1994

Underground explosion modelling with parallel processors

Ian Conrad Piper
University of Wollongong

Recommended Citation
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
UNDERGROUND EXPLOSION MODELLING
WITH PARALLEL PROCESSORS

A thesis submitted in fulfilment of the
requirements for the award of the degree of

DOCTOR OF PHILOSOPHY

from

THE UNIVERSITY OF WOLLONGONG

by

Ian Conrad Piper, B.Sc. (U.O.W.)

Department of Computer Science
1994
This thesis is submitted to The University of Wollongong, and has not been submitted for a higher degree to any other University or Institution.

Ian Piper
February, 1994
ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my supervisor, Prof. R. W. (Bill) Upfold, for introducing me to the subject of explosion modelling and for his expert advice, assistance and encouragement throughout this project.

Thanks are also due to Prof. Gregory Doherty, who co-supervised me, for his valuable support and advice.

I also wish to acknowledge the persons and companies whose products have made the production of this work possible, particularly AutoDesk for their excellent animation system and John Krommes for his FWEB package.

Special thanks are also due to Prof. D. E. Knuth without whose initial work in the area of literate programming this project would have been impossible.

Finally I would like to thank my wife, my parents and other family members for tolerating my part-time membership of the family over the past few years.
# TABLE OF CONTENTS

Declaration .......................................................... ii
Acknowledgements ................................................. iii
Table of contents ..................................................... iv
Abstract .................................................................... vii

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2 Mathematical Background</td>
<td>6</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Governing Equations</td>
<td>2</td>
</tr>
<tr>
<td>Turbulence</td>
<td>3</td>
</tr>
<tr>
<td>Combustion</td>
<td>4</td>
</tr>
<tr>
<td>Thermodynamics</td>
<td>5</td>
</tr>
<tr>
<td>Boundary Conditions</td>
<td>6</td>
</tr>
<tr>
<td>Chapter 3 The Programs</td>
<td>20</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Data Representation</td>
<td>2</td>
</tr>
<tr>
<td>Griddler</td>
<td>7</td>
</tr>
<tr>
<td>Explode</td>
<td>8</td>
</tr>
<tr>
<td>Chapter 4 Model Validation</td>
<td>43</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Results</td>
<td>3</td>
</tr>
<tr>
<td>Chapter 5 Model Results</td>
<td>50</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Results</td>
<td>2</td>
</tr>
<tr>
<td>Chapter 6 Extensions</td>
<td>56</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Initial conditions</td>
<td>2</td>
</tr>
<tr>
<td>Boundary conditions</td>
<td>5</td>
</tr>
<tr>
<td>Geometry</td>
<td>9</td>
</tr>
<tr>
<td>Solid phase</td>
<td>12</td>
</tr>
<tr>
<td>Topology</td>
<td>13</td>
</tr>
<tr>
<td>Appendix A Gridder</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>The program</td>
<td>6</td>
</tr>
<tr>
<td>Initialisation</td>
<td>9</td>
</tr>
<tr>
<td>The input phase</td>
<td>15</td>
</tr>
<tr>
<td>Set boundary array</td>
<td>27</td>
</tr>
<tr>
<td>Set grid array</td>
<td>32</td>
</tr>
<tr>
<td>Set worker array</td>
<td>35</td>
</tr>
<tr>
<td>Set condition array</td>
<td>38</td>
</tr>
<tr>
<td>The output phase</td>
<td>41</td>
</tr>
<tr>
<td>Index</td>
<td>54</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix B Explode Root Task</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Main</td>
<td>2</td>
</tr>
<tr>
<td>Initialisation</td>
<td>3</td>
</tr>
<tr>
<td>The main loop</td>
<td>25</td>
</tr>
<tr>
<td>Packet handlers</td>
<td>40</td>
</tr>
<tr>
<td>Index</td>
<td>44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix C Explode Root Task—2D Version</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Main</td>
<td>2</td>
</tr>
<tr>
<td>Index</td>
<td>44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix D Explode Worker Task</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Main</td>
<td>2</td>
</tr>
<tr>
<td>Thread control</td>
<td>3</td>
</tr>
<tr>
<td>Initialisation</td>
<td>8</td>
</tr>
<tr>
<td>The model</td>
<td>21</td>
</tr>
<tr>
<td>Pressure</td>
<td>26</td>
</tr>
<tr>
<td>Timestep</td>
<td>29</td>
</tr>
<tr>
<td>Source terms</td>
<td>37</td>
</tr>
<tr>
<td>Artificial viscosity</td>
<td>53</td>
</tr>
<tr>
<td>Viscid fluxes</td>
<td>70</td>
</tr>
<tr>
<td>Inviscid fluxes</td>
<td>118</td>
</tr>
</tbody>
</table>
This work represents the synthesis of a number of areas including literate programming, parallel processing, scientific visualisation and turbulent, reactive fluid dynamics and the application of this synthesis to a significant physical system, gas explosions in confined areas.

This thesis presents a parallel program developed to simulate the transient physical phenomena associated with the detonation of a gaseous fuel-air mixture in a confined volume, along with the support programs developed to assist in the data input and analysis tasks. Also contained are discussions of some results produced by the simulation system, specifically, the modelling of the Norfuss 10 metre ignition tube and the modelling of a complex mine development heading based on the Moura No. 4 Mine at which a major explosion occurred in 1986. Finally a number of possible extensions and enhancements to the simulator are discussed.

The program is shown to provide close agreement with a number of physical experiments and its use as a diagnostic or predictive tool is examined.

Accompanying the printed text of the thesis is a CD-Rom containing the graphical results of the models discussed in the form of both graphs and animations of a number of the key variables in the explosion process.
Chapter 1

— Introduction —
1. INTRODUCTION. This work represents the synthesis of a number of areas including literate programming, parallel processing, scientific visualisation and turbulent, reactive fluid dynamics and the application of this synthesis to a significant physical system, gas explosions in confined areas.

This introductory chapter briefly examines these areas and notes some of the other significant work done in each of them.

2. The concept of literate programming was developed by D. E. Knuth [22] and his WEB system was used to develop the later versions of the TeX [21] and METAFONT [20] programs. The original WEB system used Pascal as the target language but subsequent work by S. Levy who developed CWEB, N. Ramsay who developed SPIDER, W. Sewell [42] who developed M-WEB and J. Krommes [23] who developed FWEB have extended the literate programming paradigm to a number of other languages including C, Fortran, Modula-2 and Ada. The programs presented in this work were developed using Krommes' FWEB system.

The underlying concept of literate programming is that the documentation and code of a program should not be developed as two essentially independent entities, but rather that a single source document, a web, be constructed in which code and documentation are brought together into an integrated whole. Two processors are then used to produce the compilable source code and the TeX documentation source; TANGLE and WEAVE. (In the present case FTANGLE and FWEAVE.) The resulting Fortran and TeX sources are then processed in the usual way to produce the executable program and the printed documentation.

The WEB system also provides a facility for reordering TANGLED code in an appropriate manner for the compiler, thus freeing the programmer from the artificial constraints to program exposition imposed by the computer language selected. The literate programming methodology provides a number of significant advantages over more conventional approaches to programming. These include greater readability, ease of modification and, somewhat surprisingly, faster program development.

In addition to producing exceptionally readable program listings, the WEAVE processor also generates an automatic index and cross reference table as well as a table of contents, all of these serving to further enhance the value of the system.
3. The first significant work in the area of parallel processing was that of A. Hoare whose *Communicating Sequential Processes* (CSP) model was used as the basis of a number of programming systems including Hoare's own Occam language and 3L's Parallel C and Parallel Fortran compilers [1].

On the hardware side the development of the Transputer by Inmos [17] provided a means for the physical realisation of the CSP methodology.

Although the transputer may no longer represent the height of performance in parallel computation it still holds a number of significant attractions as a platform upon which to develop code. First, the transputer architecture supports the scheduling and running of multiple threads, or multiple processes, at the hardware level without the need for an operating system. Second, the transputer allows changes to the connection topology of the processors in a simple manner allowing the evaluation of different message passing methods. Finally, the transputer is ideally suited to small, inexpensive prototype systems as a four processor transputer board for a PC costs only a few thousand dollars.

4. The field of scientific visualisation owes its existence to the realisation that the increasing volumes of data typical of most numerical simulations were rapidly exceeding the capacity of human interpretation by use of the conventional mechanisms of the table and the graph.

Arising from this realisation has come a wide range of visualisation techniques including the computer animation technology used in the present study.

5. The field of turbulent, reactive fluid dynamics has a long history in comparison to the other fields considered above. The Navier-Stokes equations for fluids with high Reynolds number were developed for nonreactive fluids and subsequent work by Williams [45], Oran and Boris [34] extended this early work to include consideration of reactive fluids and the numerical simulation of the processes involved.

The present work is based on the formulation developed by Patankar and Spalding [35] as modified by Hjertager [15], recast by the author to provide an explicit formulation suitable for parallel solution.

Surprisingly, to the author's knowledge, no other work has been carried out in the area of parallel solutions for reactive turbulent flow. Computational fluid dynamics (CFD) is one of the Grand Challenge areas for parallel and supercomputing but, to date, all of the work in this area has been restricted to non-reactive flows. Although not directly related to the present area, the work of Ast, Dayde and Kourta [3], Carmona [8], Deiwert [10], Kutler [24] and Mangiavacchi and Akhavan [27] all serve to provide some indication of the current state of the art in parallel CFD techniques.
Explosions in confined spaces, particularly those occurring in coal mines, have long been a matter of real concern to the community. The greatest disaster to occur on land in Australia's history, the Mt Kembla disaster, in which 96 men lost their lives was just such a coal mine explosion. The history of coal mining, both in Australia and overseas, is replete with such incidents and the loss of life from mine explosions, both methane and coal dust, reaches into the tens of thousands.

Meadows [28] notes that, as early as 1844, Michael Faraday and Charles Lyell were called upon to investigate a catastrophic mine explosion in Durham. They determined that the likely cause of this explosion was the safety lamp, invented by Davy, which miners had been using to light their pipes while underground.

Despite this loss of life, limited work has been carried out in the area of simulation of the processes involved.

Several methods capable of handling transient compressible flows with chemical reactions have been published. The majority of these methods use explicit formulation of the momentum and other conservation equations. The method of Butler and O'Rourke [7] uses the Implicit Continuous-fluid Eulerian (ICE) technique to eliminate the timestep restrictions imposed on explicit methods by the Courant criterion \( \Delta t \sim \Delta x/(c_s + \Delta u) \). This method, along with the Arbitrary Lagrangian Eulerian (ALE) technique was employed by Boni, Chapman and Schneyer [6] to study combustion in the stratified charge engine. Both Oran, Young and Boris [34] and Green, Piper and Upfold [12] have examined explosion propagation using the Flux Corrected Transport (FCT) algorithms.

A number of experimental explosion facilities exist around the world including the Bruceton and Lake Lynn mines in the United States, the Barbara mine in Poland, Buxton in the United Kingdom and Australia's Londonderry facility, but the cost of large scale, fully instrumented experimental explosions is prohibitive and the extent of the work carried out in these facilities is, of necessity, limited.

This area of research, combining great social significance with a relative shortage of data, recommended itself as being worthy of study.
7. The present work contains a discussion of the underlying mathematical model used in the simulation, the significant features of the simulation and data preparation programs, the results of the model used in the validation of the simulator and in the subsequent investigation of a real mine explosion and the consideration of a number of possible future extensions and enhancements to the simulator.

In addition to this it also contains the woven listings of 4 programs—the data preparation program, the root and worker tasks for the explosion simulator itself along with the changes required to produce the 2 dimensional version of the program used in the reported results and the graphical post processor used to develop the individual frames from which the animated sequences were created.

A CD-Rom is also included with the printed material which contains a wide range of graphical presentations of the simulation results as well as the text of this volume in a machine readable format. A number of freely distributable utility programs are also included on the CD-Rom which will permit the results to be examined on any MPC compliant personal computer with at least a 256 colour Windows 3.1 capability.

To install the CD-Rom, simply run the setup.exe program on the root directory of the CD-Rom from the Windows Program Manager; setup will make the necessary modifications to the Windows win.ini and system.ini files and create a number of application groups within the file manager.

For those without an appropriate Windows environment the ani directory on the CD-Rom contains a program, aaplay.exe, which may be used to play the animations in the tflics, vflics and cflics directories which contain the ignition tube results as well as the mflics, nflics and oflics directories which contain the mine animations.
### Chapter 2

— Mathematical Background —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>GOVERNING EQUATIONS</td>
<td>2</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>3</td>
</tr>
<tr>
<td>COMBUSTION</td>
<td>4</td>
</tr>
<tr>
<td>THERMODYNAMICS</td>
<td>5</td>
</tr>
<tr>
<td>BOUNDARY CONDITIONS</td>
<td>6</td>
</tr>
</tbody>
</table>

February 11, 1994

20:35
1. INTRODUCTION. The mathematical and numerical modelling of combustion processes in turbulent compressible flows is a complex problem of practical importance. The non-linear behaviour of such processes and the range of time scales involved in the modelling renders the task a demanding one from both a theoretical and numerical viewpoint. The key issue which must be addressed in such a model is the adequate modelling of the coupling between the fluid dynamic properties and the reaction rate mechanisms of the system.

As already noted, several prediction methods capable of handling transient compressible flows exist using various explicit formulations. All of these methods have inherent time step limitations and may therefore be unsuited to the modelling of systems in which a slow variation occurs or in which convergence to a steady state is sought. The alternative implicit solution schemes, which do not, in principle, have the time step limitation of the explicit methods, may seem attractive and indeed Computational Fluid Dynamics (CFD) methods using implicit solution schemes have been developed. The formulation due to Patankar and Spalding [35] and further developed by Magnussen and Hjertager [26] in the SIMPLE system is one such implicit scheme for incompressible reactive systems. Implicit schemes, as well as the already noted time step independence, lend themselves to solution on SIMD vector supercomputers as they require the solution of large systems of simultaneous equations. The disadvantage of implicit schemes is that they are best suited for the modelling of geometrically simple domains and are thus not appropriate for the solution of the equations of reactive turbulent flow in complex domains such as those found in underground mines.

The solution presented here is an explicit scheme which differs from both the SIMPLE method and the FCT method. The equations developed have been integrated in time by employing a modified Runge-Kutta scheme [40]. The problem domain is divided into regular, finite volume elements (finite area in the two-dimensional version).
2. GOVERNING EQUATIONS. Any general flow is governed by the Navier Stokes equations \[11\] and when written in what is called the 'conservative form' will automatically capture shock-like gradients. In flows with combustion, the energy at any location is the sum of the internal, kinetic energies and the heat of combustion.

The problem of turbulent transient flows may be approached by solving for the change over time of a series of dependent physical parameters, \( \Phi \), in the domain of interest. The time mean of a variable varying with time, \( t \), may be expressed as:

\[
\Phi(t) = \frac{1}{T} \int_{0}^{T} \Phi(t + \tau) \, d\tau
\]

where \( \Phi(t) \) is the mean value of the instantaneous value \( \Phi(t) \) averaged over the time interval \( T \). \( T \) must be chosen such that it is of sufficiently long duration to give relevant time mean values in the interval and such that it is of sufficiently short duration to ensure that significant transients are not averaged out. The equations of motion and energy may be expressed in Cartesian coordinates in the following form:

\[
\frac{\partial}{\partial t} \rho \Phi + \frac{\partial}{\partial x_j} \left( \rho u_j \Phi - \Gamma_\Phi \frac{\partial \Phi}{\partial x_j} \right) = S_\Phi
\]

where \( \rho \) is the local density, \( u_j \) the velocity components, \( \Gamma_\Phi \) are fluxes and \( S_\Phi \) are source terms.

In the present problem there are 10 such equations which must be solved. These are the equations for conservation of mass, momentum (3 equations), internal energy, fuel fraction, mixture fraction, variance in the mixture fraction, turbulent kinetic energy and rate of dissipation of turbulence.

The expressions for \( \Phi \), \( \Gamma_\Phi \) and \( S_\Phi \) for each of the equations are as follows:

<table>
<thead>
<tr>
<th>( \Phi )</th>
<th>( \Gamma_\Phi )</th>
<th>( S_\Phi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x-velocity</td>
<td>( u )</td>
<td>( \mu_t )</td>
</tr>
<tr>
<td>y-velocity</td>
<td>( v )</td>
<td>( \mu_t )</td>
</tr>
<tr>
<td>z-velocity</td>
<td>( w )</td>
<td>( \mu_t )</td>
</tr>
<tr>
<td>internal energy</td>
<td>( h )</td>
<td>( \frac{\mu_t}{\sigma_h} )</td>
</tr>
<tr>
<td>fuel fraction</td>
<td>( m_{f,u} )</td>
<td>( \frac{\mu_t}{\sigma_{f,u}} )</td>
</tr>
<tr>
<td>mixture fraction</td>
<td>( f )</td>
<td>( \frac{\mu_t}{\sigma_f} )</td>
</tr>
<tr>
<td>variance of ( f )</td>
<td>( g )</td>
<td>( \frac{\mu_t}{\sigma_g} )</td>
</tr>
</tbody>
</table>


§2 — Mathematical Background —

**GOVERNING EQUATIONS**

- **turbulent kinetic energy**  
  \( k \)  
- **rate of dissipation**  
  \( \epsilon \)

\[
G_k - \rho \epsilon = (C_1 G_k - C_2 \rho \epsilon) \frac{\epsilon}{t}
\]

where \( \mu_t \) and \( G_k \) are given by

\[
\mu_t = \frac{C_\mu \rho k^2}{\epsilon}
\]

and:

\[
G_k = \mu_{eff} \left\{ 2 \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial y} \right)^2 + \left( \frac{\partial w}{\partial z} \right)^2 \right] + \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right)^2 + \left( \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \right)^2 \right\}
\]

where \( \mu_{eff} \) is the effective viscosity and the following empirically derived [2] constants are used:

- \( C_\mu = 0.09 \)
- \( C_1 = 1.44 \)
- \( C_2 = 1.92 \)
- \( C_{g1} = 2.8 \)
- \( C_{g2} = 2.0 \)
- \( A = 4.0 \)
- \( \sigma_h = 0.7 \)
- \( \sigma_{fu} = 0.7 \)
- \( \sigma_f = 0.7 \)
- \( \sigma_g = 0.7 \)
- \( \sigma_k = 0.9 \)
- \( \sigma_\epsilon = 1.22 \)

For each finite volume in the model domain, the governing equations are applied and simplified using Green’s Theorem, taking the form:

\[
\frac{d}{dt}(\Phi V) + Q(\Phi) = 0
\]

where \( Q \) is a flux based operator and \( V \) is a finite volume.

It should be noted that this results in the reduction of the governing equations from Partial Differential Equations to a set of coupled difference equations which may be expressed as Ordinary Differential Equations and solved for \( \Phi \) by methods such as the Runge-Kutta algorithm:

The operator \( Q \) is based on the fluxes across the discrete volume boundaries and takes the following form:

\[
Q = \Delta t \left( \frac{\Delta \Gamma_{xz}}{\Delta x} - \frac{\Delta \Gamma_{zx}}{\Delta x} + \frac{\Delta \Gamma_{yz}}{\Delta y} + \frac{\Delta \Gamma_{zy}}{\Delta y} + \frac{\Delta \Gamma_{wz}}{\Delta z} + \frac{\Delta \Gamma_{zw}}{\Delta z} \right) + \frac{\Delta t}{V} \Gamma_a + \Delta t S
\]

where \( \Gamma_a \) is an artificial viscosity term used as a Total Variance Diminishing (TVD) mechanism, \( \Gamma_i \) and \( \Gamma_v \) are the inviscid and viscous flux terms, \( S \) is the source term, \( \Delta x \), \( \Delta y \) and \( \Delta z \) are the Cartesian dimensions of the finite volume \( V \).
The solution scheme used in the present work is based on the following single step numerical integration method [40]:

$$
\Psi_1 = f_1(\Phi_t) + f_2(\Phi_t)
$$

$$
\Psi_2 = f_1(\Phi_t + \frac{\Delta t}{4} \Psi_1) + f_2(\Phi_t)
$$

$$
\Psi_3 = f_1(\Phi_t + \frac{\Delta t}{3} \Psi_2) + f_2(\Phi_t)
$$

$$
\Psi_4 = f_1(\Phi_t + \frac{\Delta t}{2} \Psi_3) + f_2(\Phi_t)
$$

$$
\Phi_{t+\Delta t} = \Phi_t + \Delta t \Psi_4.
$$

The basis for this partitioning into fourth order and first order expressions is that the stiff terms, those with high eigenvalues, are collected into the $f_2$ component while the other terms are collected into the $f_1$ component.

In the present case, $f_1$ involves the source and inviscid flux terms and $f_2$ the artificial viscosity and viscid flux terms.

This partitioning into the turbulent and laminar components of the model results in a solution scheme which combines very high numerical stability (the method has proven to be stable for a range of initial turbulence values spanning six orders of magnitude from laminar to highly turbulent flow) with the ability to capture both the overall pattern and the observed transients found in published experimental results.

Testing of this scheme with a range of Courant numbers from 0.9 to 0.1 indicates good convergence and stability properties. In contrast to this, early work by the author in which all terms were combined into the fourth order computation proved to be extremely unstable.
3. TURBULENCE. In order to solve the governing equations, models must be established for the fluxes which incorporate the effect of turbulence. The general form of the flux is as follows:

$$\Gamma_\Phi = -\frac{\mu_{\text{eff}}}{\sigma_\Phi} \frac{\partial \Phi}{\partial x_j}$$

where $\mu_{\text{eff}}$ is the effective viscosity and $\sigma_\Phi$ is the effective Prandtl/Schmidt number. The effective turbulent viscosity is given by:

$$\mu_{\text{eff}} = \mu_1 + \rho u_t l$$

where $\mu_1$ is the laminar, molecular viscosity, $u_t$ is the turbulence velocity and $l$ is a length scale.

As the values of $u_t$ and $l$ are not known, a means of determining them is required. This may be done through application of the $k$-$\epsilon$ model of Launder and Spalding [25]. The turbulence velocity is related to the kinetic energy of turbulence, $k$, by:

$$u_t = \left(\frac{2}{3} k \right)^{\frac{1}{2}}$$

and the length scale is related to the kinetic energy of turbulence and its rate of dissipation, $\epsilon$, by:

$$l \sim \frac{k^{3/2}}{\epsilon}.$$  

Substituting these expressions into the equation for $\mu_{\text{eff}}$ gives:

$$\mu_{\text{eff}} = \mu_1 + C_\mu \rho \frac{k^2}{\epsilon}$$

where $C_\mu$ is an empirical constant taken to be 0.09 [2].

The conservation equations for $k$ and $\epsilon$ are:

$$\frac{\partial \rho k}{\partial t} + \frac{\partial}{\partial x_j} (\rho u_j k) - \frac{\partial}{\partial x_j} \left( \frac{\mu_{\text{eff}}}{\sigma_k} \frac{\partial k}{\partial x_j} \right) = G - \rho \epsilon$$
and

\[ \frac{\partial \rho \varepsilon}{\partial t} + \frac{\partial}{\partial x_j} (\rho u_j \varepsilon) - \frac{\partial}{\partial x_j} \left( \mu_{\text{eff}} \frac{\partial \varepsilon}{\partial x_j} \right) = C_1 \frac{\varepsilon}{k} G - C_2 \rho \varepsilon^2 \]

where the empirically derived constants $C_1$ and $C_2$ take the values 1.44 and 1.92 respectively and the Schmidt numbers $\sigma_k$ and $\sigma_\varepsilon$ are given the values 1.0 and 1.3 respectively. The generation rate of turbulence $G$ is derived from the expression already given above and is related to turbulence production from shear and expansion/compression effects.
4. **COMBUSTION.** The combustion model used in this work is a simple single-step irreversible reaction with finite reaction rate between fuel and oxygen. This reaction may be written:

$$1 \text{ kg fuel} + s \text{ kg oxygen} \rightarrow (1 + s) \text{ kg products}.$$  

where $s$ is the stoichiometric oxygen requirement for complete combustion of 1 kg of fuel. For a hydrocarbon $C_{nc}H_{nh}$ this may be derived from the reaction:

$$C_{nc}H_{nh} + (nc + \frac{1}{2} nh)O_2 \rightarrow (nc)CO_2 + (\frac{1}{2} nh)H_2O$$  

and the molecular weights of the various reactants.

Clearly, this reaction scheme represents a considerable simplification of the combustion process but it provides an adequate approximation and avoids the necessity to model each of the reactions actually involved in hydrocarbon combustion (23 reactions are involved in the combustion of methane [45]).

This reaction scheme results in a mixture composition which may be determined from two variables, the mass fraction of fuel, $m_{fu}$, and the mixture fraction, $f$.

The mixture fraction is given by the expression:

$$f = \frac{\beta - \beta_\infty}{\beta_0 - \beta_\infty}$$

where $\beta$ is a conserved combined function of mass fraction of fuel, $m_{fu}$, and mass fraction of oxygen, $m_{O_2}$, expressed as:

$$\beta = m_{fu} - m_{O_2}/s$$

where $\beta_0$ is the value of $\beta$ at a fuel rich reference point ($m_{fu} = 1.0$, $m_{O_2} = 0.0$) and $\beta_\infty$ is the value of $\beta$ at an oxygen rich reference point ($m_{fu} = 0.0$, $m_{O_2} \approx 0.3$, the mass fraction of oxygen in air).

The complexity inherent in an analytical expression for reaction rate based on chemical kinetics in turbulent situations renders this formulation unsuitable for solution. Consequently an alternative method must be employed.
Spalding [43] has shown experimentally that the combustion rate in flames is dependent to a large degree on hydrodynamic properties. This implies that the combustion rate is dependent on the rate of mixing between the reactants which is, in turn, dependent on the rate at which turbulent eddies dissipate. This dissipation may be expressed as:

\[ \epsilon_\Phi = D_\Phi \frac{\partial \Phi}{\partial x_k} \cdot \frac{\partial \Phi}{\partial x_k} \]

where \( D_\Phi \) is the molecular diffusion coefficient and the rate of dissipation of turbulent kinetic energy is denoted by \( \epsilon \). Thus, it may be assumed that the combustion rate, \( R_{fu} \), is proportional to the rate of dissipation of the kinetic energy of turbulence:

\[ R_{fu} = -\rho c \frac{\sqrt{g_\Phi}}{k} \]

where \( g_\Phi \) is the variance of the fluctuation. Within each discrete volume the fuel, oxidant and reaction products will be present as fluctuating quantities whose contribution to the reaction is given by:

\[ R_{fu} = -A \rho \frac{\epsilon}{k} \min(m_{fu}, m_{O_2}/s, m_b) \]

where \( m_{fu} \), \( m_{O_2} \) and \( m_b \) are the masses of fuel, oxidant and reaction product and \( A \) is a constant ranging in the literature [2] from 2.0 to 22.0 and here taken to be 4.0 as a result of comparison of simulation and experimental results.

This formulation is only valid for cases where the chemical kinetics of the system are fast, an assumption which is not the case for the conditions obtaining in an explosion. A modification of the above scheme has been proposed by Hjertager [14]. A chemical time, \( \tau_{ch} \), may be derived from the chemical kinetics of the system. In addition to this the lifetime of turbulent eddies, \( \tau_e \) may be defined. It is assumed that ignition or extinction occurs dependent on some ratio of these times:

\[ D_{le} = \frac{\tau_{ch}}{\tau_e} \]

This leads to the following reaction rate scheme:
\[ R_{fu} = \begin{cases} 0 & \text{if } \tau_{ch}/\tau_e \geq D_{ie} \\ -A\rho_k^e \min(m_{fu}, m_{O_2}/s, m_b) & \text{if } \tau_{ch}/\tau_e < D_{ie} \end{cases} \]

The eddy lifetime, \( \tau_e \) and the chemical time, \( \tau_{ch} \) are given by:

\[ \tau_e = \frac{k}{\epsilon} \quad \text{and} \quad \tau_{ch} = A_{ch} e^{\frac{E}{RT}} (\rho m_{fu})^a (\rho m_{O_2})^b \]

where \( a \) and \( b \) are derived from the partial pressures of fuel and oxygen respectively, \( E \) is the total energy density, \( R \) is the gas constant and \( T \) is the temperature.
5. THERMODYNAMICS. For the simple reaction scheme outlined above, five mass fractions may be identified; fuel, nitrogen, oxygen, carbon dioxide and water vapour.

The density of this gas mixture may be derived from the ideal gas laws as:

\[ \rho = \frac{M_{av} P}{R u T} \]

where the average species molecular weight, \( M_{av} \), is calculated from:

\[ \frac{1}{M_{av}} = \sum_{i=1}^{5} \frac{m_i}{M_i} \]

The enthalpy, \( h \), is defined as:

\[ h = c_{pm,mix} T + m_{fu} H_{fu} \]

where the mixture specific heat, \( c_{pm,mix} \), is calculated from each of the five gases present in the mixture via:

\[ c_{pm,mix} = \sum_{i=1}^{5} m_i c_{pm,i} \]

and the specific heat of species \( i \), \( c_{pm,i} \), is given by:

\[ c_{pm,i} = \frac{1}{T - T_{ref}} \int_{T_{ref}}^{T} c_{p,i} dT \]

The species specific heats are assumed to vary linearly with temperature:

\[ c_{p,i} = a_i + b_i T \]

where the \( a_i \) and \( b_i \) values are determined experimentally [19].
6. BOUNDARY CONDITIONS. Boundary conditions must be specified in order to model both confined and unconfined domains. Three general approaches to the modelling of boundaries in computational fluid dynamic systems may be taken.

1. Expand the continuum fluid variables in a superposition of expansion functions with the boundary conditions built into them.

2. Develop specific finite difference formulae for boundary cell values which combine the formulae used in the model with auxiliary relations to determine the value of the model variables outside the computational domain.

3. Extrapolate from the interior cells to guard cells outside the computational domain that effectively extend the model mesh beyond the model boundary.

Of these methods, the third, defining guard cell values, is the easiest to compute and is therefore adopted in the present work.

Three basic boundary conditions need to be implemented, a rigid boundary where a zero slip condition holds, a symmetric boundary and an inflow-outflow boundary.

The general formulation of the first two cases is essentially identical:

\[ \Phi_G = S_\Phi \Phi_{in} \]

where \( \Phi_G \) is the value of one of the conserved variables, \( \Phi \), in the guard cell, \( \Phi_{in} \) is the corresponding value in the interior cell and \( S_\Phi \) is either +1 or -1 depending on which \( \Phi \) is being considered and whether a rigid or symmetric boundary is being modelled.

Specifically, \( S_\Phi = +1 \) for all \( \Phi \) except the velocities in both cases. For the rigid boundary \( S_\Phi = -1 \) for all velocities. For the symmetric boundary \( S_\Phi = +1 \) for the velocity components parallel to the boundary and \( S_\Phi = -1 \) for the velocity component normal to the boundary.

The simulation of inflow-outflow boundary conditions presents a more complex problem as this procedure requires the representation of an effectively infinite region as a finite computational domain. The boundary conditions must effectively transmit information to and from the outside world, correctly reacting to signals coming from the computational domain.
Two techniques present themselves as possible solutions to this problem.

1. Extend the computational domain to a sufficient degree that boundary interactions do not affect the area of interest within the time modelled.

2. Analytically model the inflow-outflow boundary.

   In practice, the domain extension methodology is unsuitable, unless a non-uniform gridding method is applied, as the number of additional cells required in the model and the additional computational and memory overhead renders the problem intractable. Even if the cell sizes are progressively increased, the spatial resolution of the grid becomes inadequate to propagate signals at the wavelengths of interest.

   The second approach also has its associated problems as an analytic representation of the boundary may be difficult to derive. An inadequate formulation of the required boundary can result in spurious reflections from the boundary at some wavelengths, producing inaccurate simulation results.

   A third approach, essentially a combination of both of the above techniques, avoids the most serious problems encountered with each of the other methods. With this technique, the computational domain is extended beyond the area of interest for a relatively small finite distance and an analytical approximation is applied at the external boundary of this extended region. If we let $\Phi_{In}$ and $\Phi_G$ take their usual meanings then we can represent the boundary behaviour for most of the variables by the following expression:

   \[
   \Phi_G = \alpha \Phi_{In} + (1 - \alpha) \Phi_{At}
   \]

   where $\Phi_{At}$ is the value of the conserved variable at infinity (atmospheric condition) and $\alpha$ is set to either $K_T$, a time dependent variable, or $1 - K_T$ depending on whether an inflow or outflow condition obtains.

   The velocity is assumed to be constant across the boundary:

   \[
   \Phi_G = \Phi_{In}
   \]

   and the energy term is calculated from the interpolated pressure value:

   \[
   E_G = \frac{1}{\gamma} P_G + \frac{\rho}{2} (u^2 + v^2 + w^2).
   \]
This approach gives good agreement with published experimental results and is felt, on this basis, to provide an effective and computationally tractable mechanism for dealing with open boundaries.

To further increase the generality of the modelling scheme, additional boundary mechanisms have been developed by the author. These permit the modelling of vented systems in which a rigid boundary is destroyed at some point in the course of the explosion as a result of a pressure differential between the two sides of the boundary. The means by which this has been achieved, essentially the removal of the boundary when a set pressure differential is reached, are detailed in the discussion of the programs in the next chapter.
Chapter 3
— The Programs —
1. INTRODUCTION. The numerical solution of the equations of turbulent reactive flow presents a number of interesting challenges. First is the development of an efficient mechanism for the representation of the model data. Next is the need for a relatively straightforward means of creating the necessary input data files. Finally there are the problems associated with the production of an efficient parallel solution program.

The following sections deal with the approach taken to each of these challenges in turn.

A more comprehensive treatment of the data preparation program, gridder, and the explosion simulator, explode, may be found in the appendices where the full WEB program listings may be examined.
2. **DATA REPRESENTATION.** The data required for this simulation may be broadly grouped into four categories:

1. geometric data,

2. cell initial conditions,

3. processor placement and

4. global, model dependant, constants.

Thus, for each computational cell in the simulation, information must be provided on the location of the cell in relation to the other cells in the computational domain and the boundaries, the initial conditions obtaining within the cell and the processor on which computation for this cell will be carried out. As already noted a further set of model dependant constants must be made available for all cells on each processor. A further constraint is that the volume of input data should be minimised in order to optimise the mass storage and memory requirements and the initialisation time of the simulator.

The approach taken here is to produce three data files, a geometry file containing connectivity and placement information as well as a pointer to the appropriate initial condition set for each cell, a condition file containing the sets of initial conditions to be applied and a constants file containing the model dependant constants. The detail of each of these files is described in the following sections.
3. The geometry file must provide the simulator with information for each cell in the computational domain. In the case of the three-dimensional model this includes the identity of each of the six orthogonal neighbours. A naive approach therefore, might be to supply a set of six integers \( n_d \) where \( d \) ranges through \( x^- \) (left), \( x^+ \) (right), \( y^- \) (back), \( y^+ \) (front), \( z^- \) (below) and \( z^+ \) (above).

This simple scheme does not however completely specify the required geometric information. In addition to the neighbour data, indicators of the presence of boundaries are required. One possible solution would be the provision of six additional integer variables, \( b_d \), where each \( b \) indicates whether a boundary lies in direction \( d \) from the cell and, if so, what type of boundary it is. Thus we might code \( b \) as 0 if no boundary is present, as 1 for a rigid, zero slip boundary, 2 for a symmetric boundary, 3 for an inflow-outflow boundary and so on.

As the solution method to be employed involves multiple parallel processors, further information must be provided to enable the placement of each cell on the appropriate processor and to inform the program of the location of the neighbour cells. This could be achieved by the inclusion of a further seven integer values \( p_c \) and \( p_d \) where \( p_c \) is the identification number for the processor on which the current cell is to be placed and \( p_d \) are the six processor id’s for the neighbouring cells.

Finally the initial physical conditions in each cell must be specified. This could be carried out by the inclusion of ten additional values for each cell, being the values of the density, \( \rho \), the three velocity components, \( u, v \) and \( w \), the internal energy, \( h \), the fuel and mixture fractions, \( m_{fu} \) and \( f \), the variance of the mixture fraction, \( g \), and the turbulence parameters, \( k \) and \( e \).
4. The data representation scheme outlined in the preceding section involves the provision of a total of twenty nine values for each cell. When this is multiplied by the thousands of cells in a typical model the total volume of data reaches considerable size. Clearly, if a more compact data representation scheme can be found it will be of considerable benefit.

It should be noted that, when considering the neighbours, a cell has either another cell or a boundary contiguous with each side. Thus it is possible to combine the $n$ and $b$ arrays by the simple mechanism of using a positive number to represent the id of a neighbouring cell and a negative number to represent a boundary condition.

As a finite number of cells are to be allocated to each processor the processor id may also be incorporated into this neighbour value by combining the cell and processor numbers into a single integer. Thus for each processor the cells local to that processor are numbered from 1 and the global identifier for the cell is derived from the expression:

$$g = Np + l$$

where $g$ is the global cell identifier, $N$ is the maximum number of cells per processor, $p$ is the processor number on which a given cell is to be sited and $l$ is the local cell number. In the present study $N = 1000$ giving a global identifier of, for example, 5297 the meaning; cell 297 on processor 5.

5. In a simulation of this nature a relatively small number of distinct sets of initial conditions are typically found within the cells of the computational domain. There exists, therefore, a considerable quantity of redundant data in the preceding coding scheme. Replacing the ten initial condition values with a single initial condition set identifier and subsequently broadcasting a small number of initial condition data sets to all processors eliminates this redundancy.

By employing this coding scheme, the twenty nine values required for each cell can be reduced to a total of only eight, namely the global cell id, $g$, six hybrid neighbour/boundary id’s, and the initial condition set id.
6. A further complication arises with the addition of vents, boundaries which are destroyed when a given pressure differential exists across them. In this situation a means must be developed which allows the identity of the neighbouring cell to be specified so that an appropriate connection between cells is established when the barrier is destroyed. This issue is resolved by encoding a vent boundary with the negative of the neighbour cell id. As the cell id's are greater than 1000, this scheme allows the maintenance of the required information and does not clash with the other boundary conditions.

Boundaries of this type are treated in exactly the same way as rigid, no slip, boundaries with the addition of a regular check on the pressure differential between the current cell and the cell on the other side of the vent. When the pressure difference between the cells exceeds some predefined value the sign of the neighbour id is reversed effectively eliminating the boundary. This process will also occur in the cell on the other side of the boundary.
7. **GRIDDER.** Once a data structure has been determined, the next task is to develop a mechanism for specifying the computational domain in a relatively simple way. Manually creating each of the eight variables for several thousand cells would be both a time consuming and error prone process. Clearly, a better solution to the problem of data preparation must be sought.

It was therefore decided, particularly given that all of the required data were capable of a geometric representation, to develop a means for specifying all of the required information in a single graphical format.

One of the most popular graphical input tools currently available is the computer aided draughting package AutoCAD. It was decided that this software package would provide an ideal medium for the preparation of the required input data. Two features of the AutoCAD software lent themselves particularly to the task at hand; layers and the DXF output file format.

The AutoCAD layer facility, shared with a number of other CAD packages, allows the creation of graphical entities on a series of numbered layers which, taken together, form a single drawing. The attractive feature of this ability provided by AutoCAD to group elements of the model in a structured manner was that it enabled each of the disparate but related elements of the model to be prepared in a single structure while retaining the ability to distinguish, via the layer number, the precise role each graphical entity was to play in the final data set.

The structure of the graphical representation of the model is as follows:

1. Layers 1, 2, 3... each contain a series of horizontal and vertical vectors which, taken together, specify the boundary conditions to be applied in the region to be modelled. Vectors on each layer correspond to sections of the boundary at which a particular boundary condition is to be applied; layer 1 to a rigid barrier, layer 2 to an open surface, layer 3 to a symmetric boundary and so on. Layer 9 is used to represent vented boundaries where the identity of the neighbour cell must be retained as well as the existence of the boundary. The set of vectors on each layer forms one or more closed contours.

2. Layer 0 contains a series of points which specify the locations within the area of interest at which sample output from the simulation is required. Layer 0 also contains a series of horizontal and vertical edges which specify the model domain. These vectors when traced around the contour in an anti-clockwise sense enclose the region of interest to their left.

3. Each of the layers -1, -2, -3... contains one or more closed regions, again consisting of a series of horizontal and vertical vectors, within which an alternate set of initial conditions is to be applied. This permits a single input file to contain the data required for a number of initial condition sets and/or a
complex system of initial conditions for a single simulation. Regions of the model not within such areas are to have a default initial condition set applied.

4. Each of the layers 1000, 2000, 3000... again contains one or more closed regions. In this case each such set of regions encloses that part of the model which is to be processed on a particular worker processor; regions within the areas on layer 1000 to be processed on worker 1, regions on layer 2000 by worker 2 and so on.

The other feature of AutoCAD which particularly lent itself to the preparation of input data sets was the DXF (Drawing Exchange Format) output file format. This format, again supported by a number of CAD packages, contains all of the information required to reproduce the input drawing in a format which makes it readily amenable to processing by other non-graphical programs.

Each graphical entity in the DXF file consists of a textual type indicator followed by a set of parameters which detail the particulars of that entity, each parameter also being preceded by a numerical type indicator. This structure permits a program to rapidly scan through a DXF file processing only those graphical entities with which it is concerned and skipping over all other file contents.

For example the DXF file entry for a line entity is as follows.

```
LINE
  8
  1000
  10
  1.0
  20
  5.0
  30
  0.0
  11
  7.0
  21
  5.0
  31
  0.0
  0
```

In common with all DXF file entities the line starts with a type identifier, 'LINE', followed by a series of
code/value pairs. Thus, the line above is on layer 1000, code 8, and lies between the points (1.0, 5.0, 0.0), codes 10, 20 and 30, and (7.0, 5.0, 0.0), codes 11, 21 and 31. The final 0 marks the end of the structure.

Structural information on each layer is encoded into one of a number of arrays, grid contains the model domain, condition contains the condition set data, horizontal boundary and vertical boundary contain the boundary conditions for horizontal and vertical boundaries respectively and worker number contains the index of the processor on which each cell is to be placed. For each of the arrays there is a one-to-one correspondence between the computational cells and the array entries.

For those lines which enclose a region, i.e. those defining the model domain, the condition sets and the worker numbers, the boundary vectors must be transformed into region entries in the appropriate array. The means by which this is accomplished is best understood by reference to the following example.

Let us consider the following simple geometry:

![Simple geometry diagram]

If this geometry is embedded in the grid and the vertical vectors are eliminated, the following structure emerges.

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```
Each vector is processed in turn. For right-pointing vectors, 1 is added to each cell above the vector; for left-pointing vectors, 1 is subtracted. This process is shown in the following series of figures:
As can be seen, at the end of this process each cell within the computational domain contains a one with all other cells set to zero. It should be noted that this process works successfully regardless of the order in which the vectors are processed.

Although the vertical vectors are not used by the program, they are useful as a means of visually verifying the correctness of the data file and are retained in the DXF file for this reason.

This grid filling algorithm is equally appropriate for the case where a number of regions, each with a different associated value are to be processed. In this case, the value associated with the region, rather then 1, is added and subtracted in a similar manner. This modified procedure is used to store the condition set and processor placement information for each cell.

It should be noted that, with the multi-value version of the fill algorithm, contiguous region boundaries must be specified twice, once for each of the two abutting regions. Clearly the vectors along this edge will be in opposite directions.

Once this processing is completed for each entry in the DXF file, the entries in the grid array are scanned and each non-zero cell is cross referenced against the corresponding entry in the worker_number array and is
assigned a unique identification number. This number is composed of a local sequence number, maintained independently for each processor, plus 1000 times the processor number. Thus, for each processor, a set of cells, numbered sequentially from $n001$, where $n$ is the processor number, is produced.

These cells may then be output with neighbour and condition information by correlation of the updated grid array with the horizontal boundary, vertical boundary and condition arrays.
8. **EXPLODE.** The implementation of the mathematical model, presented in the preceding chapter, involves the resolution of a number of issues, particularly where the solution mechanism is to be a parallel program.

The particular questions which are to be addressed are as follows.

1. How to distribute the computational load among the available processors?

2. How to implement the communication necessary between processors?

3. How to implement the boundary conditions in an efficient manner?

Each of these broad areas involve the solution of a number of problems and the following sections deal with these and their resolution in the *exployd* program.
§9 — The Programs —

9. A number of potential approaches to the distribution of the processing load among the available processors are available for consideration. These approaches may be broadly categorised into three classes.

1. The processor farm. This methodology carries out parallel computations by sending packets of work from a ‘master’ processor to several ‘slave’ processors. Each ‘slave’ carries out the necessary work and then transmits the results back to the ‘master’. The advantage of this approach is that it tends to minimise total processor idle time as each ‘slave’ is supplied with a new packet to process as soon as it has finished with the previous one. This approach to parallel computation is particularly well suited to problems where minimal amounts of data are required for the processing of each packet and where there is no interaction between the packets.

2. The processor pipeline. This approach operates by allocating some part of the computation to each processor. Packets are passed from processor to processor with each performing the next stage in the computation. The advantage of this approach is that, in real-time applications, the rate at which packets may be processed may be readily tuned by varying the amount of work performed at each stage in the pipe. As with the processor farm, this technique is best suited to problems where there is no interaction between packets.

3. The processor map. This technique involves the computation for a fixed data domain being carried out on each processor. A ‘root’ processor is responsible for any necessary process synchronisation between the ‘worker’ processors. The advantage of this approach is that the volume of communication between processors is minimised. This computational strategy is particularly suited to problems in which large volumes of data are required for each cell or where there is a degree of interdependence between cells.

The present problem is most closely matched to the third of these solution techniques, the processor map. As has already been foreshadowed in the discussion of the gridded program, the assignment of regions of contiguous cells to each processor allows the computation to proceed, to a large degree, independently on each processor. The only remaining problem is how to deal with the inevitable case where neighbouring cells lie on different processors.

The standard approach to this problem is to duplicate cells on more than one processor, updating cell contents at the end of each computational phase. Thus, if a set of cells processed by processor 1 has neighbours on processor 2, these cells are duplicated on the second processor as a set of passive ‘ghost’ cells. Similarly the cells from processor 2 are set up as ghosts on processor 1. This approach to domain decomposition is described in more detail by Carmona in his paper on the MACH2 code [8].

It was felt that this technique contained several shortcomings, specifically:
1. the concentration of all interprocess communication in the update phase represented a mechanism which had a high probability of swamping the interprocessor communication channels, particularly where these channels were of a relatively low speed;

2. the additional data storage requirements for the maintenance of the 'ghost' cells reduced the total model size for any given processor and memory configuration;

3. the data preparation procedure becomes more complex, requiring the identification and processing of the 'ghost' cells; and

4. the algorithmic complexity of the solution procedure is substantially increased, requiring additional code to handle the incorporation and update of 'ghost' cells in the model.

A better strategy for addressing this concern was, therefore, sought. Ideally, this strategy should incorporate the following features.

1. Interprocess communication should occur in a relatively continuous, asynchronous manner without the need for an end of phase update process.

2. Each cell should be stored only once, on a single processor.

3. The processing of each cell should, as far as possible, proceed in the same manner regardless of whether neighbouring cells are local to that cell’s processor.

The solution scheme adopted for the `explode` program satisfies each of the above criteria and, in addition, allows for further simplification of the algorithm when the question of boundary cells is considered.

The principal features of this solution scheme are as follows.

1. All access to neighbour information takes place via calls to a single subroutine, `fill_array`. Thus, in all other parts of the `explode` program, there is no distinction between cells with local neighbours and those which have some or all of their neighbours positioned on another processor.

2. Within `fill_array`, information for remotely located cells is obtained, as required, from the other processors.

3. As will be seen in the later discussion on the communications mechanism, the impact of interprocessor data requests on the processor servicing the request and on other, intermediate relay processors is very low.
10. In addition to the interprocessor communication necessary to support the provision of neighbour information, a number of other classes of data must be propagated to the 'worker' processors. These communications needs fall into a number of broad categories:

1. initialisation data including both processor specific local information relating to each cell and global information required by all 'worker' processors;

2. control information required for the synchronisation of processing between 'worker' processors;

3. output data from the 'worker' processors to the 'root' processor; and

4. neighbour cell data as noted above.

Given the requirement that data requests should not interrupt processing on the 'worker' processors, a communications model based upon message passing principles and multi-threaded processes was developed.

Each of the 'worker' processes consists of three asynchronous independent process threads or co-processes, communicating with each other via internal, memory based channels and with the other processors via external links. The arrangement of these threads and their relationship to other processors is shown below.

The processors in the present version of this program are arranged serially in a circular topology with each 'worker' processor communicating with the preceding and following processors. In the case of the first worker the upstream connection is to the 'root' processor and for the last worker the downstream connection is also to the root.

Message flow is unidirectional from processor to processor thus, in general, most packets will be processed
in some manner by all processors in the ring. This arrangement, involving as it does all processors in each message, adds some overhead to the operation of the simulation, particularly when the relatively slow speed (10MBits/sec) of the serial links between processors is considered. However, for a small number of ‘worker’ processors, this overhead is tolerable when considered against the per packet cost of a more complex routing algorithm. An alternative topology for larger systems of processors in which message transport is more nearly optimal will be discussed in a later chapter.

Message flow between the threads on each processor is also unidirectional, following the direction of the arrows in the preceding figure. As already noted, this message flow is asynchronous and, therefore, the switch thread must always be capable of receiving data from either the buffer or worker threads. This is accomplished by using the ALT construct in which the switch thread waits for messages from either of the internal channels connected to it.

The detail of message flow depends on two conditions:

1. the type of message involved; and

2. the role played by the processor in relation to the message.

This may best be explained by a more detailed examination of the messages and their routes through the threads.

Each message is implemented as a pair of packets. The first packet is of fixed size and contains the packet type and the size of the following data packet. This approach of using two packets per message is required as the communication model supported by the compiler used in this work requires that the process receiving a packet specify its size in the receipt routine. As packet sizes range from 0 to 292 bytes this approach is more efficient than that of using a single, fixed size packet large enough to contain the longest message required.

There are a total of 11 data packet types as noted below:

1. worker_count_type, used during initialisation to determine the number of ‘worker’ processors;

2. geometry_data_type, used during initialisation to transmit cell geometry to the ‘worker’ processes;

3. condition_data_type, used during initialisation to transmit initial condition data to the ‘worker’ processes;

4. constant_data_type, used during initialisation to transmit model dependent constants to the ‘worker’ processes;
5. `synch.type`; sent by the ‘root’ processor to the ‘worker’ processes to synchronise processing at the start of each phase of the computation, i.e. a query;

6. `timestep.type`; this packet is sent by the ‘root’ processor around all of the ‘worker’ processes and contains a global minimum timestep value on return to the ‘root’ process, i.e. a broadcast;

7. `timeset.type`, sent by the ‘root’ process to all ‘worker’ processes to update the local timestep to the value determined by the `timestep.type` packet;

8. `step_complete.type`, one packet of this type is sent by each ‘worker’ process to the ‘root’ process at the end of each phase of the computation enabling the ‘root’ process, by counting these packets, to determine when all ‘worker’ processes have completed a phase;

9. `report.type`, sent by the ‘root’ process to a ‘worker’ process and back, this packet is used to both request and supply the current values for a specific cell for the purpose of data logging; and

10. `data_request.type`, sent by a ‘worker’ process to request data values for a neighbour cell.

The routing of these packets and the operations, if any, performed on them by the ‘worker’ processes are as follows.

The `worker_count.type` packet is set to 0 by the ‘root’ process and sent to the first ‘worker’ processor in the ring. The buffer thread receives the packet and sends it to the worker thread. The worker thread adds 1 to the count and notes this value as its processor id. The packet is then sent by the worker thread to the switch thread which, in turn, passes it to either the buffer thread of the next processor or to the ‘root’ process. On return to the ‘root’ process, the packet contains the total number of ‘worker’ processes.

The `geometry.data.type` packets are sent, one packet for each cell in the computational domain, from the ‘root’ process to the ‘worker’ processes. On receipt, the buffer thread sends the packet to the worker thread which, if the cell is to be processed by this processor will store the data. If the packet is not destined for the current processor, the packet is sent from the worker thread, via the switch thread to the next processor.

The `condition.data.type` and `constant.data.type` packets are both handled in the same way. The packets are sent from the ‘root’ processor to the first ‘worker’ process. Each ‘worker’ process in turn routes the packet from the buffer thread via the worker thread to the switch thread and then on to the next ‘worker’ process or, in the case of the last ‘worker’ process, to the ‘root’ process. The worker thread of each process copies the packet contents to the appropriate data structures.
Process synchronisation is achieved by the use of two packet types; the `synch_type` packet and the `step_complete_type`. At the beginning of each computational phase the 'root' process sends a `synch_type` packet to the 'worker' processes. The `worker` threads receive and forward these packets and then start the computation of the next phase. Each `worker` thread, on completion of the computation, sends a `step_complete_type` packet to the 'root' process. These packets are routed directly from the `buffer` thread to the `switch` thread of intervening 'worker' processes. The 'root' process counts the `step_complete_type` packets and, when it has received as many packets as there are 'worker' processes, initiates the next phase with another `synch_type` packet.

