Met O (APR) Turbulence and Diffusion Note No. 240

Large-eddy simulation on a massively parallel computer

by

A.R. Brown, M.E.B. Gray and M.K. MacVean

2nd July 1997

Met O (APR)
(Atmospheric Processes Research)
Meteorological Office
London Road
Bracknell
Berks, RG12 2SZ

Note

Met O (APR) TDN-240

Large-eddy simulation on a massively parallel computer

A.R. Brown, M.E.B. Gray and M.K. MacVean

Contents

Large-eddy simulation on a massively parallel computer

A.R. Brown, M.E.B. Gray and M.K. MacVean

2nd July 1997

Abstract

The ways in which the large-eddy model has been changed in order to run on the massively parallel T3E supercomputer are detailed. This is not intended to be a complete documentation of the model and should be used in conjunction with TDN 213. However, because of a considerable number of changes since that document was written, updated lists of model parameters and namelist variables are also given.

1 Introduction

The present Met O(APR) large-eddy model was originally coded by S.H.Derbyshire for the CRAY-YMP, based on the IBM code of P.J.Mason. The code has evolved over the years, with added functionality, and maintenance simplified through use of the Cray Update facility. A relatively steady state was reached with Version 1.4, for which documentation is available (TDN 213; Derbyshire *et al.*, 1994), and various studies using this version of the model have appeared in the literature. Version 1.5, released in August 1996, was intended to be an interim version incorporating various updates to Version 1.4 that it was thought helpful to have in the base code, prior to a major re-write at Version 2.0. This re-write was necessary because of the decision to purchase a CRAY T3E which, unlike the YMP (and the C90 which followed it), is a massively parallel, distributed memory machine. A single job can run on a large number of processing elements (PEs), each of which has direct access to its own memory, but which can only access the memory of other PEs through explicit calls ('message passing'). The aim of the present document is to detail the changes which have been made to the code to enable it to run efficiently on such a computer. Of necessity, much of the discussion is specific to the large-eddy model, but it is hoped that some of the more general issues relating to domain decomposition and the handling of input and output may be relevant to people wishing to port other large models to massively parallel machines.

The structure of the remainder of this document is as follows. Section (2) describes the way in which a problem is split across a number of processors, and Section (3) describes how the model structure achieves this. The pressure solver is probably the single most difficult issue as it requires Fourier transforms of data which are distributed across different PEs, and it is described in detail in Section (4). I/O issues are detailed in Section (5), while Section (6) discusses code efficiency, and Section (7) describes the testing of the new code. Finally, up-to-date lists of model parameters and namelist variables are given in the appendices.

Details of the GC (Generalized Communication) routines which are used as an interface to the PVM (Parallel Virtual Machine) message passing system are not given here, but documentation can be found in Amundsen and Skålin (1996). As currently coded the model cannot be used with the alternative (slightly faster) SHMEM (Shared Memory) message passing system but changes to allow its use may be made at some future date.

2 Domain decomposition

The momentum (u_i) and conserved scalar (q_n) equations used by the model are, as given by Equations (9) and (12) of TDN 213 with the material derivatives expanded using Equation (11), the following :

$$\frac{\partial u_i}{\partial t} = -\frac{\partial}{\partial x_i}(p'/\rho_s) - u_i\frac{\partial u_i}{\partial x_i} + \delta_{i3}B' + \rho_s^{-1}\frac{\partial \tau_{ij}}{\partial x_j} - 2\varepsilon_{ijk}\Omega_j u_k = -\frac{\partial}{\partial x_i}(p'/\rho_s) + s_i \qquad (1)$$

and

$$\frac{\partial q_n}{\partial t} = -u_i\frac{\partial q_n}{\partial x_i} - \rho_s^{-1}\frac{\partial h_{n,i}}{\partial x_i} \qquad (2)$$

Here τ_{ij} is the subgrid stress tensor and $h_{n,i}$ is the subgrid scalar flux of q_n. Note that all of the terms which make up s_i on the right-hand side of Equation (1), can be calculated without the use of information from more than two grid-points away from the point currently being stepped. The same is true of the terms on the right-hand side of Equation (2). Hence the domain can be decomposed into a number of sub-domains (with overlapping halos), and the source terms for the points in each sub-domain can be calculated concurrently by separate processing elements in a massively-parallel processing (MPP) machine. As the calculation of these source terms typically accounts for most of the CPU time used by the large-eddy model, this sort of decomposition can be expected to lead in a significant reduction in the total elapsed time when performing a given simulation. It may also be possible to perform simulations which were previously impossible due to memory constraints when running on a single processor.

There are a large number of possible domain decompositions. For example, it would be possible to attempt a full 3-dimensional decomposition, with each PE responsible for a small cube of points within the domain. Alternatively, a 2D decomposition might split the domain in the $x-$ and $y-$directions, but keep all vertical columns on a single PE. Multiple directional decompositions have the advantage that, for a given domain size and number of processors, they lead to fewer internal boundaries than a 1D decomposition. This potentially reduces the amount of communication between processors which is required. However, the coding is much more complex, and it was decided to use a 1D decomposition for the large-eddy model. This was made particularly straightforward by the 'slice' structure of the existing model, and, as discussed in Section (6), the efficient performance obtained appears to justify the decision.

Figure (1) shows the domain decomposition used. Here a run involving IIP slices is split over 4 PEs (NPES=4). Each PE is responsible for IIPEP=IIP/4 slices. In order to step the fields on each slice, information is required from two slices either side, and so each PE also has a double halo on each side, reset at the beginning of each time step, to enable it to calculate the source terms for its slices 1, 2, IIPEP-1 and IIPEP [1]. This structure means that many of the routines used in the calculation of the source terms (e.g. SMAG, UVWSRCE, THSOURCE, QSOURCE, MICROPHYS) are largely unaffected by the domain decomposition, and a user who wishes only to make minor changes to these routines may be able to do so without a detailed understanding of the changes required elsewhere in the model to allow running on a massively parallel machine.

Of course, not all of the calculations can be done in parallel. In particular, the solving of the Poisson-like elliptic equation for p'/ρ_s requires the values of the source terms (s_i) from all points in the domain, and information must be exchanged between PEs through message passing. However, even here parts of the calculation can be done in parallel (e.g. the tridiagonal solver which works independently on each column of data) and, once the pressure field has been calculated, the pressure source terms can be be calculated independently on each PE.

