARAMETER IS DETERMINED BY NEWTON-RAPHSON ITERATION
c
      x1 = xinit
      iter = 0
   10 continue
      iter = iter + 1
```

```
      f = (1. + x1 * (bb - dx)) * (1. - x1 * bb)**yy
     &  - (1. - x1 * (bb - dx)) * (1. + x1 * bb)**yy
      df = (bb - dx) * (1. - x1 * bb)**yy
     &   + (bb - dx) * (1. + x1 * bb)**yy
     &   - (bb - dr) * (1. + x1 * (bb - dx)) * (1. - x1 * bb)**y1
     &   - (bb - dr) * (1. - x1 * (bb - dx)) * (1. + x1 * bb)**y1
      x2 = x1 - f / df
      xeps = abs (x1 - x2)
      x1 = x2
      if (xeps .gt. epps .and. iter .lt. 10) goto 10
      zz = x1 * bb
      hk = 0.5 / zz * log((1. + zz) / (1. - zz))
      end
c***********************************************************************
      subroutine chiptemp
c
c     THIS SUBROUTINE CALCULATES THE CHIP TEMPERATURES (DIMENSIONAL)
c
      implicit real*8 (a-h,o-z)
      common/ng/dxx(40),dyy(40),dzz(40),x(40),y(40),z(40),
     &          dxxs(40),dyys(40),dzzs(40)
      common/parm/ra,pr,sorsum,dtime,xper,roll,dt_inv,xtime
      common/tol/small,eps,sormax
      common/dims/h,wth,bth,hchip,wchip,bchip,bsub
      common/count/nt,nmax,imax,itmax,krun,nprint
      common/mscn/smp(12,16,16),resorm(93),
     &            du(12,16,16),dv(12,16,16),
     &            dw(12,16,16),pp(12,16,16)
      common/limits/ni,nip1,nim1,nj,njp1,njm1,nk,nkp1,nkm1,
     &              nip2,njp2,nkp2,iter,nnmax
      common/fv_init/tod(12,16,16),uod(12,16,16),vod(12,16,16),
     &               wod(12,16,16),pod(12,16,16)
      common/fv_cur/t(12,16,16),u(12,16,16),v(12,16,16),
     &              w(12,16,16),p(12,16,16)
      COMMON/RHOCP/RHS,RHC
      common/condu/alpha(12,16,16),rho(12,16,16),visco(12,16,16)
      common/chip/ibgn,iend,jbgn(3),jend(3),kbgn(3),kend(3),nchp,
     &            isub,ichip,jchip,kchip
      common/diff/alc,als,thot,tcool,tavg
      common/array/njchip,nkchip
      COMMON/POWER/QQQ,QCOND
C
      dimension tt(3,3),vol(3,3)
C
C
      DO 15 M=1,NJCHIP
         DO 15 N=1,NKCHIP
            VOL(M,N) = 0.
            TT(M,N) = 0.
   15 CONTINUE
C
      do 20 m=1,njchip
         do 20 n=1,nkchip
            do 10 i=ibgn,iend
               do 10 j=jbgn(m),jend(m)
                  do 10 k=kbgn(n),kend(n)
                     tt(m,n) = tt(m,n) + t(i,j,k) * dxx(i)
     &                         * dyy(j) * dzz(k)
                     vol(m,n) = vol(m,n) + dxx(i) * dyy(j)
     &                          * dzz(k)
```

```
 10         continue
            TT(M,N) = TT(M,N) / VOL(M,N) * QCOND
 20   continue
      print *,'THE TEMPERATURE RISES IN THE CHIP ARE:'
      WRITE(6,200) TT
200   FORMAT (4(F17.6))
      end
```

REFERENCES

Arroyo M. P. and Saviron J. M., "Rayleigh-Bénard Convection in a Small Box: Spatial Features and Thermal Dependence of the Velocity Field", Journal of Fluid Mechanics, Vol. 235, pp. 325-348, 1992.

Deng, Y, Glimm J. and Sharp D. H., "Perspectives on Parallel Computing", Daedalus, Vol. 121, pp 31-52, 1992.

Fletcher, C. A. J., Computational Techniques for Fluid Dynamics, Vol. 1, Springer-Verlag, 1988.

Joshi, Y., Kelleher M. D., Powell M. and Torres, E. I., "Natural Convection Heat Transfer from an Array of Rectangular Protrusions in an Enclosure Filled with Dielectric Liquid", Presented at the ASME Winter Annual Meeting, 1991.

Leonard B.P., "A Convectively Stable, Third-Order Accurate Finite-Difference Method for Steady Two-Dimensional Flow and Heat Transfer", Numerical Properties and Methodologies in Heat Transfer, edited by Shih, T. M., Hemisphere Publishing Corporation, Washington D.C., pp. 211-226, 1983.

Liu, K. V., A Numerical, Analytical and Experimental Investigation of the Radiation-Convection Interaction in a Diffusion Flame Adjacent to a Vertical Flat Plate, Ph.D Thesis, Department of Aerospace and Mechanical Engineering, University of Notre Dame, 1979.

Patankar, S. V., Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, Washington D.C., 1980.

Patankar, S. V., "Recent Developments in Computational Heat Transfer", Journal of Heat Transfer, pp. 1037-1045, Vol. 110, 1988.

Roberts, G. O., "Computational Meshes for Boundary Layer Problems", Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics, Vol. 8, edited by M. Holt, Springer-Verlag, pp. 171-177, 1970.

Stone, H. J., SIAM Journal for Numerical Analysis, Vol. 5, pp. 530-558, 1968.

Van Doormal, J. P. and Raithby, G. D., "Enhancement of the Simple Method for Predicting Incompressible Fluid Flows", Numerical Heat Transfer, Vol. 17, pp. 147-163, 1984.