The computational algorithm employed in this simulation uses an adaptive time step mechanism in which the chosen time step is taken as the minimum calculated timestep over all cells in the computational domain. To determine this global minimum and to broadcast it to all 'worker' processes is the task of the `timestep_type` and `timeset_type` packets. The process of time step determination is initiated by the 'root' process sending a `timestep_type` packet containing the `min_timestep` value. Each `worker` thread calculates the local minimum timestep over the cells in its control and, if this is less than the current contents of the packet, updates the value in the `timestep_type` packet before sending it to the next process. On return to the 'root' process the packet contains the global minimum required. This value is now routed to all 'worker' processes as the content of a `timeset_type` packet from which each `worker` thread extracts a copy.

The timestep for each cell is given by the expression:

\[ \Delta t = \frac{CAX}{c_s + AV} \]

where \( C \) is the Courant number, \( c_s \) is the local speed of sound and \( AX \) and \( AV \) are distance and velocity norms given by:

\[ AX = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

and

\[ AV = \sqrt{u^2 + v^2 + w^2} \]

This expression results in a conservative timestep for which computational stability is high. Thus, recalculation and refinement of the timestep is not needed within an iteration of the model and the consequent
§10 — The Programs —

Periodically, the ‘root’ process requests data for specific cells in the computational domain. These data are written to file by the ‘root’ process for subsequent analysis or as the basis of the animation sequences. This data collection is accomplished via the agency of the report_type packet. One packet of this type is sent from the ‘root’ process to the ‘worker’ processes for each cell for which data are required. The packet content is, at this point, simply the identification number of the cell of interest, the remainder of the packet being empty. The buffer thread of each process in turn examines this cell number and, if the cell is local to this process, fills the remainder of the packet with the requested data. The packet is then sent to the switch thread and from there to the next process.

Finally, the packet mechanism is used for the determination of neighbour information during the various computational phases. This process, achieved through the data_request_type packet, represents the most complex of the packet routing tasks. Apart from the ‘root’ process, which has no interest in these packets and merely passively routes them back to the first ‘worker’ process, three distinct cases exist.

1. The packet originated at this ‘worker’ process.
2. The requested data is resident on this ‘worker’ process.
3. Neither of the above is true.

The data requests from the originating ‘worker’ process are sent from the fillArray subroutine in the worker thread as a data_request_type packet. This packet is sent to the switch thread and thence to the buffer thread of the next process. The buffer thread of each process examines the packet contents to determine the identity of the requesting and serving processes. If neither of these match the current process identifier, the packet is sent to the switch thread unchanged (case 3). When the local process number matches the server process, the buffer thread fills the packet with the data associated with the cell of interest before sending the packet to its switch thread (case 2). The packet proceeds around the processor ring until it arrives back at the buffer thread of the originating process. From here the packet is sent to the worker thread where fillArray extracts the required data.

As may be seen from the above description, the worker thread of each process receives only those packets which contain relevant data. All other packets pass through only the buffer and switch threads of the process. This mechanism enables the elimination of almost all code involving packet manipulation from the worker thread and thus keeps the mathematical modelling independent of the underlying messaging system. Thus, not only is the code kept relatively simple, but the operation of the thread based messaging
The system is able to run independently of the computational aspects of the system. The scheduling of the three threads comprising the 'worker' process is achieved by the program via a combination of calls to the `f77_thread_deschedule` subroutine and the inbuilt automatic descheduling by the transputer of any thread waiting for data.

Once started each thread runs independently sharing the transputer. A thread will continue to run until it is descheduled either by the process of sending a message to another thread or by an explicit call to the `f77_thread_deschedule` subroutine. If the threads were to be run at the `f77_thread_noturg` priority instead of the default `f77_thread_urgent` a processor moderated time slicing mechanism would also be invoked to cause thread descheduling. As there is only a single processor intensive thread in the worker task this additional mechanism is not warranted in the present case.

A further advantage of this uncoupling of the computational and messaging aspects of the program is that it allows the replacement of the message routing code with no change to the `worker` thread. As will be seen in a later chapter, the simple ring processor topology may be significantly modified for large numbers of processors with minimal coding overhead.

Indeed, even if the program were to run on a shared memory, multi processor system or, at the other extreme, on a set of loosely coupled processors on a network in the style of the portable virtual machine (PVM) paradigm, the `worker` thread should need little or no change although significant modification of the buffer and switch mechanisms may well be needed.
11. As with the implementation of the messaging strategy the issue of dealing with boundary conditions in this program was approached with the goal of, as far as possible, isolating the bulk of the processing from the detail of handling this complication.

The fill_array subroutine in the worker thread, already used to abstract the message passing aspects from the main processing task, is used to manage the boundary condition code.

As the need arises to obtain the conditions applying in each neighbour of a given cell on a particular process, calls to fill_array are made to determine the relevant values. This subroutine is called with a number of arguments, the most important of which are:

1. W.in, an array containing the identity and data associated with the current cell;
2. W.out, an array into which the neighbouring cell's condition data are to be placed; and
3. direction, an indicator of which neighbour is required.

The array entry W.in(direction) contains the identification number of the requested neighbour cell.

Within fill_array, three cases exist:

1. if W.in(direction)/MAX_CELL = my_worker.id, the cell in the given direction from the current call is local to the current processor;
2. if W.in(direction)/MAX_CELL ≠ my_worker.id, the cell in the given direction from the current call is located on another processor; and
3. if W.in(direction) < 0, there is a boundary in the required direction from the current cell.

In the first case, where the required data is local, fill_array merely copies the data from the local work space into W.out. The second case, where the data lies on another processor, is handled via the data_request_type packet mechanism. The final case, where there is a boundary present, involves the determination of the boundary type and the filling of W.out with appropriate values. As explained in the preceding chapter, for most boundary conditions the required values are either copies of the data in W.in, possibly with sign changes or, in the case of an open boundary, a linear combination of the values in W.in and W.at, the atmospheric conditions.

Some additional complexity occurs in the case of a vented boundary. Here the value to be returned depends on whether the vent blows out. Thus, before W.out can be evaluated the program must determine
the pressure differential between the current cell and the cell abs(W_in(direction)). Depending on whether this cell is local or remote the same mechanisms are used as in the non-boundary case to determine the pressure, i.e. either local lookup or use of the data_request_type packet. If the pressure difference is less than \(K_{vent}\), the failure pressure value, the values returned are derived as in the case of a solid boundary. If the difference is greater than \(K_{vent}\) the contents of \(W_{in}(direction)\) is replaced with \(-W_{in}(direction)\) and the values for this cell are returned in \(W_{out}\). Clearly, once the vent fails, the situation is identical to the normal neighbour cases.

In order to calculate some of the required results in the simulation, cells in addition to the six (four in the two dimensional code) immediate orthogonal neighbours are required. Because the structure of \(W_{out}\) is identical to that of \(W_{in}\) this may be accomplished simply by calling fill_array with \(W_{out}\) in place of \(W_{in}\) and an additional receiving array.

Extension of the model by the addition of extra boundary types is made easy by this localisation, permitting, in the extreme case, the implementation of boundary functions of significant complexity which might model fuel leakage through walls or other active boundary conditions.

The preceding discussion serves to highlight the significant innovations in this work. The reader is referred to the appendices which contain the WEB program listings for a complete understanding of the code.
# Chapter 4

— Model Validation —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>RESULTS</td>
<td>3</td>
</tr>
</tbody>
</table>

February 11, 1994

22:02
1. **INTRODUCTION.** One problem which arises in the process of validation of a model for explosion processes is the limited quantity of experimental data which are available. Only one fairly complete series of experimental results for a large-scale explosion has been published, those dealing with the Raufoss experimental facility in Norway and, as will be seen later, even these results are to some extent of questionable merit in some respects.

A less complete set of experimental results has also been published by the U.S. Bureau of Mines for their Bruceton and Lake Lynn experimental mines but these data are again of a somewhat limited nature.

In this chapter the validation of the model is described using the Raufoss results as a basis for comparison.
2. The Raufoss explosion facility is a 10 metre long, 2.5 metre diameter cylindrical steel tube, closed at one end. Obstructing annular rings may be fitted along the tube occluding 50% of the cross-sectional area of the tube to induce additional turbulence. The tube may be filled with an explosive fuel-air mixture which may be ignited near the closed end of the tube using either a planar array of igniters or a single axially located one.

The computational model used in this discussion was a 2 dimensional slice through the tube with a 0.2 metre grid size and sampling points as shown on the following diagram. A rectangular Cartesian coordinate system was used for this simulation. Although a true cylindrical coordinate system would have been more appropriate it was decided that the additional coding was not justified given the principal aims of the simulation system.

Recent work by Green [13], based on the system presented here, has shown that the variations in temperature and velocity experienced between a rectangular approximation such as the one used here and a true cylindrical representation amount to some 10% with the higher values being associated with the cylindrical case. Pressure, being a derived variable, experiences a rise of some 30%.

Work carried out by the computational fluid dynamics group at Harwell [18] suggests similar, relatively small differences are to be expected between rectangular and cylindrical representations of an explosion model although the existence of an anomalous burning pattern in the cylindrical case raises some question as to the reliability of these findings.

These findings suggest that it should be possible to infer the behaviour of a complex three dimensional mine geometry from an examination of the results of a two dimensional computation, at least as far as determining the pattern and magnitude of the principal events.

<table>
<thead>
<tr>
<th>11x</th>
<th>12x</th>
<th>13x</th>
<th>14x</th>
<th>15x</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x</td>
<td>6x</td>
<td>7x</td>
<td>8x</td>
<td>9x</td>
</tr>
<tr>
<td>1x</td>
<td>2x</td>
<td>3x</td>
<td>4x</td>
<td></td>
</tr>
</tbody>
</table>

An additional sampling point (point 16) was located 9 metres beyond the end of the tube along the centre line. The model domain extended 10 metres beyond the open end of the tube and 2.4 metres to either side.
3. RESULTS. A total of 19 runs of the simulation were conducted with differing initial conditions. The various initial condition values used are given in the following table.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Ignition Source</th>
<th>Fuel</th>
<th>$k, \epsilon$</th>
<th>% Fuel</th>
<th>$k_{\text{stub}}$</th>
<th>$k_{\text{max fuel}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.00001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>7.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>8.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>10.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Planar</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>11.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>9.5%</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>13</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.00001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.001</td>
<td>9.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>7.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>8.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>10.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>Point</td>
<td>CH$_4$</td>
<td>0.0001</td>
<td>11.5%</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>Planar</td>
<td>C$_3$H$_8$</td>
<td>0.0001</td>
<td>4.5%</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

For each of these 19 runs data were collected for density, pressure, temperature, axial velocity, radial velocity, energy, fuel fraction and mixture fraction at each of the 16 sampling points. In each run a period of 0.5 seconds was simulated with sample values being extracted each millisecond.
4. The graphical results from these runs are presented on the accompanying CD-Rom in the folder Thesis Tube Charts as Run1 through Run19. These graphs may be viewed from within Windows after the CD-Rom is installed.

In addition to these files, an additional set of comparison charts have been produced. These are to be found in the Compare file and consist of the following data sets.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Data sets compared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varying $k, c$</td>
<td>Sets 2, 3 and 4</td>
</tr>
<tr>
<td>Varying fuel concentration</td>
<td>Sets 6, 7, 2, 8 and 9</td>
</tr>
<tr>
<td>Planar vs. Point ignition source</td>
<td>Sets 2 and 11</td>
</tr>
<tr>
<td>Varying location in tube</td>
<td>Set 2</td>
</tr>
<tr>
<td>Varying fuel type</td>
<td>Sets 2 and 19</td>
</tr>
</tbody>
</table>

5. In addition to the result graphs, a number of animations have been produced which are also on the accompanying CD-Rom. These are to be found in the folders Animations - Open Tube and Animations - Blocked tube. In each of these folders is a set of 19 animations consisting of density, energy, fuel fraction, mixture fraction, pressure, temperature, x-velocity and y-velocity presented in both 2-D and 3-D modes. An additional animation in each folder shows the velocity field.

The animations in the Animations - Open Tube folder represent the initial conditions to be found in data set 2. The animations in the Animations - Blocked Tube folder are the results for a run in which the same initial conditions were taken but the model geometry was modified by completely blocking the 5 rings, the centre disk of each ring being set to fail at a pressure differential of 0.3 Bar.

A third set of animations presenting the two cases in combination are to be found in the folder Animations - Combined Tube where the open tube's results are on the left hand side and the corresponding blocked tube data are on the right.
Generally the results of the simulation show close agreement with the published experimental data. The following discussion examines some features of the results of run 2, the data set which most nearly matches the experimental conditions for the published Raufoss results.

Examination of the pressure results show excellent agreement with the equivalent experimental results including the presence of acoustic modes enhancing the explosion. The peak pressure occurs against the outer ring. In the experimental setup, this ring was made from plywood and did not survive the blast.

The failure of the outermost ring was felt to be a significant cause of the observed differences between the simulation and the Raufoss results. In order to test this assumption an additional simulation was performed with the same initial conditions as run 4 but with a modification to the model allowing the outermost ring to fail at a pressure differential of 0.3 Bar.

This change in the model resulted in a drop in peak pressure of approximately one atmosphere and the disappearance of the secondary pressure rise at the end of the tube associated with the reflection of the pressure peak from the end ring. These new results agree almost exactly with the published data.

A consideration of the velocity field shows a stable vortex behind the outward moving combustion front at 129 msec, the time of peak pressure. This vortex has moved along the tube over the preceding 20 msec to a position 7 metres along the tube. The velocity within the tube at this time is around 600 m/sec, consistent with the published results.

The peak velocity occurs 18 msec later at 147 msec. These velocities are slightly in excess of 1000 m/sec, substantially higher than the reported experimental values. There is some doubt whether the reported experimental peak velocities represent true maxima. These velocities were measured with a hot wire probe and are only reported until the time at which the signal could no longer be interpreted, presumably because of breakage of the probe. The cessation of reported values coincides with a strong reversal of flow associated with the vortex following the initial outward pressure pulse. While the absence of corresponding experimental data for these later times renders the results of the simulation open to debate, the fact that the prior close agreement is obtained suggests that these results may well be consistent with reality.

A consideration of the radial velocity along the centreline for data set 4, the low initial turbulence case, shows the presence of strong symmetrical ringing due to transverse acoustic modes coupling to the flow. In this type of system, acoustic coupling is an additional explosion enhancing mechanism, the other being obstacle induced turbulence.

If the fuel fraction is examined from 111 msec to 121 msec in the animation of data set 2 the transition
from laminar to turbulent combustion may be seen. Up to 111 msec the combustion of the fuel has taken
the form of a smooth, uniform flame front progressing from a planar to a hemispherical domain. As a result
of the progressively increasing turbulence in the system, in part induced by the rings, burning becomes
non-uniform. Consumption of fuel between the rings proceeds at a much higher rate than near the walls.
Once again this agrees with the published data.

The results from the blocked tube show the slower development of the explosion in this case. They also
show the greater final levels of pressure and turbulence which would be expected with the subsequent failure
of the barriers. One unusual result found in this simulation was the partial failure of the obstruction in ring
3, the central portion failing at a much later time than the surrounding annulus.

A surprising result is to be found in the comparison of the results for planar and point ignition sources.
Contrary to initial expectation, and to some published results, the only apparent effect of a change in
ignition source is to produce a delay in detonation, the profiles being shifted by some 170 msec. On further
consideration, these results are less unlikely than would be expected, the chemical energy liberated from the
explosion being the major determinant of the process. The differences in the published results may well be
due to varying initial conditions of induced turbulence in the two cases, the point ignition source causing
less initial turbulence.

7. In summary, the simulation of the 10 metre ignition tube at Raufoss shows close agreement with the
published data including the presence of acoustic modes evident in the pressure profiles. The velocity profiles
agree with the published data both in shape and magnitude over the reported times. The break in reported
data appears to correspond to a strong vortex behind the initial outward pressure pulse.

A significant problem in the validation of theoretical and numerical models of this nature is the paucity of
experimental data on large scale explosions suitable for calibrating simulations of this type and complexity.
This problem becomes even more apparent when the results of a larger scale simulation are considered.
# Chapter 5

— Model Results —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>RESULTS</td>
<td>2</td>
</tr>
</tbody>
</table>

February 11, 1994

22:31
1. **INTRODUCTION.** A more comprehensive test of the simulation's capacity was afforded by the following model, a section of a mine consisting of two development headings with two crosscuts. A continuous miner (A), two shuttle cars (B) and a conveyor (C) are positioned in the south heading with a diesel rover (D) in the north heading.

The computational model used a 1 metre grid size and sampling points as shown on the following diagram.

The model extended some 40 metres to the east of the area shown along the two roadways to an inflow/outflow boundary.

The geometry of the headings and the positions of the vehicles correspond closely to the configuration found at Moura No. 4 mine in which an explosion occurred in July 1986. Much forensic evidence was available for this explosion and it was felt that the results could be used to partially validate the model for such a complex geometry.

A brick stopping with a failure pressure of 0.3 Bar (s) was placed in the east crosscut.
2. RESULTS. A total of 5 runs of the simulation were conducted with differing initial conditions. The various initial condition values used are given in the following table.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Grid Size</th>
<th>Courant Number</th>
<th>Fuel Distribution</th>
<th>Boundary Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1m.</td>
<td>0.9</td>
<td>Uniform</td>
<td>Undamped</td>
</tr>
<tr>
<td>2</td>
<td>1m.</td>
<td>0.2</td>
<td>Uniform</td>
<td>Undamped</td>
</tr>
<tr>
<td>3</td>
<td>2m.</td>
<td>0.9</td>
<td>Uniform</td>
<td>Undamped</td>
</tr>
<tr>
<td>4</td>
<td>1m.</td>
<td>0.9</td>
<td>Non-uniform</td>
<td>Undamped</td>
</tr>
<tr>
<td>5</td>
<td>1m.</td>
<td>0.9</td>
<td>Uniform</td>
<td>Damped</td>
</tr>
</tbody>
</table>

For each of these 5 runs data were collected for density, pressure, temperature, axial velocity, radial velocity, energy, fuel fraction and mixture fraction at each of the 8 sampling points. In each run a period of 2 seconds was simulated with sample values being extracted each 10 milliseconds.

In all runs except run 4, a stoichiometric mixture of methane and air filled the mine headings. In run 4, only the northwest corner of the mine contained such a mixture, the concentration of methane decreasing to zero in a series of regions towards the southeast.

In run 2, the Courant number was reduced from 0.9 to 0.2 to examine the degree of dependence of the model on timestep variations.

In run 3, the grid size was increased to examine the degree of scale invariance in the simulation. This change in grid size also required some simplification of the model boundary to align it to the changed grid.

In run 5 the rigid boundary conditions were changed to force $k, \epsilon$ damping.

All other model conditions were the same as those used in the Raufoss ignition tube simulation.

Ignition occurred in a band of cells to the west of the northwest intersection.
3. The graphical results from these runs are presented on the accompanying CD-Rom in the folder Thesis Mine Charts as Run1 through Run5. These graphs may be viewed from within Windows after the CD-Rom is installed.

In addition to the result graphs, a number of animations have been produced which are also on the accompanying CD-Rom. These are to be found in the folders Animations - Mine, Animations - Mine - Low Fuel and Animations - Mine - Damped corresponding to runs 1, 4 and 5 respectively.

In each of these folders is a set of 19 animations consisting of density, energy, fuel fraction, mixture fraction, pressure, temperature, x-velocity and y-velocity presented in both 2-D and 3-D modes. An additional animation in each folder shows the velocity field. The results for mixture fraction are only shown for the low fuel case as they remain at a constant value throughout the other two simulations.

4. The results of the simulation show considerable agreement with the forensic evidence available from the Moura explosion. The following discussion examines some features of the results of run 1.

The explosion developed along the western crosscut and along the north heading. At 390 msec, the pressure pulse reached the northeast intersection and a spherical shockwave is clearly evident at this location. A second, reflected, shock is also visible in the southwest intersection. This reflection and the high pressure at the western end of the southern heading would not have occurred at Moura as these roadways were not blocked as is the case with the simulation.

If the velocity is examined at the same time (390 msec), a high negative x-velocity may be seen in the northern roadway due to partial reflection of pressure from the rover and the corners of the heading. The high y-velocity associated with the reflected shockwave in the southwest corner may also be clearly seen. The high x-velocity in the southern roadway is a consequence of the constriction in the airway between the two intersections east of sample point 2.
5. The main pressure front travelled south along the eastern crosscut, destroying the stopping and striking the shuttle car at 430 msec after ignition. The subsequent sustained pressure differential across the shuttle car would be enough to rotate the eastern end of the car to the south. This was seen in the actual explosion at Moura. This sustained pressure arises from two sources, south along the east crosscut and east along the south roadway. In general the regions of sustained high pressure observed in the simulation were consistent with the movement of and damage to equipment at Moura. Of particular note in this context is the extreme pressure at the west end of the conveyor. At Moura this boot-end, weighing several tonnes, was blown some distance along the roadway and the beltway itself was severely damaged for a distance in excess of 100 metres.

6. In the south heading the principal flow is to the northern side of the continuous miner. A pool of unburnt gas remained on the southern side of the miner for a second or so. This differential burning has a marked effect on the temperature as can be seen 500 msec after ignition. This temperature differential persists for a period of some 500 msec, due to the trapping of a stable pocket of burnt gas between two opposing flows. This pocket eventually dissipates towards the west. It is interesting to note that evidence of significant heating and coking of coal dust and subsequent flow in this area was evident following the Moura explosion along with evidence of strong differential heating between the northern and southern surfaces of the miner.

7. The peak transient pressures in the simulation reach 3500 KPa while major sustained pressure peaks lasting several hundreds of milliseconds reach levels between 1000 and 1300 KPa. These values seem somewhat high but given the large quantity of stoichiometric methane present and the limited experimental or other data available for explosions on this scale there is nothing to assert that the pressures are over predicted. Indeed, the recent failure of the Buxton explosion gallery during tests with very large fuel volumes [39] strongly suggests the validity of the results obtained in these simulations.

8. Comparison of the results of run 1 with those of run 3 illustrate that variation in grid size has little effect on the simulation with variations of at most 20%, some portion of which may be attributed to the change in geometry. Comparison of runs 1 and 2 show that lowering the Courant number and the subsequent reduction in timestep has the effect of smoothing the explosion processes but does not significantly affect either peak values or overall shape of the results.
9. Varying the amount and distribution of methane present in the simulation has, not surprisingly, a more dramatic effect on peak pressures. In an examination of the results of run 4, sustained pressures reach a maximum of around 500 KPa. This simulation involves gas quantities and produces results comparable to explosions undertaken in experimental facilities.

It is worth noting however that, even with this much less violent explosion, the pressure and temperature distributions are still consistent with some of the forensic evidence available from the Moura site.

10. Changing the boundary conditions to allow damping of turbulence once again results in lower peak values, though not to the extent found in the low fuel case. Transient pressure peaks of the order of 2000 KPa are observed with sustained peak values of some 800 to 1000 KPa. The most significant effects of this change are to be seen in the turbulence results, not surprisingly, and in the rate of fuel combustion, a variable closely related to turbulence levels.

Once more, the same gross features are present in the results as in the other cases, features which are consistent with the Moura evidence.

11. In addition to the forensic evidence from Moura, a 1/50 scale model of a section of the mine was constructed at the University of Wollongong. This model consisted of a 25mm thick craftwood base upon which a 50mm layer of craftwood was fixed producing a scale model with 100mm×50mm airways. A 25mm thick sheet of perspex provided a transparent upper surface which allowed observation of the flame spread resulting from the ignition of a fuel-air mixture within the model. Spark igniters were positioned at various locations in the perspex sheet allowing a number of different ignition points to be examined. The results of the tests were recorded on video tape for later analysis.

Although complete instrumentation of the model was not carried out, the subsequent results allowed the determination of flow and temperature distributions by observation of flame motion and colour. As with the forensic evidence, the results from the physical model testing served to provide general support to the results of the simulation as presented here.
Chapter 6
— Extensions —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1 57</td>
</tr>
<tr>
<td>INITIAL CONDITIONS</td>
<td>2 58</td>
</tr>
<tr>
<td>BOUNDARY CONDITIONS</td>
<td>5 60</td>
</tr>
<tr>
<td>GEOMETRY</td>
<td>9 61</td>
</tr>
<tr>
<td>SOLID PHASE</td>
<td>12 62</td>
</tr>
<tr>
<td>TOPOLOGY</td>
<td>13 63</td>
</tr>
</tbody>
</table>

February 11, 1994
23:03
1. INTRODUCTION. This chapter considers a number of possible extensions to the explosion simulator. The changes considered fall into a number of broad categories.

i. Initial conditions. This includes extension of the model to allow for the inclusion of randomly and smoothly varying initial conditions, enabling the modelling of systems which more nearly resemble those found in real situations.

ii. Boundary conditions. This includes the addition of active boundaries and the use of boundary functions to couple the simulation to some other physical process.

iii. Geometry. This includes the extension of the model to allow non-uniform cell sizes and the partial obstruction of cells.

iv. Solid Phase. The inclusion of statistical or particle tracking and the associated reaction modelling.

v. Topology. This involves improvements to the message routing algorithm.

The following sections examine each of these areas in turn.

It should be noted that, although each of these extensions would serve to enhance the performance or flexibility of the simulator, the implementation of the proposals discussed in this chapter are outside the scope of the present work.
2. **INITIAL CONDITIONS.** In the present version of the program the initial conditions are constrained to take a small finite set of values.

This restriction, while permitting a wide range of problems to be addressed, limits the use of the model to the consideration of problems in which this assumption of discrete uniform regions holds. Thus, while appropriate for the simulation of experimental systems, the current model is not especially suitable for the modelling of real systems.

An extension of the current initial condition mechanism could remedy these perceived shortcomings.

The utilisation of negative condition indices could, in a manner analogous to the use of negative connectivity array entries, be used along with some additions to the condition set data to signal the presence of extended condition sets in the following manner.

3. **Random Conditions.**

If the condition set data, $c_i$, are augmented by the addition of a set of standard deviation values $\sigma_i$ then a set of randomly distributed values could be placed in the affected cells by generating a normally distributed random sample $z_i$ for each variable in each cell and setting the value of the given condition in that cell to $c_i + z_i \sigma_i$. Clearly the use of a zero value for $\sigma_i$ would result in no variation for the corresponding $c_i$, allowing the present uniform initial conditions to be included in a natural manner.

The use of this device would readily allow the introduction of the sorts of small random variations in initial conditions typical of real explosions.
4. Uniformly Varying Conditions.

Another, more complex, extension would allow the modelling of situations in which the initial conditions follow a smooth gradient over some spatial distribution.

By utilising some of the unfilled entries in \( W_p \) and setting the condition code to a negative value, modification of the program could allow such variation to occur.

First it should be noted that, at present, only 8 entries in \( W_p \) are initialised by the root task for each cell. By increasing the size of the geometry data packet from 8 to 12 and using the additional 4 values to hold \( \delta_x, \delta_y \) and \( \delta_z \), the effective distance of the cell from the point at which maximum condition value occurs in each of the \( x, y \) and \( z \) directions and \( g \) the gradient of the variation in initial conditions, the addition of smoothly variant initial conditions could be carried out in the following manner.

If the expression \( \delta = \sqrt{\delta_x^2 + \delta_y^2 + \delta_z^2} \) is evaluated, the cell value for each condition variable is given by the expression \( (1 - \frac{\delta}{g})(c_i + z_i\sigma_i) \) where \( c_i \), \( z_i \) and \( \sigma_i \) are defined as in the preceding section. It should be noted that the simpler expression \( (1 - \frac{\delta}{g})c_i \) could be used if random variation of initial conditions was not to be used.

It should also be noted that \( \delta_x, \delta_y \) and \( \delta_z \) need not simply represent a linear distance, they could in fact be arbitrarily complex functions.
5. **BOUNDARY CONDITIONS.** One rich field for extension of the present code is in the area of boundary conditions. Modifications in this area should involve relatively minor changes to the code as all such changes will only affect the `fill_array` subroutine.

6. The first class of extensions in this area involves the addition of active boundaries, similar to the current inflow outflow condition. These would involve the modification of the boundary condition, possibly in response to conditions in the bounded cell. Examples of this type of boundary might include a provision for fuel injection across a rigid boundary, simulating the leakage of gas through the mine wall, or the injection of inert or combustible material into the bounded cell when some predetermined velocity or temperature condition was reached, simulating the picking up of material from the mine wall or the evaporation of some volatile material.

7. A second category of extension in boundary modelling is the addition of wall functions, additional code to allow for energy transfer mechanisms between the mine walls and the enclosed reactive cells. The present model does not allow for such energy loss to the walls except for the simple turbulence limiting code in the case of boundary condition 4.

8. The final potential extension to the model involves the dedication of some processors to the modelling of a boundary process, an independent task which models some specific physical system which feeds into the explosion domain. One such boundary process might be one which models the effect of a roof fall in a section of the mine, passing velocity, pressure and other flow related data into the current simulator via an appropriate boundary condition.
9. **GEOMETRY.** The addition of enhancements to the underlying geometric basis of the computational model could be used to allow a greater range of problems to be addressed. This section examines some possible developments in this area.

The current model is based on a constant size, uniform cell structure with each cell being fully open. The implications of relaxation of these constraints will now be considered.

10. The use in the current model of a constant cell size imposes restrictions on the physical extent of the system that may be modelled given that the cell size must be made sufficiently small to allow the capture of significant data in the areas of greatest activity. The inflow outflow boundary condition attempts to address this difficulty to some extent by providing a reasonable extrapolation of the internal conditions out to the model exterior.

This mechanism is not however wholly satisfactory, particularly when an extremely energetic process reaches such a boundary, as transient behaviour is not readily dealt with.

Modification of the geometry to allow variation of cell size throughout the region being modelled would allow a small cell size in areas of interest along with a larger cell size in more remote areas of the model. This change would permit the extension of model size without resorting to either an inappropriately large fixed cell size or to an unreasonable number of processors.

11. The modelling of obstructions to flow in the present program is constrained by the requirement that cells be fully open. Thus placing a restriction on the nature of the obstructions that can be modelled. If the program is to be used to model explosions in other industrial contexts than the mining industry, such as the petrochemical industry, the addition of some mechanism which permits the modelling of sub-cell sized obstructions is needed.

One possible solution to this problem is to decouple the cross-sectional area of the cell from the cell size. The addition of an obstruction factor to each of the three x, y and z directions would achieve this by replacing the current expression $A_x = \Delta y \Delta z$ with an expression of the form $A_x = o_x \Delta y \Delta z$, where $o_x$ is the obstruction factor.
12. **SOLID PHASE.** The program in its present form models only gaseous material. Solid material, both reactive and inert, plays a major role in explosions, both as a fuel and as a retarding mechanism. The extension of the program to incorporate particulate systems is, therefore, a worthwhile consideration.

Two approaches suggest themselves to the solution of this problem. The first is to incorporate a statistical mechanism for particle distribution, the second is to use a particle tracking mechanism. In both cases the contribution of the particulate matter to the reaction would be based on the concentration of particle species within a cell, which concentration would be modified over time by the fluid dynamical behaviour of the gaseous mode.

The inclusion of solid phase calculation to the program could thus be realised via the addition of a secondary process over each cell, coupled to the gas phase calculations already in place. This extra process could be implemented on a series of additional processors communicating over the spare serial links on each worker processor. Thus one additional transputer for each four workers could be added to cover the solid phase calculation. The alternative computational strategy of adding the solid phase calculation directly to the current worker task has the advantage of minimising communication overhead but at the cost of reducing the number of cells which may be handled by each processor.
13. **TOPOLOGY.** The program in its present form utilises a simple ring structure to connect the processors with inter-processor communication taking place in a single direction.

As each message travels around the entire ring the total time used in processing messages is given by the expression

\[ T_{tot} = N_p(T_{pack} + T_{fill} + T_{unpack} + N_r T_{route}) \]

where \( T_{pack} + T_{unpack} \) represents the time taken by the originating processor in packing and unpacking the packet, \( T_{fill} \) is the time taken by the server processor to fill the request and \( T_{route} \) is the time spent by each of the other workers in routing the packet around the ring. The routing time on the root processor is not included in \( T_{tot} \) as the root processor is not in a position to be doing any useful work at this point.

14. As the number of workers involved in routing the packet \( N_r \) and the number of packets \( N_p \) increases, the total time spent in packet handling rises to the point where the bulk of the available processing time is used in packet handling. It should be noted that in the case of the present ring topology \( N_r = N - 2 \) where \( N \) is the total number of workers.

This escalation of \( T_{tot} \) can be addressed via two mechanisms, the reduction of \( N_p \) and the reduction of \( N_r \).

The reduction of \( N_p \) can be achieved by increasing the number of cells on each processor but the resulting increase in solution time severely limits this utility of this approach.

The reduction of \( N \) is clearly not a solution to the problem as this has the effect of reducing the model size. Thus a mechanism which changes the relationship between \( N_r \) and \( N \) is required.

15. A standard solution to this problem is the use of the hypercube connection scheme where each processor in a network of \( N = 2^n \) processors is connected to \( n \) neighbours, each processor occupying a vertex of an \( n \)-dimensional hypercube. This connection scheme reduces \( N_r \) to \( n - 2 = \log_2 N - 2 \) resulting in a massive reduction of \( T_{route} \) and hence in \( T_{tot} \) for large \( N \).

Two difficulties arise in this scheme when applied to the present problem. First, the message flow along the edges of the hypercube is bidirectional, requiring a more complex routing code. Second and more important, this connection scheme requires \( n \) links to each processor effectively reducing the total possible processors to 16 given that the transputer has only four communications links.

Clearly then, the hypercube is not the appropriate topology for the present problem.
16. An alternative scheme more suited to the transputer architecture is the rectangular mesh whereby \( N = m \times n \) processors are connected such that processor \((i, j)\) is in direct communication with processors \((i \pm 1, j)\) and \((i, j \pm 1)\).

This scheme again requires bidirectional communication but reduces \( N_r \) to an average value of \( \frac{m+n}{2} \approx \sqrt{N} \).

This value of \( N_r \) while clearly better than \( N - 2 \) is far from the \( \log_2 N \) achieved with the hypercube.

17. One superficially attractive scheme is to connect the processors in a manner which models, as nearly as possible, the geometric relationship between the groups of cells modelled on each transputer. With this approach \( N_r \approx 0 \) as each worker is ideally directly connected to the processors from which it requires data.

There is however a significant problem with this approach, even assuming that the ideal \( N_r = 0 \) connection scheme can be attained. This is the requirement that the physical connections between processors be rewired for each new model. A time consuming and potentially error prone procedure more suited to the days of ENIAC than to the present.

18. The solution to this problem may well lie in the following connection scheme, one which provides \( N_r = n - 2 \) for \( N = 2^n \) processors, allows unidirectional message flow and requires only four links per processor for any \( n \).

Each processor is assigned a number \( i \) in the range \( 0 \leq i \leq N - 1 \) and the processors are connected such that

\[
i \to \frac{i}{2}, \quad i \to \frac{N + i}{2}, \quad i \to 2i \pmod{N}, \quad \text{and} \quad i \to 3 \cdot 2i + 1 \pmod{N}.
\]

Note that each worker number \( i \) is an \( n \)-digit binary number.

Now consider the route from processor \( i \) to processor \( j \). First construct the new index \( k_0 = Nj + i \), a 2n-digit binary number, then construct the series \( k_m \) using the relationship \( k_m = \frac{k_{m-1}}{2}, m = 1, 2, \ldots, n \).

If \( k_m^* \) is now defined as \( k_m^* = k_m \pmod{N} \) it can be seen that \( k_0^* = i \) and \( k_n^* = j \). Furthermore, using the connection scheme described above either \( k_{m-1}^* \to k_m^* \) or \( k_{m-1}^* \to k_m^* \) for each \( m, 1 \leq m \leq n \), the specific link used (0 or 1) depending on the value of the high order binary digit in each successive \( k_m^* \). Thus, for any \( 0 \leq i, j < N \) a connection exists involving exactly \( n \) links.
The following example may serve to clarify this mechanism.

Let \( n = 3 \), then the links between the processors \( i = 0, 1, \ldots, 7 \) are as follows.

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Or, considered graphically,

![Diagram](image)

Thus, if a message from processor 1 to processor 3 and back is considered, the sequence of transfers is as follows.

\[
0 \rightarrow 1 \rightarrow 4 \rightarrow 6 \rightarrow 3 \quad \text{and} \quad 3 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0
\]

It should be noted that, for any \( n \), there will always be 2 unused links on each of processor 0 and processor \( N - 1 \). These links would be connected to the root processor in the same way as in the ring topology.

The connection scheme presented here exhibits the same worst case behaviour as the hypercube topology and, for large \( N \) the additional complexity in the message routing procedure is more than compensated for by the reduction in \( N_r \). Although the simple scheme given here does not always give the shortest possible route from \( i \) to \( j \) it allows for a implementation with a minimal additional computational overhead.
1. BIBLIOGRAPHY. The following list contains the books and journal articles which were used as background to this work.


Science and Technology. 27, 159-170.


Appendix A
— Griddor —

INTRODUCTION .................................................................................................................. 1 72
THE PROGRAM .................................................................................................................. 6 75
INITIALISATION .................................................................................................................. 9 77
THE INPUT PHASE .......................................................................................................... 15 81
SET BOUNDARY ARRAY .................................................................................................... 27 86
SET GRID ARRAY .............................................................................................................. 32 88
SET WORKER ARRAY ......................................................................................................... 35 90
SET CONDITION ARRAY ................................................................................................. 38 92
THE OUTPUT PHASE ......................................................................................................... 41 94
INDEX .................................................................................................................................. 54 100

February 12, 1994
0:36
1. INTRODUCTION. One of the first problems that must be confronted in the task of preparing input data for the explosion modeller is that of developing a concise mode for the representation of all of the required data.

Consideration of the requirements of the simulation process indicated a need for the following data items.

i. The geometric structure of the region to be simulated.

ii. The conditions to be applied to each boundary of the region.

iii. The conditions to be initially set at each point within the region.

iv. The position of each of the sample points within the region.

v. The disposition of sections of the region between multiple worker tasks.

Clearly, although the simulator itself utilises input data in numerical form, this is not an ideal format for the initial preparation of the input data, being both time consuming and error prone in its preparation. It was therefore decided, particularly given that all of the required data were capable of a geometric representation, to develop a means for specifying all of the the required information in a single graphical format.

2. One of the most popular graphical input tools currently available is the computer aided draughting package AutoCAD. It was decided that this software package would provide an ideal medium for the preparation of the required input data.

Two features of the AutoCAD software lent themselves particularly to the task at hand; layers and the DXF output file format.
3. The AutoCAD layer facility, shared with a number of other CAD packages, allows the creation of graphical entities on a series of numbered layers which, taken together, form a single drawing. The attractive feature of this ability provided by AutoCAD to group elements of the model in a structured manner was that it enabled each of the disparate but related elements of the model to be prepared in a single structure while retaining the ability to distinguish, via the layer number, the precise rôle each graphical entity was to play in the final data set.

The structure of the graphical representation of the model is as follows:—

i. Layers 1, 2, 3... each contain a series of horizontal and vertical vectors which, taken together, specify the boundary conditions to be applied in the region to be modelled. Vectors on each layer correspond to sections of the boundary at which a particular boundary condition is to be applied; layer 1 to a rigid barrier, layer 2 to an open surface, layer 3 to a symmetric boundary and so on. The set of vectors on all such layers form one or more closed contours.

ii. Layer 0 contains a series of points which specify the locations within the area of interest at which sample output from the simulation is required. Layer 0 also contains a series of horizontal and vertical edges which specify the model domain. These vectors when traced around the contour in an anti-clockwise sense enclose the region of interest to their left.

iii Each of the layers −1, −2, −3... contains one or more closed regions, again consisting of a series of horizontal and vertical vectors, within which an alternate set of initial conditions are to be applied. This permits a single input file to contain the data required for a number of initial condition sets and/or a complex system of initial conditions for a single simulation. Regions of the model not within such areas are to have a default initial condition set applied.

iv. Each of the layers 1000, 2000, 3000... again contains one or more closed regions. In this case each such set of regions encloses that part of the model which is to be processed on a particular worker processor; regions within the areas on layer 1000 to be processed on worker 1, regions on layer 2000 by worker 2 and so on.
4. The other feature of AutoCAD which particularly lent itself to the preparation of input data sets was the DXF (Drawing Exchange Format) output file format.

This format, again supported by a number of CAD packages, contains all of the information required to reproduce the input drawing in a format which makes it readily amenable to processing by other non-graphical programs.

Each graphical entity in the DXF file consists of a textual type indicator followed by a set of parameters which detail the particulars of that entity, each parameter also being preceded by a numerical type indicator. This structure permits a program to rapidly scan through a DXF file processing only those graphical entities with which it is concerned and skipping over all other file contents.

5. Given the input file structure described above, a program is now required to produce the digital input file needed for the simulation program. The program which achieves this task, *gridder*, is described and specified in the following sections.
6. THE PROGRAM. This program takes an AutoCAD DXF input file containing the data described in the preceding sections and produces gridded output suitable for input to the explosion simulator.

Two constants which control the maximum size of the problem must be declared. \( MAX\_GRID \) which sets the upper limit on the size of the gridded data and \( MAX\_SAMPLE \) which sets the number of allowed sample points.

As the program may be required to generate gridded data to run on an array of transputers, the maximum number of worker processes, \( MAX\_TRANSPUTER \), and the maximum number of cells per transputer, \( MAX\_CELL \), must also be declared.

Finally, the maximum number of negative layers \( MAX\_CONDITION \), corresponding to alternate initial condition sets, must also be declared.

```fortran
define MAX_GRID 200
define MAX_SAMPLE 100
define MAX_TRANSPUTER 10
define MAX_CELL 1000
define MAX_CONDITION 10
```

7. A number of aliases for common programming idioms are also declared.

```fortran
define do_nthing continue
define prompt write(*, '(lx, a$)')
define get_string read(*, '(a)')
format prompt write
format get_string read
```
8. The skeleton of the program is presented below. The detail of the code will be expanded in the following modules.

```plaintext
program gridder,
    implicit none

    (Variables of the program 9)

    (Get input values and open input files 10)

    (Zero the arrays 14)

    (Process the input file 17)

    (Open output files 41)

    (Set the cell indices 43)

    (Write the output 44)

end
```
9. **INITIALISATION.** In addition to the input DXF file, *gridder*_8 requires a few additional values.

First, to permit mapping from a vector to a raster representation of the data, the x and y cell sizes, *x.celLsize* and *y.celLsize* must be declared.

Each entry in the condition layer array *condition.layer*, indicates whether that particular layer of the DXF file will be interpreted as containing the vector representation of an alternate condition set to be used with this particular output data set.

The initial non-blank sequence of characters in the variable *base.name* will hold the base file name. This name with the string '.*dxf' appended to it should be the input DXF file.

It is convenient to define a number of macros to simplify the process of generating file names from the variable *base.name*.

```
define base_len index(base_name, 'U') - 1
define base_file base_name(1: base_len)
define fullname(a) base_file || a
define open_unit(a,b) open(unit = a, file = fullname(b))
format open_unit open
```

(Variables of the program 9) =

- real *x.cell.size*, *y.cell.size*
- character*1 condition_layer(MAX_CONDITION)
- integer condition_index
- logical more_conditions
- character*40 base_name

See also sections 12, 13, 16, 19, 21, 25, 27, 29, 42, 45, and 52.

This code is used in section 8.
10. The required input parameters are now read in.

   define input_file 10

   (Get input values and open input files 10) ≡
   prompt 'Enter input file base name:'
   get_string base_name
   open_unit (input_file, '.dxf')
   prompt 'Enter grid X cell size:'
   read (*,*) x_cell_size
   prompt 'Enter grid Y cell size:'
   read (*,*) y_cell_size

   (Get active condition layers 11)

This code is used in section 8.

11. The program must now proceed to accept a series of zero or more active condition layer numbers.

   Each entry in the condition_layer array will contain either a '0' or a '1'. A '1' entry indicates that regions on the corresponding layer are to be flagged on output as containing an alternate initial condition set. A '0' entry indicates that the layer in question is to be ignored for the purposes of data preparation.

   (Get active condition layers 11) ≡
   do condition_index = 1, MAX_CONDITION
      condition_layer(condition_index) = '0'
   enddo
   prompt 'Enter condition layer numbers (0 to terminate):'
   more_conditions = T
   do while (more_conditions)
      read (*,*) condition_index
      if (condition_index ≡ 0) then
         more_conditions = F
      else
         condition_layer(-condition_index) = '1'
      endif
   enddo

This code is used in section 10.
12. The array $grid$ will contain the gridded representation of the mine interior, the array $condition$ will be used to determine which cells are within the explosion initiator area, the array $horizontal\_boundary$ will contain the gridded representation of the horizontal boundaries and the array $vertical\_boundary$ will contain the gridded representation of the vertical boundaries.

To conserve space, the arrays $condition$, $horizontal\_boundary$ and $vertical\_boundary$ will be declared as character variables and the following three macros will be used to manipulate their contents.

The macros $set\_char\_to\_int$, $add\_int\_to\_char$ and $get\_int\_from\_char$ simplify the task of storing, manipulating and extracting numerical data from character variables.

\[
\text{define } set\_char\_to\_int (array, value) \text{ array } = \text{char}(ichar('0') + value)
\]
\[
\text{define } add\_int\_to\_char (array, value) \text{ array } = \text{char}(ichar(array) + value)
\]
\[
\text{define } get\_int\_from\_char (array) \text{ (ichar(array) — ichar('0'))}
\]

{Variables of the program 9} +≡

integer $grid(0: MAX\_GRID, 0: MAX\_GRID)$
character*1 $condition(0: MAX\_GRID, 0: MAX\_GRID)$
character*1 $horizontal\_boundary(0: MAX\_GRID, 0: MAX\_GRID)$
character*1 $vertical\_boundary(0: MAX\_GRID, 0: MAX\_GRID)$
integer $ix, iy$

13. In addition to these arrays, two more are required to enable output suitable for processing on an array of transputers to be produced. The array $worker\_number$ will contain the worker number on which each cell is to reside and the array $counters$ will contain the count of cells allocated to each worker.

{Variables of the program 9} +≡

character*1 $worker\_number(0: MAX\_GRID, 0: MAX\_GRID)$
integer $counters(0: MAX\_TRANSPUTER)$
integer $worker\_index$
§14  — Gridder —

14. All the arrays must be initialised. In the case of the grid and counters arrays, the initial value will be 0. The other four arrays; condition, horizontal_boundary, vertical_boundary and worker_number will be initialised to '0'.

(Zero the arrays 14) ≡

\[
\begin{align*}
&\text{do } ix = 0, MAX\text{-GRID} \\
&\quad \text{do } iy = 0, MAX\text{-GRID} \\
&\quad \quad \text{grid}(ix, iy) = 0 \\
&\quad \quad \text{condition}(ix, iy) = '0' \\
&\quad \quad \text{horizontal\_boundary}(ix, iy) = '0' \\
&\quad \quad \text{vertical\_boundary}(ix, iy) = '0' \\
&\quad \quad \text{worker\_number}(ix, iy) = '0' \\
&\quad \text{enddo} \\
&\text{enddo} \\
&\text{do } worker\_index = 0, MAX\text{-TRANSPUTER} \\
&\quad \text{counters}(worker\_index) = 0 \\
&\text{enddo}
\end{align*}
\]

This code is used in section 8.
15. THE INPUT PHASE. The input file is an AutoCAD DXF file which contains vector data defining the mine geometry, the regions of the mine to be processed on each worker and the initial explosion condition areas as well as a set of point data representing the sampling points.

As has been noted above, to represent the varying boundary condition values, use has been made of the layer facility in AutoCAD; layer 1 corresponding to a boundary condition value of $-1$, layer 2 to boundary condition $-2$ and so on. Layers of the form $1000n$ are used to define the region of the mine layout to be processed on worker $n$. Negative layers are used to contain vector data related to different areas in which an explosion may be initiated; each negative layer corresponding to one set of initial conditions. Thus a single data file can contain a number of initial condition sets which are able to be selected for output through the value of the variable `condition_layer`.

16. The state of the variable `more_data` is used to signal the end of the input DXF file, the variable `n_of_points` is a counter of the number of sample points specified and the character variable `input_line` will be used to hold input from the DXF file.

(Variables of the program 9) \(+\equiv\)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical</td>
<td><code>more_data</code></td>
</tr>
<tr>
<td>integer</td>
<td><code>n_of_points</code></td>
</tr>
<tr>
<td>character</td>
<td><code>input_line</code></td>
</tr>
</tbody>
</table>

17. As stated above, the DXF file contains all of the required geometric data needed to fully specify a model, the interior volume, the worker assignments, the boundary conditions, the initial internal conditions and the points at which sample data output are to be produced.

This information may be broadly categorised as vector and point data, representing region and boundary information and sample data locations respectively.

(Process the input file 17) \(\equiv\)

```
    n_of_points = 0
    more_data = T
    do while (more_data)
        (Process the next input file item 18)
    enddo
    close(unit = input_file)
```

This code is used in section 8.
18. The input DXF file must now be sequentially scanned, processing each record type in an appropriate manner.

\[
\begin{align*}
\text{(Process the next input file item 18) } & \equiv \\
\text{read} & \quad (\text{input}\_\text{file}, 'a') \quad \text{input}\_\text{line} \\
\text{if} & \quad (\text{input}\_\text{line} \equiv 'EOF') \quad \text{then} \\
\text{more\_data} & = \text{false} \\
\text{elseif} & \quad (\text{input}\_\text{line} \equiv 'LINE') \quad \text{then} \\
\quad & \quad \{ \text{Process line input 20} \} \\
\text{elseif} & \quad (\text{input}\_\text{line} \equiv 'POINT') \quad \text{then} \\
\quad & \quad \{ \text{Process point input 26} \} \\
\text{endif}
\end{align*}
\]

This code is used in section 17.

19. The variable \textit{layer} is used to hold the layer in the DXF file on which a vector or point is declared. The variables \textit{start.x}, \textit{start.y}, \textit{end.x} and \textit{end.y} represent the start and end coordinates of a vector or, in the case of a sample pair, the former pair, \textit{x.start} and \textit{y.start}, represent its coordinates.

\[
\begin{align*}
\{ \text{Variables of the program 9} \} & \equiv \\
\text{integer} & \quad \textit{layer} \\
\text{real} & \quad \textit{start.x}, \textit{start.y}, \textit{end.x}, \textit{end.y}
\end{align*}
\]
20. Line entries in the DXF file contain data corresponding to a mine boundary in the case of data on positive layers where \( \text{layer} < 1000 \), worker assignments where \( \text{layer} \geq 1000 \) or to the edge of a condition set in the case of negative layers. In the case of a mine boundary, the raster equivalent of the input vector must be stored in the grid array and the appropriate values set into one of the arrays \( \text{horizontal}.\text{boundary} \) and \( \text{vertical}.\text{boundary} \). Worker selection data are stored in the array \( \text{worker}.\text{number} \). For a condition vector, assuming that the level is one selected for this run of grider, the raster equivalent is stored into the condition array.

The first input image in a vector record contains the string 'LINE'. Subsequent images contain data relating to the layer on which the vector occurs (prefixed by an image containing '08'), the starting x-coordinate (prefixed by '10'), the starting y-coordinate (prefixed by '20') and the ending x- and y-coordinates (prefixed by '11' and '21').

Once the vector has been read, the raster data (if any) is stored into the appropriate array.

\[
\begin{align*}
\text{(Process line input 20)} & \equiv \\
\text{(Get layer 22)} & \\
\text{(Get start point 23)} & \\
\text{(Get end point 24)} & \\
\text{if (layer} & \geq 1000 \text{) then} \\
\text{(Process worker entry 35)} & \\
\text{elseif (layer} & > 0 \text{) then} \\
\text{(Process boundary entry 28)} & \\
\text{elseif (layer} & \equiv 0 \text{) then} \\
\text{(Process geometry entry 32)} & \\
\text{elseif (condition}.\text{layer}(\text{-layer}) & \neq '\text{0}' \text{) then} \\
\text{(Process condition entry 38)} & \\
\text{endif} & \\
\end{align*}
\]

This code is used in section 18.
21. The following modules scan the DXF file for key codes and set appropriate variables.