[1] Calculation of backscatter source terms requires information from three slices either side and so triple halos are used

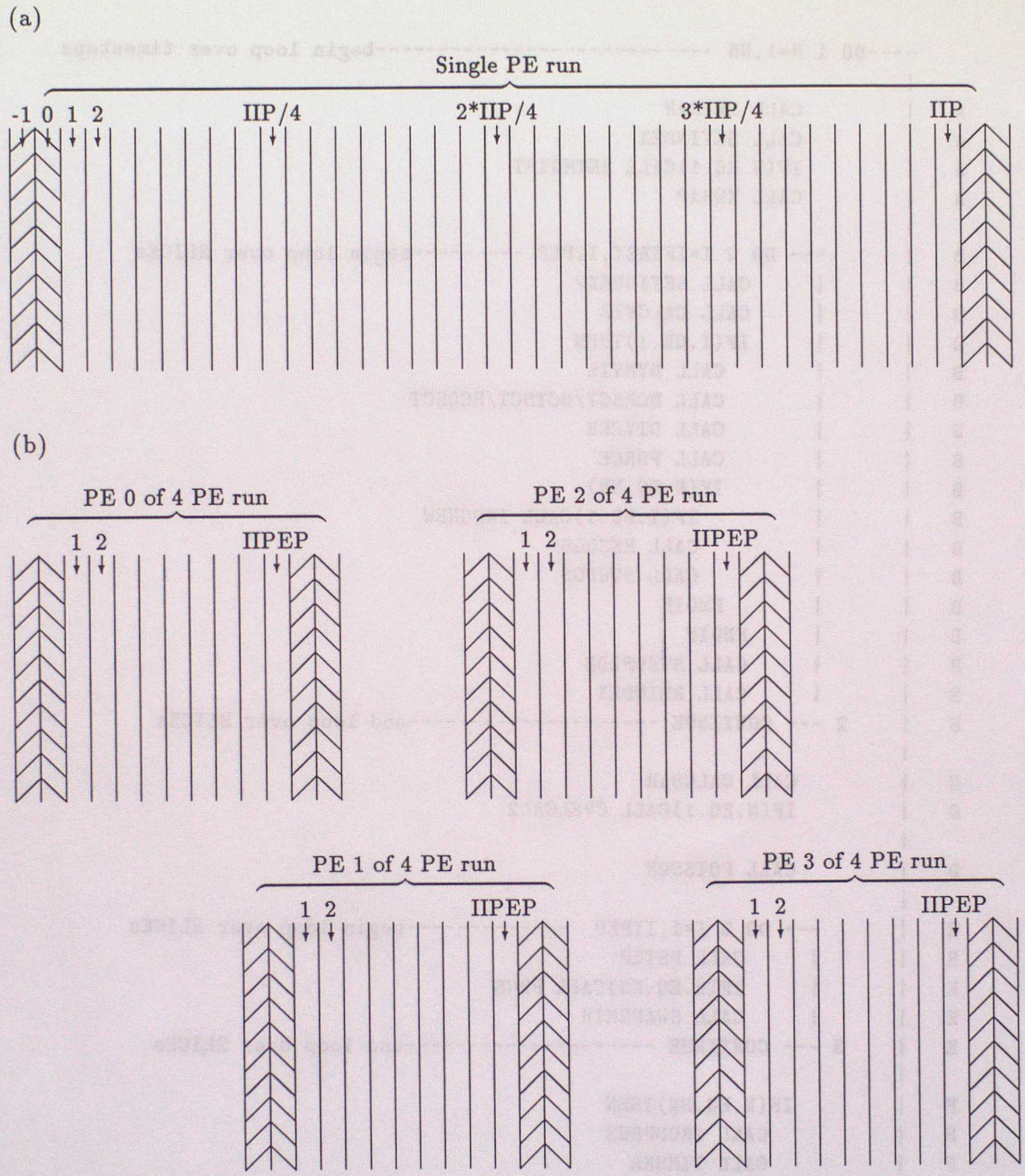

Figure 1: Explanation of domain decomposition. (a) Illustration of a single processor run. Slices 1 to IIP are stepped, with the aid of halo slices (shown filled) -1, 0, IIP+1 and IIP+2. Halo slice -1 is a copy of IIP-1, 0 is IIP, IIP+1 is 1 and IIP+2 is 2 in order to enforce periodicity. (b) The same run, now performed on 4 PEs. Each PE deals with IIPEP (=IIP/4) slices, with a double halo at each side. For example, PE 2 steps its slices 1 to IIPEP (slices 2*IIP/4+1 to 3*IIP/4 of the single processor problem). It obtains its halo slices -1 and 0 by copying slices IIPEP-1 and IIPEP from PE 1, and IIPEP+1 and IIPEP+2 by copying slices 1 and 2 from PE 3. The external halos (slices -1 and 0 of PE 0 and IIPEP+1 and IIPEP+2 of PE 4) are set to impose the periodicity as before.

```
       ----DO 1 N=1,NN --------------------------begin loop over timesteps
           |
     A     |        CALL SETRAN
     A     |        CALL SETINDEX
     A     |        IF(N.EQ.1)CALL SETMOIST
     A     |        CALL XWRAP
           |
     B     |      --- DO 2 I=IFIRST,IIPEP ---------begin loop over SLICEs
     B     |      |     CALL SETINDEX2
     B     |      |     CALL CALCVIS
     B     |      |     IF(I.GE.1)THEN
     B     |      |       CALL DYNVIS
     B     |      |       CALL BCKSCT/BCTSCT/BCQSCT
     B     |      |       CALL DIVERR
     B     |      |       CALL PSRCE
     B     |      |       IF(N.EQ.NN)
     B     |      |         IF(I.EQ.1)CALL INDGNEW
     B     |      |         CALL RESDGS
     B     |      |         CALL SUBDGS
     B     |      |       ENDIF
     B     |      |     ENDIF
     B     |      |     CALL STEPFLDS
     B     |      |     CALL REINDEX
     B     |    2 --- CONTINUE --------------------end loop over SLICEs
           |
     C     |        CALL CALCBAR
     C     |        IF(N.EQ.1)CALL CVELGAL2
           |
     D     |        CALL POISSON
           |
     E     |      --- DO 3 I=1,IIPEP  -------------begin loop over SLICEs
     E     |      |     CALL PSTEP
     E     |      |     IF(N.EQ.NN)CALL PDGS
     E     |      |     CALL SWAPSMTH
     E     |    3 --- CONTINUE --------------------end loop over SLICEs
           |
     F     |        IF(N.EQ.NN)THEN
     F     |          CALL GROUPDGS
     F     |          CALL TIMSER
     F     |        ENDIF
     F     |        IF(N.EQ.1)CALL TESTCFL
           |
       ----1 CONTINUE -------------------------------end loop over timesteps

         CALL AVDG
         CALL TESTTIME
```

Figure 2: Subroutine NNSTEPS in Version 2.0.

3 Subroutine NNSTEPS in Version 2.0

Subroutine NNSTEPS is the central subroutine of the model, and an understanding of it is the key to understanding the way the model works. It has undergone considerable restructuring, both to allow running on a massively parallel machine, and to remove features no longer required. These include the 'packed arrays' (so that the main model fields are now stepped directly), and the expanded arrays previously set up in SET2D (superfluous on a scalar machine). A desirable side-effect of the changes is that they have made the code generally more straightforward and easier to understand.

Figure (2) shows the structure of subroutine NNSTEPS. The letters at the left edge of the diagram show how the routine can be split into six main sections, A–F. The purpose and structure of each of these is now discussed in turn.

3.1 Section A – Initialization

SETRAN	Set up random numbers for backscatter (called if IBSCATP.EQ.1).
SETINDEX	Set up pointers for the NVISP slices of viscosities and the NDISP slices of dissipations (as before). The main fields are now accessed directly (see SETINDEX2 in Section B).
SETMOIST	Recalculate QSAT and various other related quantities using current mean states rather than reference states (called if IQLCALCP.EQ.1).
XWRAP	Wrapping in the x-direction. Halo slices I=−1,0,IIPEP+1 and IIPEP+2 (and −2 and IIPEP+2 if using backscatter) are updated on each PE. This involves message passing (subroutines WRAPSEND and WRAPRECV) unless NPES=1 (in which case it can be done directly).

3.2 Section B – Calculation of source terms and stepping

SETINDEX2	Set up pointers for main fields. Note that the main fields are now used directly, rather than by copying slices from 'packed storage' as was done previously.
CALCVIS	Calculate viscosities (on slice I+1 without backscatter; on I+2 with backscatter.
DYNVIS	Calculate source terms (except pressure and backscatter).
BCKSCT BCTSCT BCQSCT	Calculate backscatter source terms.
DIVERR	Calculate divergence error of main fields and place in pressure source (P).
PSRCE	Calculate divergence of source of terms, and add it to pressure source. Previously called from DYNVIS but moved to be after calculation of backscatter source terms.
INDGNEW RESDGS SUBDGS	Calculation of diagnostics (non-pressure) for N.EQ.NN. Results are held on each PE in DGNEW and represent averages over the part of the domain on that PE in each case.

STEPFLDS	Previously called WRTFLDS but renamed as main fields are now stepped directly, rather than writing 'packed fields'. This means that no slice can be stepped until it is no longer required on the present timestep – accordingly NSLP=3 slices of source terms are stored, and slice I−2 is stepped for I=3 to IIPEP, with IIPEP−1 and IIPEP additionally stepped for I=IIPEP. The Galilean transformation is also added back on to horizontal velocity fields. CVELGAL is still called for N.EQ.1 but its calculation of new Galilean transformation and CFL criteria can only be partially completed at this point – information is required from all PEs to complete the calculation, and this is done in CVELGAL2 in Section C (outside the main loop over I). Similarly, calculation of BARS is deferred until CALCBAR in Section C.
REINDEX	Reset (shuffle) the pointers for the slices of viscosity and dissipation.

3.3 Section C – Calculation of means and CFL criteria

CALCBAR	Calculate mean fields. The results should always be reproducible for a fixed number of PEs, but identical results across a variable number of PEs can only be obtained by using IREPBARP=1 (slight overhead). The mean fields are not strictly required every step unless the damping layer is used, but at present are calculated anyway for convenience
CVELGAL2	Complete calculation of new Galilean transformation and CFL criteria (started in CVELGAL which is called from STEPFLDS in Section B).

3.4 Section D – Pressure solver

POISSON	Complete pressure source calculation. CALL FFANAL to perform 2D-Fourier transform, use tridiagonal solver and then call FFSYNTH to perform inverse 2D-Fourier transform. More details can be found in Section (4).

3.5 Section E – Pressure stepping

PSTEP	Subtracts new Galilean transformation and performs pressure stepping. Also sets level 1 velocities.
PDGS	Calculates diagnostics involving pressure (averages across each PE at this stage).
SWAPSMTH	Swaps and time-smooths main fields and BARS.

3.6 Section F – Miscellaneous

GROUPDGS	Completes calculation of diagnostics by combining DGNEWs from all PEs. Also calculates those TKE budget diagnostics which could not be calculated until various domain-averaged quantities were available. Results will be reproducible for a fixed number of processors, but will depend on NPES.

TIMSER	Calculate time series.
TESTCFL	Calculate new timestep.

4 Pressure Solver

The calculation of the pressure term, $[\frac{\partial}{\partial x_i}(p'/\rho_s)]$, in equation (1) requires the solution of a Poisson-like elliptic equation. This is solved using a Fourier-transform in the horizontal and a tridiagonal matrix solver in the vertical. The horizontal Fourier transformations, the calculation of the elliptic equation source terms and the maximum divergence error calculation are all affected by the move to a MPP machine.

4.1 Horizontal Fourier Transform

Horizontal data is partitioned between the PEs, but all the data is required to perform a 2-D fast Fourier transform (FFT). Machine specific utility routines for 2-D FFTs exist on the T3E. These perform a single FFT in one direction, rearrange the data so that the x- and y-directions are swapped, perform the second FFT operation, and return the data to its original position. Because the pressure solver requires two 2-D FFTs (converting the real data to wavespace and then back again) the shifting of data back to the original position is a redundant time-consuming overhead. The preferred approach is to perform the 2-D FFT explicitly by use of single FFTs (which are not necessarily machine specific) and message passing using only one data rearrangement per transformation.

As the single FFTs only require data in one direction, the decomposition of the domain into I slices makes the y-direction FFT trivial. To perform the x-direction FFT the data must be rearranged so that the domain can be decomposed into J slices, as illustrated in Figure (3). Not only does data need to be transferred between PEs but, in order to ensure faster floating point arithmetic and FFTs, the order of the pressure array must be changed. Pressure (and pressure source) is held in an array P(J,K,I). This is transposed to an array PT(I,K,J) for the x-direction FFT and tridiagonal calculation. This transposition is performed by decomposing each PE's P array into NPES sections in the y-direction and transposing each section to PT in turn, using the new subroutine TRANSPOSE. The data is transferred as illustrated in Figure (3c). The algorithm loops over NPES and is shown below. If the PE number matches the loop counter (NP) then that PE receives data messages, else the PE sends messages to the PE identified by NP. A separate loop controls the receipt of messages. The entire procedure is reversed for the inverse FFTs after completion of the tridiagonal solver. This message sending algorithm scales as NPES(NPES-1) and hence becomes a proportionately larger overhead as NPES increases.

```
      DO NP=0,NPES-1
        IF(MYPE.NE.NP)THEN
C---------------------------------------
C Send packets to all other processors
C---------------------------------------
          JSTART=NP*JJPEP
          CALL TRANSPOSE(....)
          CALL GC_RSEND(....)
        ELSE
C-------------------------
C Do local transposition :
```

```
C--------------------------
        JSTART=MYPE*JJPEP
        CALL TRANSPOSE(....)

       ! transfer data to PT

       ENDIF
     ENDDO
C----------------------------------------
C Receive packets from the other processors
C----------------------------------------
     DO NP=0,NPES-1
       IF(MYPE.NE.NP)THEN
         CALL GC_RRECV(....)

       ! transfer data to PT

       ENDIF
     ENDDO
```

When the model is used in 2-D (y-z) mode on a single PE neither message passing nor transposing are required. However, because the tridiagonal algorithm is set up for using PT the pressure source data must be transferred from P to PT. PT is dimensioned with intelligent parameter statements so that it is PT(IIP,KKP,JJPEP) in 3-D and PT(JJP,KKP,1) in 2-D.

4.1.1 Elliptic Equation Source Terms

The source term for the elliptic equation contains

$$\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}\right) \tag{3}$$

Because the domain is decomposed in the x-direction this requires message passing for the IIPEP slice of the source term. Rather than passing the $\partial u/\partial t$ data between PEs, part of the $\partial/\partial x$ term is calculated on the adjacent PE and stored in the P halo. This is then transferred to the appropriate PE at the beginning of POISSON to complete the pressure source calculation.