Because vector and point records in the DXF file may contain fields other than those required for this program, for example z-coordinates, it is not possible to simply read in the required sequence of values. These modules repeatedly read code-value pairs until the input code value read in is equal to the required code value.

\[
\text{(Variables of the program 9) } + \Xi \\
\quad \text{integer input code}
\]

22. First, the layer value, corresponding to an input code of 8.

\[
\text{(Get layer 22) } \equiv \\
\quad \text{input code } = 0 \\
\quad \text{do while (input code } \neq 8) \\
\quad \quad \text{read (input file,*) input code} \\
\quad \quad \text{read (input file,*) layer} \\
\quad \text{enddo}
\]

This code is used in section 20.

23. Next the starting coordinates with input codes of 10 and 20.

\[
\text{(Get start point 23) } \equiv \\
\quad \text{input code } = 0 \\
\quad \text{do while (input code } \neq 10) \\
\quad \quad \text{read (input file,*) input code} \\
\quad \quad \text{read (input file,*) start x} \\
\quad \text{enddo}
\]

\[
\text{input code } = 0 \\
\quad \text{do while (input code } \neq 20) \\
\quad \quad \text{read (input file,*) input code} \\
\quad \quad \text{read (input file,*) start y} \\
\quad \text{enddo}
\]

This code is used in sections 20 and 26.
24. And last, the end coordinates with input codes of 11 and 21.

(Get end point 24) ≡

\[
\text{input\_code} = 0 \\
\text{do} \text{ while (input\_code \neq 11)} \\
\quad \text{read (input\_file,*) input\_code} \\
\quad \text{read (input\_file,*) end\_x} \\
\text{endo} \\
\text{input\_code} = 0 \\
\text{do} \text{ while (input\_code \neq 21)} \\
\quad \text{read (input\_file,*) input\_code} \\
\quad \text{read (input\_file,*) end\_y} \\
\text{endo}
\]

This code is used in section 20.

25. The arrays \textit{sample\_x} and \textit{sample\_y} contain the integer equivalent to the sample point coordinates.

(Variables of the program 9) +≡

\[
\text{integer sample\_x(MAX\_SAMPLE)} \\
\text{integer sample\_y(MAX\_SAMPLE)}
\]

26. In a similar manner to vector data, point data in the DXF file consists of a keyword, in this case ‘POINT’ followed by the x- and y-coordinates of the point (prefixed by ‘10’ and ‘20’ respectively). The x and y indices corresponding to the point are calculated and stored in the next sequential location in the \textit{sample\_x} and \textit{sample\_y} arrays.

(Process point input 26) ≡

\[
\text{n\_of\_points}++ \\
\text{(Get start point 23)} \\
\text{sample\_x(n\_of\_points) = start\_x/y.cell\_size} \\
\text{sample\_y(n\_of\_points) = start\_y/y.cell\_size}
\]

This code is used in section 18.
27. **SET BOUNDARY ARRAY**. Once a vector is read in from the DXF file, it remains to insert a raster representation into the horizontal boundary or vertical boundary arrays.

The variables `istart-X, istart-y, i.end-X` and `i.end-y` are the grid indices corresponding to the starting and ending vector coordinates `start.x, start.y, end.x` and `end.y`

(Variables of the program 9) +=

integer `istart-x, istart-y, i.end-x, i.end-y`

28. The vector data in the DXF file is structured such that when each area is traced in an anti-clockwise sense, the interior of the region is to the left of each vector. Thus, four cases can be distinguished, one for each of the vector directions; up, down, left and right. For each case, the boundary value equivalent for this vector, passed in the layer variable, will be stored in the row of cells corresponding to the vector’s location in one of the arrays vertical boundary or horizontal boundary.

(Process boundary entry 28) =

`istart.x = start.x / x.cellsize + 0.5`
`istart.y = start.y / y.cellsize + 0.5`
`i.end.x = end.x / x.cellsize + 0.5`
`i.end.y = end.y / y.cellsize + 0.5`

if `istart.x = i.end.x` then
    (Do vertical boundary vector 30)
elseif `istart.y = i.end.y` then
    (Do horizontal boundary vector 31)
else
    do nothing /* Possible input error? Diagonal line? */
endif

This code is used in section 20.

29. The following scratch variables are required for the vector processing.

(Variables of the program 9) +=

integer `ix1, ix2, iy1, iy2, jy, incr`
§30 — Gridder — SET BOUNDARY ARRAY

30. In the case of a vertical boundary the boundary condition value is inserted in each appropriate cell in the array vertical_boundary.

(Do vertical boundary vector 30) ≡

if \( i_{start.y} < i_{end.y} \) then /* VECTOR IS UP */
   \[ \begin{align*}
   ix &= i_{start.x} \\
   iy1 &= i_{start.y} \\
   iy2 &= i_{end.y} - 1
   \end{align*} \]
else /* VECTOR IS DOWN */
   \[ \begin{align*}
   ix &= i_{start.x} \\
   iy1 &= i_{end.y} \\
   iy2 &= i_{start.y} - 1
   \end{align*} \]
endif

do iy = iy1, iy2
   set_char_to_int(vertical_boundary(ix,iy), layer)
dendo

This code is used in section 28.

31. In the case of a horizontal boundary the boundary condition value is similarly inserted in each appropriate cell in the array horizontal_boundary.

(Do horizontal boundary vector 31) ≡

if \( i_{start.x} < i_{end.x} \) then /* VECTOR IS RIGHT */
   \[ \begin{align*}
   ix1 &= i_{start.x} \\
   ix2 &= i_{end.x} - 1 \\
   iy &= i_{start.y}
   \end{align*} \]
else /* VECTOR IS LEFT */
   \[ \begin{align*}
   ix1 &= i_{end.x} \\
   ix2 &= i_{start.x} - 1 \\
   iy &= i_{start.y}
   \end{align*} \]
endif

do ix = ix1, ix2
   set_char_to_int(horizontal_boundary(ix,iy), layer)
dendo

This code is used in section 28.
32. **SET GRID ARRAY.** This processing involves the addition of 1 to each cell above an edge in the array *grid* for right pointing vectors and the subtraction of 1 from each such cell for left pointing vectors.

The result of this procedure is to set all internal cell values to 1 and all exterior cell values to 0.

The truth of the assertion that the above procedure in fact achieves the desired result may be easily shown.

The algorithm is closely related to that of finding the area of a set of closed regions bounded by horizontal and vertical edges. Clearly, the area of such a set is equal to the sum of all areas between the upper boundaries of such regions and an exterior horizontal line above all such region boundaries subtracted from the sum of all areas between the lower boundaries of such regions and the same exterior upper horizontal line.

By noting that for an anti-clockwise set of boundaries each upper boundary will point to the left and each lower boundary to the right, the congruence of the algorithm and the above area-finding technique will become clear.

\[
\begin{align*}
\text{Process geometry entry } 32 \equiv & \\
\text{istart-}x &= \text{start-}x/x_\text{.cell.size} + 0.5 \\
\text{istart-}y &= \text{start-}y/y_\text{.cell.size} + 0.5 \\
\text{i.end-}x &= \text{end-}x/x_\text{.cell.size} + 0.5 \\
\text{i.end-}y &= \text{end-}y/y_\text{.cell.size} + 0.5 \\
\text{if } (\text{istart-}x = \text{i.end-}x) & \text{ then} \\
& \text{(Do vertical geometry vector 33)} \\
\text{elseif } (\text{istart-}y = \text{i.end-}y) & \text{ then} \\
& \text{(Do horizontal geometry vector 34)} \\
\text{else} & \\
& \text{do.nothing} /* \text{ POSSIBLE INPUT ERROR? DIAGONAL LINE? */} \\
\text{endif} \\
\end{align*}
\]

This code is used in section 20.

33. **As the interior region of the model is bounded by a set of closed curves containing only vertical and horizontal edges, and given the region filling algorithm detailed above, no action need be taken for vertical edges.**

\[
\begin{align*}
\text{Do vertical geometry vector 33} \equiv & \\
\text{do.nothing} \\
\end{align*}
\]

This code is used in section 32.
Diagonal vectors are processed to set the interior region.

\[
\text{(Do diagonal geometry vector 34) } \equiv \\
\begin{align*}
\text{if } (i_{\text{start}.x} < i_{\text{end}.x}) \quad &/ \quad \text{VECTOR IS RIGHT} * / \\
&i_{x1} = i_{\text{start}.x} \\
i_{x2} = i_{\text{end}.x} - 1 \\
i_{y} = i_{\text{start}.y} \\
in_{c} = 1 \\
\text{else } &/ \quad \text{VECTOR IS LEFT } * / \\
i_{x1} = i_{\text{end}.x} \\
i_{x2} = i_{\text{start}.x} - 1 \\
i_{y} = i_{\text{start}.y} \\
in_{c} = -1 \\
\end{align*}
\]

\text{endif}

\text{do } i_{x} = i_{x1}, i_{x2}

\text{do } j_{y} = i_{y}, \text{MAX}_\text{GRID}

\text{grid} (i_{x}, j_{y}) = \text{grid} (i_{x}, j_{y}) + i_{c} \\
\text{endo}

\text{endo}

This code is used in section 32.
35. **SET WORKER ARRAY.** Vectors on layers 1000, 2000, 3000 etc. are used to define areas of the geometry array for each worker. Cells within the 1000 region are to be processed on worker 1, cells in the 2000 region on worker 2 and so on.

This code essentially mirrors that required to handle the grid array except that there is no boundary condition data to process and that the value of the worker, `worker_number` is added to or subtracted from each cell. This procedure will result in each cell containing the number of the worker on which it is to be processed.

\[
\text{(Process worker entry 35) } \equiv \\
\quad \text{i.start.x} = \text{start.x}/x\_cell.size + 0.5 \\
\quad \text{i.start.y} = \text{start.y}/y\_cell.size + 0.5 \\
\quad \text{i.end.x} = \text{end.x}/x\_cell.size + 0.5 \\
\quad \text{i.end.y} = \text{end.y}/y\_cell.size + 0.5 \\
\quad \text{if } (\text{i.start.x} \equiv \text{i.end.x}) \text{ then} \\
\quad \quad \text{(Do vertical worker vector 36)} \\
\quad \text{else} \\
\quad \quad \text{(Do horizontal worker vector 37)} \\
\quad \text{endif}
\]

This code is used in section 20.

36. In the case of the `worker_number` array, because finding the interior region is all that is required, there is no need to process vertical vector data.

\[
\text{(Do vertical worker vector 36) } \equiv \\
\quad \text{do\_nothing}
\]

This code is used in section 35.
37. For a horizontal boundary, for the reasons stated earlier, only those cells above the boundary need to be processed.

This processing is essentially identical to that for the grid array.

(Do horizontal worker vector 37) \(\equiv\)

```plaintext
if (i.start.x < i.end.x) then /* VECTOR IS RIGHT */
  il = i.start.x
  i2 = i.end.x - 1
  iy = i.start.y
  incr = layer/1000
else /* VECTOR IS LEFT */
  il = i.end.x
  i2 = i.start.x - 1
  iy = i.start.y
  incr = -layer/1000
endif

do ix = il, i2
  do jy = iy, MAXGRID
    add_int_to_char(worker.number(ix,jy),incr)
  enddo
enddo
```

This code is used in section 35.
38. **SET CONDITION ARRAY.** Once a vector is read in from the DXF file, it remains to insert a raster representation into the *condition* array.

This code is essentially the same as that required to handle the *worker_number* array except that the condition code rather than the worker number is to be stored.

\[
\begin{align*}
\text{(Process condition entry 38) } & \equiv \\
\text{ } & i.\text{start}.x = \frac{\text{start}.x}{x.\text{cell.size}} + 0.5 \\
\text{ } & i.\text{start}.y = \frac{\text{start}.y}{y.\text{cell.size}} + 0.5 \\
\text{ } & i.\text{end}.x = \frac{\text{end}.x}{x.\text{cell.size}} + 0.5 \\
\text{ } & i.\text{end}.y = \frac{\text{end}.y}{y.\text{cell.size}} + 0.5 \\
\text{if } (i.\text{start}.x \equiv i.\text{end}.x) \text{ then} & \\
\text{ } & \text{(Do vertical condition vector 39)} \\
\text{else} & \\
\text{ } & \text{(Do horizontal condition vector 40)} \\
\text{endif}
\end{align*}
\]

This code is used in section 20.

39. In the case of the *condition* array, as with the *worker_number* array, there is no need to process vertical vector data.

\[
\begin{align*}
\text{(Do vertical condition vector 39) } & \equiv \\
\text{ } & \text{do.nothing}
\end{align*}
\]

This code is used in section 38.
40. Again, only those cells above the boundary need to be processed.

This processing is again essentially identical to that for the grid array.

\[
\begin{align*}
\text{(Do horizontal condition vector 40) } & \equiv \\
\text{if } (i._{\text{start}}.x < i._{\text{end}}.x) \text{ then } /* \text{VECTOR IS RIGHT */} & \\
& \quad \text{ixl} = i._{\text{start}}.x \\
& \quad \text{ix2} = i._{\text{end}}.x - 1 \\
& \quad \text{iy} = i._{\text{start}}.y \\
& \quad \text{incr} = -\text{layer} \\
\text{else } /* \text{VECTOR IS LEFT */} & \\
& \quad \text{ixl} = i._{\text{end}}.x \\
& \quad \text{ix2} = i._{\text{start}}.x - 1 \\
& \quad \text{iy} = i._{\text{start}}.y \\
& \quad \text{incr} = \text{layer} \\
\text{endif} \\
\text{do } \text{ix} = \text{ixl}, \text{ix2} & \\
\text{do } \text{jy} = \text{iy}, \text{MAX GRID} & \\
& \quad \text{add.int.to.char(condition(ix, jy), incr)} \\
\text{endo\ndo} \\
\end{align*}
\]

This code is used in section 38.
41. **THE OUTPUT PHASE.** The gridder program uses two output files. The geometry file (unit 11) contains the coordinates of each cell in the mine interior along with the connectivity data for each cell. The sample file (unit 12) contains the cell numbers at which sample values are to be reported.

   define geometry_output 11
   define sample_output 12

(Open output files 41) ≡

   open_unit (geometry_output, '.geo')
   open_unit (sample_output, '.smp')

This code is used in section 8.

42. A variable to hold the number of cells in the interior of the boundary is required.

(Variables of the program 9) +≡

   integer n_of_cells
43. At this stage, grid is one at each interior cell and is zero elsewhere. Numbers of the form layer are present in the horizontal boundary and vertical boundary arrays at all boundary cells and again zeros elsewhere. The array worker_number will contain the index of the worker process on which each cell is to be processed.

The grid array can now be scanned replacing the ones with a series of numbers of the form \( n \times \text{MAX.CELL} + i \), where \( n \) is the worker number and \( i \) is the sequential cell number of each worker.

If the output is not to be produced for a transputer array, the ones in the grid array must be replaced with a sequence of ascending integers.

\[
\text{(Set the cell indices 43) } \equiv \\
\text{n.of.cells} = 0 \\
\text{do } ix = 0, \text{MAX.GRID} \\
\text{do } iy = 0, \text{MAX.GRID} \\
\text{if } (\text{grid}(ix,iy) > 0) \text{ then} \\
\quad \text{n.of.cells}++ \\
\quad \text{worker.index} = \text{get.int.from.char(} \text{worker.number}(ix,iy) \text{)} \\
\quad \text{counters(} \text{worker.index} \text{)}++ \\
\quad \text{grid}(ix,iy) = \text{MAX.CELL*worker.index + counters(} \text{worker.index} \text{)} \\
\text{endif} \\
\text{enddo} \\
\text{enddo} \\
\]

This code is used in section 8.
44. Finally, one more pass is made over the grid array. For each positive cell value entries are written to the connectivity data file.

\[(\text{Write the output 44}) \equiv \]
\begin{verbatim}
write(\text{geometry.output, *}) \text{n.of.cells}
do \text{ix} = 0, \text{MAX.GRID}
do \text{iy} = 0, \text{MAX.GRID}
\text{if}(\text{grid}(\text{ix}, \text{iy}) > 0) \text{then}
\quad (\text{Write connectivity file 46})
\text{endif}
enddo
enddo
\end{verbatim}

\[(\text{Write sample file 53})\]

This code is used in section 8.

45. For each interior cell, the x- and y-coordinates of its lower left corner, and of its centre, the lengths of its horizontal and vertical sides, the values contained in each of its four neighbouring cells and the initial condition index are written to the geometry file.

\[(\text{Variables of the program 9}) + \equiv \]
\begin{verbatim}
real \text{xcorner}, \text{ycorner}
integer \text{ihere}, \text{iright}, \text{ileft}, \text{ibottom}, \text{itop}
\end{verbatim}

46. The connectivity array may now be written.

\[(\text{Write connectivity file 46}) \equiv \]
\begin{verbatim}
\text{ihere} = \text{grid}(\text{ix}, \text{iy})
(\text{Get the right neighbour index 47})
(\text{Get the left neighbour index 48})
(\text{Get the bottom neighbour index 49})
(\text{Get the top neighbour index 50})
(\text{Get the condition value 51})
\text{xcorner} = \text{ix}\times\text{z.cell.size}
\text{ycorner} = \text{iy}\times\text{y.cell.size}
write(\text{geometry.output, *}) \text{ihere, xcorner, ycorner, z.cell.size, y.cell.size, iright, ileft, itop, ibottom, condition_index}
\end{verbatim}

This code is used in section 44.
47. To determine the right hand neighbour index to the cell in \( \text{grid}(ix, iy) \) the \( \text{vertical boundary}(ix + 1, iy) \) entry must be examined. If it is non-zero, the cell has a boundary to its right and it is this boundary value that must be output. Otherwise the value of \( \text{grid}(ix + 1, iy) \) will be written.

An exception to the above is the case where the boundary value is equal to 9. In this case the boundary represents a pressure dependent vent and the neighbour index is set to \(-\text{grid}(ix + 1, iy)\).

\[
\text{(Get the right neighbour index 47) } \equiv \\
\quad \text{if } (\text{vertical boundary}(ix + 1, iy) \equiv '9') \text{ then} \\
\quad \quad \text{iright} = -\text{grid}(ix + 1, iy) \\
\quad \text{else if } (\text{vertical boundary}(ix + 1, iy) \neq '0') \text{ then} \\
\quad \quad \text{iright} = -\text{get int from char(vertical boundary}(ix + 1, iy)) \\
\quad \text{else} \\
\quad \quad \text{iright} = \text{grid}(ix + 1, iy) \\
\text{endif}
\]

This code is used in section 46.

48. Similarly for the left hand neighbour, the value of \( \text{vertical boundary}(ix, iy) \) must be examined.

\[
\text{(Get the left neighbour index 48) } \equiv \\
\quad \text{if } (\text{vertical boundary}(ix, iy) \equiv '9') \text{ then} \\
\quad \quad \text{ileft} = -\text{grid}(ix - 1, iy) \\
\quad \text{else if } (\text{vertical boundary}(ix, iy) \neq '0') \text{ then} \\
\quad \quad \text{ileft} = -\text{get int from char(vertical boundary}(ix, iy)) \\
\quad \text{else} \\
\quad \quad \text{ileft} = \text{grid}(ix - 1, iy) \\
\text{endif}
\]

This code is used in section 46.
49. For the bottom neighbour, $horizontal.boundary(ix,iy)$ is examined.

\[
\text{(Get the bottom neighbour index 49) } \equiv \\
\text{if } (\text{horizontal.boundary}(ix,iy) \equiv '9') \text{ then} \\
\quad ibottom = -\text{grid}(ix,iy - 1) \\
\text{else if } (\text{horizontal.boundary}(ix,iy) \neq '0') \text{ then} \\
\quad ibottom = -\text{get.int.from.char(horizontal.boundary}(ix,iy)) \\
\text{else} \\
\quad ibottom = \text{grid}(ix,iy - 1) \\
\text{endif}
\]

This code is used in section 46.

50. Finally, for the top neighbour $horizontal.boundary(ix,iy+1)$ is examined.

\[
\text{(Get the top neighbour index 50) } \equiv \\
\text{if } (\text{horizontal.boundary}(ix,iy + 1) \equiv '9') \text{ then} \\
\quad itop = -\text{grid}(ix,iy + 1) \\
\text{else if } (\text{horizontal.boundary}(ix,iy + 1) \neq '0') \text{ then} \\
\quad itop = -\text{get.int.from.char(horizontal.boundary}(ix,iy + 1)) \\
\text{else} \\
\quad itop = \text{grid}(ix,iy + 1) \\
\text{endif}
\]

This code is used in section 46.

51. In addition a condition value is required to indicate which initial condition set is to be associated with the cell. The value of $condition.index$ for each is derived from the corresponding entry in the $condition$ array.

\[
\text{(Get the condition value 51) } \equiv \\
\quad condition.index = \text{get.int.from.char(condition}(ix,iy))
\]

This code is used in section 46.

52. The variable $point.index$ is needed for the following module.

\[
\text{(Variables of the program 9) } \equiv \\
\quad \text{integer } point.index
\]
53. Finally, the cell numbers corresponding to the sampling points are written to the sample file.

\[(\text{Write sample file 53}) \equiv \]
\begin{verbatim}
write(sample_output, *) n_of_points
do point_index = 1, n_of_points
    write(sample_output, *) grid(sample_x(point_index), sample_y(point_index))
enddo
\end{verbatim}

This code is used in section 44.
54. INDEX. The index contains references to all variables and macro definitions in the program. Underlined module numbers refer to the module in which an indexed item is first referenced.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Module Numbers</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_int_to_char</td>
<td>12, 37, 40</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>AutoCAD</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>base_file</td>
<td>9, 10</td>
<td></td>
</tr>
<tr>
<td>base_len</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>base_name</td>
<td>9, 10</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>condition</td>
<td>12, 14, 20, 38, 39, 40, 43</td>
<td></td>
</tr>
<tr>
<td>condition_index</td>
<td>9, 11, 46, 51</td>
<td></td>
</tr>
<tr>
<td>condition_layer</td>
<td>9, 11, 15, 20</td>
<td></td>
</tr>
<tr>
<td>counters</td>
<td>13, 14, 43</td>
<td></td>
</tr>
<tr>
<td>do_nothing</td>
<td>7, 28, 32, 33, 36, 39</td>
<td></td>
</tr>
<tr>
<td>DXF</td>
<td>2, 9, 15, 16, 18, 20, 21, 26, 28, 38</td>
<td></td>
</tr>
<tr>
<td>end_x</td>
<td>19, 24, 27, 28, 32, 35, 38</td>
<td></td>
</tr>
<tr>
<td>end_y</td>
<td>19, 24, 27, 28, 32, 35, 38</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>full_name</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>geometry_output</td>
<td>41, 44, 46</td>
<td></td>
</tr>
<tr>
<td>get_int_from_char</td>
<td>12, 43, 47, 48, 49, 50, 51</td>
<td></td>
</tr>
<tr>
<td>get_string</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>grid</td>
<td>12, 14, 20, 32, 34, 35, 37, 40, 43, 44, 46, 47, 48, 49, 50, 53</td>
<td></td>
</tr>
<tr>
<td>gridder</td>
<td>5, 8, 9</td>
<td></td>
</tr>
<tr>
<td>horizontal_boundary</td>
<td>12, 14, 20, 27, 28, 31, 43, 49, 50</td>
<td></td>
</tr>
<tr>
<td>i_end_x</td>
<td>27, 28, 31, 32, 34, 35, 37, 38, 40</td>
<td></td>
</tr>
<tr>
<td>i_end_y</td>
<td>27, 28, 30, 32, 35, 38</td>
<td></td>
</tr>
<tr>
<td>i_start_x</td>
<td>27, 28, 30, 31, 32, 34, 35, 37, 38, 40</td>
<td></td>
</tr>
<tr>
<td>i_start_y</td>
<td>27, 28, 30, 31, 32, 34, 35, 37, 38, 40</td>
<td></td>
</tr>
<tr>
<td>ibottom</td>
<td>45, 46, 49</td>
<td></td>
</tr>
<tr>
<td>ichar</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>there</td>
<td>45, 46</td>
<td></td>
</tr>
<tr>
<td>ileft</td>
<td>45, 46</td>
<td>48</td>
</tr>
<tr>
<td>incr</td>
<td>29, 34, 37, 40</td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>output_code</td>
<td>21, 22, 23, 24</td>
<td></td>
</tr>
<tr>
<td>output_file</td>
<td>10, 17, 18, 22, 23, 24</td>
<td></td>
</tr>
<tr>
<td>output_line</td>
<td>16, 18</td>
<td></td>
</tr>
<tr>
<td>irtight</td>
<td>45, 46</td>
<td>47</td>
</tr>
<tr>
<td>istop</td>
<td>45, 46</td>
<td>50</td>
</tr>
<tr>
<td>iz</td>
<td>12, 14, 30, 31, 34, 37, 40, 43, 44, 46, 47, 48, 49, 50, 51</td>
<td></td>
</tr>
<tr>
<td>iz1</td>
<td>29, 31, 34, 37, 40</td>
<td></td>
</tr>
<tr>
<td>iz2</td>
<td>29, 31, 34, 37, 40</td>
<td></td>
</tr>
<tr>
<td>iy</td>
<td>12, 14, 30, 31, 34, 37, 40, 43, 44, 46, 47, 48, 49, 50, 51</td>
<td></td>
</tr>
<tr>
<td>iy1</td>
<td>29, 30</td>
<td></td>
</tr>
<tr>
<td>iy2</td>
<td>29, 30</td>
<td></td>
</tr>
<tr>
<td>jj</td>
<td>29, 34, 37, 40</td>
<td></td>
</tr>
<tr>
<td>layer</td>
<td>19, 20, 22, 28, 30, 31, 37, 40, 43</td>
<td></td>
</tr>
<tr>
<td>MAX_CELL</td>
<td>6, 43</td>
<td></td>
</tr>
<tr>
<td>MAX_CONDITION</td>
<td>6, 9, 11</td>
<td></td>
</tr>
<tr>
<td>MAX_GRID</td>
<td>6, 12, 13, 14, 34, 37, 40, 43, 44</td>
<td></td>
</tr>
<tr>
<td>MAX_SAMPLE</td>
<td>6, 25</td>
<td></td>
</tr>
<tr>
<td>MAX_TRANSPUTER</td>
<td>6, 13, 14</td>
<td></td>
</tr>
<tr>
<td>more_conditions</td>
<td>9, 11</td>
<td></td>
</tr>
<tr>
<td>more_data</td>
<td>16, 17, 18</td>
<td></td>
</tr>
<tr>
<td>n_of_cells</td>
<td>42, 43, 44</td>
<td></td>
</tr>
<tr>
<td>n_of_points</td>
<td>16, 17, 26, 53</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>open</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>open_unit</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>point_index</td>
<td>52, 53</td>
<td></td>
</tr>
<tr>
<td>prompt</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>sample_output</td>
<td>41, 53</td>
<td></td>
</tr>
<tr>
<td>sample_x</td>
<td>25, 26</td>
<td>53</td>
</tr>
<tr>
<td>sample_y</td>
<td>25, 26</td>
<td>53</td>
</tr>
<tr>
<td>set_char_to_int</td>
<td>12, 30, 31</td>
<td></td>
</tr>
<tr>
<td>start_x</td>
<td>19, 23, 26, 27, 28, 32, 35, 38</td>
<td></td>
</tr>
<tr>
<td>start_y</td>
<td>19, 23, 26, 27, 28, 32, 35, 38</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>9, 17</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>vertical_boundary</td>
<td>12, 14, 20, 27, 28, 30, 43, 47, 48</td>
<td></td>
</tr>
<tr>
<td>while</td>
<td>11, 17, 22, 23, 24</td>
<td></td>
</tr>
<tr>
<td>worker_index</td>
<td>13, 14, 43</td>
<td></td>
</tr>
<tr>
<td>worker_number</td>
<td>13, 14, 20, 35, 36, 37, 38, 39, 43</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>x_cellsize</td>
<td>9, 10, 26, 28, 32, 35, 38, 46</td>
<td></td>
</tr>
<tr>
<td>x_start</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>xcorner</td>
<td>45, 46</td>
<td></td>
</tr>
<tr>
<td>y_cellsize</td>
<td>9, 10, 26, 28, 32, 35, 38, 46</td>
<td></td>
</tr>
<tr>
<td>y_start</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>ycorner</td>
<td>45, 46</td>
<td></td>
</tr>
</tbody>
</table>
(Do horizontal boundary vector 31) Used in section 28.
(Do horizontal condition vector 40) Used in section 38.
(Do horizontal geometry vector 34) Used in section 32.
(Do horizontal worker vector 37) Used in section 35.
(Do vertical boundary vector 30) Used in section 28.
(Do vertical condition vector 39) Used in section 38.
(Do vertical geometry vector 33) Used in section 32.
(Do vertical worker vector 36) Used in section 35.
(Get active condition layers 11) Used in section 10.
(Get end point 24) Used in section 20.
(Get input values and open input files 10) Used in section 8.
(Get layer 22) Used in section 20.
(Get start point 23) Used in sections 20 and 26.
(Get the bottom neighbour index 49) Used in section 46.
(Get the condition value 51) Used in section 46.
(Get the left neighbour index 48) Used in section 46.
(Get the right neighbour index 47) Used in section 46.
(Get the top neighbour index 50) Used in section 46.
(Open output files 41) Used in section 8.
(Process boundary entry 28) Used in section 20.
(Process condition entry 38) Used in section 20.
(Process geometry entry 32) Used in section 20.
(Process line input 20) Used in section 18.
(Process point input 26) Used in section 18.
(Process the input file 17) Used in section 8.
(Process the next input file item 18) Used in section 17.
(Process worker entry 35) Used in section 20.
(Set the cell indices 43) Used in section 8.
(Variables of the program 9, 12, 13, 16, 19, 21, 25, 27, 29, 42, 45, 52) Used in section 8.
(Write connectivity file 46) Used in section 44.
(Write sample file 53) Used in section 44.
(Write the output 44) Used in section 8.
(Zero the arrays 14) Used in section 8.

COMMAND LINE: "C:\BIN\FWEAVE.EXE griddler".
WEB FILE: "griddler.web".
CHANGE FILE: (none).
GLOBAL LANGUAGE: FORTRAN.
1. INTRODUCTION. The *explode* program consists of two tasks, the root task and the worker task. The root task runs on a single transputer, the one connected to the PC and is responsible for all program input/output as well as for the synchronisation of the processing carried out by the worker task, one copy of which runs on each remaining transputer in the computational ring.

The transputers in the present version of this program are arranged serially in a circular topology with each worker transputer communicating with the preceding and following processors. In the case of the first worker the upstream connection is to the root processor and for the last worker the downstream connection is also to the root.

Message flow is unidirectional from transputer to transputer thus, in general, most packets will be processed in some manner by all processors in the ring. This arrangement, involving as it does all processors in each message, adds some overhead to the operation of the simulation; particularly when the relatively slow speed (10MBits/sec) of the serial links between processors is considered. However, for a small number of worker processors, this overhead is tolerable when considered against the per packet cost of a more complex routing algorithm. An alternative topology for larger systems of processors in which message transport is more nearly optimal will be discussed in a later section.
2. MAIN. The top level structure of the root task is essentially simple. First the data structures required for the simulation are initialised by the process of sending data packets out to the worker tasks. Then the root task synchronizes the computational stages for each time step, triggering data report generation as required.

```fortran
define MAX_SAMPLE 100
define prompt write(*, '(lx,ua$)')
define get_string read(*, '(a)')
format prompt write
format get_string read

program explroot,
  implicit none
  include 'chan.inc'
  (Variables of the program 4)
  (Perform initialisation 5)
  do while(time < finish_time)
    (Perform the main loop 27)
      enddo
  end
```

§2 — Explode Root Task —
3. **INITIALISATION.** The initialisation phase of the root task involves the setting up of the communication channels to the worker processes, the determination of the number of workers in the ring and the input and transmission of the initial data. These data consist of four elements, the geometry of the model, the condition sets, the constant data and the sample collection points.

4. The following variables are used to track the progress of the simulation over time. The variable `time` represents the current value of elapsed time from the start of the simulation, `finish_time` is the time at which the simulation is to be terminated, `report_time` is the time for which the next set of sample data are to be collected and `report_time_step` is the interval between sample data collections.

\[
\{\text{Variables of the program 4}\} \equiv 
\begin{align*}
\text{real} & \quad \text{time, finish\_time, report\_time, report\_time\_step} \\
\end{align*}
\]

See also sections 6, 9, 11, 13, 15, 18, 21, 23, 26, 28, 32, 34, and 41.

This code is used in section 2.

5. The initialisation code is fairly self explanatory. Once the channels have been established and the data broadcast to the workers the simulation is ready to run. All that remains is to initialise the value of `time` and set up `report_time` to the value for which the first results will be sampled.

\[
\{\text{Perform initialisation 5}\} \equiv 
\begin{align*}
\{\text{Declare packet types 7}\} \\
\{\text{Declare packet sizes 8}\} \\
\{\text{Set up the channels 10}\} \\
\{\text{Determine the number of workers 12}\} \\
\{\text{Open the input data files 14}\} \\
\{\text{Get the sample data 16}\} \\
\{\text{Broadcast the initial data 17}\} \\
\text{time} = 0.0 \\
\text{report\_time} = \text{report\_time\_step}
\end{align*}
\]

This code is used in section 2.

6. Communication between the processes is via a packet protocol. Each type of data element which traverses the processor network has its own particular size, `packet\_size`, and type, `packet\_type`.

\[
\{\text{Variables of the program 4}\}^+ \equiv 
\begin{align*}
\text{integer} & \quad \text{packet\_size, packet\_type}
\end{align*}
\]
7. The values of packet_type for each class of packet are defined as follows.

(Delclare packet types 7) \equiv

- define worker_count_type 1
- define geometry_data_type 2
- define condition_data_type 3
- define constant_data_type 4
- define synch_type 5
- define timestep_type 6
- define timeset_type 7
- define step_complete_type 8
- define report_type 9
- define data_request_type 10
- define debug_type 11

This code is used in section 5.

8. The values of packet_size are now defined. It should be noted that the sizes of the step completion and data request packets are not defined in this program as the root process does not originate packets of either of these types.

(Delclare packet sizes 8) \equiv

- define count_packet_size 1
- define geometry_packet_size 8
- define condition_packet_size 10
- define constant_packet_size 73
- define synch_packet_size 0
- define timestep_packet_size 2
- define timeset_packet_size 1
- define report_packet_size 13
- define debug_packet_size 18

This code is used in section 5.
§9 — Explode Root Task —

9. The variables `to_ring` and `from_ring` will contain the channel addresses of the outgoing and incoming connections to the ring of worker tasks respectively.

\[
\text{(Variables of the program 4) } \equiv \\
\text{integer } to\_ring, from\_ring
\]

10. These channel addresses are now initialised. The functions \texttt{f77\_chan\_out\_port()} and \texttt{f77\_chan\_in\_port()} are part of the standard parallel processing support libraries provided with the 3L Parallel Fortran compiler. The functions return the \textit{binding} of the specified port, the address of the channel word with which the port is associated.

\[
\text{(Set up the channels 10) } \equiv \\
\text{to\_ring } = \texttt{f77\_chan\_out\_port(2)} \\
\text{from\_ring } = \texttt{f77\_chan\_in\_port(3)}
\]

This code is used in section 5.

11. In order to track the progress of the simulation and to ensure that all worker processes are maintained in a state of synchronisation it is necessary for the root task to determine the value of the variable `n\_of\_workers`, the number of worker tasks.

\[
\text{(Variables of the program 4) } \equiv \\
\text{integer } n\_of\_workers
\]

12. This determination involves sending out a packet of type `worker\_count\_type` containing a zero and extracting the total number of workers from the packet after it has traversed the ring.

\[
\text{(Determine the number of workers 12) } \equiv \\
\text{packet(1) } = 0 \\
\text{packet\_size } = \text{count\_packet\_size} \\
\text{packet\_type } = \text{worker\_count\_type} \\
\text{(Send packet to ring 42)} \\
\text{(Get packet from ring 43)} \\
\text{n\_of\_workers } = \text{packet(1)}
\]

This code is used in section 5.
§13 — Explode Root Task — INITIALISATION 108

13. The initial non-blank sequence of characters in the variable base.name will hold the base file name. This variable with an appropriate extension will give the names of each of the files used by the program.

It is convenient to define a number of macros to simplify the process of generating file names from the variable base.name.

define geometry_file 10
define condition_file 11
define sample_file 12
define constants_file 13
define sample_data_out 14
define base_len index(base.name, 'u') - 1
define base_file base.name(1 : base_len)
define fullname(a) base_file || a
define open_unit(a, b, c) open(unit = a, file = full.name(b), status = c)
define open_unit.binary(a, b, c) open(unit = a, file = full.name(b), status = c, form = 'unformatted')

format open_unit open
format open_unit.binary open

(Variables of the program 4) +≡

character*40 base.name

14. The input and output file streams may now be opened. There are four input streams containing the geometry, condition set, sample data points and model constants. The single output file contains all of the sample data from the simulation.

(Open the input data files 14) ≡

prompt(‘Enter input file base name: ’)
get_string(base.name)
open_unit(geometry_file, '.geo', 'old')
open_unit(condition_file, '.con', 'old')
open_unit(sample_file, '.smp', 'old')
open_unit(constants_file, '.cnst', 'old')
open_unit_binary(sample_data_out, '.dat', 'unknown')

This code is used in section 5.
15. The array `sample` will contain the indices of the points or point ranges for which samples are to be taken. The variable `n.of.samples` contains the number of entries in `sample`. The `sample.count` variable contains the actual number of samples to be taken at each appropriate time step.

```plaintext
Variables of the program 4) +\equiv

integer sample(MAX.SAMPLE)
integer n.of.samples, sample.index, sample.count
```

16. The input file `sample.file` contains `n.of.samples + 1` integer data values. The first entry in the file is `n.of.samples`, the remaining entries indicate the indices of the sampling points. If a `sample.file` entry is a positive integer the value read in is the index of a sample point. If a negative entry is encountered, the absolute value of this entry together with the subsequent value in the file define a range of points to be sampled.

As an example consider a `sample.file` containing the following sequence of entries 4, 1134, -2033, 2052, 3072. In this case `n.of.samples` will take the value 4 and the points sampled will be 1134, 2033 through to 2052 inclusive and 3072. Thus `sample.count` will take the value 22.

```plaintext
Get the sample data 16) \equiv
read(sample.file,*) n.of.samples
sample.index = 0
sample.count = 0
do while (sample.index < n.of.samples)
    sample.index++
    read(sample.file,*) sample(sample.index)
if(sample(sample.index) > 0) then /* SINGLE SAMPLE POINT */
    sample.count++
else /* SAMPLE POINT RANGE */
    sample.index++
    read(sample.file,*) sample(sample.index)
    sample.count += sample(sample.index) + sample(sample.index - 1) + 1
endif
endo
dowrite(sample.data_out) sample.count
```

This code is used in section 5.
17. It is now necessary to send all initialisation data to the worker processes. This involves transmitting
the data from the connectivity array file `geometry.file`, the condition set file `condition_file` and the model
dependent constants from `constants.file`.

(Broadcast the initial data 17) ≡
  (Send the geometry data 19)
  (Send the condition data 22)
  (Send the constants 24)

This code is used in section 5.

18. The connectivity file `geometry.file` contains \texttt{n_of_cells} records, one for each cell in the model. Each cell
record contains 14 variables.

The cell index, \texttt{ihere}, is an integer of the form \(1000t + l\) where \(t\) is the index of the transputer on which
this particular cell is to be processed and \(l\) is the local cell index on transputer \(t\). Thus for example an \texttt{ihere}
value of 4152 indicates that this is cell 152 on transputer 4.

The three values \texttt{zcorner}, \texttt{ycorner} and \texttt{zcorner} give the coordinates of the cell's corner point, \texttt{x.mid_point},
\texttt{y.mid_point} and \texttt{z.mid_point} provide the coordinates of the cell's centre and \texttt{x.cell_size}, \texttt{y.cell_size} and
\texttt{z.cell_size} give the dimensions of the cell.

The next six entries \texttt{east}, \texttt{west}, \texttt{south}, \texttt{north}, \texttt{top} and \texttt{bottom} provide information about the six orthogonal
neighbours of the cell. These integer values are interpreted in the following way. If another cell abuts the
current cell the value of the corresponding neighbour entry is that cell's \texttt{ihere} value. A negative neighbour
entry implies that a boundary lies in that direction from the current cell.

The final element in the record is \texttt{condition_code}, an index into the set of initial conditions which indicates
what the conditions are for this cell at time 0.

(Variables of the program 4) \(\equiv\)

  integer \texttt{ihere}
  real \texttt{zcorner}, \texttt{ycorner}, \texttt{zcorner}, \texttt{x.mid_point}, \texttt{y.mid_point}, \texttt{z.mid_point}, \texttt{x.cell_size}, \texttt{y.cell_size}, \texttt{z.cell_size}
  integer \texttt{east}, \texttt{west}, \texttt{south}, \texttt{north}, \texttt{top}, \texttt{bottom}, \texttt{condition_code}
  integer \texttt{n_of_cells}, \texttt{cell_index}
19. The initialisation of the geometry data for the worker processes is achieved by sending out $n_{\text{of}\_\text{cells}}$ packets to the worker ring, each packet containing the geometry information for a single cell.

It should be noted that each packet proceeds around the ring only as far as its target worker. Thus no packets will be returned to the root task in this process.

(Send the geometry data 19) \equiv
\begin{verbatim}
read(geometry_file,+) n_of_cells
    do cell_index = 1, n_of_cells
        read(geometry_file,+) xcorner, ycorner, zcorner, x_cellsize, y_cellsize, z_cellsize, east, west, north, south, top, bottom, condition_code
        (Pack the geometry data 20)
        packet_size = geometry_packet_size
        packet_type = geometry_data_type
        (Send packet to ring 42)
    enddo
\end{verbatim}

This code is used in section 17.

20. Not all of the data in geometry_file is relevant to the simulation. Only the 8 significant data elements are sent to the worker. The absence of $x_{\text{cellsize}}$, $y_{\text{cellsize}}$, and $z_{\text{cellsize}}$ from the packet is due to the fact that the current version of the simulation assumes constant cell size for all cells. Thus these variables are sent to the workers once only as part of the model dependent constant data. The modification of the code to accommodate variable cell dimensions would be a relatively straightforward process.

(Pack the geometry data 20) \equiv
\begin{verbatim}
packet(1) = ihere
packet(2) = east
packet(3) = west
packet(4) = north
packet(5) = south
packet(6) = top
packet(7) = bottom
packet(8) = condition_code
\end{verbatim}

This code is used in section 19.
21. Associated with each cell was a pointer to the initial condition set for that cell, \textit{condition\_code}. The root task must now send out the \texttt{n\_of\_conditions} sets of initial conditions to each worker in the ring where the worker tasks will allocate appropriate values from these sets to each cell.

\begin{verbatim}
(integer of the program 4) +==
   integer n\_of\_conditions, condition\_index, condition\_subscript
\end{verbatim}

22. The following code reads in and transmits the initial conditions for the simulation to the worker processes. Note that in this case, unlike that of the geometry initialisation, each packet completes a full circuit of the ring as a given condition set may be applicable to cells on any or all workers.

\begin{verbatim}
(Send the condition data 22) ==
   packet\_type = condition\_data\_type
   packet\_size = condition\_packet\_size
   read(condition\_file,*) n\_of\_conditions
   packet(1) = n\_of\_conditions
   (Send packet to ring 42)
   (Get packet from ring 43)
   do condition\_index = 0, n\_of\_conditions - 1
      read(condition\_file,*) (packet(condition\_subscript), condition\_subscript = 1, condition\_packet\_size)
      (Send packet to ring 42)
      (Get packet from ring 43)
   enddo
\end{verbatim}

This code is used in section 17.

23. The last aspect of the model initialisation phase is to transmit the model dependent constants to the worker tasks.

\begin{verbatim}
(Variables of the program 4) +==
   integer constant\_index
\end{verbatim}
The following code transmits the model constants to the workers. As noted above the variables \( x_{\text{cell size}}, y_{\text{cell size}} \) and \( z_{\text{cell size}} \) are sent as part of the model constants as a fixed cell size is used by the simulation. Once again, the model constant packet makes a complete circuit of the processor ring as all processors need a copy of these data.

\[
\text{(Send the constants 24) } \equiv \\
\quad \text{packet}(1) = x_{\text{cell size}} \\
\quad \text{packet}(2) = y_{\text{cell size}} \\
\quad \text{packet}(3) = z_{\text{cell size}} \\
\text{read}(\text{constants file},*) \text{ finish time, max timestep, min timestep, report time step} \\
\text{read}(\text{constants file},*) (\text{packet(constant index)}, \text{constant index} = 4, 73) \\
\quad \text{packet size} = \text{constant packet size} \\
\quad \text{packet type} = \text{constant data type} \\
\text{(Send packet to ring 42)} \\
\text{(Get packet from ring 43)} \\
\quad \text{n. of stages} = \text{packet}(73)
\]

This code is used in section 17.
25. **THE MAIN LOOP.** This module controls the main program flow. It serves to synchronise the operation of each of the worker tasks, ensuring that each phase of the calculation is completed on all workers before proceeding to the next.

26. The variables `stage` and `n_of_stages` are used to control the Runge-Kutta solution for the source terms.

```plaintext
(integer stage, n_of_stages)
```
27. Each time step involves the following series of operations.

1. Setting up the new time step value.

2. Calculating the artificial viscosity term.

3. Calculating the viscid flux.

4. The Runge-Kutta solution involving
   a. Calculation of the source terms.
   b. Calculation of the new $W$ work arrays.
   c. Updating of the $W$ arrays.

5. Reporting results. This occurs only when the current elapsed time is equal to the next report time value. In this case the value of $report\_time$ is incremented by $report\_time\_step$ ready for the next report.

(Perform the main loop 27) ≡

( Set up time step 29 )

(Synchronise calculation 35) /* FOR ARTIFICIAL VISCOSITY PHASE I */
(Synchronise calculation 35) /* FOR ARTIFICIAL VISCOSITY PHASE II */
(Synchronise calculation 35) /* FOR VISCID FLUX */
do stage = 1, n_of_stages /* 1, 2, or 4 */
   (Synchronise calculation 35) /* FOR SOURCE TERMS */
   (Synchronise calculation 35) /* CALCULATE NEW W VALUES */
   (Synchronise calculation 35) /* FOR ARRAY UPDATES */
endo

if ($time \geq report\_time$) then
   (Get data from workers and store it to file 36)
   $report\_time +:= report\_time\_step$
endif

This code is used in section 2.
28. The variable *timestep* holds the value of the next time increment in the simulation. The variables *max\_timestep* and *min\_timestep* are used to serve as a set of bounds on the *timestep* value. In the case of an out of bounds *timestep* value the integer variable *timecell* will indicate the index of the model cell at which the out of bounds condition occurs.

(Variables of the program 4) \(\equiv\)

real \(timestep, \ max\_timestep, \ min\_timestep\)

integer \(timecell\)
29. In this phase the root task sends out a packet of type timestep_type. Each worker in the ring will receive this packet in turn and modify its contents, placing the minimum local value of timestep and its corresponding value of timecell into the packet if the local timestep value is less than the current packet contents. Thus, by the time it has circumnavigated the ring, the packet contains the global minimum timestep value.

If the new value of timestep is within bounds the root task proceeds to broadcast this value to the workers via a packet of type timeset_type. Otherwise the program terminates in error.

Finally time the value of total elapsed time is incremented by the timestep value.

(Setup time step 29) ≡

\[
\begin{align*}
\text{packet}._{size} &= \text{timestep}._{packet}._{size} \\
\text{packet}._{type} &= \text{timestep}._{type} \\
\text{packet}(_{1}) &= \text{max}._{timestep} \\
\text{packet}(_{2}) &= -1 \\
(\text{Send packet to ring 42}) \\
(\text{Get packet from ring 43}) \\
\text{timestep} &= \text{packet}(_{1}) \\
\text{timecell} &= \text{packet}(_{2}) \\
\text{if} (\text{timestep} \geq \text{max}._{timestep}) \text{ then} \\
\quad (\text{Do max timestep error 30}) \\
\text{endif} \\
\text{if} (\text{timestep} \leq \text{min}._{timestep}) \text{ then} \\
\quad (\text{Do min timestep error 31}) \\
\text{endif} \\
\text{packet}(_{1}) &= \text{timestep} \\
\text{packet}._{size} &= \text{timeset}._{packet}._{size} \\
\text{packet}._{type} &= \text{timeset}._{type} \\
(\text{Send packet to ring 42}) \\
(\text{Get packet from ring 43}) \\
\text{time} &= \text{time} + \text{timestep}
\end{align*}
\]

This code is used in section 27.
§30 — Explode Root Task — THE MAIN LOOP 118

30. If the minimum value of \( \textit{timestep} \) is greater than or equal to \( \textit{max}_t \textit{imestep} \) the following code prints an appropriate error report and retrieves and writes the values for the cell at which the error occurred.

\[
\text{(Do max timestep error 30) } \equiv \\
\text{write(\(\ast\),\(\ast\)) '***uERRORu—uTimestepu>u\textit{max}_u***'} \\
\text{write(\(\ast\),\(\ast\)) '***uCELLu=U', timecell} \\
\text{write(\(\ast\),\(\ast\)) 'Timestepu=U', timestep} \\
\text{write(\(\ast\),\(\ast\)) 'Max_uuu_uuu_uuu_uuu_uuu_uuu=U', max\textit{_timestep}} \\
\text{packet(1) = timecell} \\
\text{packet\_size = report\_packet\_size} \\
\text{packet\_type = report\_type} \\
\text{\{Send packet to ring 42\}} \\
\text{\{Get packet from ring 43\}} \\
\text{\{Write sample data to file 39\}} \\
\text{\{Do debug stuff 33\}} \\
\text{stop} \\
\]

This code is used in section 29.

31. The following code similarly reports the data for the case where \( \textit{timestep} \) is less than or equal to \( \textit{min}_t \textit{imestep} \). In practice neither of these modules should be invoked as they indicate an instability in the solution space.

\[
\text{(Do min timestep error 31) } \equiv \\
\text{write(\(\ast\),\(\ast\)) '***uERRORu—uTimestepu<\textit{min}'} \\
\text{write(\(\ast\),\(\ast\)) '***uCELLu=U', timecell} \\
\text{write(\(\ast\),\(\ast\)) 'Timestepu=U', timestep} \\
\text{write(\(\ast\),\(\ast\)) 'Min_uuu_uuu_uuu_uuu_uuu_uuu=U', min\textit{_timestep}} \\
\text{packet(1) = timecell} \\
\text{packet\_size = report\_packet\_size} \\
\text{packet\_type = report\_type} \\
\text{\{Send packet to ring 42\}} \\
\text{\{Get packet from ring 43\}} \\
\text{\{Write sample data to file 39\}} \\
\text{\{Do debug stuff 33\}} \\
\text{stop} \\
\]

This code is used in section 29.
32. The variables `worker.index` and `debug.index` are used to index into the loops controlling the debug code which follows.

\[
\text{(Variables of the program 4) } +\equiv \\
\begin{align*}
\text{integer } & \text{worker.index, debug.index} \\
\end{align*}
\]

33. In the event of a `timestep` error the values for all variables at the cell representing the local minimum for each worker are collected and written.

The parameter `MAX_CELL` controls the maximum number of cells which may be processed on a single worker transputer.

\[
\text{define } \text{MAX\_CELL } 1000
\]

\[
\text{(Do debug stuff 33) } \equiv \\
\begin{align*}
\text{do } & \text{worker.index } = 1, \text{n.of.workers} \\
\text{packet.size } = & \text{debug.packet.size} \\
\text{packet.type } = & \text{debug.type} \\
\text{packet(1) } = & \text{worker.index}\times \text{MAX\_CELL} \\
\text{(Send packet to ring 42)} \\
\text{(Get packet from ring 43)} \\
\text{write(sample.data.out) } & \text{packet(debug.index), debug.index } = 1,18 \\
\text{enddo}
\end{align*}
\]

This code is used in sections 30 and 31.

34. The variable `number.complete` is used to track the completion status of the worker tasks.

\[
\text{(Variables of the program 4) } +\equiv \\
\begin{align*}
\text{integer } & \text{number.complete} \\
\end{align*}
\]
Each phase of the calculation is handled by this module. First a packet of type `synch.type` is sent around the processor ring. On receipt of this packet each worker task will commence calculation of the next phase. On completion of this phase the worker will transmit a packet of type `step.complete.type`. The root task monitors the traffic on the ring and counts the number of `step.complete.type` packets received in the variable `number.complete`. When this count is equal to `n.of.workers` all workers have completed the current calculation step and control can return to the main loop for the next phase of the process.

In the process of calculation each worker task will require the values associated with cells on other workers. The worker will send out a packet of type `data.request.type` which will be filled by the appropriate worker for each such data set needed. As these packets pass around the ring they are received and passed on by the worker task.

(Synchronise calculation 35) ≡

```
packet.size = synch.packet.size
packet.type = synch.type
(Send packet to ring 42)
(Get packet from ring 43)
number.complete = 0
do while (number.complete < n.of.workers)
  (Get packet from ring 43)
  if (packet.type ≡ step.complete.type) then
    number.complete ++
  elseif (packet.type ≡ data.request.type) then
    (Send packet to ring 42)
  else
    write(*,*) 'Packet uerror, utype=u', packet.type
  endif
enddo
```

This code is used in section 27.
36. Whenever the current value of time equals or exceeds report_time the following code is invoked to obtain report data from each of the sample points. As already explained the sample array contains the indices for both single sample points and arrays of sample points.

\[
\text{(Get data from workers and store it to file 36)} \equiv \\
\text{write} (\text{sample\_data\_out}) \text{ report\_time, time} \\
\text{sample\_index} = 0 \\
\text{do while} (\text{sample\_index} < \text{n\_of\_samples}) \\
\text{sample\_index}++ \\
\text{if} (\text{sample}(\text{sample\_index}) > 0) \text{ then} \\
\text{(Get single sample point data 37)} \\
\text{else} \\
\text{(Get multiple sample point data 38)} \\
\text{endif} \\
\text{endo}\text{do}
\]

This code is used in section 27.

37. If the current sample value is positive data for a single cell are required. In this case a packet of type report\_type is sent around the worker ring. The appropriate worker will fill the packet with the required data and it will continue round the ring back to the root task where the data will be written to file.

\[
\text{(Get single sample point data 37)} \equiv \\
\text{packet}(1) = \text{sample}(\text{sample\_index}) \\
\text{packet\_size} = \text{report\_packet\_size} \\
\text{packet\_type} = \text{report\_type} \\
\text{(Send packet to ring 42)} \\
\text{(Get packet from ring 43)} \\
\text{(Write sample data to file 39)}
\]

This code is used in section 36.
§38 — Explode Root Task —

38. If the current sample value is negative it and its successor define a range of samples to be taken. In this case a packet of type report_type is circulated for each cell in this range.

\[(\text{Get multiple sample point data 38}) \equiv\]

\[
\begin{align*}
\text{sample\_index} & \text{++} \\
\text{do } \text{sample\_count} & = \text{sample\_index} - 1, \text{sample\_index} \\
\text{packet\_size} & = \text{report\_packet\_size} \\
\text{packet\_type} & = \text{report\_type} \\
(\text{Send packet to ring 42}) \\
(\text{Get packet from ring 43}) \\
(\text{Write sample data to file 39}) \\
\end{align*}
\]

enddo

This code is used in section 36.

39. At this point packet contains the values of the \(W\) array entries for the current time step and sample point. These values are written to the file sample.data.out.

\[(\text{Write sample data to file 39}) \equiv\]

\[
\text{write(sample.data.out)} (\text{packet(condition\_subscript)}, \text{condition\_subscript} = 1, \text{packet\_size})
\]

This code is used in sections 30, 31, 37, and 38.
40. PACKET HANDLERS. The following modules control the transmission and receipt of packets around the ring of workers. Each packet consists of two parts, the packet header and the packet proper.

41. The two components of the packet are declared. The array header which will contain the packet_size and packet_type and the array packet which will contain the packet data. The size of packet represents an upper bound as only packet_size items will be sent or received for a given packet_type.

\[(\text{Variables of the program} 4) \equiv\]
\[
\begin{align*}
\text{integer} & \quad \text{header}(2) \\
\text{real} & \quad \text{packet}(100)
\end{align*}
\]

42. The process of transmitting a packet to the ring involves two steps.

1. Preparing and sending the header.

2. Sending the packet.

The fixed size header is sent first to inform the worker tasks of the size and type of the packet which follows. This protocol allows for the transmission of variable sized packets.

The packet sizes in the calls to \texttt{f77-chan.out-message}, part of the standard parallel processing support libraries provided with the 3L Parallel Fortran compiler, are 4 times the size of the relevant message as the length count is in bytes.

\[(\text{Send packet to ring} 42) \equiv\]
\[
\begin{align*}
\text{header}(1) & \equiv \text{packet.size} \\
\text{header}(2) & \equiv \text{packet.type} \\
\text{call} & \quad \text{f77-chan.out-message(8, header, to.ring)} \\
\text{if} & \quad (\text{packet.size} > 0) \quad \text{then} \\
& \quad \text{call} \quad \text{f77-chan.out-message(4*packet.size, packet, to.ring)} \\
\text{endif}
\end{align*}
\]

This code is used in sections 12, 19, 22, 24, 29, 30, 31, 33, 35, 37, and 38.
43. The process by which a packet is received from the ring is similar to the transmission process. First the header is received and unpacked and then the packet of size packet.size is received. Again the message length parameter in f77.chan.in.message is in bytes.

(Get packet from ring 43) =

call f77.chan.in.message(8, header, from_ring)

packet.size = header(1)

packet.type = header(2)

if (packet.size > 0) then

call f77.chan.in.message(4*packet.size, packet, from_ring)

endif

This code is used in sections 12, 22, 24, 29, 30, 31, 33, 35, 37, and 38.
44. INDEX. The index contains references to all variables and macro definitions in the program. Underlined module numbers refer to the module in which an indexed item is first referenced.

```
base_file: 13.
base_len: 13.
base_name: 13, 14.
bottom: 18, 19, 20.
cell_index: 18, 19.
condition_code: 18, 19, 20, 21.
condition_data_type: 7, 22.
condition_file: 13, 14, 17, 22.
condition_index: 21, 22.
condition_packet_size: 8, 22.
condition_subscript: 21, 22, 39.
constant_data_type: 7, 24.
constant_index: 23, 24.
constant_packet_size: 8, 24.
constants_file: 13, 14, 17, 24.
count_packet_size: 8, 12.
data_request_type: 7, 35.
debug_index: 32, 33.
debug_packet_size: 8, 33.
debug_type: 7, 33.
east: 18, 19, 20.
explode: 1.
exproot2: 2.
file: 13.
finish_time: 2, 4, 24.
form: 13.
from_ring: 9, 10, 43.
full_name: 13.
f77.chan_in_message: 43.
f77.chan_in_port: 10.
f77.chan_out_message: 42.
f77.chan_out_port: 10.
geometry_data_type: 7, 19.
geometry_file: 13, 14, 17, 18, 19, 20.
geometry_packet_size: 8, 19.
get_string: 2.
header: 40, 41, 42, 43.
here: 18, 19, 20.
include: 2.
index: 13.
MAX_CELL: 33.
MAX_SAMPLE: 2, 15.
max_timestep: 24, 28, 29, 30.
min_timestep: 24, 28, 29, 31.
 n_of_cells: 18, 19.
 n_of_conditions: 21, 22.
 n_of_samples: 15, 16, 36.
 n_of_stages: 24, 26, 27.
 n_of_workers: 11, 12, 33, 35.
 none: 2.
north: 18, 19, 20.
number_complete: 34, 35.
open: 13.
open_unit: 13.
open_unit_binary: 13.
packet: 12, 20, 22, 24, 29, 30, 31, 33, 37, 38, 39, 40, 41, 42, 43.
packet_size: 6, 8, 12, 19, 22, 24, 29, 30, 31, 33, 35, 37, 38, 39, 41, 42, 43.
packet_type: 6, 7, 12, 19, 22, 24, 29, 30, 31, 33, 35, 37, 38, 41, 42, 43.
prompt: 2.
read: 2.
report_packet_size: 8, 30, 31, 37, 38.
report_time: 4, 5, 27, 36.
report_time_step: 4, 5, 24, 27.
report_type: 7, 30, 31, 37, 38.
sample: 15, 16, 36, 37, 38.
sample_count: 15, 16, 38.
sample_data_out: 13, 14, 16, 33, 36, 39.
sample_file: 13, 14, 16.
sample_index: 15, 16, 36, 37, 38.
south: 18, 19, 20.
stage: 26, 27.
status: 13.
step_complete_type: 7, 35.
synch_packet_size: 8, 35.
synch_type: 7, 35.
time: 2, 4, 5, 27, 29, 36.
timecell: 28, 29, 30, 31.
timeset_packet_size: 8, 29.
timeset_type: 7, 29.
timestep: 28, 29, 30, 31, 33.
timestep_packet_size: 8, 29.
timestep_type: 7, 29.
to_ring: 9, 10, 42.
top: 18, 19, 20.
unit: 13.
west: 18, 19, 20.
while: 2, 16, 35, 36.
worker_count_type: 7, 12.
```
worker_index: 32, 33.
write: 2.

x.cellsize: 18, 19, 20, 24.
x.mid_point: 18.
x.corner: 18, 19.

y.cellsize: 18, 19, 20, 24.
y.mid_point: 18.
y.corner: 18, 19.

z.cellsize: 18, 19, 20, 24.
z.mid_point: 18.
z.corner: 18, 19.
§44 — Explode Root Task —

- Broadcast the initial data 17) Used in section 5.
- Declare packet sizes 8) Used in section 5.
- Declare packet types 7) Used in section 5.
- Determine the number of workers 12) Used in section 5.
- Do debug stuff 33) Used in sections 30 and 31.
- Do max timestep error 30) Used in section 29.
- Do min timestep error 31) Used in section 29.
- Get data from workers and store it to file 36) Used in section 27.
- Get multiple sample point data 38) Used in section 36.
- Get packet from ring 43) Used in sections 12, 22, 24, 29, 30, 31, 33, 35, 37, and 38.
- Get single sample point data 37) Used in section 36.
- Get the sample data 16) Used in section 5.
- Open the input data files 14) Used in section 5.
- Pack the geometry data 20) Used in section 19.
- Perform initialisation 5) Used in section 2.
- Perform the main loop 27) Used in section 2.
- Send packet to ring 42) Used in sections 12, 19, 22, 24, 29, 30, 31, 33, 35, 37, and 38.
- Send the condition data 22) Used in section 17.
- Send the constants 24) Used in section 17.
- Set up the channels 10) Used in section 5.
- Set up time step 29) Used in section 27.
- Synchronise calculation 35) Used in section 27.
- Variables of the program 4, 6, 9, 11, 13, 15, 18, 21, 23, 26, 28, 32, 34, 41) Used in section 2.
- Write sample data to file 39) Used in sections 30, 31, 37, and 38.

COMMAND LINE: "C:\BIN\FWEAVE.EXE explroot".
WEB FILE: "explroot.web".
CHANGE FILE: (none).
GLOBAL LANGUAGE: FORTRAN.
Appendix C

— Explode Root Task—2D Version —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1 129</td>
</tr>
<tr>
<td>MAIN</td>
<td>2 130</td>
</tr>
<tr>
<td>INDEX</td>
<td>44 134</td>
</tr>
</tbody>
</table>

February 12, 1994
3:56
INTRODUCTION. The following chapter contains only the listings of those root task modules for which there are differences between the 2D and 3D code.

The explode program consists of two tasks, the root task and the worker task. The root task runs on a single transputer, the one connected to the PC and is responsible for all program input/output as well as for the synchronisation of the processing carried out by the worker task, one copy of which runs on each remaining transputer in the computational ring.

The transputers in the present version of this program are arranged serially in a circular topology with each worker transputer communicating with the preceding and following processors. In the case of the first worker the upstream connection is to the root processor and for the last worker the downstream connection is also to the root.

Message flow is unidirectional from transputer to transputer thus, in general, most packets will be processed in some manner by all processors in the ring. This arrangement, involving as it does all processors in each message, adds some overhead to the operation of the simulation; particularly when the relatively slow speed (10MBits/sec) of the serial links between processors is considered. However, for a small number of worker processors, this overhead is tolerable when considered against the per packet cost of a more complex routing algorithm. An alternative topology for larger systems of processors in which message transport is more nearly optimal will be discussed in a later section.
2. **MAIN.** The top level structure of the root task is essentially simple. First the data structures required for the simulation are initialised by the process of sending data packets out to the worker tasks. Then the root task synchronizes the computational stages for each time step, triggering data report generation as required.

```fortran
define MAX_SAMPLE 100
define prompt write(*,'(lx,ua$)')
define get_string read(*,'(a)')
format prompt write
format get_string read
program explroot_2d,
  implicit none
  include 'chan.inc'
  (Variables of the program 4)
  (Perform initialisation 5)
do while(time < finish.time)
  (Perform the main loop 27)
  enddo
end
```
The initialisation of the geometry data for the worker processes is achieved by sending out \( n_{of\_cells} \) packets to the worker ring, each packet containing the geometry information for a single cell.

It should be noted that each packet proceeds around the ring only as far as its target worker. Thus no packets will be returned to the root task in this process.

In the 2D code the variables relating to the third spatial dimension, \( z_{cell\_size} \), \( top \) and \( bottom \) are eliminated.

(Send the geometry data 19*)

```plaintext
read (geometry_file, *) n_of_cells

do cell_index = 1, n_of_cells
    read (geometry_file, *) ihere, xcorner, ycorner, x_cellsize, y_cellsize, east, west, north, south, condition_code

    (Pack the geometry data 20*)
    packet_size = geometry_packet_size
    packet_type = geometry_data_type

    (Send packet to ring 42)
enddo
```

This code is used in section 17.
20* Not all of the data in `geometry_file` is relevant to the simulation. Only the 8 significant data elements are sent to the worker. The absence of `x.cell.size`, `y.cell.size` and `z.cell.size` from the packet is due to the fact that the current version of the simulation assumes constant cell size for all cells. Thus these variables are sent to the workers once only as part of the model dependent constant data. The modification of the code to accommodate variable cell dimensions would be a relatively straightforward process.

The sixth and seventh elements of `packet` are zero filled as the variables `top` and `bottom` are not used in the 2D code.

\[
\text{(Pack the geometry data 20*}) \equiv \\
\begin{align*}
\text{packet}(1) &= \text{ihere} \\
\text{packet}(2) &= \text{east} \\
\text{packet}(3) &= \text{west} \\
\text{packet}(4) &= \text{north} \\
\text{packet}(5) &= \text{south} \\
\text{packet}(6) &= 0 \\
\text{packet}(7) &= 0 \\
\text{packet}(8) &= \text{condition\_code}
\end{align*}
\]

This code is used in section 19*.
The following code transmits the model constants to the workers. As noted above the variables \(x_{\text{cell.size}}, y_{\text{cell.size}}\) and \(z_{\text{cell.size}}\) are sent as part of the model constants as a fixed cell size is used by the simulation. Once again, the model constant packet makes a complete circuit of the processor ring as all processors need a copy of these data.

The third element of packet is zero filled as the \(z_{\text{cell.size}}\) variable is not used in the 2D code.

\[
\text{(Send the constants 24*) } \equiv \\
packet(1) = x_{\text{cell.size}} \\
packet(2) = y_{\text{cell.size}} \\
packet(3) = 0 \\
\text{read (constants.file, *) finish.time, max.timestep, min.timestep, report.time.step} \\
\text{read (constants.file, *) (packet(constant.index), constant.index = 4, 73)} \\
packet.size = constant.packet.size \\
packet.type = constant.data.type \\
\text{(Send packet to ring 42)} \\
\text{(Get packet from ring 43)} \\
n_{\text{of.stages}} = packet(73)
\]

This code is used in section 17.
44* INDEX. The index contains references to all variables and macro definitions in the program. Underlined module numbers refer to the module in which an indexed item is first referenced.

The following sections were changed by the change file: 1, 2, 19, 20, 24, 44.

```
base_file: 13.
base_len: 13.
base_name: 13, 14.
bottom: 18, 19* 20*.
cell_index: 18, 19*.
condition_code: 18, 19* 20* 21.
condition_data_type: 7, 22.
condition_file: 13, 14, 17, 22.
condition_index: 21, 22.
condition_packetsize: 8, 22.
condition_subscript: 21, 22, 39.
constant_data_type: 7, 24*.
constant_index: 23, 24*.
constant_packet_size: 8, 24*.
constant_file: 13, 14, 17, 24*.
count_packet_size: 8, 12.
data_request_type: 7, 35.
debug_index: 32, 33.
debug_packet_size: 8, 33.
debug_type: 7, 33.
east: 18, 19* 20*.
explode: 1*.
explrooL2d 2: 2*.
file: 13.
finish_time: 2* 4, 24*.
form: 13.
from_ring: 9, 10, 43.
full_name: 13.
f77ChanIn_message: 43.
f77Chan_in_port: 10.
f77ChanOut_message: 42.
f77Chan_out_port: 10.
geometry_data_type: 7, 19*.
geometry_file: 13, 14, 17, 18, 19* 20*.
geometry_packet_size: 8, 19*.
get_string: 2*.
header: 40, 41, 42, 43.
ithere: 18, 19* 20*.
includE: 2*.
index: 13.
MAX_CELL: 33.
MAX_SAMPLE: 2* 15.
max_timestep: 24* 28, 29, 30.
n_of_cells: 18, 19*.
n_of_conditions: 21, 22.
n_of_samples: 15, 16, 36.
n_of_stages: 24* 26, 27.
n_of_workers: 11, 12, 33, 35.
one: 2*.
north: 18, 19* 20*.
number_complete: 34, 35.
open: 13.
open_unit: 13.
open_unit_binary: 13.
packet: 12, 20* 22, 24* 29, 30, 31, 33, 37, 38, 39, 40, 41, 42, 43.
packet_size: 6, 8, 12, 19* 22, 24* 29, 30, 31, 33, 35, 37, 38, 39, 41, 42, 43.
packet_type: 6, 7, 12, 19* 22, 24* 29, 30, 31, 33, 35, 37, 38, 41, 42, 43.
prompt: 2*.
read: 2*.
report_packet_size: 8, 30, 31, 37, 38.
report_time: 4, 5, 27, 36.
report_time_step: 4, 5, 24* 27.
report_type: 7, 30, 31, 37, 38.
sample: 15, 16, 36, 37, 38.
sample_count: 15, 16, 38.
sample_data_out: 13, 14, 16, 33, 36, 39.
sample_file: 13, 14, 16.
sample_index: 15, 16, 36, 37, 38.
south: 18, 19* 20*.
stage: 26, 27.
status: 13.
step_complete_type: 7, 35.
synch_packet_size: 8, 35.
synch_type: 7, 35.
time: 2* 4, 5, 27, 29, 36.
timecell: 28, 29, 30, 31.
timeset_packet_size: 8, 29.
timeset_type: 7, 29.
timestep: 28, 29, 30, 31, 33.
timestep_packet_size: 8, 29.
timestep_type: 7, 29.
to_ring: 9, 10, 42.
top: 18, 19* 20*.
unit: 13.
west: 18, 19* 20*.
```
while: 2*, 16, 35, 36.
worker_count_type: 7, 12.
worker_index: 32, 33.
write: 2*

x.cell.size: 18, 19*, 20*, 24*
x.mid_point: 18.
x.corner: 18, 19*
y.cell.size: 18, 19*, 20*, 24*
y.mid_point: 18.
y.corner: 18, 19*

z.cell.size: 18, 19*, 20*, 24*
z.mid_point: 18.
z.corner: 18.
§44 — Explode Root Task—2D Version —

(Broadcast the initial data 17) Used in section 5.
(Declare packet sizes 8) Used in section 5.
(Declare packet types 7) Used in section 5.
(Determine the number of workers 12) Used in section 5.
(Do debug stuff 33) Used in sections 30 and 31.
(Do max timestep error 30) Used in section 29.
(Do min timestep error 31) Used in section 29.
(Get data from workers and store it to file 36) Used in section 27.
(Get multiple sample point data 38) Used in section 36.
(Get packet from ring 43) Used in sections 12, 22, 24*, 29, 30, 31, 33, 35, 37, and 38.
(Get single sample point data 37) Used in section 36.
(Get the sample data 16) Used in section 5.
(Open the input data files 14) Used in section 5.
(Pack the geometry data 20*) Used in section 19*.
(Perform initialisation 5) Used in section 2*.
(Perform the main loop 27) Used in section 2*.
(Send packet to ring 42) Used in sections 12, 19*, 22, 24*, 29, 30, 31, 33, 35, 37, and 38.
(Send the condition data 22) Used in section 17.
(Send the geometry data 19*) Used in section 17.
(Setup the channels 10) Used in section 5.
(Setup time step 29) Used in section 27.
(Synchronise calculation 35) Used in section 27.
(Variables of the program 4, 6, 9, 11, 13, 15, 18, 21, 23, 26, 28, 32, 34, 41) Used in section 2*.
(Write sample data to file 39) Used in sections 30, 31, 37, and 38.

COMMAND LINE: "C:\BIN\FWEAVE.EXE explroot explroot.2d".
WEB FILE: "explroot.web".
CHANGE FILE: "explroot.2d".
GLOBAL LANGUAGE: FORTRAN.
Appendix D
— Explode Worker Task —

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>MAIN</td>
<td>2</td>
</tr>
<tr>
<td>THREAD CONTROL</td>
<td>3</td>
</tr>
<tr>
<td>INITIALISATION</td>
<td>8</td>
</tr>
<tr>
<td>THE MODEL</td>
<td>21</td>
</tr>
<tr>
<td>PRESSURE</td>
<td>26</td>
</tr>
<tr>
<td>TIMESTEP</td>
<td>29</td>
</tr>
<tr>
<td>SOURCE TERMS</td>
<td>37</td>
</tr>
<tr>
<td>ARTIFICIAL VISCOSITY</td>
<td>53</td>
</tr>
<tr>
<td>VISCID FLUXES</td>
<td>70</td>
</tr>
<tr>
<td>INVISCID FLUXES</td>
<td>118</td>
</tr>
<tr>
<td>UPDATE PHASE</td>
<td>122</td>
</tr>
<tr>
<td>FILL ARRAY</td>
<td>125</td>
</tr>
<tr>
<td>BUFFER</td>
<td>137</td>
</tr>
<tr>
<td>SWITCH</td>
<td>158</td>
</tr>
<tr>
<td>PACKET HANDLING</td>
<td>159</td>
</tr>
<tr>
<td>INDEX</td>
<td>167</td>
</tr>
</tbody>
</table>

February 12, 1994
11:10
1. **INTRODUCTION.** As already noted, the explode program consists of two tasks, the root task and the worker task. This chapter details the structure of the worker task.

The worker task consists of three threads or co-routines. The main thread is responsible for the calculation of the simulation results for the model and the other two threads, the switch thread and the buffer thread, are responsible for maintaining the necessary communication between copies of the worker task on each transputer and between the worker and root tasks.

The following diagram shows the relationship between the threads in the worker task and the direction of packet flow between the threads.
2. **MAIN.** The top level structure of the worker task is shown below.

Following the various data declaration modules the program must first start the two communication threads.

Once this has been achieved it is necessary to determine the worker number for the present task. As a number of messages, both from the root and other worker tasks, must be addressed to a specific worker task this numbering is essential to the operation of the simulation.

Now that the worker task has established its identity the initialisation process proper can begin. This is achieved in two phases.

First the initial data are read in from the root task via the processor ring, these data being the connectivity and initial conditions for each cell as well as a set of global data values.

Secondly a number of model dependent constants are calculated.

The initialisation having been completed the worker task’s main thread enters an infinite loop wherein the main calculation stages are accomplished.

The three subroutine declarations consist of `fillarray`, a subroutine used by the main thread to access information from neighbouring cells and `buffer_thread` and `switch_thread`, the communications co-routines.

The parameter `MAX_CELL` specifies the maximum number of cells which may be processed on a given transputer.

```program exptwork,```

```define MAX_CELL 1000```

```implicit none```

```{ Include files 4 }```

```{ Variables of main 5 }```

```{ Common semaphore 6 }```

```{ Other common data 9 }```

```{ Start the communication threads 7 }```

```{ Determine worker id 11 }```

```{ Get initial data 12 }```

```{ Derive pseudo constants 20 }```
§2 — Explode Worker Task —

\[\text{do while}(T) \quad /* \text{Do forever} */\]
\[
\text{(Do the main loop 24)}
\]
\[\text{enddo}\]
\[\text{end}\]

(Fill subroutine 125)

(Buffer subroutine 137)

(Switch subroutine 158)
3. THREAD CONTROL. As noted above, in addition to the computational thread, the worker task consists of two additional communication threads. This section of code controls the initialisation of these threads.

4. The 3L Parallel Fortran compiler used for this program provides a number of subroutines to support the parallel operation. In order to use these routines it is necessary to define a number of constants and data types. The three include files 'chan.inc', 'thread.inc' and 'sema.inc' provide the definitions and declarations necessary to support the channel, thread and semaphore routines respectively.

\[
\text{(Include files 4) } \equiv \\
\text{ include 'chan.inc'} \\
\text{ include 'thread.inc'} \\
\text{ include 'sema.inc'}
\]

This code is used in sections 2, 125, 137, and 158.
5. The following variables are required to support the two communication threads. Three inter-thread channels \textit{buffer} to \textit{main}, \textit{buffer} to \textit{switch} and \textit{main} to \textit{switch} are declared. These channel words provide a common access area for each pair of communicating threads.

Communication via these internal channels requires that the address of the channel be known to the program. The variables \textit{from buffer}, \textit{to switch} and \textit{b.switch} will hold the addresses of \textit{buffer} to \textit{main}, \textit{main} to \textit{switch}, and \textit{buffer} to \textit{switch} respectively.

Each thread must be assigned an execution priority. In this case the communication threads will be given the same priority as the main computational thread. The variable \textit{priority} is used to hold this value.

Each thread must also be allocated a workspace, an integer array used for the storage of local variables. The arrays \textit{buffer.work} and \textit{switch.work} are used as the workspaces for the buffer and switch threads respectively.

\texttt{define WORK\_SIZE 4000}

(Variables of main $s$) $\equiv$

\begin{verbatim}
integer buffer.to.main, buffer.to.switch, main.to.switch
integer from.buffer, to.switch, b.switch
integer priority
integer buffer.work(WORK\_SIZE), switch.work(WORK\_SIZE)
external buffer.thread, switch.thread
\end{verbatim}

See also sections 10, 13, 15, 17, 19, 23, 27, 30, 32, 34, 38, 40, 42, 44, 46, 48, 50, 55, 63, 65, 67, 71, 73, 88, 90, 94, 96, 100, 104, 106, 110, 114, 116, and 122.

This code is used in section 2.

6. The variable \textit{flag} is shared by the main and buffer threads to control communications access between both of these threads and the switch thread.

(Common semaphore $c$) $\equiv$

\begin{verbatim}
integer flag(f77_sema_size)
common /semaphore/ flag
\end{verbatim}

This code is used in sections 2, 125, and 137.
Before actually starting the buffer and switch threads a few additional tasks must be performed.

First the semaphore variable `flag` must be initialised using the `f77.sema.init` subroutine. This variable is used to control access to the switch thread from each of the main and buffer threads. As both of these threads need to communicate with the switch thread `flag` serves to provide an access queueing mechanism.

Secondly the thread priority is established.

Finally the addresses of the three internal channels are determined and the channels are initialised.

These preliminary tasks having been completed the two communication threads may now be started. The `f77.thread.start` subroutine, as well as specifying the thread routine, workspace and priority, also allows the passing of the subroutine arguments to the new thread. In the case of the `buffer.thread` subroutine these consist of the addresses of the two internal channels it uses as well as the work array `W` to which it shares access with the main thread. In the case of `switch.thread` only the internal channel addresses need to be passed.

Once started each thread runs independently sharing the transputer. A thread will continue to run until it is descheduled either by the process of sending a message to another thread or by an explicit call to the `f77.thread.deschedule` subroutine. If the threads were to be run at the `f77.thread.noturg` priority instead of the default `f77.thread.urgent` a processor moderated time slicing mechanism would also be invoked to cause thread descheduling. As there is only a single processor intensive thread in the worker task this additional mechanism is not warranted in the present case.

(Start the communication threads 7) \(\equiv\)

```fortran
    call f77.sema.init(flag, 1)
    priority = f77.thread.priority()
    from.buffer = f77.chan.address(buffer.to.main)
    call f77.chan.init(from.buffer)
    b.switch = f77.chan.address(buffer.to.switch)
    call f77.chan.init(b.switch)
    to.switch = f77.chan.address(main.to.switch)
    call f77.chan.init(to.switch)
    call f77.thread.start(buffer.thread, buffer.work, WORK.SIZE, priority, 3, from.buffer, b.switch, W)
    call f77.thread.start(switch.thread, switch.work, WORK.SIZE, priority, 2, to.switch, b.switch)
```

This code is used in section 2.
8. **INITIALISATION.** The initialisation phase of the worker task involves a number of tasks. First, the identity of the processor on which this copy of the worker task must be determined, second, data sent from the root task must be stored in the appropriate data structures and finally, a number of additional model dependent constants must be derived.

9. The variable `my.worker.id` is used to store the identification number of the processor on which a given copy of the worker task is running. This value is used to determine which elements of the model are to be processed by each instance of the task.

The array `debug` is used to store debugging information for retrieval by the root task in the event of a subsequent program failure.

(Other common data 9) ≡

```plaintext
integer my.worker.id
real debug(18)
common /worker.id/  my.worker.id
common /debug.common/  debug
```

This code is used in sections 2, 125, and 137.
10. The variables relevant in the handling of communication packets between tasks are defined here. The type codes of the various packet types are also defined at this point.

```
define worker_count_type 1
define geometry_data_type 2
define condition_data_type 3
define constant_data_type 4
define synch_type 5
define timestep_type 6
define timeset_type 7
define step_complete_type 8
define report_type 9
define data_request_type 10
define debug_type 11

(integers of main 5) +¥
  integer packet_size, packet_type, header(2)
  real packet(100)
```
11. The process of determining the identity of the current worker task involves the fetching of a packet of `worker_count_type` from the upstream processor via the buffer thread and incrementing its contents by one. This modified packet is then sent on to the next processor via the switch task.

As the root task initialises the packet contents to zero, and as the workers are arranged in a simple ring, this mechanism serves the double duty of identifying each worker and providing a count of the workers to the root task.

An alternative mechanism would be to configure an additional dummy channel and use the 3L configuration processor to bind the worker number to this channel. Although this is marginally more efficient it is felt that the present method has the advantage of greater clarity. If a more complex processor topology were to be used, however, a mechanism such as this would be appropriate.

\[
\text{(Determine worker id 11)} \equiv \\
\text{(Get packet from buffer 162)} \\
\text{packet(1)++} \\
\text{my\_worker\_id = packet(1)} \\
\text{(Send packet to switch 163)}
\]

This code is used in section 2.

12. Any potential identity crises having been resolved the task of loading the initial data for the simulation may proceed.

Three data sets are required for the model.

i. The geometry data which define the extent and connectivity of the model.

ii. The condition data which define the initial conditions for each cell in the model.

iii. The constant data which are common to all cells.

\[
\text{(Get initial data 12)} \equiv \\
\text{(Get the geometry data 14)} \\
\text{(Get the condition data 16)} \\
\text{(Get constant data 18)}
\]

This code is used in section 2.
§13 — Explode Worker Task —

13. The array $W$ will ultimately hold all of the data specific to each cell, both the connectivity and condition data as well as some intermediate results.

The variable $n_{of\text{-}cells}$ will be used to accumulate the total number of cells on this instance of the worker task.

In order to minimise processor idle time it is important that the work carried out by each worker processor be approximately equal thus the assignment of cells to the individual worker tasks performed by the *gridder* program should result in $n_{of\text{-}cells}$ being roughly the same for each worker.

The variable $cell$ is used throughout this task as a pointer to the current cell.

(Variables of $main$ $s$) $+$

```plaintext
real $W(64, MAX\_CELL)$
integer $n_{of\text{-}cells}$
integer $cell$
integer $worker$
logical $finished$
```
§14 — Explode Worker Task — INITIALISATION

14. A packet of type `geometry.data_type` is sent from the root task to the ring of workers for each cell in the model. Each worker in turn will examine the packet to determine whether the cell is one which it is to operate on. If the cell does belong to the current task the connectivity data are extracted into the appropriate locations and a pointer to the condition set associated with the cell is also extracted from the packet and stored. If the cell is not destined for the current task, the packet is sent downstream to the remaining workers.

This sequence of receiving and storing or forwarding packets continues until a packet of a type other than `geometry.data_type` is received.

The macro $W.p(i)$ is defined to allow a simplified means of referring to the data associated with the current cell.

\[
\text{define } W.p(i) \equiv W(i, cell)
\]

\[
\text{(Get the geometry data 14) } \equiv \\
\text{finished } = \mathcal{F}\\
\text{do while } (\neg \text{finished})\\
\text{(Get packet from buffer 162) } \\
\text{if } (\text{packet_type } \neq \text{geometry.data_type}) \text{ then} \\
\quad (\text{Send packet to switch 163}) \\
\quad \text{finished } = T \\
\text{else} \\
\quad \text{worker } = \text{packet(1)}/\text{MAX.CELL} \\
\quad \text{if } (\text{worker } \equiv \text{my.worker_id}) \text{ then} \\
\quad \quad \text{cell } = \text{mod}(\text{int(packet(1)}), \text{MAX.CELL}) \\
\quad \quad W.p(17) = \text{packet(2) } /* \text{EAST NEIGHBOUR */} \\
\quad \quad W.p(18) = \text{packet(3) } /* \text{WEST NEIGHBOUR */} \\
\quad \quad W.p(19) = \text{packet(4) } /* \text{NORTH NEIGHBOUR */} \\
\quad \quad W.p(20) = \text{packet(5) } /* \text{SOUTH NEIGHBOUR */} \\
\quad \quad W.p(21) = \text{packet(6) } /* \text{TOP NEIGHBOUR */} \\
\quad \quad W.p(22) = \text{packet(7) } /* \text{BOTTOM NEIGHBOUR */} \\
\quad \quad W.p(21) = \text{packet(8) } /* \text{CONDITION CODE */} \\
\quad \quad n.of.cells = \text{cell} \\
\quad \text{else} \\
\quad \quad (\text{Send packet to switch 163}) \\
\quad \text{endif} 
\]
§14 — Explode Worker Task —

endif
enddo

This code is used in section 12.

15. The variable $n_{of\text{-}conditions}$ is used to hold the number of distinct sets of initial conditions to be applied to the cells in the model.

(Variables of main 5) $+\equiv$

integer $n_{of\text{-}conditions$

integer $condition\_set$
16. A total of \( n_{\text{of}} \cdot \text{conditions} + 1 \) packets of type \( \text{condition\_data\_type} \) are now received from the root task. The first of these packets is a dummy containing \( n_{\text{of}} \cdot \text{conditions} \). The remaining packets contain the initial values of the 10 variables \( \rho \) through \( \rho P \) for each condition code. The array of cells on each worker is scanned and each cell with a matching condition code is initialised.

\[
\text{(Get the condition data 16)} \equiv \\
\text{n\_of\_conditions} = \text{packet(1)} \\
\text{do condition\_set} = 0, n_{\text{of}} \cdot \text{conditions} - 1 \\
\text{(Get packet from buffer 162)} \\
\text{(Send packet to switch 163)} \\
\text{do cell} = 1, n_{\text{of}} \cdot \text{cells} \\
\text{if (condition\_set} \equiv \text{W\_p(11)) then} \\
\text{W\_p(1)} = \text{packet(1)} /* \rho */ \\
\text{W\_p(2)} = \text{packet(2)} /* \rho U */ \\
\text{W\_p(3)} = \text{packet(3)} /* \rho V */ \\
\text{W\_p(4)} = \text{packet(4)} /* \rho W */ \\
\text{W\_p(5)} = \text{packet(5)} /* h */ \\
\text{W\_p(6)} = \text{packet(6)} /* \rho m_{JU} */ \\
\text{W\_p(7)} = \text{packet(7)} /* \rho f */ \\
\text{W\_p(8)} = \text{packet(8)} /* \rho g */ \\
\text{W\_p(9)} = \text{packet(9)} /* \rho k */ \\
\text{W\_p(10)} = \text{packet(10)} /* \rho P */ \\
\text{W\_p(11)} = 0.0 /* P */ \\
\text{W\_p(12)} = 0.0 /* \text{MEAN MOL\_WT} */ \\
\text{W\_p(13)} = 0.0 /* m_{O2} */ \\
\text{W\_p(14)} = 0.0 /* r_x */ \\
\text{W\_p(15)} = 0.0 /* r_y */ \\
\text{W\_p(16)} = 0.0 /* r_z */ \\
\text{endif} \\
\text{enddo} \\
\text{enddo}
\]

This code is used in section 12.
17. The following variables are the model dependent constants. Their meanings are noted in the following section.

\begin{verbatim}
(Variables of main 5) +

real x_cellsize, y_cellsize, z_cellsize

real K_a, K_ach, K_alpha(4), K_alpha_h, K_b, K_cfl, K_cpa(5), K_cpb(5), K_d, K_die, K_ebu, K_eonr, K_gamma, K_gindex, K_g1, K_g2, K_hc, K_k2, K_k4, K_nc, K_nh, K_prn, K_prnt, K_ratox, K_ru, K_sigma(6:10), K_tau_1, K_tau_2, K_tref, K_1, K_110, K_145, K_2, K_maxeddy, K_mazk, K_mazeps, K_maxfuel, K_so, K_salpha, K_sbeta, K_vent

real W_at(15)

integer n_of_stages

integer index
\end{verbatim}
18. A single packet of type `constant_data_type` is received from the upstream processor via the buffer thread, its contents are extracted into the appropriate variables and the packet retransmitted to the downstream processor via the switch thread.

As all cells are of the same dimensions the \( x \), \( y \), and \( z_{\text{cell.size}} \) values are set via this packet rather than being transmitted with each of the geometry packets.

\[
\text{(Get constant data 18)} \equiv \\
\text{(Get packet from buffer 162)} \\
\text{(Send packet to switch 163)} \\
x_{\text{cell.size}} = \text{packet}(1) \\
y_{\text{cell.size}} = \text{packet}(2) \\
z_{\text{cell.size}} = \text{packet}(3) \\
K.a = \text{packet}(4) \quad /* \text{PARTIAL PRESSURE EXPONENT FOR FUEL REACTION RATE}*/ \\
K.a\text{ch} = \text{packet}(5) \quad /* \text{PRE-ACTIVATION ENERGY CONSTANT}*/ \\
K.alpha(1) = \text{packet}(6) \quad /* \text{RUNGE KUTTA PARAMETERS}*/ \\
K.alpha(2) = \text{packet}(7) \\
K.alpha(3) = \text{packet}(8) \\
K.alpha(4) = \text{packet}(9) \\
K.alpha.h = \text{packet}(10) \quad /* \text{ARTIFICIAL VISCOSITY CALCULATION CONSTANT}*/ \\
K.b = \text{packet}(11) \quad /* \text{PARTIAL PRESSURE EXPONENT FOR OXYGEN REACTION RATE}*/ \\
K.cfl = \text{packet}(12) \quad /* \text{COURANT NUMBER}*/ \\
K.cpa(1) = \text{packet}(13) \quad /* \text{SPECIFIC HEAT CONSTANTS}*/ \\
K.cpa(2) = \text{packet}(14) \\
K.cpa(3) = \text{packet}(15) \\
K.cpa(4) = \text{packet}(16) \\
K.cpa(5) = \text{packet}(17) \\
K.cpb(1) = \text{packet}(18) \quad /* \text{TEMPERATURE DEPENDENT SPECIFIC HEAT CONSTANTS}*/ \\
K.cpb(2) = \text{packet}(19) \\
K.cpb(3) = \text{packet}(20) \\
K.cpb(4) = \text{packet}(21) \\
K.cpb(5) = \text{packet}(22) \\
K.d = \text{packet}(23) \quad /* \text{EDDY VISCOSITY CONSTANT}*/ \\
K.die = \text{packet}(24) \quad /* \text{EXTINCTION CONSTANT FOR REACTION}*/ \\
K.ebu = \text{packet}(25) \quad /* \text{EDDY BREAKUP REACTION RATE CONSTANT}*/ \\
K.eonr = \text{packet}(26) \quad /* \text{EXPONENTIAL REACTION RATE}*/ 
\]
\$18 \quad \text{— Explore Worker Task —} \quad \text{INITIALISATION \hspace{1cm} 153}

\[ K_{\text{gamma}} = \text{packet}(27) \quad /\!* \text{gamma} */\]
\[ K_{\text{gindex}} = \text{packet}(28) \quad /\!* \text{TEMPERATURE EXPONENT FOR LAMINAR VISCOITY CALCULATION} */\]
\[ K_{\text{g1}} = \text{packet}(29) \]
\[ K_{\text{g2}} = \text{packet}(30) \quad /\!* \text{CONSTANTS FOR MIXTURE FRACTION VARIANCE CALCULATION} */\]
\[ K_{\text{hc}} = \text{packet}(31) \quad /\!* \text{HEAT OF COMBUSTION OF FUEL} */\]
\[ K_{\text{k2}} = \text{packet}(32) \quad /\!* \text{ARTIFICIAL VISCOITY CALCULATION CONSTANTS} */\]
\[ K_{\text{k4}} = \text{packet}(33) \]
\[ K_{\text{nc}} = \text{packet}(34) \quad /\!* \text{NUMBER OF CARBON ATOMS IN FUEL MOLECULE} */\]
\[ K_{\text{nh}} = \text{packet}(35) \quad /\!* \text{NUMBER OF HYDROGEN ATOMS IN FUEL MOLECULE} */\]
\[ K_{\text{prn}} = \text{packet}(36) \quad /\!* \text{PRANDTL NUMBER} */\]
\[ K_{\text{prnt}} = \text{packet}(37) \quad /\!* \text{TURBULENT PRANDTL NUMBER} */\]
\[ K_{\text{ratox}} = \text{packet}(38) \quad /\!* \text{RATIO OF N}_2 \text{ TO O}_2 \text{ IN AIR} */\]
\[ K_{\text{ru}} = \text{packet}(39) \quad /\!* \text{GAS CONSTANT} */\]
\[ K_{\text{sigma}(6)} = \text{packet}(40) \quad /\!* \text{SHEAR STRESS FACTORS} */\]
\[ K_{\text{sigma}(7)} = \text{packet}(41) \]
\[ K_{\text{sigma}(8)} = \text{packet}(42) \]
\[ K_{\text{sigma}(9)} = \text{packet}(43) \]
\[ K_{\text{sigma}(10)} = \text{packet}(44) \]
\[ K_{\text{tau}.1} = \text{packet}(45) \quad /\!* \text{OPEN BOUNDARY CONDITION TIME DEPENDENCY CONSTANTS} */\]
\[ K_{\text{tau}.2} = \text{packet}(46) \]
\[ K_{\text{tref}} = \text{packet}(47) \quad /\!* \text{REFERENCE TEMPERATURE} */\]
\[ K_{\cdot 1} = \text{packet}(48) \quad /\!* \text{ */} \]
\[ K_{\cdot 110} = \text{packet}(49) \quad /\!* \text{LAMINAR VISCOITY CONSTANTS} */\]
\[ K_{\cdot 145} = \text{packet}(50) \]
\[ K_{\cdot 2} = \text{packet}(51) \]
\[ K_{\cdot \text{maxeddy}} = \text{packet}(52) \quad /\!* \text{UPPER LIMIT ON EDDY VISCOITY} */\]
\[ K_{\cdot \text{maxk}} = \text{packet}(53) \quad /\!* \text{UPPER LIMIT ON TURBULENT KINETIC ENERGY} */\]
\[ K_{\cdot \text{maxeps}} = \text{packet}(54) \quad /\!* \text{UPPER LIMIT ON} \epsilon */\]
\[ K_{\cdot \text{maxfuel}} = \text{packet}(55) \quad /\!* \text{UPPER LIMIT ON FUEL REACTION RATE} */\]
\[ K_{\cdot \text{so}} = \text{packet}(56) \quad /\!* \text{FUNDAMENTAL LAMINAR BURNING VELOCITY OF FUEL} */\]
\[ K_{\cdot \text{salpha}} = \text{packet}(57) \quad /\!* \text{TEMPERATURE DEPENDENCE FACTOR FOR BURNING VELOCITY} */\]
\[ K_{\cdot \text{sbeta}} = \text{packet}(58) \quad /\!* \text{PRESSURE DEPENDENCE FACTOR FOR BURNING VELOCITY} */\]
\[ K_{\cdot \text{vent}} = \text{packet}(59) \quad /\!* \text{FAILURE PRESSURE FOR VENTS} */\]
\[ \text{do index} = 1,13 \]
\[ W_{\cdot \text{at}}(\text{index}) = \text{packet}(59 + \text{index}) \quad /\!* \text{ATMOSPHERIC CONDITIONS} */\]
enddo

n_of_stages = packet(73)  /* number of stages in Runge-Kutta solution */

This code is used in section 12.

19. The following variables are either derived from the preceding data or are invariant physically derived constants.

{ Variables of main 5 } +≡

real cell_volume, Fuel.molwt, O2.molwt, N2.molwt, CO2.molwt, H2O.molwt, Prod.molwt, x_1, x_2,

real CO2.m
20. The following section sets the value of a number of constants, a mixture of model dependent terms and physical invariants.

These include $\text{Fuel.molwt}$, the molecular weight of the $C_{n_c}H_{n_h}$ hydrocarbon fuel molecule, $O_2.molwt$ the molecular weight of the $O_2$ molecule, $N_2.molwt$ the molecular weight of the $N_2$ molecule, $CO_2.molwt$ the molecular weight of the $CO_2$ molecule, $H_2O.molwt$ the molecular weight of the $H_2O$ molecule and $\text{Prod.molwt}$ the mean molecular weight of the combustion products. The combustion model used in this program is a simple, single step process $C_{n_c}H_{n_h} + (n_c + n_h/2)O_2 \rightarrow (n_c)CO_2 + (n_h/2)H_2O$.

The variable $CO_2.m$ contains the fraction of $CO_2$ by mass in the products.

The $K.stoi$ variable contains the stoichiometric ratio.

The variables $O_2.infty$, $\text{Fuel.infty}$, $O_2.zero$ and $\text{Fuel.zero}$ contain the mass fractions for oxygen and fuel in pure air ($O_2.infty$, $\text{Fuel.infty}$) and pure fuel ($O_2.zero$, $\text{Fuel.zero}$). These variables along with $N.infty$, $beta.infty$ and $beta.zero$ are used in the combustion rate calculation.

The remaining expressions give the pressure, $W.at(11)$, mean molecular weight $W.at(12)$, and nitrogen content, $W.at(14)$, for the environment external to the model space. It should be noted that these $W.at$ data need not be set to normal atmospheric conditions if the inflow outflow boundary conditions represent flow to and from another section of the mine.

(Derive pseudo constants 20) $\equiv$

\[
\begin{align*}
\text{cell.volume} &= x.cell.size \cdot y.cell.size \cdot z.cell.size \\
\text{Fuel.molwt} &= 12.0 \cdot K.nc + K.nh \\
O_2.molwt &= 32.0 \\
N_2.molwt &= 28.0 \\
CO_2.molwt &= 44.0 \\
H_2O.molwt &= 18.0 \\
CO_2.m &= K.nc \cdot CO_2.molwt / (K.nc \cdot CO_2.molwt + 0.5 \cdot K.nh \cdot H_2O.molwt) \\
x.1 &= \text{fuel.molwt} / (\text{fuel.molwt} + (K.nc + 0.25 \cdot K.nh) \cdot O_2.molwt + K.ratox \cdot (K.nc + 0.25 \cdot K.nh) \cdot N_2.molwt) \\
x.2 &= (K.nc + 0.25 \cdot K.nh) \cdot O_2.molwt / (\text{fuel.molwt} + (K.nc + 0.25 \cdot K.nh) \cdot O_2.molwt + K.ratox \cdot (K.nc + 0.25 \cdot K.nh) \cdot N_2.molwt) \\
\text{Prod.molwt} &= (K.nc \cdot CO_2.molwt + 0.5 \cdot K.nh \cdot H_2O.molwt) / (K.nc + 0.5 \cdot K.nh) \\
K.stoi &= x.2 / x.1 \\
O_2.infty &= 0.209 \cdot O_2.molwt / (0.209 \cdot O_2.molwt + 0.791 \cdot N_2.molwt) \\
O_2.zero &= 0.0
\end{align*}
\]
\[ FueLinfty = 0.0 \]
\[ FueLzero = 1.0 \]
\[ N2.infty = 1 - O2.infty \]
\[ beta.infty = FueLinfty - O2.infty / K.stoi \]
\[ beta.zero = FueLzero - O2.zero / K.stoi \]
\[ W.at(11) = (K.gamma - 1.0) \times (W.at(5) - 0.5 \times (W.at(2)^2 + W.at(3)^2 + W.at(4)^2) / W.at(1)) \]
\[ W.at(14) = N2.infty \times (1 - W.at(7) / W.at(1)) \]
\[ W.at(15) = \max(0.0, (1.0 - W.at(6) / W.at(1) - K.stoi \times W.at(13) - W.at(14))) \]
\[ W.at(12) = 1.0 / (W.at(6) / (W.at(1) \times Fuel.molwt) + K.stoi \times W.at(13) / O2.molwt + W.at(14) / N2.molwt + W.at(15) / Prod.molwt) \]

This code is used in section 2.
21. THE MODEL. The underlying physical model for the simulation is based on the normal hydrodynamic equations for fluid flow plus additional equations for the fuel mass fraction, the mixture fraction, the variance in the mixture fraction and a $k$-$\epsilon$ model of turbulence.

The computational model represents a recasting of the model, proposed by Patankar and Spalding and modified by Hjertager, into an explicit formulation designed to allow efficient parallel computation.

Each of the ten governing equations are cast in the following common form.

$$\frac{\partial}{\partial t} \rho \Phi + \frac{\partial}{\partial x_j} \left( \rho u_j \Phi - \Gamma_\Phi \frac{\partial \Phi}{\partial x_j} \right) = S_\Phi$$

The expressions for $\Phi$, $\Gamma_\Phi$ and $S_\Phi$ for each of the equations are as follows.

<table>
<thead>
<tr>
<th></th>
<th>$\Phi$</th>
<th>$\Gamma_\Phi$</th>
<th>$S_\Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>$1$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>x-velocity</td>
<td>$u$</td>
<td>$\mu_t$</td>
<td>$-\frac{\partial P}{\partial x} + \frac{\partial}{\partial x} \left( \mu_t \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( \mu_t \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial z} \left( \mu_t \frac{\partial u}{\partial z} \right)$</td>
</tr>
<tr>
<td>y-velocity</td>
<td>$v$</td>
<td>$\mu_t$</td>
<td>$-\frac{\partial P}{\partial y} + \frac{\partial}{\partial x} \left( \mu_t \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left( \mu_t \frac{\partial v}{\partial y} \right) + \frac{\partial}{\partial z} \left( \mu_t \frac{\partial v}{\partial z} \right)$</td>
</tr>
<tr>
<td>z-velocity</td>
<td>$w$</td>
<td>$\mu_t$</td>
<td>$-\frac{\partial P}{\partial z} + \frac{\partial}{\partial x} \left( \mu_t \frac{\partial w}{\partial x} \right) + \frac{\partial}{\partial y} \left( \mu_t \frac{\partial w}{\partial y} \right) + \frac{\partial}{\partial z} \left( \mu_t \frac{\partial w}{\partial z} \right)$</td>
</tr>
<tr>
<td>internal energy</td>
<td>$h$</td>
<td>$\mu \sigma_h$</td>
<td>$0$</td>
</tr>
<tr>
<td>fuel fraction</td>
<td>$m_{fu}$</td>
<td>$\frac{\mu_t}{\sigma_m}$</td>
<td>$-A \rho \min(m_{fu}, m_{O2x}, m_b)$</td>
</tr>
<tr>
<td>mixture fraction</td>
<td>$f$</td>
<td>$\mu \sigma_f$</td>
<td>$0$</td>
</tr>
<tr>
<td>variance of $f$</td>
<td>$g$</td>
<td>$\frac{\mu_t}{\sigma_g}$</td>
<td>$C_g^1 \left[ \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 + \left( \frac{\partial f}{\partial z} \right)^2 \right] - C_g^2 \rho g \frac{f}{k}$</td>
</tr>
<tr>
<td>turbulent kinetic energy</td>
<td>$k$</td>
<td>$\frac{\mu_k}{\sigma_k}$</td>
<td>$G_k - \rho \epsilon$</td>
</tr>
<tr>
<td>rate of dissipation</td>
<td>$\epsilon$</td>
<td>$\frac{\mu_e}{\sigma_e}$</td>
<td>$(C_1 G_k - C_2 \rho \epsilon) \frac{\epsilon}{k}$</td>
</tr>
</tbody>
</table>

Where

$$\mu_t = \frac{C_\mu \rho k^2}{\epsilon},$$

$$G_k = \mu \left\{ 2 \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial y} \right)^2 + \left( \frac{\partial w}{\partial z} \right)^2 \right] + \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right)^2 + \left( \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \right)^2 \right\}$$

and

$$C_\mu = 0.09 \quad C_1 = 1.44 \quad C_2 = 1.92 \quad C_{g1} = 2.8 \quad C_{g2} = 2.0 \quad A = 4.0$$

$$\sigma_h = 0.7 \quad \sigma_{fu} = 0.7 \quad \sigma_f = 0.7 \quad \sigma_g = 0.7 \quad \sigma_k = 0.9 \quad \sigma_\epsilon = 1.22$$
22. The solution technique in this program uses a Total Variance Diminishing scheme based on finite volume discretization utilising a fourth order Runge-Kutta algorithm. The method is applicable as the volume discretisation converts the partial differential equations into ordinary differential equations via arguments based on Green’s theorem. The resulting explicit formulation is stable over a wide range of values which strongly suggests that this assumption of effective separability holds.

The solution scheme uses the following Runge-Kutta step for each cell

\[ W_{n+1} = W_n - \alpha Q(W_n). \]

The operator \( Q \) involves the fluxes across the cell boundaries

\[ Q = \frac{\Delta t}{\Delta V} \left\{ \Delta y \Delta z (\Delta F - \Delta R) + \Delta z \Delta x (\Delta G - \Delta S) + \Delta x \Delta y (\Delta H - \Delta T) \right\} + \frac{\Delta t}{\Delta V} ART + \Delta t \Delta C \]

where \( ART \) is an artificial viscosity term, \( F, G \) and \( H \) are the inviscid flux terms, \( R, S \) and \( T \) are the viscid flux terms and \( C \) is the source term. The detail of these terms will be covered later in the code where these terms are actually calculated.

23. The variable \( stage \) is used as an index into the Runge-Kutta solution loop.

\[
(\text{Variables of } \text{main} \ s) + \equiv \\
\quad \textbf{integer} \ stage
\]
24. The main loop of the worker task involves a number of phases of calculation.

First the current $\Phi$ values are stored to $\Phi_{old}$, $k$ and $\epsilon$ are checked and if necessary brought back to reasonable values and the timestep for the next iteration is determined.

Secondly the artificial viscosity and viscid flux components are calculated.

Finally the fourth order Runge-Kutta algorithm is invoked to calculate the inviscid flux and source components and to update the current $\Phi$'s.

\[
\text{(Do the main loop 24) } \equiv \\
\text{(Store current } W.p \text{ to } W.\text{old 25)} \\
\text{(Calculate pressure and mass terms 26)} \\
\text{(Verify } k, \epsilon \text{ 28)} \\
\text{(Set up timestep 29)} \\
\text{(Do artificial viscosity calculation 53)} \\
\text{(Do viscid flux calculation 70)} \\
\text{do stage = 1, n.of.stages } \ast 1, 2, \text{ OR 4 } \ast/ \\
\text{(Calculate pressure and mass terms 26)} \\
\text{(Verify } k, \epsilon \text{ 28)} \\
\text{(Do source term calculation 37)} \\
\text{(Do inviscid fluxes 118)} \\
\text{(Do update arrays 123)} \\
\text{enddo}
\]

This code is used in section 2.
§25 — Explode Worker Task —

25. At the beginning of each iteration the current $\Phi$ values stored in $W(1)$ through to $W(10)$ must be saved for each cell into $W(55)$ through to $W(64)$. It should be noted that the values stored in $W$ are in fact $\rho \Phi$.

The macro defining $W_{old}$ provides a clearer indication of this process than would otherwise be the case.

define $W_{old}(i) \equiv W(54 + i, cell)$

(Store current $W_p$ to $W_{old}$ 25)

\[
\begin{align*}
\text{do} & \quad \text{cell} = 1, n_{\text{of cells}} \\
\text{do} & \quad \text{index} = 1, 10 \\
& \quad W_{old}(\text{index}) = W_p(\text{index}) \\
& \text{endo}
\end{align*}
\]

endo

This code is used in section 24.
26. **PRESSURE.** The following sections perform the calculation of cell pressures and mass terms as well as checking the values of \( k \) and \( \epsilon \) for appropriate bounds.

The cell pressure \( P \) is stored in \( W(11) \) and is derived from the following relation

\[
P = (\gamma - 1)\frac{\rho h - \frac{1}{2}((\rho u)^2 + (\rho v)^2 + (\rho w)^2)}{\rho}.
\]

The stoichiometric oxygen mass \( m_{O_2} \), is stored in \( W(13) \) and is derived as follows

\[
m_{O_2} = m_f u - \beta_{\infty} - f(\beta_0 - \beta_{\infty}).
\]

The mean molecular weight of the cell contents is stored in \( W_p(12) \) and is derived from the concentration terms for fuel, \( W_p(6) \), oxygen, \( W_p(13) \), nitrogen, \( W_p(23) \) and reaction products, \( W_p(24) \).

(Calculate pressure and mass terms 26) \( \equiv \)

\[
\begin{align*}
    W_p(11) &= (\text{K.gamma} - 1.0)*(W_p(5) - 0.5*(W_p(2)^2 + W_p(3)^2 + W_p(4)^2)/W_p(1)) \\
    W_p(13) &= (W_p(6)/W_p(1) - \text{beta.infty} - W_p(7)*(\text{beta.zero} - \text{beta.infty})/W_p(1)) \\
    W_p(23) &= \text{N2.infty}*(1 - W_p(7)/W_p(1)) \\
    W_p(24) &= \max(0.0, (1.0 - W_p(6)/W_p(1) - \text{K.stoi} * W_p(13) - W_p(23))) \\
    W_p(12) &= 1.0/(W_p(6)/(W_p(1)*\text{Fuel.molwt}) + \text{K.stoi} * W_p(13)/\text{O2.molwt} + W_p(23)/\text{N2.molwt} + W_p(24)/\text{Prod.molwt})
\end{align*}
\]

enddo

This code is used in section 24.

27. The variable \( eddy \) contains the eddy viscosity for the cell.

(Variables of main 5) \( +\equiv \)

    real eddy
28. In order to minimise the likelihood of instability in the solution the values of \( k \) and \( \epsilon \) are subjected to bounds checking.

If either of \( k \) or \( \epsilon \) are \( < 10^{-10} \) they are reset to this minimum value. If the eddy viscosity \( \mu_t = \frac{C_p \rho k^2}{\epsilon} \) is greater than the maximum allowed value \( \text{eddymax} \) then \( k \) and \( \epsilon \) are recalculated such that the new eddy viscosity is set equal to the maximum allowed subject to the condition \( \epsilon \leq \epsilon_{\text{max}} \). Where \( \text{eddyl} \leq \text{eddymax} \) the values of \( k \) and \( \epsilon \) are adjusted such that the eddy viscosity is retained at its current value and \( \epsilon \leq \epsilon_{\text{max}} \).

Constraint is applied to \( \epsilon \) rather than to \( k \) as the critical process in the development of computational instability is the rate of dissipation of turbulence.

\( \text{(Verify } k, \text{ epsilon } 28) \equiv \)

\[
\begin{align*}
\text{do cell } &= 1, \text{n_of_cells} \\
\text{if (} W.P(10) / W.P(1) < 1.0 \cdot 10^{-10} \text{) then} \\
&\quad W.P(10) = W.P(1) \cdot 10^{-10} \\
\text{endif} \\
\text{if (} W.P(9) / W.P(1) < 1.0 \cdot 10^{-10} \text{) then} \\
&\quad W.P(9) = W.P(1) \cdot 10^{-10} \\
\text{endif} \\
\text{eddyl} &= K.d \cdot W.P(9)^2 / W.P(10) \\
\text{if (eddyl} > K.\text{mazeddy) then} \\
&\quad \text{eddyl} = K.\text{mazeddy} \\
&\quad W.P(10) = K.d \cdot W.P(9)^2 / \text{eddyl} \\
\text{if (} W.P(10) > W.P(1) \times K.\text{mazeps) then} \\
&\quad W.P(9) = \sqrt{W.P(1) \times K.\text{mazeddy} \times K.\text{mazeps} / K.d} \\
&\quad W.P(10) = W.P(1) \times K.\text{mazeps} \\
\text{else} \\
&\quad W.P(9) = \sqrt{K.\text{mazeddy} \times W.P(10) / K.d} \\
\text{endif} \\
\text{else} \\
\text{if (} W.P(10) > W.P(1) \times K.\text{mazeps) then} \\
&\quad W.P(9) = \sqrt{W.P(1) \times \text{eddyl} \times K.\text{mazeps} / K.d} \\
&\quad W.P(10) = W.P(1) \times K.\text{mazeps} \\
\text{endif} \\
\text{endif} \\
\text{enddo}
\]
This code is used in section 24.
29. **TIMESTEP.** The following sections calculate a preferred timestep value $dt_i$ for each cell. The value $\Delta t = \min dt_i$ is determined and the minimum value of $\Delta t$ across all workers is used as the next timestep value.

(Set up timestep 29) $\equiv$