The order of the calculation in PSRCE is changed slightly to mimic this separate summation so as to ensure bit reproducibility between identical runs with different NPES:

```
                                 ! Bracketing and order of calculation
         P(J,K,I) = (P(J,K,I) +  ! ensures reproducibility between
     &    CX*SU(J,K) +           ! runs with different NPES
     &    CY*(SV(J,K)-SV(J-1,K))+
     &    4.*(TZC2(K)*SW(J,K)-TZC1(K)*SW(J,K-1)) )
     &     - ( CX*SUM1(J,K) )
```

4.1.2 Divergence Error Calculation

In order to output the maximum divergence error in the model the local maximum divergence error on each PE needs to be calculated. The global max-min routines do not output upon

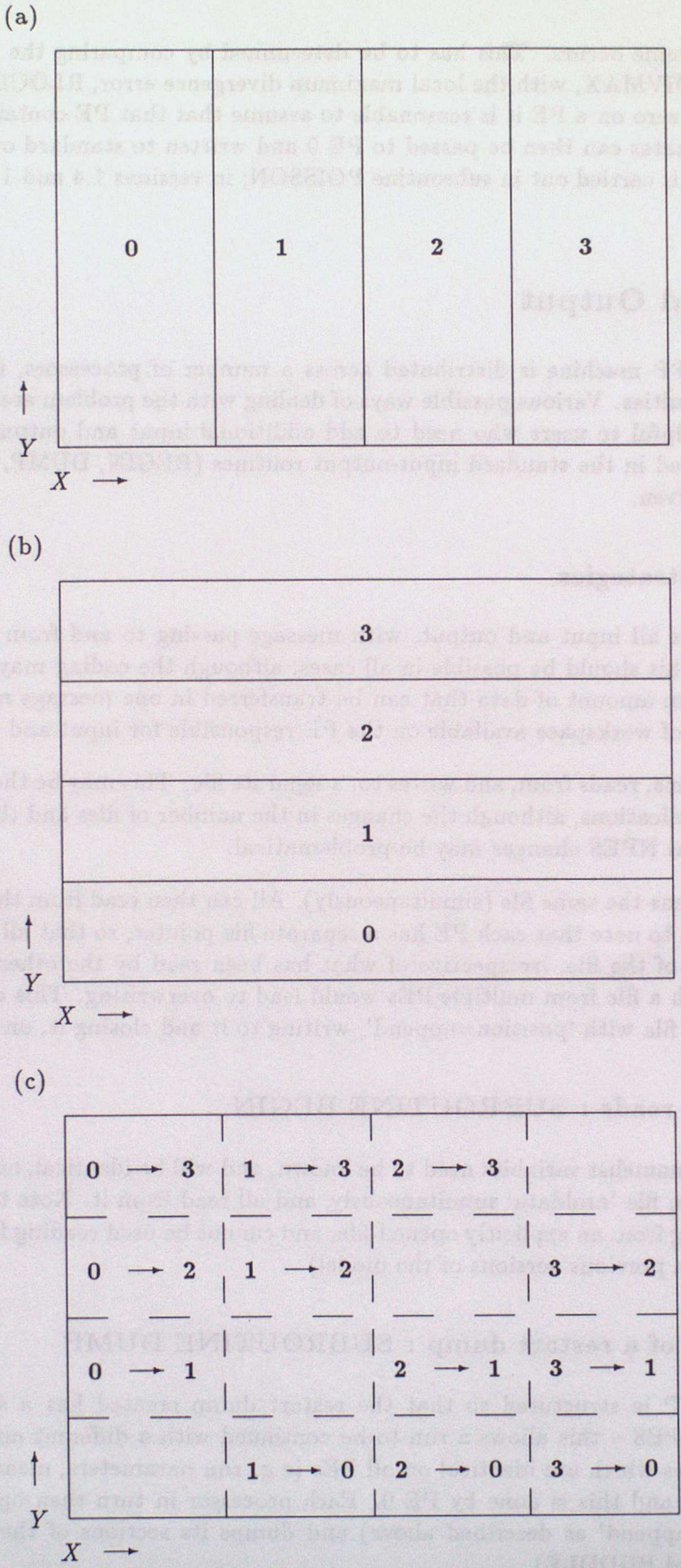

Figure 3: Schematic diagram of the arrangement of data in a horizontal slice of the model for four processors: (a) standard arrangement for *x*-direction FFT; (b) arrangement for *y*-direction FFT; (c) data transfers required to go from (a) to (b).

which PE the extreme occurs. This has to be determined by comparing the global maximum divergence error, DIVMAX, with the local maximum divergence error, RLOCDIVMAX. If they are equal and non-zero on a PE it is reasonable to assume that that PE contains the maximum value. The coordinates can then be passed to PE 0 and written to standard output. In version 2.0 this procedure is carried out in subroutine POISSON; in versions 1.4 and 1.5 this took place in DIVERR.

5 Input and Output

As data on an MPP machine is distributed across a number of processors, input and output presents new difficulties. Various possible ways of dealing with the problem are suggested below, as this may be helpful to users who need to add additional input and output. More specifics on the strategy used in the standard input-output routines (BEGIN, DUMP, RETRIEVE and DIAG) are then given.

5.1 Possible strategies

- One PE does all input and output, with message passing to and from other PEs where necessary. This should be possible in all cases, although the coding may be complex, and the maximum amount of data that can be transferred in one message may be limited by the amount of workspace available on the PE responsible for input and output.
- Each PE opens, reads from, and writes to, a separate file. This may be the easiest approach for some applications, although the changes in the number of files and the amount of data in each file as NPES changes may be problematical.
- Each PE opens the same file (simultaneously). All can then read from that file although it is important to note that each PE has a separate file pointer, so that all will start reading at the start of the file, irrespective of what has been read by the others. Attempting to write to such a file from multiple PEs would lead to overwriting. This can be avoided by opening the file with 'position=append', writing to it and closing it, on each PE in turn.

5.2 Namelist reads : SUBROUTINE BEGIN

The values of the namelist variables need to be known, and will be identical, on all PEs. Accordingly, all PEs open file 'nmldata' simultaneously, and all read from it. Note that this approach only works reading from an explicitly opened file, and cannot be used reading from unit 6 (stdin) which was done in previous versions of the model)

5.3 Creation of a restart dump : SUBROUTINE DUMP

Subroutine DUMP is structured so that the restart dump created has a structure which is independent of NPES – this allows a run to be continued with a different number of PEs if so desired. Quantities which are identical on all PEs (e.g. run parameters, mean fields) need only be dumped once, and this is done by PE 0. Each processor in turn then opens the dump file (with 'position=append' as described above) and dumps its sections of the distributed fields (U,V,W,TH,Q and PUDDLE).

5.4 Reading of a restart dump : SUBROUTINE RETRIEVE

Subroutine RETRIEVE is rather more complicated than DUMP, as for reading, there is no analogue of the method whereby data could be dumped to the same file by different PEs simply by repeatedly closing and re-opening the file. Accordingly message passing is used.

All PEs open the dump file and read the quantities which are required by them all (e.g. run parameters, mean fields). The last processor (NPES-1) then reads the part of the distributed fields required by PE 0, and sends that data to PE 0. It repeats this process for each PE in turn, until it reads its own part of the distributed fields, at which point the retrieve operation is complete.

5.5 Diagnostics dump : SUBROUTINE DIAG

Subroutine DIAG has undergone major changes in the rewrite to version 2.0. Part of this is due to the MPP aspects, but mostly it is a desire to tidy up the routine and produce a diagnostics file which is compatible with the PV-Wave based graphics package DaViE. This requires the output of a character header to the diagnostics file (or to a separate file which can later be appended to the main diagnostics file). Here, only details relevant to MPP machines and new diagnostic control parameters will be discussed.