```latex
do cell = 1, n_of_cells
  (Calculate mach\_I 31)
  (Calculate f\_vel, f\_size 33)
  (Calculate dt 35)
enddo
(Send and set timestep 36)
```

This code is used in section 24.

30. The variable $mach\_I$ is the local speed of sound in the cell.

(Variables of main 5) $\equiv$

```latex
real mach\_I
```

31. This value is calculated as $mach = \sqrt{\frac{\gamma p}{\rho}}$.

(Calculate mach\_I 31) $\equiv$

```latex
mach\_I = sqrt(K\_gamma*W\_p(11)/W\_p(1))
```

This code is used in section 29.

32. The variables $f\_vel$ and $f\_size$ are the magnitude of the total velocity and cell diagonal respectively.

(Variables of main 5) $\equiv$

```latex
real f\_vel, f\_u, f\_v, f\_w, f\_size
```
§33 — Explode Worker Task —

33. These values are calculated as

\[ \text{vel} = \sqrt{u^2 + v^2 + w^2} \]

and

\[ \text{size} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}. \]

(Calculate \( f_{\text{vel}}, f_{\text{size}} \) \( \equiv \))

\[
\begin{align*}
  f_u &= \left( W_p(2) / W_p(1) \right) \\
  f_v &= \left( W_p(3) / W_p(1) \right) \\
  f_w &= \left( W_p(4) / W_p(1) \right) \\
  f_{\text{vel}} &= \sqrt{f_u^2 + f_v^2 + f_w^2} \\
  f_{\text{size}} &= \sqrt{x\text{-cellsize}^2 + y\text{-cellsize}^2 + z\text{-cellsize}^2}
\end{align*}
\]

This code is used in section 29.

34. The variable \( dt \) is the local preferred timestep \( \delta t_i \), the variable \( \text{timestep} \) is the minimum timestep value for the cells on this processor and the variable \( \text{timecell} \) holds the cell number for which \( \delta t_i \) is minimised.

(Variables of main s \( \equiv \))

\[
\begin{align*}
  \text{real} & \ dt, \ timestep \\
  \text{integer} & \ \text{timecell}
\end{align*}
\]
35. The value of the local timestep $dt$ is derived from the expression $\delta t_i = \frac{CFL \times \text{size}}{\text{vel} \times \text{mach}}$ where $CFL$ is the Courant number.

A number of related values are stored in the $\text{debug}$ array for the cell at which the minimum timestep was set. In the event that the minimum timestep is found to be out of range this debugging information will be collected by the root task and printed.

\[ \text{(Calculate } dt \text{ 35)} \equiv \]
\[ dt = K \cdot \text{cfl} \cdot f \cdot \text{size} / (f \cdot \text{vel} + \text{mach} \cdot 1) \]

\begin{verbatim}
if (cell \equiv 1) then
    timestep = dt
    timecell = MAX_CELL*my.worker_id + cell
    debug(1) = timecell
    debug(2) = timestep
    debug(3) = K \cdot \text{cfl}
    debug(4) = f \cdot \text{size}
    debug(5) = f \cdot \text{vel}
    debug(6) = \text{mach} \cdot 1
    debug(7) = f \cdot u
    debug(8) = f \cdot v
    debug(9) = f \cdot w
    debug(10) = x \cdot \text{cell} \cdot \text{size}
    debug(11) = y \cdot \text{cell} \cdot \text{size}
    debug(12) = z \cdot \text{cell} \cdot \text{size}
    debug(13) = W_p(1)
    debug(14) = W_p(2)
    debug(15) = W_p(3)
    debug(16) = W_p(4)
    debug(17) = W_p(11)
    debug(18) = K \cdot \gamma
else
    if (dt < timestep) then
        timestep = dt
        timecell = MAX_CELL*my.worker_id + cell
        debug(1) = timecell
        debug(2) = timestep
\end{verbatim}
This code is used in section 29.

36. Now that the local minimum timestep has been determined it remains to find the global minimum across all processors. The following section of code achieves this and subsequently receives this global minimum from the root task for use as the timestep for all workers for the next iteration.

\[
\text{(Send and set timestep 36) } \equiv \\
\text{(Get packet from buffer 162)} \\
\text{if (packet(1) > timestep) then} \\
\quad \text{packet(1) = timestep} \\
\quad \text{packet(2) = timecell} \\
\text{endif} \\
\text{(Send packet to switch 163)} \\
\text{(Get packet from buffer 162)} \\
\text{(Send packet to switch 163)} \\
\text{timestep = packet(1)}
\]

This code is used in section 29.
37. SOURCE TERMS. The source terms $S_\Phi$ are now calculated. As this calculation involves the asynchronous transmission of packets between worker tasks the calculations are initially brought into step by the root task sending a packet of type `synch.type`. At the end of the calculation each worker send a packet of type `step.complete.type` to the root task. On receipt of $n\_of\_workers$ such packets the root task can initiate the next phase of the calculation.

The source term calculation, in common with the later calculation of the artificial viscosity and viscid and inviscid fluxes requires that data be collected from neighbouring cells. The computational domain for the source terms is shown in the following diagram. The letter $P$ denotes the current cell.

```
(Do source term calculation 37) ≡
  (Get synch 160)
  do cell = 1, n\_of\_cells
    (Fill near arrays 39)
    (Calculate $\mu_{eff}$ for $C$ array 41)
    (Calculate dx, dy, dz for $C$ array 43)
    (Calculate ux through wz for $C$ array 45)
    (Calculate fx through gz for $C$ array 47)
    (Calculate $\tau_{eh}$ and $\tau_e$ 49)
    (Calculate $p_k$ 51)
    (Calculate $C$ array 52)
  enddo
  (Send done 161)
```

This code is used in section 24.
38. The source term calculation requires that the values of the six orthogonally surrounding cells be determined. These values are stored in the arrays \( W_e, W_w, W_n, W_s, W_t \) and \( W_b \) for the east, west, north, south, top and bottom neighbours respectively.

The macro definitions of \( EAST \) through to \( BOTTOM \) refer to the location in the \( W \) array which contains the identity of the corresponding neighbouring cell.

```plaintext
define EAST 17
define WEST 18
define NORTH 19
define SOUTH 20
define TOP 21
define BOTTOM 22
```

\( (\text{Variables of } \text{main } 5) \) +=

```plaintext
real \( W_e(24), W_w(24), W_n(24), W_s(24), W_t(24), W_b(24) \)
```

39. The subroutine \( fillarray \) is invoked six times, once for each of the orthogonal neighbours. This routine fills the array contained in the second argument, \( W.direction \), with the values associated with the appropriate neighbour to the current cell \( W.p \) passed in the first argument.

As the majority of the arguments to \( fillarray \) are the same for all calls, the macro \( FILL \) is declared.

```plaintext
define FILL(source, destination, direction) call fillarray(source,destination,direction,from_buffer,
to_switch,timestep,K.tau_1,K.tau_2,K.hc,K.gamma,Kstoi,N2.infty,beta.infty,beta.zero,
Fuel.molwt,O2.molwt,N2.molwt,Prod.molwt,K.vent,W.at,W)
```

\( \text{(Fill near arrays 39)} \) =

```plaintext
FILL(W.p(1), W.e(1), EAST)
FILL(W.p(1), W.w(1), WEST)
FILL(W.p(1), W.n(1), NORTH)
FILL(W.p(1), W.s(1), SOUTH)
FILL(W.p(1), W.t(1), TOP)
FILL(W.p(1), W.b(1), BOTTOM)
```

This code is used in sections 37, 53, 56, 74, and 118.
40. (Variables of main 5) \[+\]

\[
\begin{align*}
\textbf{real} & \quad \text{Temp}_p, \text{mu}_l, \text{mu}_t, \text{mu}_\text{eff}, s_u
\end{align*}
\]

41. The temperature in the cell, \(\text{Temp}_p\), is now calculated along with the effective viscosity, \(\text{mu}_\text{eff}\), which is the sum of the laminar, \(\text{mu}_l\), and turbulent, \(\text{mu}_t\), viscosities and the laminar burning velocity, \(s_u\).

\[
\begin{align*}
\text{(Calculate } \mu_{\text{eff}} \text{ for } C \text{ array 41}) & \equiv \\
\text{Temp}_p & = W_p(11)*W_p(12)/(W_p(1)*K.ru) \\
\text{mu}_l & = K.145/(\text{Temp}_p + K.110) * \text{Temp}_p K.gindex \\
\text{mu}_t & = K.d*W_p(9)^2/W_p(10) \\
\text{mu}_\text{eff} & = \text{mu}_l + \text{mu}_t \\
s_u & = k.so*(\text{temp}_p/K.tref) K.salpha*(W_p(11)/W.at(11)) K.sbeta
\end{align*}
\]

This code is used in section 37.

42. The variables \(dx\), \(dy\) and \(dz\) contain the \(x\)-, \(y\)- and \(z\)-dimensions on which the flux calculations are based.

(Variables of main 5) \[+\]

\[
\begin{align*}
\textbf{real} & \quad dx, dy, dz
\end{align*}
\]

43. As fluxes are calculated from cell centres the \(dx\), \(dy\) and \(dz\) values are set to be the distance from the centre of one neighbouring cell to the centre of the opposite neighbour via the following expression \(dx = \Delta x + \frac{1}{2}(\Delta x^- + \Delta x^+).\)

The simpler calculation in this section is based on the fact that all cells are of the same size.

(Calculate \(dx\), \(dy\), \(dz\) for \(C\) array 43) \[\equiv\]

\[
\begin{align*}
dx & = 2.0*x.cellsize \\
dy & = 2.0*y.cellsize \\
dz & = 2.0*z.cellsize
\end{align*}
\]

This code is used in section 37.
44. The following variables hold the velocity fluxes.

(Variables of main s) +≡

real ux, uy, uz, vx, vy, vz, wx, wy, wz

45. The nine velocity fluxes ux through wz are given by the general equation $\Delta u = \frac{u^+ - u^-}{dx}$ for each of u, v and w in each of the three directions.

(Calculate ux through wz for C array 45) ≡

$$ux = (W_e(2)/W_e(1) - W_w(2)/W_w(1))/dx$$
$$uy = (W_n(2)/W_n(1) - W_s(2)/W_s(1))/dy$$
$$uz = (W_t(2)/W_t(1) - W_b(2)/W_b(1))/dz$$
$$vx = (W_e(3)/W_e(1) - W_w(3)/W_w(1))/dx$$
$$vy = (W_n(3)/W_n(1) - W_s(3)/W_s(1))/dy$$
$$vz = (W_t(3)/W_t(1) - W_b(3)/W_b(1))/dz$$
$$wx = (W_e(4)/W_e(1) - W_w(4)/W_w(1))/dx$$
$$wy = (W_n(4)/W_n(1) - W_s(4)/W_s(1))/dy$$
$$wz = (W_t(4)/W_t(1) - W_b(4)/W_b(1))/dz$$

This code is used in section 37.

46. The following variables hold the mixture fraction and variance fluxes.

(Variables of main s) +≡

real fx, fy, fz, gx, gy, gz
§47 — Explode Worker Task —

47. The mixture fraction fluxes are given by \( \Delta f = \frac{E^+ - E^-}{dx} \).

In a similar fashion the variance fluxes are given by \( \Delta g = \frac{\sigma^+ - \sigma^-}{dx} \).

\(<\text{Calculate } f_x \text{ through } g_z \text{ for } C \text{ array 47}>\) \equiv

\[
\begin{align*}
    f_x &= (W_e(7)/W_e(1) - W_w(7)/W_w(1))/dx \\
    f_y &= (W_n(7)/W_n(1) - W_s(7)/W_s(1))/dy \\
    f_z &= (W_t(7)/W_t(1) - W_b(7)/W_b(1))/dz \\
    g_x &= (W_e(8) - W_w(8))/dx \\
    g_y &= (W_n(8) - W_s(8))/dy \\
    g_z &= (W_t(8) - W_b(8))/dz
\end{align*}
\]

This code is used in section 37.

48.

\(<\text{Variables of main 5}>\) \equiv

\[
\begin{align*}
    \text{real } &\tau_e, \tau_{ch}
\end{align*}
\]

49. The eddy lifetime \( \tau_e \) and the chemical induction time \( \tau_{ch} \) are now calculated from the expressions

\(<\text{Calculate } \tau_{ch} \text{ and } \tau_e \text{ 49}>\) \equiv

\[
\begin{align*}
    \tau_e &= W_p(9)/W_p(10) \\
    \tau_{ch} &= K_{ach}\exp(K_{eonr}/(K_{ru}\times Temp_p))\times W_p(6)^{K-a}\times(K_{stoi}\times(W_p(5) - W_p(1)\times beta_{infty} + W_p(7)\times(beta_{infty} - beta_{zero}))^{K-b}}
\end{align*}
\]

This code is used in section 37.

50.

\(<\text{Variables of main 5}>\) \equiv

\[
\begin{align*}
    \text{real } &D, S_sq, \rho_{-k}
\end{align*}
\]
§51 — Explode Worker Task —

51.

(Calculate $\rho_k 51$) ≡

\[
D = ux + vy + wz
\]

\[
S_{sq} = (uy + vz)^2 + (uz + wx)^2 + (vz + wy)^2 + 2.0(uz^2 + vy^2 + wz^2) - 2.0/3.0*D^2
\]

\[
\rho_{o.k} = \mu_{eff} * S_{sq} - 2.0/3.0*W_{p}(0)*D
\]

This code is used in section 37.
52. The actual source terms $S_f$ may now be calculated. The macro $C_p$ reflects the storage location of the source terms in the $W$ array.

define $C_p(i) = W(24 + i, cell)$

(Calculate $C$ array 52) \( \equiv \)

\[
\begin{align*}
C_p(1) &= 0.0 \\
C_p(2) &= g_x \\
C_p(3) &= g_y \\
C_p(4) &= g_z \\
C_p(5) &= (w_p(2) g_x + w_p(3) g_y + w_p(4) g_z) / w_p(1) \\
\text{if} (\tau_{ch}/\tau_{e} < K_{die}) \text{ then} \\
\text{if} (m_{eff} > 5.0 * m_{ul}) \text{ then} \\
C_p(6) &= -K_{ebu} * w_p(1) * w_p(10) * \min(w_p(6)/w_p(1), w_p(13), w_p(24)) / w_p(9) \\
\text{elseif} (w_p(24) > 0.0) \text{ then} \\
C_p(6) &= -s_{u} * w_p(1) * \min(w_p(6)/w_p(1), w_p(13), w_p(24)) * (x_{cellsize} + y_{cellsize} + z_{cellsize}) \\
\text{else} \\
C_p(6) &= 0.0 \\
\text{endif} \\
\text{else} \\
C_p(6) &= 0.0 \\
\text{endif} \\
\text{if} (\abs(C_p(6)) > K_{maxfuel}) \text{ then} \\
C_p(6) &= -K_{maxfuel} \\
\text{endif} \\
C_p(7) &= 0.0 \\
C_p(8) &= K_{g1} * m_{eff} * (f_x^2 + f_y^2 + f_z^2 - K_{g2} * w_p(8) * w_p(10)/w_p(9)) \\
C_p(9) &= \rho_{k} - w_p(10) \\
C_p(10) &= w_p(10)/w_p(9) * (\rho_{k} K_{i} - w_p(10) * K_{2}) \\
C_p(5) &= -C_p(8) * K_{hc} \\
\end{align*}
\]

This code is used in section 37.
53. **ARTIFICIAL VISCOSITY.** The determination of the artificial viscosity terms is carried out in a similar manner to that of the source terms in that the calculation involves asynchronous packet transmission which must be monitored for completion by the root task.

The artificial viscosity calculation uses a two phase process. First the values of $r_x$, $r_y$ and $r_z$ are determined for each cell. When this process is complete for each worker the remaining calculation is done. Each of these phases is synchronised across all workers.

As in the case of the source terms, the artificial viscosity calculation involves the use of data from neighbouring cells. The computational domain for the artificial viscosity terms is shown in the following diagram.

\[
\begin{align*}
\text{Domain for } ART &= \quad (\text{Do artificial viscosity calculation 53}) \equiv \\
&= (\text{Get synch 160}) \\
&\quad \text{do } cell = 1, n_{\text{of cells}} \\
&\quad \quad (\text{Fill near arrays 39}) \\
&\quad \quad (\text{Calculate r terms 54}) \\
&\quad \text{enddo}
\end{align*}
\]
§53 — Explode Worker Task —

(Send done 161)

(Get synch 160)

do cell = 1, n_of_cells

(Fill off-element arrays for ART calculation 56)

(Calculate r maxima 64)

(Calculate E terms 66)

(Calculate Q terms 68)

(Calculate ART 69)

endo

(Send done 161)

This code is used in section 24.

54. The values for the flux gradient sensors \( r_x \), \( r_y \) and \( r_z \) are derived from the pressure in the cell and its orthogonal neighbours in the following manner. These values are stored in \( W \) (14, \( W(15) \) and \( W(16) \).

\[
    r_P = \frac{|P^- + P^+ - 2P|}{|P^- + P^+ + 2P|}
\]

(Calculate \( r \) terms 54) \( \equiv \)

\[
    W_p(14) = \frac{abs(W_w(11) + W_e(11) - 2.0*W_p(11))}{abs(W_w(11) + W_e(11) + 2.0*W_p(11))}
\]

\[
    W_p(15) = \frac{abs(W_s(11) + W_n(11) - 2.0*W_p(11))}{abs(W_s(11) + W_n(11) + 2.0*W_p(11))}
\]

\[
    W_p(16) = \frac{abs(W_b(11) + W_t(11) - 2.0*W_p(11))}{abs(W_b(11) + W_t(11) + 2.0*W_p(11))}
\]

This code is used in section 53.

55. Six additional cell data sets are required for the calculation of the artificial viscosity terms. These are the values for the cells two steps away in each of the orthogonal directions. The variables \( W_fe \) through \( W_fb \) will hold these far cell values.

(Variables of main 5) \( \equiv \)

real \( W_fe(24), W_fw(24), W_fn(24), W_fs(24), W_ft(24), W_fb(24) \)
56. The twelve neighbouring data sets required for the artificial viscosity calculation are now assembled.

\[
\text{Fill off-element arrays for } ART \text{ calculation 56} \equiv \\
(\text{Fill near arrays 39}) \\
(\text{Fill } W.fe \ 57) \\
(\text{Fill } W fw \ 58) \\
(\text{Fill } W.fn \ 59) \\
(\text{Fill } W.fs \ 60) \\
(\text{Fill } W.ft \ 61) \\
(\text{Fill } W.fb \ 62)
\]

This code is used in section 53.

57. In determining the values for the far cells two possible cases exist. If the current cell has a boundary on the corresponding side then the far cell is assumed to be identical to the equivalent near neighbour. If this is not the case then \textit{fillarray} is called in the expected manner.

The far west cell is the first to be filled.

\[
(\text{Fill } W.fe \ 57) \equiv \\
\text{if } (W.p(EAST) < 0) \text{ then} \\
\hspace{1em} FILL(W.p(1), W.fe(1), EAST) \\
\text{else} \\
\hspace{1em} FILL(W.e(1), W.fe(1), EAST) \\
\text{endif}
\]

This code is used in section 56.
58. The remaining five far cells are filled in the same manner.

The far west cell is filled.

\[
\text{\{Fill } W_{fw}\text{ 58\} } \equiv \\
\text{if } (W_p(WEST) < 0) \text{ then } \\
\text{ FILL}(W_p(1), W_{fw}(1), WEST) \\
\text{else } \\
\text{ FILL}(W_w(1), W_{fw}(1), WEST) \\
\text{endif}
\]

This code is used in section 56.

59. The far north cell is filled.

\[
\text{\{Fill } W_{fn}\text{ 59\} } \equiv \\
\text{if } (W_p(NORTH) < 0) \text{ then } \\
\text{ FILL}(W_p(1), W_{fn}(1), NORTH) \\
\text{else } \\
\text{ FILL}(W_n(1), W_{fn}(1), NORTH) \\
\text{endif}
\]

This code is used in section 56.

60. The far south cell is filled.

\[
\text{\{Fill } W_{fs}\text{ 60\} } \equiv \\
\text{if } (W_p(SOUTH) < 0) \text{ then } \\
\text{ FILL}(W_p(1), W_{fs}(1), SOUTH) \\
\text{else } \\
\text{ FILL}(W_s(1), W_{fs}(1), SOUTH) \\
\text{endif}
\]

This code is used in section 56.
§75 — Explode Worker Task —

Depending on the presence of boundaries a number of possible ways exist to determine the values in a diagonal neighbour. In the case of the north-east cell the following strategy is used.

If boundaries exist to both the north and east of the current cell then the north east cell is assumed to be identical to the north cell. If the eastern cell side is a boundary then the north east cell value is determined via the north cell. In any other case the north east cell values are determined via the east neighbour.

The following sections fill the twelve diagonal arrays in a similar manner.

The north east array is filled.

\[
(Fill \ W_{ne} \ 75) \equiv \\
\text{if } (W.p(EAST) < 0) \text{ then} \\
\quad \text{if } (W.p(NORTH) < 0) \text{ then} \\
\quad \quad FILL(W.p(1), W_{ne}(1), NORTH) \\
\quad \text{else} \\
\quad \quad FILL(W.n(1), W_{ne}(1), EAST) \\
\quad \text{endif} \\
\text{else} \\
\quad FILL(W.e(1), W_{ne}(1), NORTH) \\
\text{endif}
\]

This code is used in section 74.

The south east array is filled.

\[
(Fill \ W_{se} \ 76) \equiv \\
\text{if } (W.p(EAST) < 0) \text{ then} \\
\quad \text{if } (W.p(SOUTH) < 0) \text{ then} \\
\quad \quad FILL(W.p(1), W_{se}(1), SOUTH) \\
\quad \text{else} \\
\quad \quad FILL(W.s(1), W_{se}(1), EAST) \\
\quad \text{endif} \\
\text{else} \\
\quad FILL(W.e(1), W_{se}(1), SOUTH) \\
\text{endif}
\]

This code is used in section 74.
77. The north west array is filled.

\[
\text{\{Fill } W_{nw} \text{ 77\}} \equiv \\
\text{if } (W_p(NORTH) < 0) \text{ then } \\
\text{ if } (W_p(WEST) < 0) \text{ then } \\
\quad FILL(W_p(1), W_{nw}(1), WEST) \\
\text{ else } \\
\quad FILL(W_w(1), W_{nw}(1), NORTH) \\
\text{ endif } \\
\text{ else } \\
\quad FILL(W_n(1), W_{nw}(1), WEST) \\
\text{ endif }
\]

This code is used in section 74.

78. The south west array is filled.

\[
\text{\{Fill } W_{sw} \text{ 78\}} \equiv \\
\text{if } (W_p(SOUTH) < 0) \text{ then } \\
\text{ if } (W_p(WEST) < 0) \text{ then } \\
\quad FILL(W_p(1), W_{sw}(1), WEST) \\
\text{ else } \\
\quad FILL(W_w(1), W_{sw}(1), SOUTH) \\
\text{ endif } \\
\text{ else } \\
\quad FILL(W_s(1), W_{sw}(1), WEST) \\
\text{ endif }
\]

This code is used in section 74.
§79 — Explode Worker Task —

79. The top east array is filled.

\[(\text{Fill } W.te\ 79) \equiv \]
\[
\text{if } (W.p(\text{EAST}) < 0) \text{ then }
\]
\[
\text{if } (W.p(\text{TOP}) < 0) \text{ then }
\]
\[
\text{FILL}(W.p(1), W.te(1), \text{TOP})
\]
\[
\text{else}
\]
\[
\text{FILL}(W.t(1), W.te(1), \text{EAST})
\]
\[
\text{endif}
\]
\[
\text{else}
\]
\[
\text{FILL}(W.e(1), W.te(1), \text{TOP})
\]
\[
\text{endif}
\]

This code is used in section 74.

80. The bottom east array is filled.

\[(\text{Fill } W.be\ 80) \equiv \]
\[
\text{if } (W.p(\text{EAST}) < 0) \text{ then }
\]
\[
\text{if } (W.p(\text{BOTTOM}) < 0) \text{ then }
\]
\[
\text{FILL}(W.p(1), W.be(1), \text{BOTTOM})
\]
\[
\text{else}
\]
\[
\text{FILL}(W.b(1), W.be(1), \text{EAST})
\]
\[
\text{endif}
\]
\[
\text{else}
\]
\[
\text{FILL}(W.e(1), W.be(1), \text{BOTTOM})
\]
\[
\text{endif}
\]

This code is used in section 74.
61. The far top cell is filled.

\[\text{Fill } W_{ft} \text{ } 61 \equiv\]

\[
\text{if } (W_p(TOP) < 0) \text{ then } \\
\quad FILL(W_p(1), W_{ft}(1), TOP) \\
\text{else } \\
\quad FILL(W_t(1), W_{ft}(1), TOP) \\
\text{endif}
\]

This code is used in section 56.

62. Finally the far bottom cell is filled.

\[\text{Fill } W_{fb} \text{ } 62 \equiv\]

\[
\text{if } (W_p(BOTTOM) < 0) \text{ then } \\
\quad FILL(W_p(1), W_{fb}(1), BOTTOM) \\
\text{else } \\
\quad FILL(W_b(1), W_{fb}(1), BOTTOM) \\
\text{endif}
\]

This code is used in section 56.

63. These variables will hold the local flux gradient sensor maxima.

\[\text{Variables of main 5 } +\equiv\]

\[
\text{real } r_{xe}, r_{xw}, r_{yn}, r_{ys}, r_{xt}, r_{zb}
\]
64. The flux gradient sensors \( r_P \) for the cell and its neighbours are now used to calculate local maxima \( r_{xe} \) through \( r_{zb} \) using the following general expressions.

\[
\begin{align*}
  r^- &= \max (r_P^-, r_P, r_P, r_P^+) \\
  r^+ &= \max (r_P^-, r_P, r_P^-, r_P^+) \\
  r_{xe} &= \max (W.w(14), W_p(14), W_e(14), W.fe(14)) \\
  r_{zw} &= \max (W.fw(14), W_w(14), W_p(14), W_e(14)) \\
  r_{yn} &= \max (W.s(15), W_p(15), W_n(15), W.fn(15)) \\
  r_{ys} &= \max (W.fs(15), W.s(15), W_p(15), W_n(15)) \\
  r_{zt} &= \max (W.b(16), W_p(16), W.t(16), W.ft(16)) \\
  r_{zb} &= \max (W.fb(16), W_b(16), W_p(16), W.t(16))
\end{align*}
\]

This code is used in section 53.

65. (Variables of hypo 5) \( +\equiv \)

\[ \text{real } V.t, E.e, E.w, E.n, E.s, E.t, E.b \]

66. The artificial viscosity components are formulated in terms of three sets of two orthogonal fluxes, \( E \) and \( Q \). The following code sections calculate the inward and outward components of these fluxes.

First the six \( E \) values are derived from the expression

\[
E = \frac{V}{\Delta t} \min(0.5, k^2r)^
\]

(\text{Calculate } E \text{ terms 66) } \equiv \)

\[
\begin{align*}
  V.t &= \text{cell volume/timestep} \\
  E.e &= V.t \times \min(0.5, r_{xe} \times K.k2) \\
  E.w &= V.t \times \min(0.5, r_{zw} \times K.k2) \\
  E.n &= V.t \times \min(0.5, r_{yn} \times K.k2) \\
  E.s &= V.t \times \min(0.5, r_{ys} \times K.k2) \\
  E.t &= V.t \times \min(0.5, r_{zt} \times K.k2) \\
  E.b &= V.t \times \min(0.5, r_{zb} \times K.k2)
\end{align*}
\]

This code is used in section 53.
(Variables of main 5) +

real $Q_e$, $Q_w$, $Q_s$, $Q_n$, $Q_t$, $Q_b$

68. Next the $Q$ values are derived from the expression

$$Q = \frac{V}{\Delta t} \max(0, k_4 k_{alpha} r x).$$

(Calculate $Q$ terms 68) $\equiv$

\begin{align*}
Q_e &= V.t * \max(0.0, K.k4 - K.alpha_h * r.xe) \\
Q_w &= V.t * \max(0.0, K.k4 - K.alpha_h * r.xw) \\
Q_n &= V.t * \max(0.0, K.k4 - K.alpha_h * r.yn) \\
Q_s &= V.t * \max(0.0, K.k4 - K.alpha_h * r.ys) \\
Q_t &= V.t * \max(0.0, K.k4 - K.alpha_h * r.zt) \\
Q_b &= V.t * \max(0.0, K.k4 - K.alpha_h * r.zb)
\end{align*}

This code is used in section 53.
74. The calculation of the $R$, $S$ and $T$ components of the viscid flux each require data from some of the twelve diagonal cells in addition to the six orthogonal neighbours. The following sections acquire these data.

\[
\begin{align*}
\text{(Fill off-element arrays for } R, S \text{ and } T \text{ calculations 74)} & \equiv \\
\text{(Fill near arrays 39)} & \\
\text{(Fill } W_{ne} 75) & \\
\text{(Fill } W_{se} 76) & \\
\text{(Fill } W_{nw} 77) & \\
\text{(Fill } W_{sw} 78) & \\
\text{(Fill } W_{te} 79) & \\
\text{(Fill } W_{be} 80) & \\
\text{(Fill } W_{tw} 81) & \\
\text{(Fill } W_{bw} 82) & \\
\text{(Fill } W_{tn} 83) & \\
\text{(Fill } W_{bn} 84) & \\
\text{(Fill } W_{ts} 85) & \\
\text{(Fill } W_{bs} 86) & \\
\end{align*}
\]

This code is used in section 70.
71. (Variables of \textit{main 5} \(\equiv\))
   \begin{verbatim}
   real Temp_avg, cp_p
   \end{verbatim}

72. The value of the cell temperature \(\text{Temp}_p\) along with the cell specific heat are now calculated.

\begin{verbatim}
\text{(Calculate Temp}_p, cp_p \(\equiv\) )
\text{Temp}_p = W_p(11)*W_p(12)/(W_p(1)*K_ru)
\text{Temp}_avg = 0.5*(\text{Temp}_p + K_tref)
\text{cp}_p = 1000.0*(W_p(6)/W_p(1)*(K_cpa(1) + K_cpb(1)*Temp_avg) + K_stoi*W_p(13)*(K_cpa(2) + K_cpb(2)*Temp_avg) + W_p(23)*(K_cpa(3) + K_cpb(3)*Temp_avg) + W_p(24)*CO2_m*(K_cpa(4) + K_cpb(4)*Temp_avg) + W_p(24)*(1 - CO2_m)*(K_cpa(5) + k_cpb(5)*Temp_avg))
\end{verbatim}

This code is used in section 70.

73. The following arrays will contain the data from the diagonal neighbour cells.

\begin{verbatim}
\text{(Variables of \textit{main 5} \(\equiv\) )}
\text{real W_ne(24), W_se(24), W_te(24), W_be(24), W_nw(24), W_tn(24), W_bn(24), W_tw(24), W_ts(24), W_bw(24), W_sw(24), W_bs(24)}
\end{verbatim}
70. **VISCID FLUXES.** The calculation of the viscid flux terms is also performed within a synchronised block.

Once again, data from neighbouring cells is required for these calculations. The computational domain for the viscid fluxes is shown in the following diagram.

![Diagram of computational domain for R, S and T]

(Do viscid flux calculation 70) ≡

(Get synch 160)

**do** cell = 1, n_of_cells

(Calculate Temp_p, cp_p 72)

(Fill off-element arrays for R, S and T calculations 74)

(Calculate R array 87)

(Calculate S array 98)

(Calculate T array 108)

**enddo**

(Send done 161)

This code is used in section 24.
69. Finally the artificial viscosity terms may be calculated from the $E$ and $Q$ values in the following manner.

$$ART_i = \sum_{x,y,z} (E\Phi)_i^+ - (Q\Phi)_i^-$$

where

$$(E\Phi)_i^+ = E_i^+(\Phi_i^+ - \Phi_i^-) - E_i^-(\Phi_i^+ - \Phi_i^-)$$

and

$$(Q\Phi)_i^+ = Q_i^+(\Phi_i^+ - 3\Phi_i^+ + 3\Phi_i^--\Phi_i^-) - Q_i^-(\Phi_i^+ - 3\Phi_i^+ + 3\Phi_i^- - \Phi_i^-)$$

The macro $ARST$ is used to indicate the storage locations into which the artificial viscosity and viscous fluxes are stored.

**define** $ARST.p(i) \ W(34 + i, cell)$

(Calculate $ART \ 69$)

**do** $index = 1, 10$

\[
ARST.p(index) = E.e*(W.e(index) - W.p(index)) - E.w*(W.p(index) - W.w(index)) - Q.e*(W.se(index) - 3.0*W.e(index) + 3.0*W.p(index) - W.w(index)) + Q.w*(W.e(index) - 3.0*W.p(index) + 3.0*W.w(index) - W.w(index)) - E.s*(W.p(index) - W.s(index)) - Q.n*(W.fn(index) - 3.0*W.n(index) + 3.0*W.p(index) - W.s(index)) + Q.s*(W.n(index) - 3.0*W.p(index) + 3.0*W.s(index) - W.fs(index)) + E.t*(W.t(index) - W.p(index)) - E.b*(W.p(index) - W.b(index)) - Q.t*(W.ft(index) - 3.0*W.t(index) + 3.0*W.p(index) - W.b(index)) + Q.b*(W.t(index) - 3.0*W.p(index) + 3.0*W.b(index) - W.fb(index))
\]

**endo**

This code is used in section 53.
§81 — Explode Worker Task —

81. The top west array is filled.

(Fill $W_{tw}$ 81) ≡

\[
\begin{align*}
&\text{if } (W_p(TOP) < 0) \text{ then} \\
&\quad \text{if } (W_p(WEST) < 0) \text{ then} \\
&\quad \quad \text{FILL}(W_p(1), W_{tw}(1), WEST) \\
&\quad \text{else} \\
&\quad \quad \text{FILL}(W_w(1), W_{tw}(1), TOP) \\
&\text{else} \\
&\quad \text{FILL}(W_t(1), W_{tw}(1), WEST)
\end{align*}
\]

This code is used in section 74.

82. The bottom west array is filled.

(Fill $W_{bw}$ 82) ≡

\[
\begin{align*}
&\text{if } (W_p(BOTTOM) < 0) \text{ then} \\
&\quad \text{if } (W_p(WEST) < 0) \text{ then} \\
&\quad \quad \text{FILL}(W_p(1), W_{bw}(1), WEST) \\
&\quad \text{else} \\
&\quad \quad \text{FILL}(W_w(1), W_{bw}(1), BOTTOM) \\
&\text{else} \\
&\quad \text{FILL}(W_b(1), W_{bw}(1), WEST)
\end{align*}
\]

This code is used in section 74.
§83 — Explode Worker Task —

83. The top north array is filled.

\[
\text{\{Fill } W_{tn} \text{ 83\}} \equiv \\
\quad \text{if } (W_p(NORTH) < 0) \text{ then} \\
\quad \quad \text{if } (W_p(TOP) < 0) \text{ then} \\
\quad \quad \quad \text{FILL}(W_p(1), W_{tn}(1), TOP) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{FILL}(W_t(1), W_{tn}(1), NORTH) \\
\quad \text{endif} \\
\quad \text{else} \\
\quad \quad \text{FILL}(W_n(1), W_{tn}(1), TOP) \\
\text{endif}
\]

This code is used in section 74.

84. The bottom north array is filled.

\[
\text{\{Fill } W_{bn} \text{ 84\}} \equiv \\
\quad \text{if } (W_p(NORTH) < 0) \text{ then} \\
\quad \quad \text{if } (W_p(BOTTOM) < 0) \text{ then} \\
\quad \quad \quad \text{FILL}(W_p(1), W_{bn}(1), BOTTOM) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{FILL}(W_b(1), W_{bn}(1), NORTH) \\
\quad \text{endif} \\
\quad \text{else} \\
\quad \quad \text{FILL}(W_n(1), W_{bn}(1), BOTTOM) \\
\text{endif}
\]

This code is used in section 74.
§85 — Explode Worker Task —

85. The top south array is filled.

\[(\text{Fill } W_{ts} \text{ 85}) \equiv \]

\[
\begin{align*}
\text{if } (W_p\text{(TOP)} < 0) & \text{ then} \\
\quad & \text{if } (W_p\text{(SOUTH)} < 0) \text{ then} \\
\quad & \text{ } FILL(W_p(1), W_{ts}(1), \text{SOUTH}) \\
\quad & \text{else} \\
\quad & \text{ } FILL(W_s(1), W_{ts}(1), \text{TOP}) \\
\text{else} & \\
\quad & \text{ } FILL(W_t(1), W_{ts}(1), \text{SOUTH})
\end{align*}
\]

This code is used in section 74.

86. The bottom south array is filled.

\[(\text{Fill } W_{bs} \text{ 86}) \equiv \]

\[
\begin{align*}
\text{if } (W_p\text{(BOTTOM)} < 0) & \text{ then} \\
\quad & \text{if } (W_p\text{(SOUTH)} < 0) \text{ then} \\
\quad & \text{ } FILL(W_p(1), W_{bs}(1), \text{SOUTH}) \\
\quad & \text{else} \\
\quad & \text{ } FILL(W_s(1), W_{bs}(1), \text{BOTTOM}) \\
\text{else} & \\
\quad & \text{ } FILL(W_b(1), W_{bs}(1), \text{SOUTH})
\end{align*}
\]

This code is used in section 74.
§87 — Explode Worker Task —

The viscid flux is expressed in terms of three components, $R$, $S$ and $T$ in each of the three orthogonal directions.

As each of $R$, $S$ and $T$ are calculated in a similar manner only the detail of the $R$ calculation will be given.

\[
(R) = (\text{Calculate } \mu_{\text{eff}} \text{ for } R \text{ array } 87) = (\text{Calculate } dx, dy, dz \text{ for } R \text{ array } 93) = (\text{Calculate } ux \text{ through } wz \text{ for } R \text{ array } 95) = \text{Fill the } R \text{ array } 97)
\]

This code is used in section 70.

§88. The following variables represent the mean effective turbulent viscosity and mean turbulent heat flux for the cell taken with each of its orthogonal neighbours.

\[
\text{real } \mu_{h.e}, \mu_{h.w}, \mu_{h.n}, \mu_{h.t}, \mu_{h.b}
\]

\[
\text{real } \mu_{\text{eff.e}}, \mu_{\text{eff.w}}, \mu_{\text{eff.n}}, \mu_{\text{eff.s}}, \mu_{\text{eff.t}}, \mu_{\text{eff.b}}
\]

§89. The mean effective turbulent viscosity and mean turbulent heat flux are now calculated for the cell and its eastern and western neighbours.

\[
(\text{Calculate } \mu_{\text{eff}} \text{ for } R \text{ array } 89) = (\text{Calculate } Temp_e, cp_e 91)
\]

\[
\mu_l = K_145*(0.5*(\text{Temp.p} + \text{Temp.e}))K_{\text{gindex}}/(0.5*(\text{Temp.p} + \text{Temp.e}) + K_{110})
\]

\[
\mu_t = K_d*0.5*(W_p(9) + W_e(9))^2/(W_p(10) + W_e(10))
\]

\[
\mu_{h.e} = 0.5*(cp_p + cp_e)*(\mu_l/K_{\text{prn}} + \mu_t/K_{\text{prnt}})
\]

\[
\mu_{\text{eff.e}} = \mu_l + \mu_t
\]

\[
(\text{Calculate } Temp_w, cp_w 92)
\]

\[
\mu_l = K_145*(0.5*(\text{Temp.p} + \text{Temp.w}))K_{\text{gindex}}/(0.5*(\text{Temp.p} + \text{Temp.w}) + K_{110})
\]

\[
\mu_t = K_d*0.5*(W_p(9) + W_w(9))^2/(W_p(10) + W_w(10))
\]

\[
\mu_{h.w} = 0.5*(cp_p + cp_w)*(\mu_l/K_{\text{prn}} + \mu_t/K_{\text{prnt}})
\]

\[
\mu_{\text{eff.w}} = \mu_l + \mu_t
\]

This code is used in section 87.
§90 — Explode Worker Task — VISCID FLUXES 193

90.

\{ Variables of main 5 \} \equiv

\textbf{real} Temp\_e, cp\_e, Temp\_w, cp\_w

91.

\{ Calculate Temp\_e, cp\_e \} \equiv

\text{Temp\_e } = \frac{W\_e(11)W\_e(12)}{(W\_e(1)K\_ru)}

\text{Temp\_avg } = 0.5(\text{Temp\_e } + K\_tref)

\text{cp\_e } = 1000.0\frac{(W\_e(6)/W\_e(1))(K\_cpa(1) + K\_cpb(1)\text{Temp\_avg}) + K\_stoiW\_e(13)(K\_cpa(2) + K\_cpb(2)\text{Temp\_avg}) + W\_e(23)(K\_cpa(3) + K\_cpb(3)\text{Temp\_avg}) + W\_e(24)CO2\_m(K\_cpa(4) + K\_cpb(4)\text{Temp\_avg})}{W\_e(24)(1 - CO2\_m)(K\_cpa(5) + K\_cpb(5)\text{Temp\_avg})}

This code is used in section 89.

92.

\{ Calculate Temp\_w, cp\_w \} \equiv

\text{Temp\_w } = \frac{W\_w(11)W\_w(12)}{(W\_w(1)K\_ru)}

\text{Temp\_avg } = 0.5(\text{Temp\_w } + K\_tref)

\text{cp\_w } = 1000.0\frac{(W\_w(6)/W\_w(1))(K\_cpa(1) + K\_cpb(1)\text{Temp\_avg}) + K\_stoiW\_w(13)(K\_cpa(2) + K\_cpb(2)\text{Temp\_avg}) + W\_w(23)(K\_cpa(3) + K\_cpb(3)\text{Temp\_avg}) + W\_w(24)CO2\_m(K\_cpa(4) + K\_cpb(4)\text{Temp\_avg})}{W\_w(24)(1 - CO2\_m)(K\_cpa(5) + K\_cpb(5)\text{Temp\_avg})}

This code is used in section 89.
93. The $R$ fluxes are calculated in a $2 \times 3 \times 3$ region thus the centre to centre distances $dx$, $dy$ and $dz$ are as given here. Once again a non-uniform cell size would necessitate a more complex calculation here.

The computational domains for $R_W$ and $R_E$, the two components of $R$, are shown in the following diagram.

(Calculate $dx$, $dy$, $dz$ for $R$ array 93) \equiv

$$dx = x\_cell\_size$$

$$dy = 2.0 \times y\_cell\_size$$

$$dz = 2.0 \times z\_cell\_size$$

This code is used in section 87.

94.

(Variables of main 5) \equiv

\textbf{real} \ ux\_e, \ uy\_e, \ uz\_e, \ vx\_e, \ vy\_e, \ vz\_e, \ wx\_e, \ wy\_e, \ wz\_e, \ tx\_e

\textbf{real} \ ux\_w, \ uy\_w, \ uz\_w, \ vx\_w, \ vy\_w, \ vz\_w, \ wx\_w, \ wy\_w, \ wz\_w, \ tx\_w
95. The velocity fluxes $u_{x,e}$ through $w_{z,e}$ and the temperature flux $t_{x,e}$ for the cell and its eastern neighbour are calculated along with the corresponding fluxes for the cell and its western neighbour.

The $x$ component of the velocity fluxes is calculated in a straightforward manner from the expression.

$$u_{x,e}^x = \frac{u_e - u}{dx}$$

The $y$ and $z$ components of the velocity fluxes are calculated as the average of two fluxes, one through the cell and the other through its eastern neighbour via the following expression.

$$u_{y,e}^y = \frac{1}{2} \left( \frac{u_n - u_s}{dy} + \frac{u_{ne} - u_{se}}{dy} \right)$$

$$u_{z,e}^z = \frac{1}{2} \left( \frac{u_n - u_s}{dz} + \frac{u_{ne} - u_{se}}{dz} \right)$$

This code is used in section 87.
96. (Variables of main 5) =
    \textbf{real} A.e, A.w, R.p.e(10), R.p.w(10)

97. The x components of the viscid flux $R_i$ are calculated as the difference between the outward fluxes $R^+_i$ and the inward fluxes $R^-_i$. The resulting values are accumulated into the $ARST$ array.

(Fill the $R$ array 97) =
\begin{align*}
A.e &= y.cell.size*z.cell.size \\
R.p.e(1) &= 0.0 \\
R.p.e(2) &= A.e*(-2.0/3.0*(0.5*(W_p(9) + W_e(9)) + mu.eff.e*(uz.e + vy.e + wz.e)) + 2.0*mu.eff.e*ux.e) \\
R.p.e(3) &= A.e*mu.eff.e*(uy.e + vz.e) \\
R.p.e(4) &= A.e*mu.eff.e*(uz.e + wx.e) \\
R.p.e(5) &= R.p.e(2)*(W_p(2) + W.e(2))/(W_p(1) + W.e(1)) + R.p.e(3)*(W_p(3) + W.e(3))/(W_p(1) + W.e(1)) + R.p.e(4)*(W_p(4) + W.e(4))/(W_p(1) + W.e(1)) + mu.h.e*tx.e*A.e \\
\text{do} \quad \text{index} = 6, 10 \\
R.p.e(index) &= \text{mu.eff.e*A.e}/K.sigma(index)*(W.e(index)/W.e(1) - W.p(index)/W.p(1))/dx \\
\text{enddo}
\end{align*}

\begin{align*}
A.w &= y.cell.size*z.cell.size \\
R.p.w(1) &= 0.0 \\
R.p.w(2) &= A.w*(-2.0/3.0*(0.5*(W_p(9) + W.w(9)) + mu.eff.w*(ux.w + vy.w + wz.w)) + 2.0*mu.eff.w*ux.w) \\
R.p.w(3) &= A.w*mu.eff.w*(uy.w + vz.w) \\
R.p.w(4) &= A.w*mu.eff.w*(uz.w + wx.w) \\
R.p.w(5) &= R.p.w(2)*(W_p(2) + W.w(2))/(W_p(1) + W.w(1)) + R.p.w(3)*(W_p(3) + W.w(3))/(W_p(1) + W.w(1)) + R.p.w(4)*(W_p(4) + W.w(4))/(W_p(1) + W.w(1)) + mu.h.w*tx.w*A.w \\
\text{do} \quad \text{index} = 6, 10 \\
R.p.w(index) &= \text{mu.eff.w*A.w}/K.sigma(index)*(W.w(index)/W.w(1) - W.p(index)/W.p(1))/dx \\
\text{enddo}
\end{align*}

\begin{align*}
\text{do} \quad \text{index} &= 1, 10 \\
ARST.p(index) &= -R.p.e(index) + R.p.w(index) \\
\text{enddo}
\end{align*}

This code is used in section 87.
98. The y components of the viscid flux $S$ are calculated in the same manner as $R$.

$$ \text{(Calculate } S \text{ array 98)} \equiv $$

$$ \{ \text{Calculate } \mu_{\text{eff}} \text{ for } S \text{ array 99} \} $$

$$ \{ \text{Calculate } dx, dy, dz \text{ for } S \text{ array 103} \} $$

$$ \{ \text{Calculate } ux \text{ through } wz \text{ for } S \text{ array 105} \} $$

$$ \{ \text{Fill the } S \text{ array 107} \} $$

This code is used in section 70.

99.

$$ \text{(Calculate } \mu_{\text{eff}} \text{ for } S \text{ array 99)} \equiv $$

$$ \{ \text{Calculate } \text{Temp}_n, \text{cp}_n \ 101 \} $$

$$ \mu_l = K_{145} \ast (0.5 \ast (\text{Temp}_p + \text{Temp}_n)) / (0.5 \ast (\text{Temp}_p + \text{Temp}_n) + K_{110}) $$

$$ \mu_t = K_d \ast 0.5 \ast (W_p(9) + W_n(9))^2 / (W_p(10) + W_n(10)) $$

$$ \mu_h = 0.5 \ast (\text{cp}_p + \text{cp}_n) \ast (\mu_l/K_{prn} + \mu_t/K_{prnt}) $$

$$ \mu_{\text{eff}} = \mu_l + \mu_t $$

$$ \{ \text{Calculate } \text{Temp}_s, \text{cp}_s \ 102 \} $$

$$ \mu_l = K_{145} \ast (0.5 \ast (\text{Temp}_p + \text{Temp}_s)) / (0.5 \ast (\text{Temp}_p + \text{Temp}_s) + K_{110}) $$

$$ \mu_t = K_d \ast 0.5 \ast (W_p(9) + W_s(9))^2 / (W_p(10) + W_s(10)) $$

$$ \mu_h = 0.5 \ast (\text{cp}_p + \text{cp}_s) \ast (\mu_l/K_{prn} + \mu_t/K_{prnt}) $$

$$ \mu_{\text{eff}} = \mu_l + \mu_t $$

This code is used in section 98.

100.

$$ \{ \text{Variables of main 5} \} \equiv $$

$$ \textbf{real} \ \text{Temp}_n, \ \text{cp}_n, \ \text{Temp}_s, \ \text{cp}_s $$
§101 — Explode Worker Task —

101.

\( \text{(Calculate } \text{Temp}_n, \text{cp}_n \text{ 101) } \equiv \)

\[
\text{Temp}_n = \frac{W_n(11) \cdot W_n(12)}{(W_n(1) \cdot K_{ru})}
\]

\[
\text{Temp}_{avg} = 0.5 \times (\text{Temp}_n + K_{tref})
\]

\[
\text{cp}_n = 1000.0 \times \left( \frac{W_n(6)}{W_n(1)} \times (K_{cpa(1)} + K_{cpb(1)} \times \text{Temp}_{avg}) + K_{stoi} \times W_n(13) \times (K_{cpa(2)} + K_{cpb(2)} \times \text{Temp}_{avg}) + W_n(23) \times (K_{cpa(3)} + K_{cpb(3)} \times \text{Temp}_{avg}) + W_n(24) \times CO_2_m \times (K_{cpa(4)} + K_{cpb(4)} \times \text{Temp}_{avg}) + W_n(24) \times (1 - CO_2_m) \times (K_{cpa(5)} + K_{cpb(5)} \times \text{Temp}_{avg}) \right)
\]

This code is used in section 99.

102.

\( \text{(Calculate } \text{Temp}_s, \text{cp}_s \text{ 102) } \equiv \)

\[
\text{Temp}_s = \frac{W_s(11) \cdot W_s(12)}{(W_s(1) \cdot K_{ru})}
\]

\[
\text{Temp}_{avg} = 0.5 \times (\text{Temp}_s + K_{tref})
\]

\[
\text{cp}_s = 1000.0 \times \left( \frac{W_s(6)}{W_s(1)} \times (K_{cpa(1)} + K_{cpb(1)} \times \text{Temp}_{avg}) + K_{stoi} \times W_s(13) \times (K_{cpa(2)} + K_{cpb(2)} \times \text{Temp}_{avg}) + W_s(23) \times (K_{cpa(3)} + K_{cpb(3)} \times \text{Temp}_{avg}) + W_s(24) \times CO_2_m \times (K_{cpa(4)} + K_{cpb(4)} \times \text{Temp}_{avg}) + W_s(24) \times (1 - CO_2_m) \times (K_{cpa(5)} + K_{cpb(5)} \times \text{Temp}_{avg}) \right)
\]

This code is used in section 99.
§103 — Explode Worker Task —

The $S$ fluxes are calculated in a $3 \times 2 \times 3$ region thus the centre to centre distances $dx$, $dy$ and $dz$ are as given here. Once again a non-uniform cell size would necessitate a more complex calculation here.

The computational domains for $S_S$ and $S_N$, the two components of $S$, are shown in the following diagram.

\[
\begin{align*}
\text{Domain for } S_S & \\
\text{Domain for } S_N
\end{align*}
\]

\{(Calculate \(dx, dy, dz\) for \(S\) array 103) \equiv \}

\[
\begin{align*}
dx &= 2.0 \cdot x \cdot \text{cell.size} \\
dy &= y \cdot \text{cell.size} \\
dz &= 2.0 \cdot z \cdot \text{cell.size}
\end{align*}
\]

This code is used in section 98.

104.

\{(Variables of \textit{main 5}) + \equiv \}

\[
\begin{align*}
\text{real } u_{x.n}, u_{y.n}, u_{z.n}, v_{x.n}, v_{y.n}, v_{z.n}, w_{x.n}, w_{y.n}, w_{z.n}, t_{y.n} & \\
\text{real } u_{x.s}, u_{y.s}, u_{z.s}, v_{x.s}, v_{y.s}, v_{z.s}, w_{x.s}, w_{y.s}, w_{z.s}, t_{y.s}
\end{align*}
\]
§105 — Explode Worker Task —

105.

(Calculate \( u_x \) through \( w_z \) for \( S \) array 105) \( \equiv \)

\[
\begin{align*}
u_{x,n} &= 0.5 \left( \frac{W_{e}(2)}{W_{e}(1)} - \frac{W_{w}(2)}{W_{w}(1)} + \frac{W_{ne}(2)}{W_{ne}(1)} - \frac{W_{nw}(2)}{W_{nw}(1)} \right) / dx \\
u_{y,n} &= \frac{W_{n}(2)/W_{n}(1) - W_{p}(2)/W_{p}(1)}{dy} \\
u_{z,n} &= 0.5 \left( \frac{W_{t}(2)}{W_{t}(1)} - \frac{W_{b}(2)}{W_{b}(1)} + W_{bn}(2)/W_{bn}(1) \right) / dz \\
u_{y,n} &= \frac{W_{n}(3)/W_{n}(1) - W_{p}(3)/W_{p}(1)}{dy} \\
u_{z,n} &= 0.5 \left( \frac{W_{e}(3)}{W_{e}(1)} - \frac{W_{w}(3)}{W_{w}(1)} + \frac{W_{ne}(3)}{W_{ne}(1)} - \frac{W_{nw}(3)}{W_{nw}(1)} \right) / dz \\
u_{y,n} &= \frac{W_{n}(4)/W_{n}(1) - W_{p}(4)/W_{p}(1)}{dy} \\
u_{z,n} &= 0.5 \left( \frac{W_{t}(4)}{W_{t}(1)} - \frac{W_{b}(4)}{W_{b}(1)} + W_{tn}(4)/W_{tn}(1) \right) / dz \\
u_{y,n} &= \frac{(Temp_{n} - Temp_{p})}{dy} \\
u_{x,s} &= 0.5 \left( \frac{W_{se}(2)}{W_{se}(1)} - \frac{W_{sw}(2)}{W_{sw}(1)} + \frac{W_{e}(2)}{W_{e}(1)} - \frac{W_{w}(2)}{W_{w}(1)} \right) / dx \\
u_{y,s} &= \frac{W_{p}(2)/W_{p}(1) - W_{s}(2)/W_{s}(1)}{dy} \\
u_{z,s} &= 0.5 \left( \frac{W_{ts}(2)}{W_{ts}(1)} - \frac{W_{bs}(2)}{W_{bs}(1)} + W_{ts}(2)/W_{ts}(1) \right) / dz \\
u_{y,s} &= \frac{W_{p}(3)/W_{p}(1) - W_{s}(3)/W_{s}(1)}{dy} \\
u_{z,s} &= 0.5 \left( \frac{W_{se}(3)}{W_{se}(1)} - \frac{W_{sw}(3)}{W_{sw}(1)} + \frac{W_{e}(3)}{W_{e}(1)} - \frac{W_{w}(3)}{W_{w}(1)} \right) / dx \\
u_{y,s} &= \frac{W_{p}(4)/W_{p}(1) - W_{s}(4)/W_{s}(1)}{dy} \\
u_{z,s} &= 0.5 \left( \frac{W_{ts}(3)}{W_{ts}(1)} - \frac{W_{bs}(3)}{W_{bs}(1)} + W_{ts}(3)/W_{ts}(1) \right) / dz \\
u_{y,s} &= \frac{W_{p}(4)/W_{p}(1) - W_{s}(4)/W_{s}(1)}{dy} \\
u_{z,s} &= 0.5 \left( \frac{W_{ts}(4)}{W_{ts}(1)} - \frac{W_{bs}(4)}{W_{bs}(1)} + W_{ts}(4)/W_{ts}(1) \right) / dz \\
u_{y,s} &= \frac{(Temp_{p} - Temp_{s})}{dy} \\
This code is used in section 98.

106.

(Variables of main 5) \( \equiv \)

real \( A.n, A.s, S.p.n(10), S.p.s(10) \)
107.  

\[ \text{\{Fill the } S \text{ array 107\}} \equiv \]

\[ A_{n} = z_{\text{cell.size}} \times x_{\text{cell.size}} \]

\[ S_{p.n}(1) = 0.0 \]

\[ S_{p.n}(2) = A_{n} \times \mu_{\text{eff.n}} \times (v_{y.n} + v_{z.n}) \]

\[ S_{p.n}(3) = A_{n} \times (-2.0/3.0 \times (0.5 \times (W_{.p}(9) + W_{.n}(9)) + \mu_{\text{eff.n}} \times (u_{x.n} + v_{y.n} + w_{z.n})) + 2.0 \times \mu_{\text{eff.n}} \times v_{y.n}) \]

\[ S_{p.n}(4) = A_{n} \times \mu_{\text{eff.n}} \times (v_{z.n} + w_{y.n}) \]

\[ S_{p.n}(5) = S_{p.n}(2) \times (W_{.p}(2) + W_{.n}(2)) / (W_{.p}(1) + W_{.n}(1)) + S_{p.n}(3) \times (W_{.p}(3) + W_{.n}(3)) / (W_{.p}(1) + W_{.n}(1)) + \mu_{h.n} \times \tau_{n} \times A_{n} \]

\[ \text{do index} = 6, 10 \]

\[ S_{p.n}(\text{index}) = \mu_{\text{eff.n}} \times A_{n} / K_{\sigma(index)} \times (W_{.n}(\text{index}) / W_{.n}(1) - W_{.p}(\text{index}) / W_{.p}(1)) / dy \]

\[ \text{enddo} \]

\[ A.s = z_{\text{cell.size}} \times x_{\text{cell.size}} \]

\[ S_{p.s}(1) = 0.0 \]

\[ S_{p.s}(2) = A.s \times \mu_{\text{eff.s}} \times (v_{y.s} + v_{z.s}) \]

\[ S_{p.s}(3) = A.s \times (-2.0/3.0 \times (0.5 \times (W_{.p}(9) + W_{.s}(9)) + \mu_{\text{eff.s}} \times (u_{x.s} + v_{y.s} + w_{z.s})) + 2.0 \times \mu_{\text{eff.s}} \times v_{y.s}) \]

\[ S_{p.s}(4) = A.s \times \mu_{\text{eff.s}} \times (v_{z.s} + v_{w.s}) \]

\[ S_{p.s}(5) = S_{p.s}(2) \times (W_{.p}(2) + W_{.s}(2)) / (W_{.p}(1) + W_{.s}(1)) + S_{p.s}(3) \times (W_{.p}(3) + W_{.s}(3)) / (W_{.p}(1) + W_{.s}(1)) + \mu_{h.s} \times \tau_{s} \times A.s \]

\[ \text{do index} = 6, 10 \]

\[ S_{p.s}(\text{index}) = \mu_{\text{eff.s}} \times A.s / K_{\sigma(index)} \times (W_{.s}(\text{index}) / W_{.s}(1) - W_{.p}(\text{index}) / W_{.p}(1)) / dy \]

\[ \text{enddo} \]

\[ \text{do index} = 1, 10 \]

\[ ARST.p(index) += S_{p.n}(\text{index}) - S_{p.s}(\text{index}) \]

\[ \text{enddo} \]

This code is used in section 98.

108.  

Finally, the \( z \) components of the viscid flux \( T \) are determined.

\( \text{\{Calculate } T \text{ array 108\}} \equiv \)

\( \text{\{Calculate } \mu_{\text{eff}} \text{ for } T \text{ array 109\}} \)

\( \text{\{Calculate } dx, dy, dz \text{ for } T \text{ array 113\}} \)

\( \text{\{Calculate } u_{x} \text{ through } w_{z} \text{ for } T \text{ array 115\}} \)

\( \text{\{Fill the } T \text{ array 117\}} \)

This code is used in section 70.
§109 — Explode Worker Task —

109.

(Calculate \( \mu_{\text{eff}} \) for \( T \) array 109) \( \equiv \)

\[
\begin{align*}
\text{(Calculate } Temp_t, cp_t \text{ 111)} & \quad \text{(Calculate } Temp_b, cp_b \text{ 112)} \\
mu_l &= K_{145} \times 0.5 \times (Temp_p + Temp_t) K_{gindex} (0.5 \times (Temp_p + Temp_t) + K_{110}) \\
mu_t &= K_0 \times 2 \times (W_p(9) + W_t(9)) (W_p(10) + W_t(10)) \\
mu_h_t &= 0.5 \times (cp_p + cp_t) \times (mu_l/K_{prn} + mu_t/K_{prnt}) \\
mu_{\text{eff} t} &= mu_l + mu_t \\
mu_b &= K_{110} \times 0.5 \times (Temp_p + Temp_b) K_{gindex} (0.5 \times (Temp_p + Temp_b) + K_{110}) \\
mu_b &= K_0 \times 2 \times (W_p(9) + W_b(9)) (W_p(10) + W_b(10)) \\
mu_h_b &= 0.5 \times (cp_p + cp_b) \times (mu_l/K_{prn} + mu_t/K_{prnt}) \\
mu_{\text{eff} b} &= mu_l + mu_t
\end{align*}
\]

This code is used in section 108.

110.

(Variables of main 5) \( + \equiv \)

\[
\begin{align*}
\text{real } Temp_t, cp_t, Temp_b, cp_b
\end{align*}
\]

111.

(Calculate \( Temp_t, cp_t \) 111) \( \equiv \)

\[
\begin{align*}
Temp_t &= W_t(11) \times W_t(12) / (W_t(1) \times K_{ru}) \\
Temp_{\text{avg}} &= 0.5 \times (Temp_t + K_{tref}) \\
cp_t &= 1000.0 \times (W_t(6) / W_t(1) \times (K_{cpa}(1) + K_{cpb}(1) \times Temp_{\text{avg}}) + K_{stoi} \times W_t(13) \times (K_{cpa}(2) + K_{cpb}(2) \times Temp_{\text{avg}}) + W_t(23) \times (K_{cpa}(3) + K_{cpb}(3) \times Temp_{\text{avg}}) + W_t(24) \times CO2.m \times (K_{cpa}(4) + K_{cpb}(4) \times Temp_{\text{avg}}) + W_t(24) \times (1 - CO2.m) \times (K_{cpa}(5) + k_{cpb}(5) \times Temp_{\text{avg}}))
\end{align*}
\]

This code is used in section 109.
112.\textsuperscript{§} Explode Worker Task

\(\text{Temp}_b = W.b(11) \times W.b(12)/(W.b(1) \times K.ru)\)

\(\text{Temp}_{\text{avg}} = 0.5 \times (\text{Temp}_b + K_{\text{ref}})\)

\(\text{cp}_b = 1000.0 \times (W.b(6)/W.b(1) \times (K_{\text{cpa}(1)} + K_{\text{cpb}(1)} \times \text{Temp}_{\text{avg}}) + K_{\text{stoi}} \times W.b(13) \times (K_{\text{cpa}(2)} + K_{\text{cpb}(2)} \times \text{Temp}_{\text{avg}}) + W.b(23) \times (K_{\text{cpa}(3)} + K_{\text{cpb}(3)} \times \text{Temp}_{\text{avg}}) + W.b(24) \times CO_2.m \times (K_{\text{cpa}(4)} + K_{\text{cpb}(4)} \times \text{Temp}_{\text{avg}}) + W.b(24) \times (1 - CO_2.m) \times (K_{\text{cpa}(5)} + k_{\text{cpb}(5)} \times \text{Temp}_{\text{avg}}))\)

This code is used in section 109.

113. The \(T\) fluxes are calculated in a \(3 \times 3 \times 2\) region thus the centre to centre distances \(dx\), \(dy\) and \(dz\) are as given here. Once again a non-uniform cell size would necessitate a more complex calculation here.

The computational domains for \(T_B\) and \(T_T\), the two components of \(T\), are shown in the following diagram.

\(\text{(Calculate } dx, dy, dz \text{ for } T \text{ array 113}) \equiv\)

\(dx = 2.0 \times x_{\text{cell.size}}\)

\(dy = 2.0 \times y_{\text{cell.size}}\)

\(dz = z_{\text{cell.size}}\)

This code is used in section 108.
\textbf{114.}

\textit{(Variables of main 5) +Ξ}
\begin{itemize}
  \item \textbf{real} $ux.t$, $uy.t$, $uz.t$, $vx.t$, $vy.t$, $vz.t$, $wx.t$, $wy.t$, $wz.t$, $tz.t$
  \item \textbf{real} $ux.b$, $uy.b$, $uz.b$, $vx.b$, $vy.b$, $vz.b$, $wx.b$, $wy.b$, $wz.b$, $tz.b$
\end{itemize}

\textbf{115.}

\textit{(Calculate ux through wz for $T$ array 115) =}
\begin{align*}
ux.t &= 0.5 \times (W_e(2)/W_e(1) - W_w(2)/W_w(1) + W.te(2)/W.te(1) - W.tw(2)/W.tw(1))/dx \\
uy.t &= 0.5 \times (W_n(2)/W_n(1) - W.s(2)/W.s(1) + W.tn(2)/W.tn(1) - W.ts(2)/W.ts(1))/dy \\
uz.t &= (W.t(2)/W.t(1) - W.p(2)/W.p(1))/dz \\
vx.t &= 0.5 \times (W_e(3)/W_e(1) - W.w(3)/W.w(1) + W.te(3)/W.te(1) - W.tw(3)/W.tw(1))/dx \\
vx.b &= 0.5 \times (W_b(3)/W_b(1) - W.bw(3)/W.bw(1) + W.e(3)/W.e(1) - W.w(3)/W.w(1))/dz \\
vx.b &= 0.5 \times (W_b(3)/W_b(1) - W.bw(3)/W.bw(1) + W.e(3)/W.e(1) - W.w(3)/W.w(1))/dz \\
vz.b &= (W.p(3)/W.p(1) - W.b(3)/W.b(1))/dz \\
tz.b &= (Temp.p - Temp.b)/dz
\end{align*}

This code is used in section 108.

\textbf{116.}

\textit{(Variables of main 5) =}
\begin{itemize}
  \item \textbf{real} $A.t$, $A.b$, $T_.p.t(10)$, $T_.p.b(10)$
\end{itemize}
§117 — Explode Worker Task —

117.

(Fill the T array 117) ≡

\[ A_t = x_{cell.size} \times y_{cell.size} \]

\[ T_p.t(1) = 0.0 \]

\[ T_p.t(2) = A_t \times \mu_{eff.t} \times (u_t + w_t) \]

\[ T_p.t(3) = A_t \times \mu_{eff.t} \times (v_t + w_t) \]

\[ T_p.t(4) = A_t \times (-2.0/3.0 \times (0.5 \times (W_p(9) + W_t(9)) + \mu_{eff.t} \times (u_t + v_t + w_t)) + 2.0 \times \mu_{eff.t} \times w_t) \]

\[ T_p.t(5) = T_p.t(2) \times (W_p(2) + W_t(2)) / (W_p(1) + W_t(1)) + T_p.t(3) \times (W_p(3) + W_t(3)) / (W_p(1) + W_t(1)) + T_p.t(4) \times (W_p(4) + W_t(4)) / (W_p(1) + W_t(1)) + \mu_h.t \times t_t \times A_t \]

\[ \text{do index} = 6,10 \]

\[ T_p.t(index) = \mu_{eff.t} \times A_t / K_{sigma(index)} \times (W_t(index) / W_t(1) - W_p(index) / W_p(1)) / dz \]

\[ \text{endo} \]

\[ A_b = x_{cell.size} \times y_{cell.size} \]

\[ T_p.b(1) = 0.0 \]

\[ T_p.b(2) = A_b \times \mu_{eff.b} \times (u_b + w_b) \]

\[ T_p.b(3) = A_b \times \mu_{eff.b} \times (v_b + w_b) \]

\[ T_p.b(4) = A_b \times (-2.0/3.0 \times (0.5 \times (W_p(9) + W_b(9)) + \mu_{eff.b} \times (u_b + v_b + w_b)) + 2.0 \times \mu_{eff.b} \times w_b) \]

\[ T_p.b(5) = T_p.b(2) \times (W_p(2) + W_b(2)) / (W_p(1) + W_b(1)) + T_p.b(3) \times (W_p(3) + W_b(3)) / (W_p(1) + W_b(1)) + T_p.b(4) \times (W_p(4) + W_b(4)) / (W_p(1) + W_b(1)) + \mu_h.b \times t_z \times A_b \]

\[ \text{do index} = 6,10 \]

\[ T_p.b(index) = \mu_{eff.b} \times A_b / K_{sigma(index)} \times (W_b(index) / W_b(1) - W_p(index) / W_p(1)) / dz \]

\[ \text{endo} \]

\[ \text{do index} = 1,10 \]

\[ ARST_p(index) += T_p.t(index) - T_p.b(index) \]

\[ \text{endo} \]

This code is used in section 108.
118. **INVISCID FLUXES.** The three inviscid flux components $F$, $G$ and $H$ in the $x$, $y$, and $z$ directions are now calculated. As with the previous flux components, values from the six orthogonal neighbour cells are required for these calculations.

The computational domain for the inviscid flux calculation is shown in the following diagram.

![Diagram of computational domain for $F$, $G$ and $H$](image)

The macro $FGH$ is declared to reflect the storage location of the accumulated inviscid fluxes in the $W$ array.

```plaintext
define $FGH.p(i) W(44 + i, \text{cell})$
```

(Do inviscid fluxes 118) \equiv

(Do cell = 1, n_of_cells)

(Fill near arrays 39)

(Calculate $F$ components 119)

(Calculate $G$ components 120)

(Calculate $H$ components 121)

endo

(Send done 161)

This code is used in section 24.
119. The x components of the inviscid flux are calculated in this section.

As in the preceding cases the flux components are expressed as the difference between the outward and inward fluxes $F^+$ and $F^-$.

The general expression for the outward fluxes $F_i^+$ is as follows

$$F_i^+ = A^+ \Phi_i^{+\star}, \quad i = 1, 2, \ldots, 10$$

where

$$A^+ = -\Delta y \Delta z u^{+\star}, \quad \Phi_i^{+\star} = \frac{1}{2} (\Phi_i + \Phi_i^+) \quad \text{and} \quad u^{+\star} = \frac{1}{2} (u + u^+).$$

The outward flux for $u$, $F_2^+$ is given by

$$F_2^+ = A^+ \Phi_i^{+\star} - P^{+\star} \Delta y \Delta z$$

where

$$P^{+\star} = \frac{1}{2} (P + P^+).$$

The outward flux for $h$, $F_5^+$ is given by

$$F_5^+ = A^+ \left( h^{+\star} + P^{+\star} \right)$$

where

$$h^{+\star} = \frac{1}{2} (h + h^+)$$

The inward fluxes $F_i^-$ are calculated in a similar manner.
§119 — Explode Worker Task —

{Calculate $F$ components 119} \equiv

\begin{align*}
A_e &= -y_{\text{cell size}} \times z_{\text{cell size}} \times 0.5 \times (W_e(2)/W_e(1) + W_p(2)/W_p(1)) \\
A_w &= -y_{\text{cell size}} \times z_{\text{cell size}} \times 0.5 \times (W_p(2)/W_p(1) + W_w(2)/W_w(1)) \\
\text{do index } &= 1,10 \\
FGH_p(\text{index}) &= A_e \times 0.5 \times (W_p(\text{index}) + W_e(\text{index})) - A_w \times 0.5 \times (W_w(\text{index}) + W_p(\text{index})) \\
\text{enddo} \\
FGH_p(2) &= 0.5 \times (W_p(11) + W_e(11)) \times y_{\text{cell size}} \times z_{\text{cell size}} - 0.5 \times (W_w(11) + W_p(11)) \times y_{\text{cell size}} \times z_{\text{cell size}} \\
FGH_p(5) &= A_e \times 0.5 \times (W_p(11) + W_e(11)) - A_w \times 0.5 \times (W_w(11) + W_p(11))
\end{align*}

This code is used in section 118.

120. The $y$ components of the viscid flux $G$ are calculated from the outward and inward fluxes in a similar fashion to $F$.

{Calculate $G$ components 120} \equiv

\begin{align*}
A_n &= -z_{\text{cell size}} \times x_{\text{cell size}} \times 0.5 \times (W_n(3)/W_n(1) + W_p(3)/W_p(1)) \\
A_s &= -z_{\text{cell size}} \times x_{\text{cell size}} \times 0.5 \times (W_p(3)/W_p(1) + W_s(3)/W_s(1)) \\
\text{do index } &= 1,10 \\
FGH_p(\text{index}) &= A_n \times 0.5 \times (W_p(\text{index}) + W_n(\text{index})) - A_s \times 0.5 \times (W_s(\text{index}) + W_p(\text{index})) \\
\text{enddo} \\
FGH_p(3) &= 0.5 \times (W_p(11) + W_n(11)) \times z_{\text{cell size}} \times x_{\text{cell size}} - 0.5 \times (W_s(11) + W_p(11)) \times z_{\text{cell size}} \times x_{\text{cell size}} \\
FGH_p(5) &= A_n \times 0.5 \times (W_p(11) + W_n(11)) - A_s \times 0.5 \times (W_s(11) + W_p(11))
\end{align*}

This code is used in section 118.
§121 — Explode Worker Task —  

121. The $z$ components of the viscid flux $H$ are calculated from the outward and inward fluxes in a similar fashion to $F$ and $G$.

\[
\begin{align*}
\text{(Calculate } H \text{ components 121)} & \equiv \\
A_t &= -x_{\text{cell size}} \cdot y_{\text{cell size}} \cdot 0.5 \cdot (W_t(4)/W_t(1) + W_p(4)/W_p(1)) \\
A_b &= -x_{\text{cell size}} \cdot y_{\text{cell size}} \cdot 0.5 \cdot (W_p(4)/W_p(1) + W_b(4)/W_b(1)) \\
\text{do index } &= 1, 10 \\
FGH_p(index) &=+ A_t \cdot 0.5 \cdot (W_p(index) + W_t(index)) - A_b \cdot 0.5 \cdot (W_b(index) + W_p(index)) \\
\text{endo} \\
FGH_p(4) &= 0.5 \cdot (W_p(11) + W_t(11)) \cdot x_{\text{cell size}} \cdot y_{\text{cell size}} - 0.5 \cdot (W_b(11) + W_p(11)) \\
FGH_p(5) &= A_t \cdot 0.5 \cdot (W_p(11) + W_t(11)) - A_b \cdot 0.5 \cdot (W_b(11) + W_p(11))
\end{align*}
\]

This code is used in section 118.
§122 — Explode Worker Task — UPDATE PHASE 210

122. UPDATE PHASE. Once all of the various flux components have been calculated the Runge-Kutta update step can be invoked to calculate the updated \( \Phi \) values.

The variable \( \text{flux} \) will be used to calculate the total flux, \( \psi \) through each cell in turn.

\( \text{Variables of } \text{main 5} \) \( \equiv \)

\[
\text{real } \text{flux}
\]

123. The total flux, \( \psi \) through the cell is given by

\[
\psi = \Delta t \left( \frac{FGH + RST + ART}{V} + C \right)
\]

where \( FGH \) are the inviscid fluxes, \( RST \) are the viscid fluxes, \( ART \) are the artificial viscosity terms and \( C \) are the source terms.

The updated \( \Phi \) values are given by

\[
\Phi^* = \Phi + \alpha_s \psi
\]

where \( \alpha_s \) is the Runge-Kutta coefficient for stage \( s \).

Once the \( \Phi^* \) values have been calculated for a given cell they must be checked to ensure that they remain within appropriate value ranges.

\( \text{Do update arrays 123} \) \( \equiv \)

\( \text{(Get synch 160)} \)

\( \text{do } cell = 1, n_{\text{of cells}} \)

\( \text{do index = 1, 10} \)

\[
\text{flux} = \text{timestep} \times (FGH.p(index) + ARST.p(index))/\text{cell volume} + \text{timestep} \times C.p(index)
\]

\[
W.p(index) = W._old(index) + K.alpha(stage) \times \text{flux}
\]

(\( \text{Check minimum values 124} \))

\( \text{enddo} \)

\( \text{enddo} \)

(\( \text{Send done 161} \))

This code is used in section 24.
Apart from the velocity components $u, v$ and $w$, the other $\Phi$ values are constrained such that $\Phi^*_i > 0$ with the tighter constraint $\Phi^*_i \geq 10^{-10}$ being applied to $m_{ju}, k$ and $\epsilon$.

(Check minimum values 124) ≡

if $(\text{index} < 2 \lor \text{index} > 4)$ then
  if $(\text{index} > 8)$ then
    if $(W_p(\text{index}) < 1 \cdot 10^{-10})$ then
      $W_p(\text{index}) = W_{old}(\text{index})$
    endif
  endif
elseif $(\text{index} \equiv 6)$ then
  if $(W_p(\text{index}) < 1 \cdot 10^{-10})$ then
    $W_p(\text{index}) = 1 \cdot 10^{-10}$
  endif
else
  if $(W_p(\text{index}) < 0.0)$ then
    $W_p(\text{index}) = 0.0$
  endif
endif
endif

This code is used in section 123.
125. **FILL ARRAY.** The *fill_array* subroutine is called by the calculation thread of the worker task whenever the values of a neighbouring cell are required. Three basic cases exist within *fill_array*.

i. The neighbouring cell is simply connected to the current cell and lies within the domain of the present worker task.

ii. The neighbouring cell is simply connected to the current cell and lies within the domain of another worker task.

iii. A boundary lies to the side of the current cell for which neighbour information is desired.

\[\text{\textit{Fill subroutine 125}}\] =

```
subroutine fill_array*(W_in , W_out, direction, from_buffer, to_switch, timestep, K_tau_1, K_tau_2, K_hi,
K_gamma, K_stoi, N2_infty, beta_infty, beta_zero, Fuel_molwt, O2_molwt, N2_molwt, Prod_molwt,
K_vent, W_at, W)

implicit none

(Include files 4)
(Other common data 9)
(Common semaphore 6)
(Variables of fill_array*, 126)

call f77_thread_deschedule
index = W_in (direction)

if (index > 0) then  /* A REAL CELL LIES IN THIS DIRECTION */
  worker = index / MAX_CELL

  if (worker == my_worker_id) then  /* CASE I. THE CELL IS LOCAL */
    (Local W_out code 127)
  else  /* CASE II. THE CELL IS REMOTE */
    (Remote W_out code 129)
  endif
else  /* CASE III. THERE IS A BOUNDARY RATHER THAN A CELL */
  (Boundary W_out code 131)
endif
return
end
```

This code is used in section 2.
126. Explode Worker Task

\( \text{Variables of } \text{fillarray}_{125\ 126} \equiv \)

- **real** \( W_{in}(\ast), W_{out}(\ast), W(64,\ast) \)
- **integer** \( \text{direction} \)
- **integer** \( \text{celLout}, \text{worker}, \text{index}, \text{index} \)
- **integer** \( \text{from}_\text{buffer}, \text{to}_\text{switch} \)
- **real** \( \text{timestep}, K_{\tau a u\ 1}, K_{\tau a u\ 2}, W_{at}(15), K_{hc}, K_{gamma}, K_{stoi}, N2_{infty}, beta_{infty}, beta_{zero}, FueL_{molwt}, O2_{molwt}, N2_{molwt}, Prod_{molwt}, K_{vent} \)

See also sections 128 and 130.

This code is used in section 125.

127. In the case where an actual neighbour cell exists and is local to this copy of the worker task all that is required is to copy the appropriate values to \( W_{out} \).

\( \text{Local } W_{out} \text{ code } 127 \equiv \)

- \( \text{celLout} = \text{mod}(\text{index}, \text{MAX\_CELL}) \)
- do \( \text{index} = 1, 24 \)
  - \( W_{out}(\text{index}) = W(\text{index}, \text{celLout}) \)
- enddo

This code is used in sections 125 and 132.

128.

\( \text{Variables of } \text{fillarray}_{125\ 126} \equiv \)

- **integer** \( \text{packet}\_\text{size}, \text{packet}_\text{type}, \text{header}(2) \)
- **real** \( \text{packet}(100) \)
§129. If an actual neighbour cell exists and is not local to this copy of the worker task a packet of type `data.request.type` is assembled containing the identity of the originating worker task and the identity of the cell for which data are required.

This packet is sent to the processor ring via the switch thread and the resulting filled packet is received from the ring via the buffer thread.

On receipt of this filled packet the data are unpacked into `W.out`.

(Variables of fillarray 125 126) \(\equiv\)

\[\begin{align*}
\text{packet.size} &= 26 \\
\text{packet.type} &= \text{data.request.type} \\
\text{header}(1) &= \text{packet.size} \\
\text{header}(2) &= \text{packet.type} \\
\text{packet}(1) &= \text{my.worker.id} \times \text{MAX.CELL} \\
\text{packet}(2) &= \text{abs}(W.in(direction)) \\
\text{(Send packet to switch 163)} \\
\text{(Get packet from buffer 162)} \\
\text{do index} &= 1, 24 \\
\text{W.out(index)} &= \text{packet(index + 2)} \\
\end{align*}\]

This code is used in sections 125 and 132.

130.

(Variables of fillarray 125 126) \(\equiv\)

\[\begin{align*}
\text{real } \alpha_1, \alpha_2
\end{align*}\]
131. The boundary code must address four basic cases.

i. A rigid boundary where a zero slip condition holds.

ii. An open boundary where inflow and outflow to atmosphere occurs.

iii. A rigid boundary where flow parallel to the boundary is allowed. This condition may also be used to model planes of symmetry within a model.

iv. A rigid, zero slip boundary where $k$ and $\epsilon$ damping occur. This is arguably a more realistic representation of case i.

A further complication is added by the existence of vent boundaries, rigid walls which will be destroyed if the pressure differential across the boundary exceeds some maximum value. These boundaries act as zero slip walls up to the point of failure and thereafter behave as for a normal cell to cell neighbour.

```plaintext
(Boundary W.out code 131) \equiv

if (index \leq -1000) then /* VENT CONDITION */
  (Handle vented boundary 132)
endif

if (index \equiv -1) then /* ZERO SLIP WALL CONDITION */
  (Handle zero slip boundary 133)
elseif (index \equiv -2) then /* INFLOW/OUTFLOW CONDITION */
  (Handle inflow outflow boundary 134)
elseif (index \equiv -3) then /* NON-ZERO SLIP WALL CONDITION */
  (Handle non-zero slip boundary 135)
elseif (index \equiv -4) then /* DAMPED ZERO SLIP WALL CONDITION */
  (Handle damped boundary 136)
endif
```

This code is used in section 125.
132. For vented boundaries the neighbour index value is the index of the cell on the other side of the boundary. The values for this neighbour must be obtained in order to determine whether wall failure has occurred. If this is the case the appropriate entry in \( W_{in} \) is modified to reflect the change in status. If no failure has occurred the index value is replaced by \(-1\) to force the zero slip boundary code to be invoked.

\[
\text{(Handle vented boundary 132) } \equiv \\
\text{index } = -\text{index} \\
\text{worker } = \text{index} / \text{MAX}_-\text{CELL} \\
\text{if } (\text{worker } \equiv \text{my}_\text{-worker}_\text{id}) \text{ then} \\
\text{(Local } W_{out} \text{ code 127)} \\
\text{else} \\
\text{(Remote } W_{out} \text{ code 129)} \\
\text{endif} \\
\text{if } (\text{abs}(W_{in}(11) - W_{out}(11)) \leq K_{vent}) \text{ then} \\
\text{index } = -1 \\
\text{else} \\
W_{in}(direction) = -W_{in}(direction) \\
\text{endif}
\]

This code is used in section 131.

133. For the zero slip boundary condition no flux is allowed across the boundary. This is achieved by setting \( \Phi_{out} \) to \( \Phi_{in} \) for each element of \( W_{out} \) except for the velocity components where \( \Phi_{out} = -\Phi_{in} \).

\[
\text{(Handle zero slip boundary 133) } \equiv \\
do \text{index } = 1, 24 \\
W_{out}(\text{index}) = W_{in}(\text{index}) \\
\text{enddo} \\
W_{out}(2) = -W_{in}(2) \\
W_{out}(3) = -W_{in}(3) \\
W_{out}(4) = -W_{in}(4)
\]

This code is used in section 131.
The behaviour of a boundary of this type is dependent on whether an inflow or outflow condition currently holds. This may be determined by consideration of the appropriate velocity component within the current cell.

The $\Phi_{\text{out}}$ values are then calculated as linear combinations of the internal and atmospheric conditions, $\Phi_{\text{in}}$ and $\Phi_{\text{at}}$.

(Handle inflow outflow boundary 134) $\equiv$

\[
\text{if } ((\text{mod}(\text{direction}, 2) \equiv 1 \land W_{\text{in}}((\text{direction} - 13)/2) < 0.0) \lor (\text{mod}(\text{direction}, 2) \equiv 0 \land W_{\text{in}}((\text{direction} - 13)/2) > 0.0)) \text{ then}
\]

\[
\begin{align*}
\alpha_1 &= 1 - K_{\tau_1} \\
\alpha_2 &= 1 - K_{\tau_2}
\end{align*}
\]

else

\[
\begin{align*}
\alpha_1 &= K_{\tau_1} \\
\alpha_2 &= K_{\tau_2}
\end{align*}
\]

endif

do index = 1, 13

\[
W_{\text{out}}(\text{index}) = (1.0 - \alpha_1) \cdot W_{\text{in}}(\text{index}) + \alpha_1 \cdot W_{\text{at}}(\text{index})
\]

endo

\[
\begin{align*}
W_{\text{out}}(2) &= W_{\text{in}}(2) \cdot W_{\text{out}}(1)/W_{\text{in}}(1) \\
W_{\text{out}}(3) &= W_{\text{in}}(3) \cdot W_{\text{out}}(1)/W_{\text{in}}(1) \\
W_{\text{out}}(4) &= W_{\text{in}}(4) \cdot W_{\text{out}}(1)/W_{\text{in}}(1) \\
W_{\text{out}}(14) &= W_{\text{in}}(14) \\
W_{\text{out}}(15) &= W_{\text{in}}(15) \\
W_{\text{out}}(16) &= W_{\text{in}}(16) \\
W_{\text{out}}(11) &= (1.0 - \alpha_2) \cdot W_{\text{in}}(11) + \alpha_2 \cdot W_{\text{at}}(11) \\
W_{\text{out}}(5) &= 1/(K_{\gamma} - 1.0) \cdot W_{\text{out}}(11) + 0.5 \cdot (W_{\text{out}}(2)^2 + W_{\text{out}}(3)^2 + W_{\text{out}}(4)^2)/W_{\text{out}}(1) \\
W_{\text{out}}(13) &= (W_{\text{out}}(6)/W_{\text{out}}(1) - beta_{\text{infty}} - W_{\text{out}}(7) \cdot (beta_{\text{zero}} - beta_{\text{infty}})/W_{\text{out}}(1)) \\
W_{\text{out}}(23) &= N_{\text{infty}} \cdot (1 - W_{\text{out}}(7)/W_{\text{out}}(1)) \\
W_{\text{out}}(24) &= \max(0.0, (1.0 - W_{\text{out}}(6)/W_{\text{out}}(1) - K_{\text{stoi}} \cdot W_{\text{out}}(13) - W_{\text{out}}(23))) \\
W_{\text{out}}(12) &= 1.0/(W_{\text{out}}(6)/(W_{\text{out}}(1) \cdot \text{Fuel.molwt}) + K_{\text{stoi}} \cdot W_{\text{out}}(13)/\text{O2.molwt} + \\
& \quad W_{\text{out}}(23)/\text{N2.molwt} + W_{\text{out}}(24)/\text{Prod.molwt})
\end{align*}
\]

This code is used in section 131.
§135 — Explode Worker Task —

135. The non-zero slip rigid boundary condition is essentially identical to the zero slip case except that the velocity components parallel to the boundary are not forced to create zero flux.

\[
\text{(Handle non-zero slip boundary 135) } \equiv \\
\text{do } index = 1, 24 \\
\quad W_{\text{out}}(index) = W_{\text{in}}(index) \\
\text{enddo} \\
W_{\text{out}}((\text{direction} - 13)/2) = -W_{\text{in}}((\text{direction} - 13)/2)
\]

This code is used in section 131.

136. The damped boundary is identical to the zero slip condition except that \(k\) and \(\epsilon\) are forced to a small value, effectively damping turbulence at the boundary.

\[
\text{(Handle damped boundary 136) } \equiv \\
\text{do } index = 1, 24 \\
\quad W_{\text{out}}(index) = W_{\text{in}}(index) \\
\text{enddo} \\
W_{\text{out}}(2) = -W_{\text{in}}(2) \\
W_{\text{out}}(3) = -W_{\text{in}}(3) \\
W_{\text{out}}(4) = -W_{\text{in}}(4) \\
W_{\text{out}}(9) = 0.000001 \\
W_{\text{out}}(10) = 0.000001
\]

This code is used in section 131.
137. BUFFER. The *buffer_thread* subroutine receives all packets from the upstream processor and routes them to either the calculation thread or the switch thread as appropriate.

\[
\text{(Buffer subroutine 137) } \equiv \\
\text{ subroutine } buffer\_thread\,(to\_main, to\_switch, W) \\
\text{ implicit none } \\
\text{ (Include files 4) } \\
\text{ (Variables of buffer\_thread, 138) } \\
\text{ (Other common data 9) } \\
\text{ (Common semaphore 6) } \\
\text{ (Initialize buffer channel addresses 139) } \\
\text{ do while(T) } /* \text{ Do FOREVER */ } \\
\text{ (Get packet from ring 140) } \\
\text{ (Process packet from ring 141) } \\
\text{ call } f77\_thread\_deschedule \\
\text{ enddo } \\
\text{ end } \\
\text{ This code is used in section 2. }
\]

138. Three channels connect the buffer thread to other processes. The *from\_ring* channel corresponds to the external link from this thread to the upstream processor. The *to\_switch* channel connects the buffer and switch threads. The *to\_main* channel connects the buffer and computational threads.

\[
\text{(Variables of buffer\_thread}_{137} \text{ 138) } \equiv \\
\text{ integer } from\_ring, to\_switch, to\_main \\
\text{ integer } packet\_size, packet\_type, header(2) \\
\text{ real } packet(100) \\
\text{ real } W(64, *) \\
\text{ See also sections 150 and 152. } \\
\text{ This code is used in section 137. } 
\]
§139 — Explode Worker Task — BUFFER

139. The value of the \textit{from\_ring} channel is set to the address of the first external link on the worker transputer. The addresses of the other two channels are passed to \textit{buffer\_thread}$_{137}$ from the main thread at startup.

\begin{Verbatim}(Initialize buffer channel addresses 139) \equiv \end{Verbatim}

\begin{Verbatim}from\_ring = f77\_chan\_in\_port(0)\end{Verbatim}

This code is used in section 137.

140. The process by which a packet is received from the ring is identical to the equivalent process in the root task. First the \textit{header} is received and unpacked and then the \textit{packet} of size \textit{packetsize} is received. The message length parameter in \textit{f77\_chan\_in\_message} is in bytes.

\begin{Verbatim}(Get packet from ring 140) \equiv \end{Verbatim}

\begin{Verbatim}call f77\_chan\_in\_message(8, header, from\_ring) \end{Verbatim}

\begin{Verbatim}packetsize = header(1) \end{Verbatim}

\begin{Verbatim}packet\_type = header(2) \end{Verbatim}

\begin{Verbatim}if (packetsize > 0) then \end{Verbatim}

\begin{Verbatim}call f77\_chan\_in\_message(4*packetsize, packet, from\_ring) \end{Verbatim}

\begin{Verbatim}endif \end{Verbatim}

This code is used in section 137.
141. Once a packet has been received from the ring it must be processed. The detail of this processing depends primarily upon the packet type.

\[
\text{(Process packet from ring 141)} \equiv \\
\begin{align*}
\text{if} & \quad (\text{packet	extunderscore type} \equiv \text{worker	extunderscore count	extunderscore type}) \text{ then} \\
& \quad \text{(Process worker	extunderscore count	extunderscore type 142)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{geometry	extunderscore data	extunderscore type}) \text{ then} \\
& \quad \text{(Process geometry	extunderscore data	extunderscore type 143)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{condition	extunderscore data	extunderscore type}) \text{ then} \\
& \quad \text{(Process condition	extunderscore data	extunderscore type 144)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{constant	extunderscore data	extunderscore type}) \text{ then} \\
& \quad \text{(Process constant	extunderscore data	extunderscore type 145)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{synch	extunderscore type}) \text{ then} \\
& \quad \text{(Process synch	extunderscore type 146)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{timestep	extunderscore type}) \text{ then} \\
& \quad \text{(Process timestep	extunderscore type 147)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{timeset	extunderscore type}) \text{ then} \\
& \quad \text{(Process timeset	extunderscore type 148)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{step	extunderscore complete	extunderscore type}) \text{ then} \\
& \quad \text{(Process step	extunderscore complete	extunderscore type 149)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{report	extunderscore type}) \text{ then} \\
& \quad \text{(Process report	extunderscore type 151)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{data	extunderscore request	extunderscore type}) \text{ then} \\
& \quad \text{(Process data	extunderscore request	extunderscore type 154)} \\
\text{elseif} & \quad (\text{packet	extunderscore type} \equiv \text{debug	extunderscore type}) \text{ then} \\
& \quad \text{(Process debug	extunderscore type 156)} \\
\text{endif}
\end{align*}
\]

This code is used in section 137.

142. Packets of type \text{worker	extunderscore count	extunderscore type} are simply sent to the main thread where they will be dealt with.

\[
\text{(Process worker	extunderscore count	extunderscore type 142)} \equiv \\
\begin{align*}
\text{(Send packet to main 166)}
\end{align*}
\]

This code is used in section 141.
143. Packets of type $geometry\_type$ are also sent to the main thread for processing.

\[
\text{(Process } geometry\_type \text{ 143) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.

144. Packets of type $condition\_type$ are also sent to the main thread for processing.

\[
\text{(Process } condition\_type \text{ 144) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.

145. Packets of type $constant\_type$ are also sent to the main thread for processing.

\[
\text{(Process } constant\_type \text{ 145) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.

146. Packets of type $synch\_type$ are also sent to the main thread for processing.

\[
\text{(Process } synch\_type \text{ 146) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.

147. Packets of type $timestep\_type$ are also sent to the main thread for processing.

\[
\text{(Process } timestep\_type \text{ 147) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.

148. Packets of type $timeset\_type$ are also sent to the main thread for processing.

\[
\text{(Process } timeset\_type \text{ 148) } \equiv \\
\text{(Send packet to main 166)}
\]

This code is used in section 141.
149. Packets of type `step_complete_type` are sent to the switch thread where they are forwarded to the root task.

\[
\langle \text{Process } \text{step\_complete\_type } 149 \rangle \equiv \\
\langle \text{Send packet to switch 163} \rangle
\]

This code is used in section 141.

150.

\[
\langle \text{Variables of buffer\_thread}_{137\ 138} \rangle \equiv \\
\langle \text{integer worker} \rangle
\]

151. When a packet of type `report_type` is received the buffer thread will fill the packet if the cell for which results are required is local to the current worker task. In any case the packet, filled or not, is sent on to the switch thread for routing to the root task.

\[
\langle \text{Process } \text{report\_type } 151 \rangle \equiv \\
\text{worker } = \text{packet}(1)/MAX\_CELL \\
\text{if } (\text{worker } \equiv \text{my\_worker\_id}) \text{ then} \\
\text{\ {Fill the result packet 153}} \\
\text{endif} \\
\langle \text{Send packet to switch 163} \rangle
\]

This code is used in section 141.

152.

\[
\langle \text{Variables of buffer\_thread}_{137\ 138} \rangle \equiv \\
\langle \text{integer cell, index} \rangle
\]
§153 — Explode Worker Task —

153. Where the cell for which result values are requested lies on the local processor, the buffer thread copies the appropriate $W$ entries into the packet.

(Fill the result packet 153) \equiv

\[
\text{cell} = \text{mod}(\text{int}(\text{packet}(1)), \text{MAX\_CELL})
\]

\[
\text{do } \text{index} = 1, \text{packet\_size} - 1
\]

\[
\text{packet}(1 + \text{index}) = W_p(\text{index})
\]

\[\text{enddo}\]

This code is used in section 151.

154. The processing of packets of type $\text{data\_request\_type}$ is somewhat more complex. Three cases exist.

i. The data request originated from this worker.

ii. The data request is for cell values local to this server.

iii. Neither of the previous conditions hold.

In the first case the packet is sent to the waiting main thread where it is received from within the \text{fill\_array125} subroutine. In the other two cases the packet is sent on to the switch thread where it will be routed to the processor ring.

In case ii the buffer thread will fill the packet from the appropriate cell before transmission of the packet.

(Process $\text{data\_request\_type}$ 154) \equiv

\[
\text{worker} = \text{packet}(1)/\text{MAX\_CELL}
\]

\[
\text{if } (\text{worker} \equiv \text{my\_worker\_id}) \text{ then}
\]

\[
(\text{Send packet to main 166})
\]

\[
\text{else}
\]

\[
\text{worker} = \text{packet}(2)/\text{MAX\_CELL}
\]

\[
\text{if } (\text{worker} \equiv \text{my\_worker\_id}) \text{ then}
\]

\[
(\text{Fill the request packet 155})
\]

\[\text{endif}\]

\[
(\text{Send packet to switch 163})
\]

\[\text{endif}\]

This code is used in section 141.
§155 — Explode Worker Task —

155. The data request packet is filled with the 24 values contained in the appropriate $W$ array entries.

(Fill the request packet 155) \equiv
\begin{align*}
  cell &= \text{mod}(\text{int}(\text{packet}(2)), \text{MAX_CELL}) \\
  \text{do} & \quad \text{index} = 1, 24 \\
  & \quad \text{packet}(2 + \text{index}) = W.p(\text{index}) \\
\end{align*}
\text{enddo}

This code is used in section 154.

156. In a similar manner to report packets, packets of type debug.type are examined to determine whether the request is targeted at the current worker task, optionally filled and returned via the switch thread to the root task.

(Process debug.type 156) \equiv
\begin{align*}
  \text{worker} &= \text{packet}(1)/\text{MAX_CELL} \\
  & \quad \text{if} \ (\text{worker} \equiv \text{my_worker_id}) \ \text{then} \\
  & \quad \quad \quad \text{(Fill the debug packet 157)} \\
  & \quad \quad \quad \text{endif} \\
  & \quad \quad \quad \text{(Send packet to switch 163)}
\end{align*}

This code is used in section 141.

157. The contents of the debug array are copied into the packet.

(Fill the debug packet 157) \equiv
\begin{align*}
  \text{do} & \quad \text{index} = 1, 18 \\
  & \quad \text{packet}(\text{index}) = \text{debug}(\text{index}) \\
\end{align*}
\text{enddo}

This code is used in section 156.
158. **SWITCH.** The `switch_thread` subroutine monitors the two internal channels connecting it to the buffer and main threads. Whenever a packet is queued to the switch thread from either source, `switch_thread` will be notified via the `f77_alt_wait` procedure which returns an index to the active channel in the variable `source`.

The switch then fetches the queued packet from the appropriate channel and sends it on to the next processor in the ring over the external channel `to_ring`.

(Switch subroutine 158) ≡