5.5.1 New Diagnostic Parameters

The diagnostic namelist variables IDGxx have been adjusted to accommodate the increased flexibility of output that DaViE permits. Fields can be output as instantaneous 1-D averages (IDGxx=1), an array of 2-D slices (IDGxx=2), or as a complete 3-D field (IDGxx=3). If IDGxx=2 or 3 the 1-D averages will also be output. It is not possible to output both 2-D slices and a complete 3-D array for the same field. 2-D slices are defined using NITEST,NJTEST and NKTEST (the number of y-z, x-z and x-y slices respectively) and XTEST, YTEST and ZTEST (arrays of the position of the required slices in metres, assuming that the centre of the horizontal domain has values of x and y of zero). A full list of arrays of the IDGxx variables is given in Appendix A.2.

In addition, model parameters, the x, y and z grids and density are automatically output in each diagnostic dump.

5.5.2 MPP Aspects

One of the requirements in producing a diagnostics file is that the reading of the dataset should be independent of NPES. In addition, fields should be in normal Cartesian order, i.e. (x,y,z) as opposed to the way in which fields are stored in the model - (J,K,I). To avoid excessive demands on memory larger fields have to be written out piecemeal rather than in entirety. This requires a greater manipulation of data and, because this is repeated for each field, leads to a modularization of DIAG. The new structure of DIAG is shown in Figure (4). Message passing algorithms are contained in routines WRTXZ, WRTYZ and WRTXY whilst the actual writing to file is carried out in WRTFB.

WRTFB

All writing is carried out through PE 0. Because the number of calls to WRTFB will depend upon the messages passed, and hence the number of processors, to maintain NPES independence

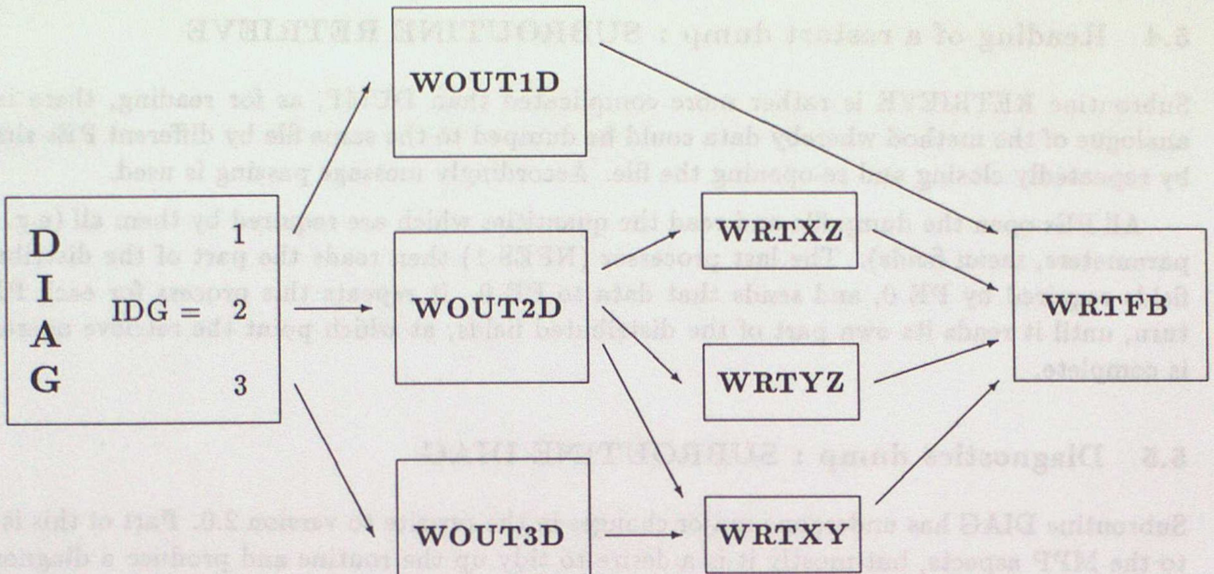

Figure 4: Schematic diagram of the new modular structure of subroutine DIAG.

a new record must not be started each time WRTFB is called, as occurred in previous versions of the LEM. WRTFB has been rewritten so that it need not start a new record on each call but can complete a previously partially filled record. (The routine was originally written by J.M. Hobson for the Blasius flow over hills code).

Because the data is "squashed" from REAL*8 to REAL*4 for output, if there is an odd number of data items a gap will appear in final data string. Although this is unimportant if a new record is to be started, it will lead to a corruption of the data if partially filled records are completed. To ensure that all fields have an even number of data points they are written out over IIDGP as opposed to IIP, where IIDGP=(IIP/2)*2 and, hence, is always even.

WRTYZ

The writing of y-z cross-sections is relatively trivial. Because the I slice is contained in entirety on a processor, all that is required is to identify the PE upon which the slice in question resides, transfer it to PE 0, and then write to file.

WRTXZ

For an x-z cross-section the data is distributed across all PEs. Each PE sends its subset of the relevant slice to PE 0 where it is assembled in a temporary array. Care is taken not to include the internal halos. Once the temporary array is complete it can be written out in the normal way.

WRTXY

WRTXY involves the most complicated message passing and is used for both horizontal slices and writing out complete 3-D fields. The data needs to be reordered as (x,y,z) and this is achieved by passing data in packets of size IIPEP by JJPEP from each PE in turn. A temporary array of dimensions (IIP,JJPEP) is filled on PE 0 which is then written out to the diagnostics file. This procedure is continued for each subsequent block of JJPEP until the complete horizontal slice is written. Halos at I=0 and I=IIDGP+1 are written in a similar way to the x-z cross-sections. In WOUT3D calls to WRTXY are looped over K=1,KKP to write out the complete 3-D field.

6 Scalability and efficient use of the code

The large-eddy model runs efficiently on the vector C90, typically at around 400 MFlops. On a single processor of the scalar T3E, the figure is closer to 60 MFlops. It is sometimes possible to make improvements in scalar performance by making changes to the code so that cache is used more efficiently, and explicit compiler directives have been placed in some of the more expensive routines (e.g. ULTFLX, SETFRI) to prevent the compiler unrolling loops in cases where this has been found to be disadvantageous. However, whilst some further improvements in scalar performance could undoubtedly be made, it seems unlikely that they would be dramatic, particularly as the optimum coding for one problem size might not be optimum for a different problem. Hence the key to obtaining reasonable throughput on the T3E is probably to ensure that the model scales well. By this it might be meant that the the elapsed time taken to solve a given problem should be proportional to (1/NPES), or alternatively that a problem of size proportional to NPES will run in the same time for any value of NPES. In practice neither of these will be achieved, and the first is a particularly difficult limit to approach. Some reasons why the model might not scale well are given below:

1. As NPES increases, the amount of time spent communicating between the PEs will increase. Note especially that the number of messages required in the pressure solver increases roughly as the square of NPES.
2. As more PEs are used to solve a fixed problem, the amount of time performing parallel work decreases. Hence the non-parallel overheads such as communication, input and output, and duplicate calculations (e.g. calculation of viscosities on halo slices) take an increasing fraction of the overall time (even before allowing for increasing amount of time performing these overheads) and the scalability suffers accordingly.
3. If not all the PEs have the same amount of work to perform, then the model can only run at the speed of the ones which have most work to do. This load imbalancing is unlikely to be a major problem for the standard large-eddy model as each PE has the same number of points and the amount of work on each point is the same, although it might become an issue in a run with complex microphysics in which extra calculations were required in some parts of the domain.