```fortran
subroutine switch_thread(from_main, from_buffer)

    implicit none

    (Include files 4)

    integer header(2), packet_size, packet_type
    real packet(100)

    integer source, from_buffer, from_main, to_ring

    to_ring = f77_chan_out_port(1)

    do while(T)
        source = f77_alt_wait(2, from_main, from_buffer)
        if (source ≡ 1) then
            (Get packet from main 164)
        else
            (Get packet from buffer 162)
        endif

        (Send packet to ring 165)
        call f77_thread_deschedule
    enddo
end

This code is used in section 2.
```
§159 — Explode Worker Task — PACKET HANDLING

159. PACKET HANDLING. The following small modules contain the code required for the various low level packet handling operations. They are collected here as they do not readily fit into any of the other sections of the worker task.

160. This code handles the receipt and transmission of the synch_type packets by the main thread. These packets are used to signal start of step to the worker tasks by the root task.

\[
\text{(Get synch 160) } \equiv \\
\quad \text{(Get packet from buffer 162)} \\
\quad \text{(Send packet to switch 163)}
\]

This code is used in sections 37, 53, 70, 118, and 123.

161. This code is invoked whenever a phase of the calculation which requires synchronisation is completed. These packets are used to signal end of step to the root task by the worker tasks.

\[
\text{(Send done 161) } \equiv \\
\quad \text{packet.size} = 0 \\
\quad \text{packet.type} = \text{step.complete.type} \\
\quad \text{header(1)} = \text{packet.size} \\
\quad \text{header(2)} = \text{packet.type} \\
\quad \text{(Send packet to switch 163)}
\]

This code is used in sections 37, 53, 70, 118, and 123.

162. This code is invoked each time a packet is to be received from the buffer thread by the main thread or the switch thread.

\[
\text{(Get packet from buffer 162) } \equiv \\
\quad \text{call f77.chan.in.message(8, header, from.buffer)} \\
\quad \text{packet.size} = \text{header(1)} \\
\quad \text{packet.type} = \text{header(2)} \\
\quad \text{if (packet.size > 0) then} \\
\quad \quad \text{call f77.chan.in.message(packet.size*4, packet, from.buffer)} \\
\quad \text{endif}
\]

This code is used in sections 11, 14, 16, 18, 36, 129, 158, and 160.
163. This code is invoked by both the main thread and the buffer thread whenever a packet is to be sent to the switch thread.

\[\text{(Send packet to switch 163)} \equiv \]
\[
\begin{align*}
\text{call } & \text{f77.sema.wait}(\text{flag}) \\
\text{call } & \text{f77.chan.out.message}(8, \text{header, to.switch}) \\
\text{if } (\text{packet.size} > 0) \text{ then} & \\
\text{call } & \text{f77.chan.out.message}(\text{packet.size*4, packet, to.switch}) \\
\text{endif} & \\
\text{call } & \text{f77.sema.signal}(\text{flag})
\end{align*}
\]

This code is used in sections 11, 14, 16, 18, 36, 129, 149, 151, 154, 156, 160, and 161.

164. This is the code used by the switch thread to receive a packet from the main thread.

\[\text{(Get packet from main 164)} \equiv \]
\[
\begin{align*}
\text{call } & \text{f77.chan.in.message}(8, \text{header, from.main}) \\
\text{packet.size} = & \text{header(1)} \\
\text{packet.type} = & \text{header(2)} \\
\text{if } (\text{packet.size} > 0) \text{ then} & \\
\text{call } & \text{f77.chan.in.message}(\text{packet.size*4, packet, from.main}) \\
\text{endif}
\end{align*}
\]

This code is used in section 158.

165. This code is used by the switch thread to send a packet to the next processor in the ring.

\[\text{(Send packet to ring 165)} \equiv \]
\[
\begin{align*}
\text{call } & \text{f77.chan.out.message}(8, \text{header, to.ring}) \\
\text{if } (\text{packet.size} > 0) \text{ then} & \\
\text{call } & \text{f77.chan.out.message}(\text{packet.size*4, packet, to.ring}) \\
\text{endif}
\end{align*}
\]

This code is used in section 158.
Finally comes the code used by the buffer thread to transmit a packet to the main thread.

(Send packet to main 166) ≡

```plaintext
  call f77_chan_out_message(8, header, to.main)
  if (packet.size > 0) then
    call f77_chan_out_message(packet.size*4, packet, to.main)
  endif
```

This code is used in sections 142, 143, 144, 145, 146, 147, 148, and 154.
### INDEX

The index contains references to all variables and macro definitions in the program.

Underlined module numbers refer to the module in which an indexed item is first referenced.

| A.b:   | 116, 117, 121. |
| A.e:   | 96, 97, 119.   |
| A.n:   | 106, 107, 120. |
| A.s:   | 106, 107, 120. |
| A.t:   | 116, 117, 121. |
| A.w:   | 96, 97, 119.   |
| abs:   | 52, 54, 129, 132. |
| alpha.1 | 130, 134.   |
| alpha.2 | 130, 134.   |
| ARST:  | 69, 97.     |
| ARST.p: | 69, 97, 107, 117, 123. |
| b.switch: | 5, 7.   |
| beta.infty: | 19, 20, 26, 39, 49, 125, 126, 134. |
| beta.zero: | 19, 20, 26, 39, 49, 125, 126, 134. |
| BOTTOM: | 38, 39, 62, 80, 82, 84, 86. |
| buffer: | 5. |
| buffer.thread: | 2, 5, 7, 137, 139. |
| buffer.to.main: | 5, 7. |
| buffer.to.switch: | 5, 7. |
| buffer.work: | 5, 7. |
| C.p:   | 52, 123.     |
| cell:  | 13, 14, 16, 25, 26, 28, 29, 35, 37, 52, 53, 69, 70, 118, 123, 152, 153, 155. |
| cell.out: | 126, 127. |
| cell.volume: | 19, 20, 66, 123. |
| condition.data.type: | 10, 16, 141, 144. |
| condition.set: | 15, 16. |
| constant.data.type: | 10, 18, 141, 145. |
| CO2.m: | 19, 20, 72, 91, 92, 101, 102, 111, 112. |
| CO2.molwt: | 19, 20. |
| cp.b:  | 109, 110, 112. |
| cp.e:  | 89, 90, 91.   |
| cp.s:  | 99, 100, 102. |
| cp.t:  | 109, 110, 111. |
| cp.w:  | 89, 90, 92.   |
| D:     | 50. |
| data.request.type: | 10, 129, 141, 154. |
| debug_common: | 9. |
| debug_type: | 10, 141, 156. |
| destination: | 39. |
| direction: | 39, 125, 126, 129, 132, 134, 135. |
| dt:    | 34, 35.     |
| dx:    | 42, 43, 45, 47, 93, 95, 97, 103, 105, 113, 115. |
| dy:    | 42, 43, 45, 47, 93, 95, 103, 105, 107, 113, 115. |
| dz:    | 42, 43, 45, 47, 93, 95, 103, 105, 113, 115, 117. |
| E.e:   | 65, 66, 69. |
| E.s:   | 65, 66, 69. |
| E.t:   | 65, 66, 69. |
| E.w:   | 65, 66, 69. |
| EAST:  | 38, 39, 57, 75, 76, 79, 80. |
| eddy:  | 27, 28.     |
| exp:   | 49. |
| explwork: | 2. |
| f.size: | 32, 33, 35. |
| f.u:   | 32, 33, 35. |
| f.v:   | 32, 33, 35. |
| f.vel: | 32, 33, 35. |
| f.w:   | 32, 33, 35. |
| FGH:   | 118. |
| FGH.p: | 118, 119, 120, 121, 123. |
| FILL:  | 39, 57, 58, 59, 60, 61, 62, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86. |
| fill.125: | 2, 39, 57, 125, 154. |
| finished: | 13, 14. |
| flag:  | 5, 7, 163.  |
| flux:  | 122, 123.   |
| from:  | 5. |
| from.buffer: | 5, 7, 39, 125, 126, 158, 162. |
| from.main: | 158, 164. |
| from.ring: | 138, 139, 140. |
| Fuel.molwt: | 19, 20, 26, 39, 125, 126, 134. |
| fuel.molwt: | 20. |
| f.x:   | 46, 47, 52. |
| f.y:   | 46, 47, 52. |
| f.z:   | 46, 47, 52. |
| f77.alt.wait: | 158. |
| f77.chan_address: | 7. |
| f77.chan.in_message: | 140, 162, 164. |
| f77.chan.in_port: | 139. |
| f77.chan_init: | 7. |
| f77.chan_out_message: | 163, 165, 166. |
| f77.chan_out_port: | 158. |
| f77.sema_init: | 7. |
| f77.sema_signal: | 163. |
| f77.sema_size: | 6. |
| f77.sema_wait: | 163. |
| f77.thread.deschedule: | 7, 125, 137, 158. |
| f77.thread_notury: | 7. |
| f77.thread_priority: | 7. |
| f77.thread_start: | 7. |
f77.thread.urgent: 7.
geometry.data.type: 10, 14, 141, 143.
griddir: 13.
gx: 46, 47, 52.
gy: 46, 47, 52.
gz: 46, 47, 52.
header: 10, 128, 129, 138, 140, 158, 161, 162, 163, 164, 165, 166.
H2O.molwt: 19, 20.
include: 4.
index: 17, 18, 19, 20, 26, 39, 49, 72, 91, 92, 101, 102, 111, 112, 125, 126, 134.
K.stoi: 19, 20, 26, 39, 49, 72, 91, 92, 101, 102, 111, 112, 125, 126, 134.
K.tau.1: 17, 18, 39, 125, 126, 134.
K.tau.2: 17, 18, 39, 125, 126, 134.
K.tref: 17, 18, 41, 72, 91, 92, 101, 102, 111, 112.
K.xvent: 17, 18, 39, 125, 126, 132.
K.I: 17, 18, 52.
K.2: 17, 18, 52.
mach.1: 30, 31, 35.
main_to_switch: 5, 7.
max: 20, 26, 64, 68, 134.
MAX.CELL: 2, 13, 14, 35, 125, 127, 129, 132, 151, 153, 154, 155, 156.
min: 52, 66.
mod: 14, 127, 134, 153, 155.
mu.eff: 40, 41, 51, 52, 107.
mu.eff.b: 88, 109, 117.
mu.eff.e: 88, 89, 97.
mu.eff.w: 88, 89, 97.
mu.eff.t: 88, 109, 117.
mu.h.t: 88, 109, 117.
mu.h.w: 88, 89, 97.
my_worker.id: 9, 11, 14, 35, 125, 129, 132, 151, 154, 156.
n.of.cells: 13, 14, 16, 25, 26, 28, 29, 37, 53, 70, 118, 123.
n.of.conditions: 15, 16.
n.of.stages: 17, 18, 24.
n.of.workers: 37.
one: 2, 125, 137, 158.
NORTH: 38, 39, 59, 75, 77, 83, 84.
N2.infty: 19, 20, 26, 39, 125, 126, 134.
N2.molwt: 19, 20, 26, 39, 125, 126, 134.
O2.infty: 19, 20.
O2.molwt: 19, 20, 26, 39, 125, 126, 134.
O2.zero: 19, 20.
packet: 10, 11, 14, 16, 18, 36, 128, 129, 138, 140, 151, 153, 154, 155, 156, 157, 158, 162, 163, 164, 165, 166.
priority: 5, 7.
Prod.molwt: 19, 20, 26, 39, 125, 126, 134.
Q.b: 67, 68, 69.
Q.e: 67, 68, 69.
Q.n: 67, 68, 69.
Q.s: 67, 68, 69.
Q.t: 67, 68, 69.
Q.w: 67, 68, 69.
R.p.w: 96, 97.
R.xe: 63, 64, 66, 68.
R.xw: 63, 64, 66, 68.
R.y: 63, 64, 66, 68.
R.y: 63, 64, 66, 68.
R.z: 63, 64, 66, 68.
report.type: 10, 141, 151.
rho.k: 50, 51, 52.
rz: 53, 54.
ry: 53, 54.
rz: 53, 54.
S.sq: 50, 51.
s.u: 40, 41, 52.
semaphore: 6.
sourcc: 39, 158.
SOUTH: 38, 39, 60, 76, 78, 85, 86.
stage: 23, 24, 123.
step.complete.type: 10, 37, 141, 149, 161.
switch.thread: 2, 5, 7, 158.
switch.work: 5, 7.
synch.type: 10, 37, 141, 146, 160.
T.p.t: 116, 117.
tau.ch: 48, 49, 52.
tau.e: 48, 49, 52.
Temp.avg: 71, 72, 91, 92, 101, 102, 111, 112.
Temp.e: 89, 90, 91, 95.
temp.p: 41.
Temp.s: 99, 100, 102, 105.
Temp.t: 109, 110, 111, 115.
Temp.w: 89, 90, 92, 95.
timecell: 34, 35, 36.
timeset.type: 10, 141, 148.
timestep: 34, 35, 36, 39, 66, 123, 125, 126.
timestep.type: 10, 141, 147.
to.main: 137, 138, 166.
to.ring: 158, 165.
to.switch: 5, 7, 39, 125, 126, 137, 138, 163.
TOP: 38, 39, 61, 79, 81, 83, 85.
tz.e: 94, 95, 97.
tz.w: 94, 95, 97.
ty.n: 104, 105, 107.
tz.b: 114, 115, 117.
tz.t: 114, 115, 117.
ux: 44, 45, 51.
ux.b: 114, 115, 117.
ux.e: 94, 95, 97.
x.t: 114, 115, 117.
ux.w: 94, 95, 97.
vy: 44, 45, 51.
vy.b: 114, 115.
vy.e: 94, 95, 97.
vy.t: 114, 115.
vy.w: 94, 95, 97.
uz: 44, 45, 51.
uz.b: 114, 115, 117.
uz.e: 94, 95, 97.
uz.t: 114, 115, 117.
uz.w: 94, 95, 97.
uz: 44, 45, 51.
uz.b: 114, 115, 117.
uz.e: 94, 95, 97.
uz.t: 114, 115, 117.
uz.w: 94, 95, 97.
V.t: 65, 66, 68.
vz: 44, 45, 51.
vz.b: 114, 115.
vz.e: 94, 95, 97.
vz.t: 114, 115.
vz.w: 94, 95, 97.
vy: 44, 45, 51.
vy.b: 114, 115, 117.
vz.e: 94, 95, 97.
vz.t: 114, 115, 117.
vz.w: 94, 95, 97.
vy: 44, 45, 51.
vy.b: 114, 115, 117.
vz.e: 94, 95, 97.
vz.t: 114, 115, 117.
vz.w: 94, 95, 97.
§167 — Explode Worker Task —

INDEX 233

worker_count_type: 10, 11, 141, 142.
wx: 44, 45, 51.
wx_b: 114, 115, 117.
wx_e: 94, 95, 97.
wx_n: 104, 105.
wx_s: 104, 105.
wx_t: 114, 115, 117.
wx_w: 94, 95, 97.
wy: 44, 45, 51.
wy_b: 114, 115, 117.
wy_e: 94, 95.
wy_s: 104, 105, 107.
wy_t: 114, 115, 117.
wy_w: 94, 95.
wx: 44, 45, 51.
wx_b: 114, 115, 117.
wx_e: 94, 95, 97.
wx_n: 104, 105.
wx_s: 104, 105.
wx_t: 114, 115, 117.
wx_w: 94, 95, 97.
w: 13, 126, 138.
w_at: 11, 18, 20, 39, 41, 125, 126, 134.
w_b: 38, 39, 45, 47, 54, 62, 64, 69, 80, 82, 84, 86, 95, 105, 109, 112, 115, 117, 121.
w_e: 73, 80, 95, 115.
w_b: 73, 80, 95, 115.
w_b: 38, 39, 45, 47, 54, 57, 64, 69, 75, 76, 79, 80, 89, 91, 95, 97, 105, 115, 119.
w_f: 55, 62, 64, 69.
w_e: 55, 57, 64, 69.
w_n: 55, 59, 64, 69.
w_s: 55, 60, 64, 69.
w_t: 55, 61, 64, 69.
w_w: 55, 58, 64, 69.
w_m: 125, 126, 129, 132, 133, 134, 135, 136.
w_n: 38, 39, 45, 47, 54, 59, 64, 69, 75, 77, 83, 84, 95, 99, 101, 105, 107, 115, 120.
w_e: 73, 75, 95, 105.
w_nw: 73, 77, 95, 105.
w_old: 25, 123, 124.
w_p: 14, 16, 25, 26, 28, 31, 33, 35, 39, 41, 49, 51, 52, 54, 57, 58, 59, 60, 61, 62, 64, 69, 72, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 89, 95, 97, 99, 105, 107, 109, 115, 117, 119, 120, 121, 123, 124, 153, 155.
w_s: 38, 39, 45, 47, 54, 60, 64, 69, 76, 78, 85, 86, 95, 99, 102, 105, 107, 115, 120.
w_e: 73, 76, 95, 105.
w_sw: 73, 78, 95, 105.
w_t: 38, 39, 45, 47, 54, 61, 64, 69, 79, 81, 83, 85, 95, 105, 109, 111, 115, 117, 121.
w_t_e: 73, 79, 95, 115.
w_t_n: 73, 83, 105, 115.
w_t_s: 73, 85, 105, 115.
w_t_w: 73, 81, 95, 115.
w_e: 38, 39, 45, 47, 54, 58, 64, 69, 77, 78, 81, 82, 89, 92, 95, 97, 105, 115, 119.
WEST: 38, 39, 58, 77, 78, 81, 82.
while: 2, 14, 137, 158.
WORK_SIZE: 5, 7.
worker: 13, 14, 125, 126, 132, 150, 151, 154, 156.
(Boundary $W_{out}$ code 131) Used in section 125.
(Buffer subroutine 137) Used in section 2.
(Calculate $\mu_{eff}$ for $C$ array 41) Used in section 37.
(Calculate $\mu_{eff}$ for $R$ array 89) Used in section 87.
(Calculate $\mu_{eff}$ for $S$ array 99) Used in section 98.
(Calculate $\mu_{eff}$ for $T$ array 109) Used in section 108.
(Calculate $p_k$ 51) Used in section 37.
(Calculate $\tau_{ch}$ and $\tau_e$ 49) Used in section 37.
(Calculate $dx$, $dy$, $dz$ for $C$ array 43) Used in section 37.
(Calculate $dx$, $dy$, $dz$ for $R$ array 93) Used in section 87.
(Calculate $dx$, $dy$, $dz$ for $S$ array 103) Used in section 98.
(Calculate $dx$, $dy$, $dz$ for $T$ array 113) Used in section 108.
(Calculate $fx$ through $gz$ for $C$ array 47) Used in section 37.
(Calculate pressure and mass terms 26) Used in section 24.
(Calculate $ux$ through $wz$ for $C$ array 45) Used in section 37.
(Calculate $ux$ through $wz$ for $R$ array 95) Used in section 87.
(Calculate $ux$ through $wz$ for $S$ array 105) Used in section 98.
(Calculate $ux$ through $wz$ for $T$ array 115) Used in section 108.
(Calculate $ART$ 69) Used in section 53.
(Calculate $C$ array 52) Used in section 37.
(Calculate $E$ terms 66) Used in section 53.
(Calculate $F$ components 119) Used in section 118.
(Calculate $G$ components 120) Used in section 118.
(Calculate $H$ components 121) Used in section 118.
(Calculate $Q$ terms 68) Used in section 53.
(Calculate $R$ array 87) Used in section 70.
(Calculate $S$ array 98) Used in section 70.
(Calculate $Temp_b$, $cp_b$ 112) Used in section 109.
(Calculate $Temp_e$, $cp_e$ 91) Used in section 89.
(Calculate $Temp_n$, $cp_n$ 101) Used in section 99.
(Calculate $Temp_p$, $cp_p$ 72) Used in section 70.
(Calculate $Temp_s$, $cp_s$ 102) Used in section 99.
(Calculate $Temp_t$, $cp_t$ 111) Used in section 109.
(Calculate $Temp_w$, $cp_w$ 92) Used in section 89.
(Calculate $T$ array 108) Used in section 70.
(Calculate $dt$ 35) Used in section 29.
(Calculate $fvel$, $fsize$ 33) Used in section 29.
(Calculate $mach_1$ 31) Used in section 29.
(Calculate $r$ maxima 64) Used in section 53.
(Calculate $r$ terms 54) Used in section 53.
(Check minimum values 124) Used in section 123.
(Common semaphore 6) Used in sections 2, 125, and 137.
(Derive pseudo constants 20) Used in section 2.
(Determine worker id 11) Used in section 2.
(Do artificial viscosity calculation 53) Used in section 24.
(Do inviscid fluxes 118) Used in section 24.
(Do source term calculation 37) Used in section 24.
(Do the main loop 24) Used in section 2.
(Do update arrays 123) Used in section 24.
(Do viscous flux calculation 70) Used in section 24.
(Fill near arrays 39) Used in sections 37, 53, 56, 74, and 118.
(Fill off-element arrays for $ART$ calculation 56) Used in section 53.
(Fill off-element arrays for R, S and T calculations 74) Used in section 70.
(Fill subroutine 125) Used in section 2.
(Fill the debug packet 157) Used in section 156.
(Fill the request packet 155) Used in section 154.
(Fill the result packet 153) Used in section 151.
(Fill the R array 97) Used in section 87.
(Fill the S array 107) Used in section 98.
(Fill the T array 117) Used in section 108.
(Fill W_be 80) Used in section 74.
(Fill W_bn 84) Used in section 74.
(Fill W_bs 86) Used in section 74.
(Fill W_bw 82) Used in section 74.
(Fill W_fb 62) Used in section 56.
(Fill W_fe 57) Used in section 56.
(Fill W_fn 59) Used in section 56.
(Fill W_fs 60) Used in section 56.
(Fill W_ft 61) Used in section 56.
(Fill W_fw 58) Used in section 56.
(Fill W_ne 75) Used in section 74.
(Fill W_nw 77) Used in section 74.
(Fill W_se 76) Used in section 74.
(Fill W_sw 78) Used in section 74.
(Fill W_te 79) Used in section 74.
(Fill W_tn 83) Used in section 74.
(Fill W_ts 85) Used in section 74.
(Fill W_tw 81) Used in section 74.
(Get constant data 18) Used in section 12.
(Get initial data 12) Used in section 2.
(Get packet from buffer 162) Used in sections 11, 14, 16, 18, 36, 129, 158, and 160.
(Get packet from main 164) Used in section 158.
(Get packet from ring 140) Used in section 137.
(Get synch 160) Used in sections 37, 53, 70, 118, and 123.
(Get the condition data 16) Used in section 12.
(Get the geometry data 14) Used in section 12.
(Handle damped boundary 136) Used in section 131.
(Handle inflow outflow boundary 134) Used in section 131.
(Handle non-zero slip boundary 135) Used in section 131.
(Handle vented boundary 132) Used in section 131.
(Handle zero slip boundary 133) Used in section 131.
(Include files 4) Used in sections 2, 125, 137, and 158.
(Initialize buffer channel addresses 139) Used in section 137.
(Local W_out code 127) Used in sections 125 and 132.
(Other common data 9) Used in sections 2, 125, and 137.
(Process packet from ring 141) Used in section 137.
(Process condition_data_type 144) Used in section 141.
(Process constant_data_type 145) Used in section 141.
(Process data_request_type 154) Used in section 141.
(Process debug_type 156) Used in section 141.
(Process geometry_data_type 143) Used in section 141.
(Process report_type 151) Used in section 141.
(Process step_complete_type 149) Used in section 141.
(Process synch_type 146) Used in section 141.
§167 — Explode Worker Task — INDEX 236

(Process timeset_type 148) Used in section 141.
(Process timestep_type 147) Used in section 141.
(Process worker_count_type 142) Used in section 141.
(Remote W.out code 129) Used in sections 125 and 132.
(Send and set timestep 36) Used in section 29.
(Send done 161) Used in sections 37, 53, 70, 118, and 123.
(Send packet to main 166) Used in sections 142, 143, 144, 145, 146, 147, 148, and 154.
(Send packet to ring 165) Used in section 158.
(Send packet to switch 163) Used in sections 11, 14, 16, 18, 36, 129, 149, 151, 154, 156, 160, and 161.
(Set up timestep 29) Used in section 24.
(Start the communication threads 7) Used in section 2.
(Store current W.p to W.old 25) Used in section 24.
(Switch subroutine 158) Used in section 2.
(Variables of buffer_thread 137 138, 150, 152) Used in section 137.
(Variables of fill_array 125 126, 128, 130) Used in section 125.
(Verify k, epsilon 28) Used in section 24.

COMMAND LINE: "C:\BIN\FWEAVE.EXE explwork".
WEB FILE: "explwork.web".
CHANGE FILE: (none).
GLOBAL LANGUAGE: FORTRAN.
INTRODUCTION. The following chapter contains only the listings of those worker task modules for which there are differences between the 2D and 3D code.

As already noted, the explode program consists of two tasks, the root task and the worker task. This chapter details the structure of the worker task.

The worker task consists of three threads or co-routines. The main thread is responsible for the calculation of the simulation results for the model and the other two threads, the switch thread and the buffer thread, are responsible for maintaining the necessary communication between copies of the worker task on each transputer and between the worker and root tasks.

The following diagram shows the relationship between the threads in the worker task and the direction of packet flow between the threads.

\[
\text{FROM PREVIOUS TASK} \\
\downarrow \\
\text{BUFFER THREAD} \\
\downarrow \\
\text{MAIN THREAD} \\
\downarrow \\
\text{SWITCH THREAD} \\
\downarrow \\
\text{TO NEXT TASK}
\]

Thread relationship.
The top level structure of the worker task is shown below.

Following the various data declaration modules the program must first start the two communication threads.

Once this has been achieved it is necessary to determine the worker number for the present task. As a number of messages, both from the root and other worker tasks, must be addressed to a specific worker task this numbering is essential to the operation of the simulation.

Now that the worker task has established its identity the initialisation process proper can begin. This is achieved in two phases.

First the initial data are read in from the root task via the processor ring, these data being the connectivity and initial conditions for each cell as well as a set of global data values.

Secondly a number of model dependent constants are calculated.

The initialisation having been completed the worker task's main thread enters an infinite loop wherein the main calculation stages are accomplished.

The three subroutine declarations consist of fill array, a subroutine used by the main thread to access information from neighbouring cells and buffer thread and switch thread, the communications co-routines.

The parameter MAX CELL specifies the maximum number of cells which may be processed on a given transputer.

define MAX CELL 1000

program explwork_2d

implicit none

(Include files 4)
(Variables of main 5)
(Common semaphore 6)
(Other common data 9)

(Start the communication threads 7)
(Determine worker id 11)
(Get initial data 12)
(Derive pseudo constants 20*)
§2 — Explode Worker Task – 2D Version —

do while(T) /* Do FOREVER */
    (Do the main loop 24)
enddo
end

(Fill subroutine 125)
(Buffer subroutine 137)
(Switch subroutine 158)
The following section sets the value of a number of constants, a mixture of model dependent terms and physical invariants.

These include \( \text{Fuelmolwt} \), the molecular weight of the \( C_{nc}H_{nh} \) hydrocarbon fuel molecule, \( O_2.molwt \), the molecular weight of the \( O_2 \) molecule, \( N_2.molwt \), the molecular weight of the \( N_2 \) molecule, \( CO_2.molwt \), the molecular weight of the \( CO_2 \) molecule, \( H_2O.molwt \), the molecular weight of the \( H_2O \) molecule and \( Prod.molwt \), the mean molecular weight of the combustion products. The combustion model used in this program is a simple, single step process \( C_{nc}H_{nh} + (nc + nh/2)O_2 \rightarrow (nc)CO_2 + (nh/2)H_2O \).

The variable \( CO_2.m \) contains the fraction of \( CO_2 \) by mass in the products.

The \( K_{stoi} \) variable contains the stoichiometric ratio.

The variables \( O_2.infty, \text{FueLinfty}, O_2.zero \) and \( Fuelzero \) contain the mass fractions for oxygen and fuel in pure air \((O_2.infty, \text{FueLinfty})\) and pure fuel \((O_2.zero, Fuelzero)\). These variables along with \( Ninfty, beta_{infty} \) and \( beta_{zero} \) are used in the combustion rate calculation.

The remaining expressions give the pressure, \( W.at(11) \), mean molecular weight \( W.at(12) \), and nitrogen content, \( W.at(14) \), for the environment external to the model space. It should be noted that these \( W.at \) data need not be set to normal atmospheric conditions if the inflow outflow boundary conditions represent flow to and from another section of the mine.

In the case of the 2D version of the program, the variable \( cell.volume \) is in fact the cell area.

\[
(\text{Derive pseudo constants 20*)} = \]
\[
cell\text{volume} = x.cell\text{size} \times y.cell\text{size}
\]
\[
\text{Fuelmolwt} = 12.0 \times K.nc + K.nh
\]
\[
O_2.molwt = 32.0
\]
\[
N_2.molwt = 28.0
\]
\[
CO_2.molwt = 44.0
\]
\[
H_2O.molwt = 18.0
\]
\[
CO_2.m = K.nc \times CO_2.molwt / (K.nc \times CO_2.molwt + 0.5 \times K.nh \times H_2O.molwt)
\]
\[
x.1 = fuel.molwt / (fuel.molwt + (K.nc + 0.25 \times K.nh) \times O_2.molwt + K.ratox \times (K.nc + 0.25 \times K.nh) \times N_2.molwt)
\]
\[
x.2 = (K.nc + 0.25 \times K.nh) \times O_2.molwt / (fuel.molwt + (K.nc + 0.25 \times K.nh) \times O_2.molwt + K.ratox \times (K.nc + 0.25 \times K.nh) \times N_2.molwt)
\]
\[
Prod.molwt = (K.nc \times CO_2.molwt + 0.5 \times K.nh \times H_2O.molwt) / (K.nc + 0.5 \times K.nh)
\]
\[
K_{stoi} = x.2 / x.1
\]
\[
O_2.infty = 0.209 \times O_2.molwt / (0.209 \times O_2.molwt + 0.791 \times N_2.molwt)
\]
\[ O_2_{\text{zero}} = 0.0 \]

\[ \text{Fuel}_{\text{infty}} = 0.0 \]

\[ \text{Fuel}_{\text{zero}} = 1.0 \]

\[ N_2_{\text{infty}} = 1 - O_2_{\text{infty}} \]

\[ \beta_{\text{infty}} = \text{Fuel}_{\text{infty}} - O_2_{\text{infty}} \cdot K_{\text{stoi}} \]

\[ \beta_{\text{zero}} = \text{Fuel}_{\text{zero}} - O_2_{\text{zero}} \cdot K_{\text{stoi}} \]

\[ W_{\text{at}(11)} = (K_{\gamma} - 1.0) \cdot (W_{\text{at}(5)} - 0.5 \cdot (W_{\text{at}(2)}^2 + W_{\text{at}(3)}^2 + W_{\text{at}(4)}^2) / W_{\text{at}(1)}) \]

\[ W_{\text{at}(14)} = N_2_{\text{infty}} \cdot (1 - W_{\text{at}(7)} / W_{\text{at}(1)}) \]

\[ W_{\text{at}(15)} = \max(0.0, (1.0 - W_{\text{at}(6)} / W_{\text{at}(1)} - K_{\text{stoi}} \cdot W_{\text{at}(13)} - W_{\text{at}(14)})) \]

\[ W_{\text{at}(12)} = 1.0 / (W_{\text{at}(6)} / (W_{\text{at}(1)} \cdot \text{Fuel} \cdot \text{molwt}) + K_{\text{stoi}} \cdot W_{\text{at}(13)} / \text{O}_2 \cdot \text{molwt} + W_{\text{at}(14)} / N_2 \cdot \text{molwt} + W_{\text{at}(15)} / \text{Prod} \cdot \text{molwt}) \]

This code is used in section 2*. 
26* PRESSURE. The following sections perform the calculation of cell pressures and mass terms as well as checking the values of \( k \) and \( \epsilon \) for appropriate bounds.

The cell pressure \( P \) is stored in \( \text{W}(11) \) and is derived from the following relation

\[
P = (\gamma - 1)\frac{\rho h - \frac{1}{2} (\rho u)^2 + (\rho v)^2 + (\rho w)^2}{\rho}.
\]

The stoichiometric oxygen mass \( m_{O_2} \), is stored in \( \text{W}(13) \) and is derived as follows

\[
m_{O_2} = m_{F} - \beta_{\infty} - f(\beta_0 - \beta_{\infty}).
\]

The mean molecular weight of the cell contents is stored in \( \text{W}_p(12) \) and is derived from the concentration terms for fuel, \( \text{W}_p(6) \), oxygen, \( \text{W}_p(13) \), nitrogen, \( \text{W}_p(23) \) and reaction products, \( \text{W}_p(24) \).

In the 2D version of the program, the \( z \) velocity term \( \rho w \) is eliminated.

(Calculate pressure and mass terms 26*)

\[
\text{do cell = 1, n.of.cells}
\]
\[
\begin{align*}
\text{W}_p(11) &= (K_{\text{gamma}} - 1.0)((\text{W}_p(5) - 0.5(\text{W}_p(2)^2 + \text{W}_p(3)^2))/\text{W}_p(1)) \\
\text{W}_p(13) &= (\text{W}_p(6)/\text{W}_p(1) - \beta_{\infty} - \text{W}_p(7)*(\beta_{\text{zero}} - \beta_{\infty})/\text{W}_p(1)) \\
\text{W}_p(23) &= N2_{\infty}*(1 - \text{W}_p(7)/\text{W}_p(1)) \\
\text{W}_p(24) &= \max(0.0, 1.0 - \text{W}_p(6)/\text{W}_p(1) - K_{\text{stoi}}*\text{W}_p(13) - \text{W}_p(23)) \\
\text{W}_p(12) &= 1.0/(\text{W}_p(6)/(\text{W}_p(1)*\text{Fuel.molwt}) + K_{\text{stoi}}*(\text{W}_p(13)/\text{O2.molwt} + \text{W}_p(23)/N2.molwt + \text{W}_p(24)/\text{Prod.molwt}))
\end{align*}
\]
\[
\text{enddo}
\]

This code is used in section 24.
These values are calculated as $\text{vel} = \sqrt{u^2 + v^2 + w^2}$ and $\text{size} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$.

In the 2D version of the program $\text{vel} = \sqrt{u^2 + v^2}$ and $\text{size} = \sqrt{\Delta x^2 + \Delta y^2}$.

\[
\text{Calculate } f_{.\text{vel}}, f_{.\text{size}} \quad 33^* \equiv \\
\quad f_{.u} = (W.p(2)/W.p(1)) \\
\quad f_{.v} = (W.p(3)/W.p(1)) \\
\quad f_{.\text{vel}} = \sqrt{(f_{.u}^2 + f_{.v}^2)} \\
\quad f_{.\text{size}} = \sqrt{x_{.\text{cell}.\text{size}}^2 + y_{.\text{cell}.\text{size}}^2}
\]

This code is used in section 29.
The value of the local timestep $dt$ is derived from the expression $\delta t = \frac{CFL \times \text{size}}{\text{vel} \times \text{mach}}$ where CFL is the Courant number.

A number of related values are stored in the debug array for the cell at which the minimum timestep was set. In the event that the minimum timestep is found to be out of range this debugging information will be collected by the root task and printed.

In the 2D version of the program debug(9) and debug(16) which are $f.w$ and $W.p(4)$ are eliminated.

(Calculate $dt$ 35*) $\equiv$

$$dt = \frac{K \times f \_ \text{size}}{(f \_vel + mach \_1)}$$

if ($cell \equiv 1$) then

    timestep = $dt$
    timecell = $MAX\_CELL \times my\_worker\_id + cell$
    debug(1) = timecell
    debug(2) = timestep
    debug(3) = $K \times f \_cfl$
    debug(4) = $f \_size$
    debug(5) = $f \_vel$
    debug(6) = $mach \_1$
    debug(7) = $f \_u$
    debug(8) = $f \_v$
    debug(9) = 0.0
    debug(10) = $x \_cell\_size$
    debug(11) = $y \_cell\_size$
    debug(12) = 0.0
    debug(13) = $W \_p(1)$
    debug(14) = $W \_p(2)$
    debug(15) = $W \_p(3)$
    debug(16) = 0.0
    debug(17) = $W \_p(11)$
    debug(18) = $K \_gamma$

else

    if ($dt < timestep$) then

        timestep = $dt$
        timecell = $MAX\_CELL \times my\_worker\_id + cell$
This code is used in section 29.
The subroutine *fillarray* is invoked six times, once for each of the orthogonal neighbours. This routine fills the array contained in the second argument, *W.direction*, with the values associated with the appropriate neighbour to the current cell *W.p* passed in the first argument.

As the majority of the arguments to *fillarray* are the same for all calls, the macro *FILL* is declared.

The arrays *W.t* and *W.b* are eliminated from the 2D calculation.

```plaintext
define FILL(source, destination, direction) call fillarray(source,destination,direction,from.buffer,
          to.switch,timestep,K.tau.1,K.tau.2,K.hc,K.gamma,K.stoi,N2.infty,beta.infty,beta.zero,
          Fuel.molwt,O2.molwt,N2.molwt,Prod.molwt,K.vent,W.at,W)
```

(Fill near arrays 39*)

```plaintext
FILL(W.p(1), W.e(1), EAST)
FILL(W.p(1), W.w(1), WEST)
FILL(W.p(1), W.n(1), NORTH)
FILL(W.p(1), W.s(1), SOUTH)
```

This code is used in sections 37, 53, 56*, 74*, and 118*.

As fluxes are calculated from cell centres the *dx*, *dy* and *dz* values are set to be the distance from the centre of one neighbouring cell to the centre of the opposite neighbour via the following expression

\[ dx = \Delta x + \frac{1}{2} (\Delta z^- + \Delta z^+) \]

The simpler calculation in this section is based on the fact that all cells are of the same size.

The calculation of *dz* is eliminated from the 2D version of the program.

(Calculate dx, dy, dz for C array 43*)

```plaintext
dx = 2.0*x.cellsize
dy = 2.0*y.cellsize
```

This code is used in section 37.
The nine velocity fluxes $u_x$ through $w_z$ are given by the general equation $\Delta u = \frac{u^+ - u^-}{dx}$ for each of $u$, $v$ and $w$ in each of the three directions.

In the 2D version of the program the variables $u_z$, $v_z$ and the $w$ components are eliminated.

(Calculate $u_x$ through $w_z$ for $C$ array 45*)

\begin{align*}
    u_x &= \left( \frac{W.e(2)}{W.e(1)} - \frac{W.w(2)}{W.w(1)} \right) / dx \\
    u_y &= \left( \frac{W.n(2)}{W.n(1)} - \frac{W.s(2)}{W.s(1)} \right) / dy \\
    v_x &= \left( \frac{W.e(3)}{W.e(1)} - \frac{W.w(3)}{W.w(1)} \right) / dx \\
    v_y &= \left( \frac{W.n(3)}{W.n(1)} - \frac{W.s(3)}{W.s(1)} \right) / dy
\end{align*}

This code is used in section 37.

The mixture fraction fluxes are given by $\Delta f = \frac{f^+ - f^-}{dx}$.

In a similar fashion the variance fluxes are given by $\Delta g = \frac{g^+ - g^-}{dx}$.

The calculation of the $z$ fluxes $f_z$ and $g_z$ is not needed for the 2D code.

(Calculate $f_x$ through $g_z$ for $C$ array 47*)

\begin{align*}
    f_x &= \left( \frac{W.e(7)}{W.e(1)} - \frac{W.w(7)}{W.w(1)} \right) / dx \\
    f_y &= \left( \frac{W.n(7)}{W.n(1)} - \frac{W.s(7)}{W.s(1)} \right) / dy \\
    g_x &= \left( W.e(8) - W.w(8) \right) / dx \\
    g_y &= \left( W.n(8) - W.s(8) \right) / dy
\end{align*}

This code is used in section 37.

The $z$ velocity fluxes, $u_z$, $v_z$, $w_x$, $w_y$ and $w_z$ are not needed for the 2D version of the program.

(Calculate $\rho_k$ 51*)

\begin{align*}
    D &= u_x + v_y \\
    S_sq &= (u_y + v_x)^2 + 2.0*(u_x^2 + v_y^2) - 2.0/3.0*D^2 \\
    rho.k &= mu_eff * S_sq - 2.0/3.0*W.p(9)*D
\end{align*}

This code is used in section 37.
The actual source terms $S_\phi$ may now be calculated. The macro $C_p$ reflects the storage location of the source terms in the $W$ array.

The $z$ velocity component of $S_\phi$, $C_p(4)$ is not calculated in the 2D case.

**define** $C_p(i) = W(24 + i, cell)$

(Calculate $C$ array 52*)

\[
\begin{align*}
C_p(1) &= 0.0 \\
C_p(2) &= gx \\
C_p(3) &= gy \\
C_p(4) &= 0.0 \\
C_p(5) &= (W_p(2)gx + W_p(3)gy)/W_p(1) \\
\end{align*}
\]

if ($\tau_{\text{ch}}/\tau_{\text{e}} < K_{\text{die}}$) then

if ($\mu_{\text{eff}} > 5.0*\mu_{\text{l}}$) then

\[
C_p(6) = -K_{ebu}W_p(1)*W_p(10)*\min(W_p(6)/W_p(1), W_p(13), W_p(24))/W_p(9)
\]

elseif ($W_p(24) > 0.0$) then

\[
C_p(6) = -s_uW_p(1)*\min(W_p(6)/W_p(1), W_p(13), W_p(24))*2.0/(x_{\text{cell size}} + y_{\text{cell size}})
\]

else

\[
C_p(6) = 0.0
\]
endif

else

\[
C_p(6) = 0.0
\]
endif

if ($\abs(C_p(6)) > K_{\text{maxfuel}}$) then

\[
C_p(6) = -K_{\text{maxfuel}}
\]
endif

$C_p(7) = 0.0$

\[
\begin{align*}
C_p(8) &= K_{g1} \mu_{\text{eff}}(fr^2 + fy^2 - K_g2*W_p(8)*W_p(10)/W_p(9)) \\
C_p(9) &= \rho_{\text{k}} - W_p(10) \\
C_p(10) &= W_p(10)/W_p(9)*(\rho_{\text{k}}*K_1 - W_p(10)*K_2) \\
C_p(5) &= -C_p(6)*K_{hc}
\end{align*}
\]

This code is used in section 37.
54. The values for the flux gradient sensors $r_x$, $r_y$ and $r_z$ are derived from the pressure in the cell and its orthogonal neighbours in the following manner. These values are stored in $W(14,15)$ and $W(16)$.

$$\begin{align*}
r_p &= \frac{|P^- + P^+ - 2P|}{|P^- + P^+ + 2P|}
\end{align*}$$

In the 2D case the $z$ sensor $r_z$ is eliminated.

(Calculate $r$ terms 54*)

$$\begin{align*}
W.p(14) &= \frac{abs(W.w(11) + W.e(11) - 2.0*W.p(11))}{abs(W.w(11) + W.e(11) + 2.0*W.p(11))}
W.p(15) &= \frac{abs(W.s(11) + W.n(11) - 2.0*W.p(11))}{abs(W.s(11) + W.n(11) + 2.0*W.p(11))}
\end{align*}$$

This code is used in section 53.

56. The twelve neighbouring data sets required for the artificial viscosity calculation are now assembled.

The far top and far bottom arrays $W.ft$ and $W.fb$ are not used in the 2D code.

(Fill off-element arrays for ART calculation 56*)

(Fill near arrays 39*)
(Fill $W.fe$ 57)
(Fill $W.fw$ 58)
(Fill $W.fn$ 59)
(Fill $W.fs$ 60)

This code is used in section 53.

61. The filling of the far top array is not required in the 2 dimension version of the program.

62. The filling of the far bottom array is not required in the 2 dimension version of the program.
The flux gradient sensors $\tau_P$ for the cell and its neighbours are now used to calculate local maxima $\tau_{xe}$ through $\tau_{zb}$ using the following general expressions.

$$r^- = \max (\tau_{P-}, \tau_{P+}, \tau_{Px}, \tau_{P+}) \quad \text{and} \quad r^+ = \max (\tau_{P-}, \tau_{P+}, \tau_{P+}, \tau_{P+}) .$$

In the 2D code, $\tau_{zt}$ and $\tau_{zb}$ are not used.

(Calculate $r$ maxima 64*) \equiv

- $\tau_{xe} = \max (W_w(14), W_p(14), W_e(14), W_{fe}(14))$
- $\tau_{xw} = \max (W_{fw}(14), W_w(14), W_p(14), W_e(14))$
- $\tau_{yn} = \max (W_s(15), W_p(15), W_n(15), W_{fn}(15))$
- $\tau_{ys} = \max (W_s(15), W_s(15), W_p(15), W_n(15))$

This code is used in section 53.

The artificial viscosity components are formulated in terms of three sets of two orthogonal fluxes, $E$ and $Q$. The following code sections calculate the inward and outward components of these fluxes.

First the six $E$ values are derived from the expression

$$E = \frac{V}{\Delta t} \min(0.5, k_2 r^\pm).$$

The $z$ components of $E$, $E_t$ and $E_b$ are not needed for the 2D case.

(Calculate $E$ terms 66*) \equiv

- $V_t = \text{cell volume/timestep}$
- $E_e = V_t \min(0.5, r_{xe} * K_{k2})$
- $E_w = V_t \min(0.5, r_{xw} * K_{k2})$
- $E_n = V_t \min(0.5, r_{yn} * K_{k2})$
- $E_s = V_t \min(0.5, r_{ys} * K_{k2})$

This code is used in section 53.
Next the $Q$ values are derived from the expression

$$ Q = \frac{V}{\Delta t} \max(0, k_4 k_{oh} r^\pm). $$

The $z$ components of $Q$, $Q_t$ and $Q_b$ are not needed for the 2D case.

(Calculate $Q$ terms 68$^*$) \equiv

$$ Q_e = V_t \max(0.0, K.k_4 - K.\alpha.h.r_xe) $$
$$ Q_w = V_t \max(0.0, K.k_4 - K.\alpha.h.r_xw) $$
$$ Q_n = V_t \max(0.0, K.k_4 - K.\alpha.h.r_yn) $$
$$ Q_s = V_t \max(0.0, K.k_4 - K.\alpha.h.r_ys) $$

This code is used in section 53.
Finally the artificial viscosity terms may be calculated from the $E$ and $Q$ values in the following manner.

\[
ART_i = \sum_{x,y,z} (E\Phi)^*_i - (Q\Phi)^*_i
\]

where

\[
(E\Phi)^*_i = E_i^+(\Phi_i^+ - \Phi_i^-) - E_i^-(\Phi_i^+ - \Phi_i^-)
\]

and

\[
(Q\Phi)^*_i = Q_i^+(\Phi_i^+ - 3\Phi_i^+ + 3\Phi_i^- - \Phi_i^-) - Q_i^-(\Phi_i^+ + 3\Phi_i^- - \Phi_i^-)
\]

The macro $ARST$ is used to indicate the storage locations into which the artificial viscosity and viscous fluxes are stored.

The $z$ components of the artificial viscosity are eliminated from the calculation in the 2D version of the program.