In spite of these problems, the scalability of the large-eddy model is encouragingly good for most practical applications. As an example, Table 1 shows timings for various 50 step runs with one active Q-field and ULTIMATE advection on scalars (without a dump). Runs A to E used an increasing number of PEs to solve proportionally larger problems (the largest being far beyond any that could be run if the dump were carried out). If the model scaled perfectly, the CPU time would increase linearly with NPES, while the elapsed time would stay constant. This is

Run	Size of run	NPES	IIPEP	Elapsed time (s)	CPU time (s)	Scalability
A	$128 \times 64 \times 61$	1	128	354	347	1.00
B	$256 \times 128 \times 61$	4	64	345	1361	1.02
C	$512 \times 256 \times 61$	16	32	356	5370	0.99
D	$1024 \times 512 \times 61$	64	16	388	23889	0.93
E	$2048 \times 1024 \times 61$	256	8	477	118900	0.75

Table 1: 50 step timing tests with the size of the problem proportional to NPES.

Run	Size of run	NPES	IIPEP	Elapsed time (s)	CPU time (s)	Scalability
B4	$256 \times 128 \times 61$	4	64	345	1361	1.00
B8	$256 \times 128 \times 61$	8	32	187	1356	0.92
B16	$256 \times 128 \times 61$	16	16	95	1398	0.91
B32	$256 \times 128 \times 61$	32	8	61	1566	0.70
B64	$256 \times 128 \times 61$	64	4	40	2266	0.54
B128	$256 \times 128 \times 61$	128	2	57	5000	0.19

Table 2: 50 step timing tests with the size of the problem constant.

approximately true for cases A, B and C, but the scalability (defined as the ratio of the elapsed time for the 1 PE run to the elapsed time using NPES) starts to fall off for cases D and E. However, even in case E the scalability remains above 0.7 and it can be concluded that the model scales well, as long as the amount of work to be performed on each PE is kept constant.

Of course, the more important type of scalability may be that of running a fixed problem on a variable number of PEs. Table 2 shows some timings from running Case B on an increasing number of PEs and the scalability figures are based on elapsed time relative to that taken with 4 PEs. In this case perfect scaling would be indicated by the CPU time being independent of NPES, while the elapsed time would be proportional to (1/NPES). The amount of parallel work on each PE decreases as NPES increases and IIPEP decreases (point (2) above), and so the scalability falls off more quickly with NPES than in Table 1. However, the use of the computer remains reasonably efficient with NPES equal to 16 or even 32, and these values are probably reasonable choices when performing a run of this sort of size, as long as IIPEP does not fall below about 6. Note the extreme case of B128 which uses twice as many PEs as B64, but actually takes more elapsed time – not an efficient use of the computer!

7 Testing of the code

The code changes from Version 1.5 were made in a sequential fashion, with over 25 intermediate versions of the model. This meant that the changes introduced at any one time usually affected only relatively small sections of the code which made testing relatively straightforward.

The final code has been tested to show that is has the following properties:

- For a run on a fixed number of PEs, the code gives reproducible answers. This might not be the case if message passing were used directly (e.g. as additions in a global sum might be performed in a different order) but is ensured by handling communication through the GC routines (Amundsen and Skålin, 1996).

- For runs in which the BARS are purely diagnostic, identical answers are obtained for any value of NPES (including one). If the BARS are used (e.g. a damping layer is used) then this is still the case as long as IREPBARP is set to 1. This independence of the results to NPES is non-trivial to obtain, and sometimes requires careful structuring of the code to ensure that the order of arithmetic operations in identical for any NPES. However ensuring that this property is maintained when making changes to the code has proved valuable in checking the logic and debugging.

- A run may be continued from a restart dump on any number of NPES, independent of the value previously used.

A Appendices

A.1 Parameters in the model

There follows a list of the parameters in the model. Many have deliberately not been given default values in order to force the user to decide on the most appropriate values for their run. Note that all parameters with no default value must be set (even if the value chosen is irrelevant), otherwise the model will fail to compile.

PARAMETER	DEFAULT	MEANING
IIP	??	Number of points in x-direction.
JJP	??	Number of points in y-direction.
KKP	??	Number of points in z-direction.
NPES	??	Number of processors. Must be a factor of both IIP and JJP.
IBSCATP	??	=1 for backscatter (3-D runs only). =0 for no backscatter.
NQSCTP	??	Number of Q-fields to be backscattered (if IBSCATP=1).
NBEGSCATP	??	Timestep to begin backscatter (if IBSCATP=1).
ITVDUVWP	??	=1 for TVD advection of momentum. =0 for centred advection of momentum.
ITVDSCALP	??	=1 for TVD advection of scalars. =0 for centred advection of scalars.
IFORSCALP	??	=1 for forward stepping of scalars. =0 for centred stepping of scalars.
IFORUVWP	??	=1 for forward stepping of momentum. =0 for centred stepping of momentum.
IBAROCLP	??	=1 gives specified geostrophic shear and consistent advection of large-scale temperature gradient. =0 for no geostrophic shear.
IUSETHP	??	=1 to use TH variable. =0 to not use TH variable.
IPASTHP	??	=1 to use passive TH variable (if IUSETHP=1). This is also the required setting if IUSETHP=0. =0 to use active TH variable (if IUSETHP=1).
IUSEQP	??	=1 to use Q variable. =0 to not use Q variable.
IPASQP	??	=1 to use passive Q variable (if IUSEQP=1). This is also the required setting if IUSEQP=0. =0 to use active Q variable (if IUSEQP=1).
NQP	??	Number of Q variables (if IUSEQP=1). Must be set to 1 if IUSEQP=0.
IANELP	??	=1 to use anelastic equations. =0 to use Boussinesq equations.
IDAMPP	??	=1 to use damping layer. =0 to not use damping layer.
INOVISP	??	=1 to get inviscid solution. =0 otherwise.

IGALOFFP	??	=1 to switch off Galilean transformation. =0 to use Galilean transformation
INOSURFP	??	=1 to switch off surface fluxes. =0 otherwise.
ISCBCP	??	=1 to use fixed flux surface boundary condition for scalars. =2 to use fixed value surface boundary condition for scalars.
ISATSURFP	??	=1 to use saturation value of Q at the surface, rather than any value entered through namelist (only relevant for ISCBCP=2). =0 otherwise.
IADJANELP	??	=1,2,3,4 for various options for setting up anelastic profiles. Irrelevant for IANELP=0.
ITSERP	??	=1 to use time series. =0 to not use time series.
NTIMP	??	Number of bins in each time series (relevant only for ITSERP=1).
NSERP	??	Number of time series (relevant only for ITSERP=1).
ISPECP	??	=1 to calculate spectra. =0 to not calculate spectra.
NSPECP	??	Number of levels at which 1-D spectra in the $y-$direction of U, V, W and TH are to be calculated (relevant only for ISPECP=1).
MOISTRIP	??	=1,2 for two different moist Richardson number schemes. Irrelevant if IPASQP=1.
IRAINP	??	=1,2 to use two different rain schemes. =0 to not use rain.
NMETEORP	??	Number of hydrometeors (Q-fields with fall velocity). Irrelevant if IRAINP=0.
IREPBARP	??	=1 to get identical BARS irrespective of NPES. =0 for calculation of BARS to be sensitive (at truncation error level) to NPES.
ITWOFILEP	??	=1 to use two dump files, writing to them alternately. =0 to use one dump file.
IQLCALCP	??	=1 to calculate liquid water by performing Taylor expansion about mean temperature. =0 to calculate liquid water by performing Taylor expansion about reference temperature.
IFBCHGP	??	=1 to get time-varying surface fluxes (with ISCBCP=1). =0 for constant surface fluxes.
NDGSP	10	Maximum number of time-averaged diagnostics.
NAPARMSP	1024	Length of APARMS array (used in DIAG).
MAXDGP	300	Maximum length of DGCNT common block.
MAXQSP	10	Maximum length of QCNT common block.
LOOKP	80	Number of values in look-up table for surface boundary condition in unstable conditions (for ISCBCP=1).
SMALLP	1.E-14	A small number.
RLARGEP	1.E37	A large number.
NTIMPDGP	100	Maximum number of diagnostics printing times.
NTIMRDGP	100	Maximum number of diagnostics resetting times.
NTIMDUMP	100	Maximum number of diagnostics dumping times.
NTMHALTP	10	Maximum number of run halting times.

NTIMHFP	20	Maximum number of specified surface flux times (for IFBCHGP=1 with ISCBCP=1).
JMINP	1	Minimum index value in loops over J (no advantage in setting to 0 on scalar machine).
JMAXP	JJP	Minimum index value in loops over J (no advantage in setting to JJP+1 on scalar machine).

Additionally various other parameters are calculated automatically from those listed in the table.

PARAMETER	MEANING
I3DP	=1 for 3-D run; =0 2-D or 1-D.
IIPEP	Number of slices per PE (=IIP/NPES).
JJPEP	=JJP/NPES. Used in pressure solver.
IIDGP	=(IIP/2)*2. Used in DIAG in setting length of 'squashed' arrays.
IDIMMINP	Minimum I value in dimensioning of main field arrays. =−1 3-D, no backscatter; =−2 3-D, backscatter; =1 2-D or 1-D.
IDIMMAXP	Maximum I value in dimensioning of main field arrays. =IIPEP+2 3-D, no backscatter; =IIPEP+3 3-D, backscatter; =1 2-D or 1-D.
IMINP	=0. Used in pressure solver.
IMAXP	=IIP+1 3-D; =JJP+1 2-D. Used in pressure solver.
IPMINP	Minimum I value in dimensioning of pressure array. =0 3-D; =1 2-D or 1-D.
IPMAXP	Maximum I value in dimensioning of pressure array. $=IIPEP+1$ 3-D; =1 2-D or 1-D.
NSP	Number of slices of source terms held.
NVISP	Number of slices of viscosities held.
NDISP	Number of slices of dissipations held.
IFIRSTP	Starting value in first loop over I in NNSTEPS.
LENP	Length of a slice of a main field.
IBSCATTP	=IBSCATP*IUSETHP. Used in dimensioning various arrays to save space if IUSETHP=0.
IBSCATQP	=IBSCATP*IUSEQP. Used in dimensioning various arrays to save space if IUSEQP=0.
NFLDSCTP	Number of fields scattered. Used in setting up random numbers.
IMICROP	=0 if IRAINP=0; =1 otherwise. Used in dimensioning of PUDDLE, and as switch for dumping and retrieving of this array.
NMETP	=NMETEORP if IMICROP=1; =1 otherwise. Used in dimensioning of PUDDLE.
NTMP	=NTIMP if ITSERP=1; =0 otherwise. Used in dimensioning time-series array to save space if ITSERP=0.
NSPP	=NSERP if ITSERP=1; =0 otherwise. Used in dimensioning time-series array to save space if ITSERP=0.
INDTVDSP	=1 if ITVDSCALP=1; =0 otherwise. Used in dimensioning various arrays to save space if ITVDSCALP=0.
INDTVDMP	=1 if ITVDUVWP=1; =0 otherwise. Used in dimensioning various arrays to save space if ITVDUVWP=0.

LWTVDP	Length of work space required for TVD routines.
LWPP	Length of work space required for pressure solver.
IWHWRKP	Pointer for whether LWPP is greater than LWTVDP.
LENWRKP	Length of work space array. Set to the larger of NWTVDP and LWPP, although a user may set it to a larger value if any of their updates require more work space.

A.2 Namelists in the model

There are a number of model variables which can be set, after compilation, through the namelists given below. Again, not all have default values, although, unlike the parameters, some may be left unset if not relevant (see below). All dimensional quantities are in S.I. units.

NAMELIST CNTRL

VARIABLE	DEFAULT	MEANING
ISTART	??	=1 if a set-up run; =0 otherwise.
NN	??	Number of steps between diagnostic evaluations.
NNDIAG	??	Number of diagnostic evaluations between dumps.
NNDUMP	??	Number of dumps for this run.

Note that the total number of steps in a run is thus NN*NNDIAG*NNDUMP (plus an additional NN in a set-up run due to additional calls to NNSTEPS from START).

NAMELIST TIMENML

VARIABLE	DEFAULT	MEANING
NTMPDG	0	Number of diagnostic printing times to be used.
TIMPDG(NTIMPDGP)	??	Array of times at which to print the diagnostics. NTMPDG values must be set.
NTMRDG	0	Number of diagnostic resetting times to be used.
TIMRDG(NTIMRDGP)	??	Array of times at which to reset the diagnostics. NTMRDG values must be set.
NTMDUM	0	Number of field dumping times to be used.
TIMDUMP(NTIMDUMP)	??	Array of times to dump fields to disk. NTMDUM values must be set.
NTMHALT	1	Number of halting times.
TIMHALT(NTMHALTP)	??	Array of times to stop job. NTMHALT values must be set. Unlike the other time arrays, the model will stop if any of the values is exceeded, so there is little point in using NTMHALT greater than 1.
ITIMPDG	0	Flag for printing diagnostics.
ITIMRDG	0	Flag for resetting diagnostics.
ITIMDUM	0	Flag for dumping fields.
ITIMHALT	0	Flag for stopping job.
NSTEPMAX	50	Maximum number of timesteps in this job.
IPRTDG	1	Controls the diagnostic output (see below).
NTMHF	0	Number of specified surface flux times.

TIMHF(NTIMHFP)	??	Array of times for time-varying surface fluxes. NTMHF values must be set.
FSHFLX_SEN(NTIMHFP)	??	Array of values of surface sensible heat flux. NTMHF values must be set.
FSHFLX_LAT (NTIMHFP,NQP)	??	Array of values of surface latent heat flux. NTMHF values must be set for each Q-field.

Diagnostic output is controlled by IPRTDG as follows: 0 for output only at times given by TIMPDG and TIMRDG, 1 for output after every job, 2 for output at set times (possibly many times within one job) to a single file, 3 for output at set times (possibly many times in one job) to separate files.

NAMELIST JOBINFO

VARIABLE	DEFAULT	MEANING
FILEA	??	Character string containing name of first dump file.
FILEB	??	Character string containing name of second dump file.
FILEZ	??	Character string containing name of file containing pointer to dump file.
MSGFILE	??	Character string containing name of message file.
USERID	??	Character string used in naming diagnostic file.