```javascript
define ARST,p(i) W(34+i, cell)
(Calculate ART 69*)

do index = 1,10
  if (index \neq 4) then
    ARST,p(index) = E.e*(W,e(index) - W.p(index)) - E.w*(W.p(index) - W.w(index)) -
    Q.e*(W.fe(index) - 3.0*W.e(index) + 3.0*W.p(index) - W.w(index)) + Q.w*(W.e(index) -
    3.0*W.p(index) + 3.0*W.w(index) - W.fe(index)) + E.n*(W.n(index) - W.p(index)) -
    E.s*(W.p(index) - W.s(index)) - Q.n*(W.fn(index) - 3.0*W.n(index) + 3.0*W.p(index) -
    W.s(index)) + Q.s*(W.n(index) - 3.0*W.p(index) + 3.0*W.s(index) - W.fs(index))
  else
    ARST,p(index) = 0.0
  endif
endo
```

This code is used in section 53.
70* **VISCID FLUXES.** The calculation of the viscid flux terms is also performed within a synchronised block.

Once again, data from neighbouring cells is required for these calculations. The computational domain for the viscid fluxes is shown in the following diagram.

The calculation of the elements of the $T$ array, the $z$ components of the viscid flux is eliminated from the 2D version of the program.

$$\text{(Do viscid flux calculation 70*) } \equiv$$

$$\text{(Get synch 160)}$$

$$\text{do cell = 1, n.of.cells}$$

$$\text{(Calculate Temp.p, cp.p 72)}$$

$$\text{(Fill off-element arrays for R, S and T calculations 74*)}$$

$$\text{(Calculate R array 87)}$$

$$\text{(Calculate S array 98)}$$

$$\text{enddo}$$

$$\text{(Send done 161)}$$

This code is used in section 24.
74* The calculation of the $R$, $S$ and $T$ components of the viscid flux each require data from some of the twelve diagonal cells in addition to the six orthogonal neighbours. The following sections acquire these data.

In the 2D version of the program only four diagonal arrays are required.

\[
\text{\{Fill off-element arrays for } R, S \text{ and } T \text{ calculations 74}\} =
\text{\{Fill near arrays 39\}}
\text{\{Fill } W_{ne} \text{ 75\}}
\text{\{Fill } W_{se} \text{ 76\}}
\text{\{Fill } W_{nw} \text{ 77\}}
\text{\{Fill } W_{sw} \text{ 78\}}
\]

This code is used in section 70*.

79* In the two dimensional version of the model, the code to fill the top and bottom diagonal elements is eliminated.

80* Module removed.

81* Module removed.

82* Module removed.

83* Module removed.

84* Module removed.

85* Module removed.

86* Module removed.
93* The $R$ fluxes are calculated in a $2 \times 3 \times 3$ region thus the centre to centre distances $dx$, $dy$ and $dz$ are as given here. Once again a non-uniform cell size would necessitate a more complex calculation here.

The computational domains for $R_W$ and $R_E$, the two components of $R$, are shown in the following diagram.

The variable $dz$ is not used in the 2D code.

(Calculate $dx$, $dy$, $dz$ for $R$ array 93*)

\[
\begin{align*}
 dx & = x_{\text{cell size}} \\
 dy & = 2.0 \times y_{\text{cell size}}
\end{align*}
\]

This code is used in section 87.
The velocity fluxes $u_{x,e}$ through $w_{z,e}$ and the temperature flux $t_{x,e}$ for the cell and its eastern neighbour are calculated along with the corresponding fluxes for the cell and its western neighbour.

The $x$ component of the velocity fluxes is calculated in a straightforward manner from the expression.

$$ u^x_e = \frac{u_x - u}{dx} $$

The $y$ and $z$ components of the velocity fluxes are calculated as the average of two fluxes, one through the cell and the other through its eastern neighbour via the following expression.

$$ u^y_e = \frac{1}{2} \left( \frac{u_n - u_s}{dy} + \frac{u_{ne} - u_{se}}{dy} \right) $$

The $u_z$ and $v_z$ variables are eliminated in the 2D code along with the six $w$ variables.

(Calculate $u_x$ through $w_z$ for $R$ array 95*)

$$ u_{x,e} = \frac{W_{e}(2)/W_{e}(1) - W_{p}(2)/W_{p}(1)}{dx} $$
$$ u_{y,e} = 0.5*\left( \frac{W_{n}(2)/W_{n}(1) - W_{s}(2)/W_{s}(1) + W_{ne}(2)/W_{ne}(1) - W_{se}(2)/W_{se}(1)}{dy} \right) $$
$$ v_{x,e} = \frac{W_{e}(3)/W_{e}(1) - W_{p}(3)/W_{p}(1)}{dx} $$
$$ v_{y,e} = 0.5*\left( \frac{W_{n}(3)/W_{n}(1) - W_{s}(3)/W_{s}(1) + W_{ne}(3)/W_{ne}(1) - W_{se}(3)/W_{se}(1)}{dy} \right) $$
$$ t_{x,e} = \frac{\text{Temp}_{e} - \text{Temp}_{p}}{dx} $$
$$ u_{x,w} = \frac{W_{p}(2)/W_{p}(1) - W_{w}(2)/W_{w}(1)}{dx} $$
$$ u_{y,w} = 0.5*\left( \frac{W_{nw}(2)/W_{nw}(1) - W_{sw}(2)/W_{sw}(1) + W_{n}(2)/W_{n}(1) - W_{s}(2)/W_{s}(1)}{dy} \right) $$
$$ v_{x,w} = \frac{W_{p}(3)/W_{p}(1) - W_{w}(3)/W_{w}(1)}{dx} $$
$$ v_{y,w} = 0.5*\left( \frac{W_{nw}(3)/W_{nw}(1) - W_{sw}(3)/W_{sw}(1) + W_{n}(3)/W_{n}(1) - W_{s}(3)/W_{s}(1)}{dy} \right) $$
$$ t_{x,w} = \frac{\text{Temp}_{e} - \text{Temp}_{p}}{dx} $$

This code is used in section 87.
The $x$ components of the viscous flux $R_i$ are calculated as the difference between the outward fluxes $R_i^+$ and the inward fluxes $R_i^-$. The resulting values are accumulated into the $ARST$ array.

The $z$ velocity component of $R$ is not calculated in the 2D case. Also the $z$ components $u_z, v_z$ and $w$ are eliminated from the calculation.

(Fill the $R$ array $97^*$) $\equiv$

\[
A_e = y.cell.size
\]

\[
R.p.e(1) = 0.0
\]

\[
R.p.e(2) = A.e*(-2.0/3.0*(0.5*(W.p(9) + W.e(9)) + mu.eff.e*(ux.e + vy.e)) + 2.0*mu.eff.e*ux.e)
R.p.e(3) = A.e*mu.eff.e*(uy.e + vx.e)
R.p.e(4) = 0.0
R.p.e(5) = R.p.e(2)*(W.p(2) + W.e(2))/(W.p(1) + W.e(1)) + R.p.e(3)*(W.p(3) + W.e(3))/(W.p(1) +
W.e(1)) + mu.h.e*tx.e*A.e
\]

\[
do index = 6, 10
R.p.e(index) = mu.eff.e*A.e/K.sigma(index)*(W.e(index)/W.e(1) - W.p(index)/W.p(1))/dx
\]

\[
A_w = y.cell.size
\]

\[
R.p.w(1) = 0.0
\]

\[
R.p.w(2) = A.w*(-2.0/3.0*(0.5*(W.p(9) -| W.w(9)) + mu.eff.w*(ux.w + vy.w)) + 2.0*mu.eff.w*ux.w)
R.p.w(3) = A.e*mu.eff.w*(uy.w + vx.w)
R.p.w(4) = 0.0
R.p.w(5) = R.p.w(2)*(W.p(2) + W.w(2))/(W.p(1) + W.w(1)) + R.p.w(3)*(W.p(3) +
W.w(3))/(W.p(1) + W.w(1)) + mu.h.w*tx.w*A.w
\]

\[
do index = 6, 10
R.p.w(index) = mu.eff.w*A.w/K.sigma(index)*(W.w(index)/W.w(1) - W.p(index)/W.p(1))/dx
\]

\[
do index = 1, 10
\]

if (index $\neq$ 4) then

\[
ARST.p(index) += -R.p.e(index) + R.p.w(index)
\]

else

\[
ARST.p(index) = 0.0
\]

endif

\enddo

This code is used in section 87.
103* The $S$ fluxes are calculated in a $3 \times 2 \times 3$ region thus the centre to centre distances $dx$, $dy$ and $dz$ are as given here. Once again a non-uniform cell size would necessitate a more complex calculation here.

The computational domains for $S_S$ and $S_N$, the two components of $S$, are shown in the following diagram.

The variable $dz$ is not used in the 2D code.

(Calculate $dx$, $dy$, $dz$ for $S$ array 103*) $\equiv$

$dz = 2.0 \times x\_cell\_size$

$dy = y\_cell\_size$

This code is used in section 98.
The $u_z$ and $v_z$ variables are eliminated in the 2D code along with the six $w$ variables.

(Calculate $u_x$ through $w_z$ for $S$ array 105*)

\begin{align*}
  u_x.n &= 0.5*(W.e(2)/W.e(1) - W.w(2)/W.w(1) + W.ne(2)/W.ne(1) - W.nw(2)/W.nw(1))/dx \\
  u_y.n &= (W.n(2)/W.n(1) - W.p(2)/W.p(1))/dy \\
  v_x.n &= 0.5*(W.e(3)/W.e(1) - W.w(3)/W.w(1) + W.ne(3)/W.ne(1) - W.nw(3)/W.nw(1))/dx \\
  v_y.n &= (W.n(3)/W.n(1) - W.p(3)/W.p(1))/dy \\
  t_y.n &= (Temp.n - Temp.p)/dy \\
  u_x.s &= 0.5*(W.se(2)/W.se(1) - W.sw(2)/W.sw(1) + W.e(2)/W.e(1) - W.w(2)/W.w(1))/dx \\
  u_y.s &= (W.p(2)/W.p(1) - W.s(2)/W.s(1))/dy \\
  v_x.s &= 0.5*(W.se(3)/W.se(1) - W.sw(3)/W.sw(1) + W.e(3)/W.e(1) - W.w(3)/W.w(1))/dx \\
  v_y.s &= (W.p(3)/W.p(1) - W.s(3)/W.s(1))/dy \\
  t_y.s &= (Temp.p - Temp.s)/dy
\end{align*}

This code is used in section 98.
The $z$ velocity component of $S$ is not calculated in the 2D case. Also the $z$ components $u_z$, $v_z$ and $w$ are eliminated from the calculation.

(Fill the $S$ array $107^*$)

\[
\begin{align*}
A_n &= x_{cell.size} \\
S_{p,n}(1) &= 0.0 \\
S_{p,n}(2) &= A_n \ast \mu_{eff.n} \ast (u_y.n + v_x.n) \\
S_{p,n}(3) &= A_n \ast (-2.0/3.0 \ast (0.5 \ast (W_p(9) + W_n(9)) + \mu_{eff.n} \ast (u_x.n + v_y.n)) + 2.0 \ast \mu_{eff.n} \ast v_y.n) \\
S_{p,n}(4) &= 0.0 \\
S_{p,n}(5) &= S_{p,n}(2) \ast (W_p(2) + W_n(2))/(W_p(1) + W_n(1)) + S_{p,n}(3) \ast (W_p(3) + W_n(3))/(W_p(1) + W_n(1)) + \mu_{h.n} \ast t_y.n \ast A_n \\
do index = 6,10 \\
S_{p,n}(index) &= \mu_{eff.n} \ast A_n / K_{sigma(index)} \ast (W_n(index)/W_n(1) - W_p(index)/W_p(1))/dy \\
enddo \\
A.s &= x_{cell.size} \\
S_{p.s}(1) &= 0.0 \\
S_{p.s}(2) &= A_s \ast \mu_{eff.s} \ast (u_y.s + v_x.s) \\
S_{p.s}(3) &= A_s \ast (-2.0/3.0 \ast (0.5 \ast (W_p(9) + W_s(9)) + \mu_{eff.s} \ast (u_x.s + v_y.s)) + 2.0 \ast \mu_{eff.s} \ast v_y.s) \\
S_{p.s}(4) &= 0.0 \\
S_{p.s}(5) &= S_{p.s}(2) \ast (W_p(2) + W_s(2))/(W_p(1) + W_s(1)) + S_{p.s}(3) \ast (W_p(3) + W_s(3))/(W_p(1) + W_s(1)) + \mu_{h.s} \ast t_y.s \ast A.s \\
do index = 6,10 \\
S_{p.s}(index) &= \mu_{eff.s} \ast A_s / K_{sigma(index)} \ast (W_s(index)/W_s(1) - W_p(index)/W_p(1))/dy \\
enddo \\
do index = 1,10 \\
if (index \neq 4) then \\
ARST_p(index) + = S_{p,n}(index) - S_{p.s}(index) \\
else \\
ARST_p(index) = 0.0 \\
endif \\
enddo
\]

This code is used in section 98.
§108 — Explode Worker Task – 2D Version —

108* The $T$ flux calculations are removed for the two dimensional version of the program.

109* Module removed.

110* Module removed.

111* Module removed.

112* Module removed.

113* Module removed.

114* Module removed.

115* Module removed.

116* Module removed.

117* Module removed.
118* INVISCID FLUXES. The three inviscid flux components $F$, $G$, and $H$ in the $x$, $y$, and $z$ directions are now calculated. As with the previous flux components, values from the six orthogonal neighbour cells are required for these calculations.

The computational domain for the inviscid flux calculation is shown in the following diagram.

![Diagram](image)

Domain for $F$, $G$ and $H$

The macro $FGH$ is declared to reflect the storage location of the accumulated inviscid fluxes in the $W$ array.

In the 2D version of the program the $z$ component of the inviscid flux, $H$, is not calculated.

**define** $FGH.p(i) \equiv W(44 + i, cell)$

(Do inviscid fluxes 118*)

(Do inviscid fluxes 118*)

Do cell = 1, n_of_cells

(Do inviscid fluxes 118*)

(Do inviscid fluxes 118*)

enddo

(Do inviscid fluxes 118*)

This code is used in section 24.
The x components of the inviscid flux are calculated in this section.

As in the preceding cases the flux components are expressed as the difference between the outward and inward fluxes $F^+$ and $F^-$.

The general expression for the outward fluxes $F_i^+$ is as follows

$$F_i^+ = A_i^+ \Phi_i^+,$$

where

$$A_i^+ = -\Delta y \Delta z u_i^+, \quad \Phi_i^+ = \frac{1}{2} (\Phi_i + \Phi_i^+) \quad \text{and} \quad u_i^+ = \frac{1}{2} (u + u^+).$$

The outward flux for $u$, $F_2^+$ is given by

$$F_2^+ = A_2^+ \Phi_2^+ - P^+ \Delta y \Delta z$$

where

$$P^+ = \frac{1}{2} (P + P^+).$$

The outward flux for $h$, $F_5^+$ is given by

$$F_5^+ = A^+ \left( h^+ + P^+ \right)$$

where

$$h^+ = \frac{1}{2} (h + h^+).$$

The inward fluxes $F_i^-$ are calculated in a similar manner.
The variables $A_e$ and $A_w$ are related to cell side length rather than cross sectional area in the 2D code. Also the $z$ components in the calculation of $F$ are eliminated.

\[
\text{(Calculate } F \text{ components 119*) } \equiv \\
A_e = -y\cdot\text{cell.size}\cdot0.5\cdot(W_e(2)/W_e(1) + W_p(2)/W_p(1)) \\
A_w = -y\cdot\text{cell.size}\cdot0.5\cdot(W_p(2)/W_p(1) + W_w(2)/W_w(1)) \\
\text{do index } = 1, 10 \\
\text{if (index } \neq 4 \text{) then} \\
\quad FGH_p(index) = A_e\cdot0.5\cdot(W_p(index) + W_e(index)) - A_w\cdot0.5\cdot(W_w(index) + W_p(index)) \\
\text{else} \\
\quad FGH_p(index) = 0.0 \\
\text{endif} \\
\text{enddo} \\
FGH_p(2) -= 0.5\cdot(W_p(11) + W_e(11))\cdot y\cdot\text{cell.size} - 0.5\cdot(W_w(11) + W_p(11))\cdot y\cdot\text{cell.size} \\
FGH_p(5) += A_e\cdot0.5\cdot(W_p(11) + W_e(11)) - A_w\cdot0.5\cdot(W_w(11) + W_p(11)) \\
\text{This code is used in section 118*.}
\]

120* The $y$ components of the viscous flux $G$ are calculated from the outward and inward fluxes in a similar fashion to $F$.

The variables $A_n$ and $A_s$ are related to cell side length rather than cross sectional area in the 2D code. Also the $z$ components in the calculation of $G$ are eliminated.

\[
\text{(Calculate } G \text{ components 120*) } \equiv \\
A_n = -x\cdot\text{cell.size}\cdot0.5\cdot(W_n(3)/W_n(1) + W_p(3)/W_p(1)) \\
A_s = -x\cdot\text{cell.size}\cdot0.5\cdot(W_p(3)/W_p(1) + W_s(3)/W_s(1)) \\
\text{do index } = 1, 10 \\
\text{if (index } \neq 4 \text{) then} \\
\quad FGH_p(index) += A_n\cdot0.5\cdot(W_p(index) + W_n(index)) - A_s\cdot0.5\cdot(W_s(index) + W_p(index)) \\
\text{else} \\
\quad FGH_p(index) = 0.0 \\
\text{endif} \\
\text{enddo} \\
FGH_p(3) -= 0.5\cdot(W_p(11) + W_n(11))\cdot x\cdot\text{cell.size} - 0.5\cdot(W_s(11) + W_p(11))\cdot x\cdot\text{cell.size} \\
FGH_p(5) += A_n\cdot0.5\cdot(W_p(11) + W_n(11)) - A_s\cdot0.5\cdot(W_s(11) + W_p(11)) \\
\text{This code is used in section 118*.}
§121 - Explode Worker Task - 2D Version -

121* The $H$ flux is not calculated in the two dimensional case.
The total flux, $\psi$ through the cell is given by

$$\psi = \Delta t \left( \frac{FGH + RST + ART}{V} + C \right)$$

where $FGH$ are the inviscid fluxes, $RST$ are the viscous fluxes, $ART$ are the artificial viscosity terms and $C$ are the source terms.

The updated $\Phi$ values are given by

$$\Phi^* = \Phi + \alpha_s \psi$$

where $\alpha_s$ is the Runge-Kutta coefficient for stage $s$.

Once the $\Phi^*$ values have been calculated for a given cell they must be checked to ensure that they remain within appropriate value ranges.

In the 2D version of the program, the $z$ velocity component $W_p(4)$ is not used.

This code is used in section 24.
For the zero slip boundary condition no flux is allowed across the boundary. This is achieved by setting \( \Phi_{out} \) to \( \Phi_{in} \) for each element of \( W_{out} \) except for the velocity components where \( \Phi_{out} = -\Phi_{in} \).

The variable \( W_{out}(4) \) is not used in the 2D case.

\[
\text{(Handle zero slip boundary 133*) \equiv}
\]

\[
\begin{align*}
& \text{do index = 1, 24} \\
& \quad W_{out}(\text{index}) = W_{in}(\text{index}) \\
& \text{enddo} \\
& W_{out}(2) = -W_{in}(2) \\
& W_{out}(3) = -W_{in}(3) \\
& W_{out}(4) = 0.0
\end{align*}
\]

This code is used in section 131.
The behaviour of a boundary of this type is dependent on whether an inflow or outflow condition currently holds. This may be determined by consideration of the appropriate velocity component within the current cell.

The $\Phi_{\text{out}}$ values are then calculated as linear combinations of the internal and atmospheric conditions, $\Phi_{\text{in}}$ and $\Phi_{\text{at}}$.

The variable $W_{\text{out}}(4)$ is not used in the 2D case.

(Handel inflow outflow boundary 134*) $\equiv$

$$\text{if } ((\text{mod}(\text{direction}, 2) \equiv 1 \text{ A } W_{\text{in}}((\text{direction} - 13)/2) < 0.0) \text{ OR } (\text{mod}(\text{direction}, 2) \equiv 0 \text{ A } W_{\text{in}}((\text{direction} - 13)/2) > 0.0)) \text{ then}$$

$$\alpha_1 = 1 - K_{\text{tau}_1}$$
$$\alpha_2 = 1 - K_{\text{tau}_2}$$

else
$$\alpha_1 = K_{\text{tau}_1}$$
$$\alpha_2 = K_{\text{tau}_2}$$

endif
do index = 1, 13
$$W_{\text{out}}(\text{index}) = (1.0 - \alpha_1) W_{\text{in}}(\text{index}) + \alpha_1 W_{\text{at}}(\text{index})$$
endo

$$W_{\text{out}}(2) = W_{\text{in}}(2) W_{\text{out}}(1)/W_{\text{in}}(1)$$
$$W_{\text{out}}(3) = W_{\text{in}}(3) W_{\text{out}}(1)/W_{\text{in}}(1)$$
$$W_{\text{out}}(4) = 0.0$$
$$W_{\text{out}}(14) = W_{\text{in}}(14)$$
$$W_{\text{out}}(15) = W_{\text{in}}(15)$$
$$W_{\text{out}}(16) = W_{\text{in}}(16)$$
$$W_{\text{out}}(11) = 1.0 - \alpha_2 W_{\text{in}}(11) + \alpha_2 W_{\text{at}}(11)$$
$$W_{\text{out}}(5) = 1/(K_{\text{gamma}} - 1.0) W_{\text{out}}(11) + 0.5(W_{\text{out}}(2)^2 + W_{\text{out}}(3)^2 + W_{\text{out}}(4)^2)/W_{\text{out}}(1)$$
$$W_{\text{out}}(13) = (W_{\text{out}}(6)/W_{\text{out}}(1) - \text{beta.infty} - W_{\text{out}}(7)/(\text{beta.zero} - \text{beta.infty}))/W_{\text{out}}(1))$$
$$W_{\text{out}}(23) = N_{\text{infty}}(1 - W_{\text{out}}(7)/W_{\text{out}}(1))$$
$$W_{\text{out}}(24) = \text{max}(0.0, (1.0 - W_{\text{out}}(6)/W_{\text{out}}(1) - K_{\text{stoi}} W_{\text{out}}(13) - W_{\text{out}}(23)))$$
$$W_{\text{out}}(12) = 1.0(W_{\text{out}}(6)/W_{\text{out}}(1) + K_{\text{stoi}} W_{\text{out}}(13)/O_{2}.\text{molwt} + W_{\text{out}}(23)/N_{2}.\text{molwt} + W_{\text{out}}(24)/\text{Prod}.\text{molwt})$$

This code is used in section 131.
The damped boundary is identical to the zero slip condition except that $k$ and $\epsilon$ are forced to a small value, effectively damping turbulence at the boundary.

The variable $W_{\text{out}}(4)$ is not used in the 2D case.

```plaintext
(\text{Handle damped boundary 136*}) \equiv

\text{do index} = 1, 24
    W_{\text{out}}(\text{index}) = W_{\text{in}}(\text{index})
\text{enddo}

W_{\text{out}}(2) = -W_{\text{in}}(2)
W_{\text{out}}(3) = -W_{\text{in}}(3)
W_{\text{out}}(4) = 0.0
W_{\text{out}}(9) = 0.000001
W_{\text{out}}(10) = 0.000001
```

This code is used in section 131.
INDEX. The index contains references to all variables and macro definitions in the program.

Underlined module numbers refer to the module in which an indexed item is first referenced.

The following sections were changed by the change file: 1, 2, 20, 26, 33, 35, 39, 43, 45, 47, 51, 52, 54, 56, 61, 62, 64, 65, 66, 68, 69, 70, 74, 79, 80, 81, 82, 83, 84, 85, 86, 93, 95, 97, 103, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 123, 133, 134, 136, 167.

A.e: 96, 97, 119
A.n: 106, 107, 120
A.s: 106, 107, 120
A.w: 96, 97, 119
abs: 52, 54, 129, 132
alpha1: 130, 134
alpha2: 130, 134
ARST: 69, 97
ARST.p: 69, 97, 107, 123
b.switch: 5, 7
beta_infty: 19, 20, 26, 39, 49, 125, 126, 134
beta.zero: 19, 20, 26, 39, 49, 125, 126, 134
BOTTOM: 38
buffer: 5
buffer_thread_137: 2, 5, 7, 137, 139
buffer_to_main: 5, 7
buffer_to_switch: 5, 7
buffer_work: 5, 7
C.p: 52, 123
cell: 13, 14, 16, 25, 26, 28, 29, 35, 37, 52, 53, 69, 70, 118, 123, 152, 153, 155
cell.out: 126, 127
cell.volume: 19, 20, 66, 123
condition.data.type: 10, 16, 141, 144
condition.set: 15, 16
constant.data.type: 10, 18, 141, 145
CO2.m: 19, 20, 72, 91, 92, 101, 102
CO2.molwt: 19, 20
cp.e: 89, 90, 91
cp.n: 99, 100, 101
cp.p: 99, 100, 102
cp.w: 89, 90, 92
D: 50
data_request.type: 10, 129, 141, 154
debug: 9, 35, 157
debug.common: 9
debug.type: 10, 141, 156
destination: 39
direction: 39, 125, 126, 129, 132, 134, 135
dt: 34, 35
dx: 42, 43, 45, 47, 93, 95, 97, 103, 105
dy: 42, 43, 45, 47, 93, 95, 103, 105, 107
dz: 42, 43, 93, 103
E.b: 65, 66
E.c: 65, 66, 69
E.n: 65, 66, 69
E.s: 65, 66, 69
E.t: 65, 66
E.w: 65, 66, 69
EAST: 38, 39, 57, 75, 76
eddy: 27, 28
exp: 49
explwork_2d: 2
f.size: 32, 33, 35
f.u: 32, 33, 35
f.v: 32, 33, 35
f.vel: 32, 33, 35
f.w: 32, 35
FGH: 118
FGH_p: 118, 119, 120, 123
FILL: 39, 57, 58, 59, 60, 75, 76, 77, 78
fill_array_125: 2, 39, 57, 125, 154
finished: 13, 14
flag: 5, 7, 163
flux: 122, 123
from: 5
from.buffer: 5, 7, 39, 125, 126, 158, 162
from.main: 158, 164
from_ring: 138, 139, 140
Fuel_infty: 19, 20
Fuel.molwt: 19, 20, 72, 91, 92, 101, 102
fuel.molwt: 20
Fuel.zero: 19, 20
fx: 46, 47, 52
fy: 46, 47, 52
fz: 46, 47
f77.alt_wait: 158
f77.chan_address: 7
f77.chan.in_message: 140, 162, 164
f77.chan.in_port: 139
f77.chan_init: 7
f77.chan.out_message: 163, 165, 166
f77.chan.out_port: 158
f77.sema_init: 7
f77.sema_signal: 163
f77.sema_size: 6
f77.sema_wait: 163
f77.thread.deschedule: 7, 125, 137, 158
f77.thread.noturg: 7
§167 Explode Worker Task - 2D Version

INDEX 272

f77.thread_priority: 7
f77.thread_start: 7
f77.thread_urgent: 7

geometry.data.type: 10, 14, 141, 143.
griddler: 13.
gx: 46, 47, 52
gy: 46, 47, 52
gz: 46, 52

header: 10, 128, 129, 138, 140, 158, 161, 162, 163, 164, 165, 166.

H2O.molwt: 19, 20

include: 4.
index: 17, 18, 25, 69, 71, 97, 107, 119, 120, 123, 124, 125, 126, 127, 129, 131, 132, 133, 134, 135, 136, 152, 153, 155, 156.

K.a: 17, 18, 49.
K.c: 17, 18, 123.
K.alfa: 17, 18, 49.
K.b: 17, 18, 49.
K.cfl: 17, 18, 35.
K.cpa: 17, 18, 28, 41, 89, 99.
K.cpb: 17, 18, 72, 91, 92, 101, 102.
K.d: 17, 18, 28, 41, 89, 99.
K.die: 17, 18, 52.
K.e: 17, 18, 52.
K.en: 17, 18, 49.
K.gamma: 17, 18, 20, 26, 31, 35, 39, 125, 126, 134.
K.gindex: 17, 18, 41, 89, 99.
K.gz: 17, 18, 52.
K.h: 17, 18, 52.
K.hc: 17, 18, 39, 52, 125, 126.
K.k: 17, 18, 66.
K.kf: 17, 18, 68.
K.m: 17, 18, 28.
K.mfs: 17, 18, 28.
K.mgz: 17, 18, 52.
K.n: 17, 18.
K.nca: 17, 18, 20.
K.nh: 17, 18, 20.
K.p: 17, 18, 49, 99.
K.pmt: 17, 18, 49, 99.
K.r: 17, 18, 20.
K.r: 17, 18, 41, 49, 72, 91, 92, 101, 102.
K.r: 17, 18, 41.

K_molwt: 19, 20.

K.s: 17, 18, 49.
K.sym: 17, 18, 97.
K.sso: 17, 18.
K.u: 19, 20, 26, 39, 49, 72, 91, 92, 101, 102, 125, 126, 134.
K.tau: 17, 18, 39, 125, 126, 134.
K.tau: 17, 18, 39, 125, 126, 134.
K.tref: 17, 18, 41, 72, 91, 92, 101, 102.
K.vent: 17, 18, 39, 125, 126, 132.
K.1: 17, 18, 52.
K.2: 17, 18, 52.
mach.1: 30, 31, 35.
main_to_switch: 5, 7.

max: 20, 26, 64, 68, 134.
MAX_CELL: 2, 13, 14, 35, 125, 127, 129, 132, 151, 153, 154, 155, 156.

min: 52, 66.
mod: 14, 127, 134, 153, 155.
mu_eff: 40, 41, 51, 52.
mu_eff_b: 88.
mu_eff_e: 88, 89, 97.
mu_eff_s: 88, 99, 107.
mu_eff_t: 88.
mu_eff_w: 88, 89, 97.
mu_h_b: 88.
mu_h_e: 88, 89, 97.
mu_h_t: 88.
mu_h_w: 88, 89, 97.
m_u: 40, 41, 52, 89, 99.
m_u: 40, 41, 89, 99.

my_worker_id: 9, 11, 14, 35, 125, 129, 132, 151, 154, 156.
n_of_cells: 13, 14, 16, 25, 26, 28, 29, 37, 53, 70, 118, 123.
n_of_conditions: 15, 16.
one: 2, 125, 137, 158.
NORTH: 38, 39, 79, 75, 77.
N2.1: 19, 20, 26, 39, 125, 126, 134.
N2.1: 19, 20, 26, 39, 125, 126, 134.
O2.1: 19, 20.
O2.1: 19, 20, 26, 39, 125, 126, 134.
O2.1: 19, 20.
packet: 10, 11, 14, 16, 18, 36, 128, 129, 138, 140, 151, 153, 154, 155, 156, 157, 158, 162, 163, 164, 165, 166.
priority: 5, 7.
Prod.molwt: 19, 20, 26, 39, 125, 126, 134.
Q.b: 67, 68
Q.e: 67, 68, 69
Q.n: 67, 68
Q.s: 67, 68, 69
Q.t: 67, 68
Q.w: 67, 68, 69
R.p.e: 96, 97
R.p.w: 96, 97
r.xe: 63, 64, 66, 68
r.wx: 63, 64, 66, 68
r.ye: 63, 64, 66, 68
r.ys: 63, 64, 66, 68
r.xw: 63, 64
r.x: 63, 64
report.type: 10, 141, 151.
rho.k: 50, 51
rx: 53, 54
ry: 53, 54
rz: 53, 54
S.p.n: 106, 107
S.p.s: 106, 107
S.sq: 50, 51
s.u: 40, 41, 52
semaphore: 6
source: 39, 158.
SOUTH: 38, 39, 60, 76, 78.
sqrt: 28, 31, 33
stage: 23, 24, 123
step.complete.type: 10, 37, 141, 149, 161.
switch.thread: 2, 5, 7, 158.
switch.work: 5, 7.
synch.type: 10, 37, 141, 146, 160.
tau.ch: 48, 49, 52
tau.e: 48, 49, 52
Temp.avg: 71, 72, 91, 92, 101, 102.
Temp.e: 89, 90, 91, 95
Temp.n: 99, 100, 101, 105
Temp.p: 40, 41, 49, 72, 89, 95, 99, 105
temp.p: 41.
Temp.s: 99, 100, 102, 105
Temp.w: 89, 90, 92.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>W.e</td>
<td>38, 39*, 45*, 47*, 54*, 57, 64*, 69*, 75, 76, 89, 91, 95*, 97*, 105*, 119*</td>
</tr>
<tr>
<td>W.fb</td>
<td>52, 56*</td>
</tr>
<tr>
<td>W.fe</td>
<td>55, 57, 64*, 69*</td>
</tr>
<tr>
<td>W.fm</td>
<td>55, 59, 64*, 69*</td>
</tr>
<tr>
<td>W.fs</td>
<td>55, 60, 64*, 69*</td>
</tr>
<tr>
<td>W.fr</td>
<td>55, 56*</td>
</tr>
<tr>
<td>W.fw</td>
<td>55, 58, 64*, 69*</td>
</tr>
<tr>
<td>W.in</td>
<td>125, 126, 129, 132, 133*, 134*, 135, 136*</td>
</tr>
<tr>
<td>W.ne</td>
<td>73, 75, 95*, 105*</td>
</tr>
<tr>
<td>W.nw</td>
<td>73, 77, 95*, 105*</td>
</tr>
<tr>
<td>W.nl</td>
<td>73</td>
</tr>
<tr>
<td>W.out</td>
<td>125, 126, 127, 129, 132, 133*, 134*, 135, 136*</td>
</tr>
<tr>
<td>W.s</td>
<td>38, 39*, 45*, 47*, 54*, 60, 64*, 69*, 76, 78, 95*, 99, 102, 105*, 107*, 120*</td>
</tr>
<tr>
<td>W.se</td>
<td>73, 76, 95*, 105*</td>
</tr>
<tr>
<td>W.sw</td>
<td>73, 78, 95*, 105*</td>
</tr>
<tr>
<td>W.t</td>
<td>38, 39*</td>
</tr>
<tr>
<td>W.te</td>
<td>73</td>
</tr>
<tr>
<td>W.tm</td>
<td>73</td>
</tr>
<tr>
<td>W.ts</td>
<td>73</td>
</tr>
<tr>
<td>W.tw</td>
<td>73</td>
</tr>
<tr>
<td>W.w</td>
<td>38, 39*, 45*, 47*, 54*, 57, 58, 64*, 69*, 77, 78, 89, 92, 95*, 97*, 105*, 119*</td>
</tr>
<tr>
<td>WEST</td>
<td>38, 39*, 58, 77, 78</td>
</tr>
<tr>
<td>while</td>
<td>2*, 14, 137, 158</td>
</tr>
<tr>
<td>WORK_SIZE</td>
<td>5, 7</td>
</tr>
<tr>
<td>worker</td>
<td>13, 14, 125, 126, 132, 150, 151, 154, 156</td>
</tr>
<tr>
<td>worker_count_type</td>
<td>10, 11, 141, 142</td>
</tr>
<tr>
<td>worker_id</td>
<td>9</td>
</tr>
<tr>
<td>wz</td>
<td>44, 51*</td>
</tr>
<tr>
<td>wz.e</td>
<td>94</td>
</tr>
<tr>
<td>wz.n</td>
<td>104</td>
</tr>
<tr>
<td>wz.s</td>
<td>104</td>
</tr>
<tr>
<td>wz.w</td>
<td>94</td>
</tr>
<tr>
<td>wy</td>
<td>44, 51*</td>
</tr>
<tr>
<td>wy.e</td>
<td>94</td>
</tr>
<tr>
<td>wy.n</td>
<td>104</td>
</tr>
<tr>
<td>wy.s</td>
<td>104</td>
</tr>
<tr>
<td>wy.w</td>
<td>94</td>
</tr>
<tr>
<td>wz</td>
<td>44, 45*, 51*</td>
</tr>
<tr>
<td>wz.e</td>
<td>94, 95*</td>
</tr>
<tr>
<td>wz.n</td>
<td>104</td>
</tr>
<tr>
<td>wz.s</td>
<td>104</td>
</tr>
<tr>
<td>wz.w</td>
<td>94</td>
</tr>
<tr>
<td>z</td>
<td>18</td>
</tr>
</tbody>
</table>
\{ Boundary \textit{W.out} code 131 \} Used in section 125.
\{ Buffer subroutine 137 \} Used in section 2*.
\{ Calculate $\mu_{eff}$ for C array 41 \} Used in section 37.
\{ Calculate $\mu_{eff}$ for R array 89 \} Used in section 87.
\{ Calculate $\mu_{eff}$ for S array 99 \} Used in section 98.
\{ Calculate $\rho_k$ 51* \} Used in section 37.
\{ Calculate $\tau_{sh}$ and $\tau_e$ 49 \} Used in section 37.
\{ Calculate dx, dy, dz for C array 43* \} Used in section 37.
\{ Calculate dx, dy, dz for R array 93* \} Used in section 87.
\{ Calculate dx, dy, dz for S array 103* \} Used in section 98.
\{ Calculate fx through gz for C array 47* \} Used in section 37.
\{ Calculate pressure and mass terms 26* \} Used in section 24.
\{ Calculate ux through wz for C array 45* \} Used in section 37.
\{ Calculate ux through wz for R array 95* \} Used in section 87.
\{ Calculate ux through wz for S array 105* \} Used in section 98.
\{ Calculate ART 69* \} Used in section 53.
\{ Calculate C array 52* \} Used in section 37.
\{ Calculate E terms 66* \} Used in section 53.
\{ Calculate F components 119* \} Used in section 118*.
\{ Calculate G components 120* \} Used in section 118*.
\{ Calculate Q terms 68* \} Used in section 53.
\{ Calculate R array 87 \} Used in section 70*.
\{ Calculate S array 98 \} Used in section 70*.
\{ Calculate Temp.e, cp.e 91 \} Used in section 89.
\{ Calculate Temp.n, cp.n 101 \} Used in section 99.
\{ Calculate Temp.p, cp.p 72 \} Used in section 70*.
\{ Calculate Temp.s, cp.s 102 \} Used in section 99.
\{ Calculate Temp.w, cp.w 92 \} Used in section 89.
\{ Calculate $dt$ 35* \} Used in section 29.
\{ Calculate f.vel, f.size 33* \} Used in section 29.
\{ Calculate mach.1 31 \} Used in section 29.
\{ Calculate $r$ maxima 64* \} Used in section 53.
\{ Calculate $r$ terms 54* \} Used in section 53.
\{ Check minimum values 124 \} Used in section 123*.
\{ Common semaphore 6 \} Used in sections 2*, 125, and 137.
\{ Derive pseudo constants 20* \} Used in section 2*.
\{ Determine worker id 11 \} Used in section 2*.
\{ Do artificial viscosity calculation 53 \} Used in section 24.
\{ Do inviscid fluxes 118* \} Used in section 24.
\{ Do source term calculation 37 \} Used in section 24.
\{ Do the main loop 24 \} Used in section 2*.
\{ Do update arrays 123* \} Used in section 24.
\{ Do viscid flux calculation 70* \} Used in section 24.
\{ Fill near arrays 39* \} Used in sections 37, 53, 56*, 74*, and 118*.
\{ Fill off-element arrays for ART calculation 56* \} Used in section 53.
\{ Fill off-element arrays for R, S and T calculations 74* \} Used in section 70*.
\{ Fill subroutine 125 \} Used in section 2*.
\{ Fill the debug packet 157 \} Used in section 156.
\{ Fill the request packet 155 \} Used in section 154.
\{ Fill the result packet 153 \} Used in section 151.
\{ Fill the R array 97* \} Used in section 87.
\{ Fill the S array 107* \} Used in section 98.
\section*{Explode Worker Task \textendash{} 2D Version}

\begin{itemize}
\item Fill $W\_fe$ \textsuperscript{57} Used in section 56*.
\item Fill $W\_fm$ \textsuperscript{59} Used in section 56*.
\item Fill $W\_fs$ \textsuperscript{60} Used in section 56*.
\item Fill $W\_fw$ \textsuperscript{58} Used in section 56*.
\item Fill $W\_ne$ \textsuperscript{75} Used in section 74*.
\item Fill $W\_nw$ \textsuperscript{77} Used in section 74*.
\item Fill $W\_se$ \textsuperscript{76} Used in section 74*.
\item Fill $W\_sw$ \textsuperscript{78} Used in section 74*.
\item Get constant data \textsuperscript{18} Used in section 12.
\item Get initial data \textsuperscript{12} Used in section 2*.
\item Get packet from buffer \textsuperscript{162} Used in sections 11, 14, 16, 18, 36, 129, 158, and 160.
\item Get packet from main \textsuperscript{164} Used in section 158.
\item Get packet from ring \textsuperscript{140} Used in section 137.
\item Get synch \textsuperscript{160} Used in sections 37, 53, 70*, 118*, and 123*.
\item Get the condition data \textsuperscript{16} Used in section 12.
\item Get the geometry data \textsuperscript{14} Used in section 12.
\item Handle damped boundary \textsuperscript{136*} Used in section 131.
\item Handle inflow outflow boundary \textsuperscript{134*} Used in section 131.
\item Handle non-zero slip boundary \textsuperscript{135} Used in section 131.
\item Handle vented boundary \textsuperscript{132} Used in section 131.
\item Handle zero slip boundary \textsuperscript{133*} Used in section 131.
\item Include files \textsuperscript{4} Used in sections 2*, 125, 137, and 158.
\item Initialize buffer channel addresses \textsuperscript{139} Used in section 137.
\item Local $W\_out$ code \textsuperscript{127} Used in sections 125 and 132.
\item Other common data \textsuperscript{9} Used in sections 2*, 125, and 137.
\item Process packet from ring \textsuperscript{141} Used in section 137.
\item Process $condition\_data\_type$ \textsuperscript{144} Used in section 141.
\item Process $constant\_data\_type$ \textsuperscript{145} Used in section 141.
\item Process $data\_request\_type$ \textsuperscript{154} Used in section 141.
\item Process $debug\_type$ \textsuperscript{156} Used in section 141.
\item Process $geometry\_data\_type$ \textsuperscript{143} Used in section 141.
\item Process $report\_type$ \textsuperscript{151} Used in section 141.
\item Process $step\_complete\_type$ \textsuperscript{149} Used in section 141.
\item Process $synch\_type$ \textsuperscript{146} Used in section 141.
\item Process $timeset\_type$ \textsuperscript{148} Used in section 141.
\item Process $timestep\_type$ \textsuperscript{147} Used in section 141.
\item Process $worker\_count\_type$ \textsuperscript{142} Used in section 141.
\item Remote $W\_out$ code \textsuperscript{129} Used in sections 125 and 132.
\item Send and set timestep \textsuperscript{36} Used in section 29.
\item Send done \textsuperscript{161} Used in sections 37, 53, 70*, 118*, and 123*.
\item Send packet to main \textsuperscript{166} Used in sections 142, 143, 144, 145, 146, 147, 148, and 154.
\item Send packet to ring \textsuperscript{165} Used in section 158.
\item Send packet to switch \textsuperscript{163} Used in sections 11, 14, 16, 18, 36, 129, 149, 151, 154, 156, 160, and 161.
\item Set up timestep \textsuperscript{29} Used in section 24.
\item Start the communication threads \textsuperscript{7} Used in section 2*.
\item Store current $W\_p$ to $W\_old$ \textsuperscript{25} Used in section 24.
\item Switch subroutine \textsuperscript{158} Used in section 2*.
\item Variables of $buffer\_thread$ \textsuperscript{157} 138, 150, 152 Used in section 137.
\item Variables of $fill\_array$ \textsuperscript{125} 126, 128, 130 Used in section 125.
\item Variables of $main$ \textsuperscript{5} 10, 13, 15, 17, 19, 23, 27, 30, 32, 34, 38, 40, 42, 44, 46, 48, 50, 55, 63, 65, 67, 71, 73, 88, 90, 94, 96, 100, 104, 106, 122 Used in section 2*.
\item Verify $k$, $epsilon$ \textsuperscript{28} Used in section 24.
\end{itemize}
COMMAND LINE: "C:\BIN\FWEAVE.EXE explwork explwork.2d".
WEB FILE: "explwork.web".
CHANGE FILE: "explwork.2d".
GLOBAL LANGUAGE: FORTRAN.
1. INTRODUCTION. The following program is used to produce the bitmap files used as the basis of the animation sequences presented on the CD-ROM. These files are subsequently converted to GIF format and combined using Autodesk Animator Pro.

Input to the program consists of three files; the geometry file, geom.file, containing \( n_{of\ samples} \) sequential \((x, y)\) coordinate pairs, a base bitmap file, base.file, a 320 \( \times \) 200 bitmap file with a colour table appropriate to the output images and the simulation data file sample.file containing \( n_{of\ frames} \) sequences of \( n_{of\ samples} \) scalar data values corresponding to the coordinates in geom.file.

In addition to these data, a number of variables are solicited from the keyboard. These are cell.size, the number of pixels used to represent each sample point (i.e. a square region of side cell.size), x.start and y.start, the origin of the bitmap in model space, min and max, the sample values corresponding to the minimum and maximum colour values respectively and \( n_{of\ frames} \), the number of sample sets in the data file (as well as the number of output frames in the animation).

Output consists of \( n_{of\ frames} \) bitmap files 000.bmp, 001.bmp, etc. each containing one frame of the final animation.
2. THE PROGRAM. The following module lays out the principal tasks to be performed by the program.

It should be noted that some routines (_setvideomode_, _remapppalette_, _setcolor_, _rectangle_ and _getpixel_) are specific to Microsoft C and would require replacement if the program were to be implemented on a different processor.

```c
#include <stdio.h>
#include <graph.h>
#include <math.h>

// Variables of the program

void main(void)
{
  // Open and read geometry file
  // Open and read bitmap file
  // _setvideomode(_MRES256COLOR);
  // /* Select 320 x 200, 256 colour VGA graphics mode */
  // Set up the palette
  // Open data input file
  // Process the frames
  // _setvideomode(_TEXT80); /* Select 80 column text mode */
}
```
3. INITIALISATION. The following modules control the initialisation of the program including the opening of the input files and the construction of the colour palette.

4. The string $fname$ is used by the program to contain the file names of each of the input files in turn. The arrays $ix$ and $iy$ contain the $n.\text{of.}\text{samples}$ coordinate pairs in screen space corresponding to the model space coordinates, $dx$, $dy$ in $geom.\text{file}$. The variables $i$, $j$ and $k$ are used as loop control variables at various places in the program.

\[
\text{(Variables of the program 4) } = \\
\text{char } fname[40]; \\
\text{FILE } *geom.\text{file}; \\
\text{long int } n.\text{of.}\text{samples}; \\
\text{int } cellsize, ix[3000], iy[3000]; \\
\text{float } dx, dy, x.\text{start}, y.\text{start}; \\
\text{int } i, j, k;
\]

See also sections 6, 8, 10, 13, 15, and 17.

This code is used in section 2.
5. This module opens the geometry coordinate file `geom_file`, transforms them from model space to screen space and stores them in the `ix` and `iy` arrays.

(Open and read geometry file 5) ≡

```
printf("Enter the geometry file name:");
scanf("%s", fname);
geom_file = fopen (fname, "r");
fscanf (geom_file, "/.Id", &n_of_samples);
printf("Enter the x and y coordinates of the model origin: ");
scanf("/.f", &x_start, &y_start);
printf("Enter the cell size:");
scanf("%d", &cell_size);
for (i = 0; i < n_of_samples; i++)
{
    fscanf (geom_file, "/,.f", &dx, &dy);
    ix[i] = (int)(cell_size*(dx - x_start));
    iy[i] = (int)(200.0 - cell_size*(dy - y_start));
}
fclose (geom_file);
```

This code is used in section 2.

6. The array `map` contains the 1078 byte bitmap header.

(Variables of the program 4) ≡

```
FILE *base_file;
unsigned char map[1078];
```
7. To simplify the code for this program it was decided that, rather than create the bitmap header structure from scratch, a standard base bitmap file would be created and the required data read in. A secondary advantage of this mechanism is that changes to the base bitmap, e.g. alternate colour maps, can be readily accomplished without modification to this program.

(Open and read bitmap file 7) ≡

```c
base_file = fopen("base.bmp","r");
for (i = 0; i < 1078; i++)
    fread(&map[i], sizeof(char), 1, base_file);
fclose(base_file);
```

This code is used in section 2.

8. The following variables are used in setting up the video palette for the screen. The required colour values are derived from the contents of the base bitmap file header contained in `map`.

(Variables of the program 4) +≡

```c
long int cval, red, green, blue;
```

9. The 1024 bytes starting at `map[54]` contain the 256 colour table entries for the bitmap. Each set of four bytes holds one byte red, green and blue values, the fourth byte being unused. The 8-bit `red`, `green` and `blue` values from `map` are scaled down to 6-bit values and combined into `cval` via the `RGB` macro. This value is then stored into the appropriate entry in the video palette via the `remappalette` function.

```c
define RGB(r,g,b) (\$3F3F3FL \& ((long)(b) << 16)\&g\&8\&(r))
```

(Set up the palette 9) ≡

```c
for (i = 0; i < 256; i++)
{
    j = 54 + 4*i;
    blue = map[j];
    green = map[j + 1];
    red = map[j + 2];
    cval = RGB(red >> 2, green >> 2, blue >> 2);
    remappalette(i, cval);
}
```

This code is used in section 2.
10. The file `sample.file` contains the simulation output sample data for a single variable (pressure, temperature, fuel fraction etc.). The `n.of.frames` sets of `n.of.samples` data values will be converted to colour values with `min` being scaled to `colour[1]` and `max` to `colour[240]`.

```c
(FILE *sample_file;
    float min, max;
)
```

11. The program now prompts for the name of the sample data file which is opened. The values of `min` and `max` are also read in. It should be noted that `min` and `max` need not represent the actual minimum and maximum sample data values. Any pair of values such that `min > max` may be entered, allowing the program to enhance the visibility of a potentially small range of data values.

```c
(Open data input file 11) ≡
    printf("Enter the sample data file name: ");
    scanf("%s", fname);
    sample_file = fopen(fname, "r");
    printf("Enter the minimum and maximum sample data values: ");
    scanf("%f%f", &min, &max);
```

This code is used in section 2.
12. THE MAIN LOOP. The remaining modules are invoked for each of the \textit{n.of.frames} sets of input data. They read in each successive set of \textit{n.of.samples} sample points, convert these values to colour indices, build the screen display and finally dump the screen into a bitmap file.

13. The variable \textit{n.of.frames} contains the number of data sets contained in the sample data file.

\begin{verbatim}
(Variables of the program 4) +≡
    long int n_of_frames;
\end{verbatim}

14. Each of the \textit{n.of.frames} sets of input data samples is converted to a bitmap file. This process is carried out in two phases; first the data is read in and an on-screen image is produced, then this image is scanned from the screen and written out to a bitmap file.

\begin{verbatim}
(Process the frames 14) ≡
    scanf("%ld", &n_of_frames);
    for(i = 0; i < n_of_frames; i++)
    {
        (Paint the screen 16)
        (Output the screen to file 18)
    }
\end{verbatim}

This code is used in section 2.

15. The variable \textit{value} holds the successive sample data values. These are mapped into corresponding colour values \textit{col}.

\begin{verbatim}
(Variables of the program 4) +≡
    short int col;
    float value;
\end{verbatim}
§16 - Bitmap - THE MAIN LOOP

16. Each of the $n_{of\ samples}$ data values for a single frame are read from sample_file and converted into a colour value, $col$, by a linear scaling of $value$ into the colour range $(1, 240)$ such that $\min \rightarrow 1$ and $\max \rightarrow 240$. Once the appropriate colour value $col$ is established a $cell\_size \times cell\_size$ square region of the screen with lower left corner at $ix[i], iy[j]$ is filled with $col$. The $\text{setcolor}$ and $\text{rectangle}$ calls perform the appropriate colour selection and fill tasks.

{Paint the screen 16} ≜

for$(j = 0; j < n_{of\ samples}; j++)$
{
    fscanf(sample_file, "%f", &value);
    col = (short)(1.0 + 240.0*(value - \min)/(\max - \min));
    if (col < 1) col = 1;
    if (col > 241) col = 241;
    setcolor(col);
    rectangle(.GFILLINTERIOR, ix[j], iy[j], ix[j] + cell\_size - 1, iy[j] + cell\_size - 1);
}

This code is used in section 14.

17. The variable $pixel$ is used to hold the colour values of successive screen locations prior to being written to the output bitmap files.

{Variables of the program 4} +≡

FILE *output;
int pixel;
18. Each of the _n_of_frames_ output files consists of the 1078-byte bitmap header, copied from the base bitmap file followed by the 320 x 200 pixel colour values.

The storage order of the pixels in the bitmap files differs from that of the screen coordinate system in that the first 320 image bytes in the output file correspond to scan line 199 of the screen image and the last 320 bytes to scan line 0. To address this difference the screen is scanned from line 199 down to line 0.

The function _getpixel(i,j)_ returns the colour index of pixel (i,j) from the screen to the variable _pixel_.

(Output the screen to file 18) ≡

```
sprintf (fname, "%03d.bmp", i);
output = fopen (fname, "wb");
for (j = 0; j < 1078; j++)
    fwrite (kmap[j], sizeof(char), 1, output);
for (j = 0; j < 200; j++)
    {
        for (k = 0; k < 320; k++)
            {
                pixel = _getpixel(k, 199 - j);
                fwrite (&pixel, sizeof(char), 1, output);
            }
    }
fclose (output);
```

This code is used in section 14.
19. INDEX. The index contains references to all variables and macro definitions in the program. Underlined module numbers refer to the module in which an indexed item is first referenced.

- _getpixel: 2, 18.
- _GFILLINTERIOR: 16.
- _MRES256COLOR: 2.
- _rectangle: 2, 16.
- _remap_palette: 2, 9.
- _setcolor: 2, 16.
- _setvideomode: 2.
- _TEXTC80: 2.

_base_file: 1, 6, 7.
_blue: 8, 9.
_cellsize: 1, 4, 5, 16.
_col: 15, 16.
_colour: 10.
_cval: 8, 9.
_dx: 4, 5.
_dy: 4, 5.
_fclose: 5, 7, 18.
_fname: 4, 5, 11, 18.
_fopen: 5, 7, 11, 18.
_fread: 7.
_fscanf: 5, 16.
_fwrite: 18.
_geom_file: 1, 4, 5.
_green: 8, 9.
_i: 4.
_iz: 4, 5, 16.
_iy: 4, 5, 16.
_j: 4.
_k: 4.
_main2: 2.
_map: 6, 7, 8, 9, 18.
_max: 1, 10, 11, 16.
_min: 1, 10, 11, 16.
_n_of_frames: 1, 10, 12, 13, 14, 18.
_n_of_samples: 1, 4, 5, 10, 12, 16.
_output: 17, 18.
_pixel: 17, 18.
_printf: 5, 11.
_red: 8, 9.
_RGB: 9.
_sample_file: 1, 10, 11, 16.
_scanf: 5, 11, 14.
_sprintf: 18.
§19 —Bitmap—

(Open and read bitmap file 7) Used in section 2.
(Open and read geometry file 5) Used in section 2.
(Open data input file 11) Used in section 2.
(Output the screen to file 18) Used in section 14.
(Paint the screen 16) Used in section 14.
(Process the frames 14) Used in section 2.
(Set up the palette 9) Used in section 2.
(Variables of the program 4, 6, 8, 10, 13, 15, 17) Used in section 2.

COMMAND LINE: "C:\BIN\FWEAVE.EXE bitmap".
WEB FILE: "bitmap.web".
CHANGE FILE: (none).
GLOBAL LANGUAGE: C.