NAMELIST INPUT

VARIABLE	DEFAULT	MEANING
NRUN	??	Run number.
NDATE	0	A number (eq. the date) to help identify the run.
TIME	0	Integration time.
Z0	??	Surface roughness length for momentum.
Z0TH	??	Surface roughness length for scalars (TH and all Q-fields).
PSF	100000	Surface pressure.
PSFR	100000	Surface reference pressure.
SHFLX_SEN	??	Surface flux of $\rho c_p \theta$ (only needs to be set for ISCBCP=1).
SHFLX_LAT(NQP)	??	Surface flux of $\rho L_v Q$ (only needs to be set for ISCBCP=1).
THSURF	??	Surface value of potential temperature (only needs to be set for ISCBCP=2).
QSURF	??	Surface value of first Q field (only needs to be set for ISCBCP=2).
RHOBOUS	??	Density for Boussinesq run (only needs to be set for IANELP=0).

NAMELIST GRID

VARIABLE	DEFAULT	MEANING
NSMTH	10	Number of 1-2-1 smoothings of grid heights.
HGD(20)	??	Array of heights for setting vertical grid. At least 1 value must be set.
KGD(20)	??	Corresponding array of K values for setting vertical grid.
ZZTOP	??	Height of domain top.

NAMELIST THPROF		
ZNREF_READ(9)	??	Array of heights for setting reference TH profile (θ_s) in anelastic run. At least one value must be set if IANELP=1.
THREF_READ(9)	??	Corresponding array of values for setting reference TH profile (θ_s).
THREF0	??	Reference TH (θ_0) for Boussinesq run. Must be set if IANELP=0.
ZNINIT_READ(9)	??	Array of heights which may be used in START for setting initial TH profile. At least one value must be set if used.
THINIT_READ(9)	??	Corresponding array of values for setting initial TH profile.

NAMELIST SUBMODEL

VARIABLE	DEFAULT	MEANING
SUBB	??	Subgrid model constant b.
SUBC	??	Subgrid model constant c.
SUBG	??	Subgrid model constant g.
SUBH	??	Subgrid model constant h.
SUBP	1.0	Subgrid model constant r; N.B. a value of 1 is currently hard-wired into the model and is used irrespective of the setting of this namelist variable.
SUBQ	1.0	Subgrid model constant r; N.B. a value of 1 is currently hard-wired into the model and is used irrespective of the setting of this namelist variable.
SUBR	4.0	Subgrid model constant r; N.B. a value of 4 is currently hard-wired into the model and is used irrespective of the setting of this namelist variable.
ATH2_N	??	Subgrid model heat flux correlation coefficient.
A2_N	??	Subgrid model stress-energy ratio.
PR_N	??	Subgrid model Prandtl number.
RIC	??	Subgrid model critical Richardson number.
RMLMAX	??	Subgrid model basic length scale λ_0.
SCT	0.0	Backscatter coefficient for momentum.
SCTT	0.0	Backscatter coefficient for TH.
SCTQ(NQSCTP)	0.0	Backscatter coefficient for Q.

NAMELIST DIAGNOST

VARIABLE	DEFAULT	MEANING
NITEST	0	Number of slices in J-K plane to be output.
XTEST(IIP)	0.0	Values of x at which to take slices (0 in centre of domain).
NJTEST	0	Number of slices in I-K plane to be output.
YTEST(JJP)	0.0	Values of y at which to take slices (0 in centre of domain).
NKTEST	0	Number of slices in I-J plane to be output.
ZTEST(KKP)	0.0	Values of z at which to take slices.
IDGU	1	Switch for U output.
IDGV	1	Switch for V output.
IDGW	0	Switch for W output.

IDGTH	1	Switch for TH output.
IDGTL	1	Switch for TL output.
IDGCL	0	Switch for cloud output.
IDGPV	0	Switch for potential vorticity output.
IDGP	1	Switch for P output.
IDGPD(NMETP)	0	Switch for PUDDLE output.
IDGQ(NQP)	1	Switch for Q output.
IDGAV	1	Switch for time-averaged diagnostics output.
NSPLEVS	0	Number of levels at which 1-D spectra of U, V, W and TH are required
ZSP(KKP)	0.0	Heights at which to calculate spectra.
IDGSP	1	Switch for spectra output.

The switches, IDGU etc, work in the following manner. 0 gives no output; 1 gives mean profiles; 2 gives 2D slices and mean profiles; 3 gives 3D fields and mean profiles. Some options may not be applicable for some fields (e.g. time-averaged diagnostics are 1D profiles and so cannot be dumped as 2D or 3D fields).

NAMELIST DYNAMICS

VARIABLE	DEFAULT	MEANING
FCORIOL	0.0001	The Coriolis parameter
UG0	??	x-component of the geostrophic wind at the surface.
VG0	??	y-component of the geostrophic wind at the surface.
DUGDZ	??	The rate of change of of the x-component of the geostrophic wind components with height. Need be set only if IBAROCLP=1.
DVGDZ	??	The rate of change of of the y-component of the geostrophic wind components with height. Need be set only if IBAROCLP=1.

NAMELIST NUMERICS

VARIABLE	DEFAULT	MEANING
DTM	??	Timestep.
TSMTH	0.01	Time-smoothing factor.
DXX	??	Horizontal grid spacing in x-direction (constant). Not required for 1D or 2D runs.
DYY	??	Horizontal grid spacing in y-direction (constant). Not required for 2D runs.
RINCMAX	0.05	Incremental factor for increasing timestep.
DTMMAX	1000.0	Maximum value for the timestep.
DTMMIN	0.01	Minimum value for the timestep.
CVISMAX	0.2	Maximum viscous CFL number.
CVELMAX	0.2	Maximum advective CFL number.
TOL	0.1	Tolerance for CFL numbers

NAMELIST PHYSICS		
VARIABLE	DEFAULT	MEANING
VK	0.4	von Karman constant.
ALPHAH	1.0	Monin-Obukhov α_h coefficient.
BETAM	4.8	Monin-Obukhov β_m coefficient.
GAMMAM	19.3	Monin-Obukhov γ_m coefficient.
BETAH	7.8	Monin-Obukhov β_h coefficient.
GAMMAH	12.0	Monin-Obukhov γ_h coefficient.
DFBMAX	0.0001	Maximum change in buoyancy flux between iterations in CHGBUOY.
CQ(NQP)	??	Array of coefficients for 'Q'-field contributions to buoyancy. The model overwrites the first element with CQ(1)=$R_v - 1$. Values must be set if NQP$\geq$ 2.

NAMELIST DAMPNML		
VARIABLE	DEFAULT	MEANING
DMPTIM	??	$1/\tau_{dmp}$, where τ_{dmp} is the damping layer time scale.
ZDMP	??	z_D, the height of the bottom of the damping layer.
HDMP	??	H_D, the damping layer height scale.

With the exceptions of CNTRL, TIMENML and JOBINFO, all of the above namelists are only read in a set-up job (ISTART=1). One other namelist, OVRIDE1, allows some of the namelist variables to be changed after the set-up.

References

Amundsen, J. and Skålin, R. (1996): GC User's Guide, Release 1.0.4, SINTEF Applied Mathematics.

Derbyshire, S.H., Brown, A.R., and Lock, A.P. (1994): The Meteorological Office Large-Eddy Simulation Model.
Met O (APR) Turbulence and Diffusion Note No. 213.